```
import cv2
import numpy as np
%matplotlib inline
# matplotlib.use('TkAgg')
import matplotlib.pyplot as plt
from matplotlib import colors, patches

plt.rcParams['figure.figsize'] = (12, 6)
```

1. Obdelava slik

1a) Preberite sliko iz datoteke umbrellas.jpg z uporabo knjižnice OpenCV (cv2.imread()) in jo prikažite z uporabo knjižnice MatPlotLib (plt.imshow()). Knjižnica OpenCV slike privzeto naloži v formatu BGR, zato jih moramo pred prikazom pretvoriti z ukazom cv2.cvtColor(image, cv2.COLOR_BGR2RGB).

```
ime_slike = 'umbrellas.jpg'
im = cv2.imread(ime_slike)
im = cv2.cvtColor(im, cv2.COLOR_BGR2RGB)
print(f'{im.shape=}')  # slika je shranjena kot visina x sirina x
kanali = h x w x 3, ker so kanali RGB
print(f'{im.dtype=}')

plt.clf()
plt.imshow(im)
plt.title('1a) Slika')
plt.show()
```

```
im.shape=(360, 640, 3)
im.dtype=dtype('uint8')
```

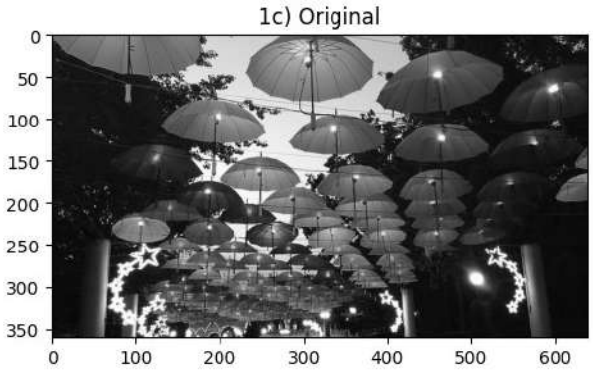
1b) Spremenite barvno sliko v sivinsko tako, da povprečite kanale in sliko zopet prikažite. Bodite pozorni na tip matrike, slika se namreč prebere v matriko tipa uint8, kjer element lahko zavzame samo vrednosti od 0 do 255. Računanje s takimi tipi je lahko problematično (npr. pri seštevanju pride do preliva, rezultat pa je napačen), zato matriko pred računanjem spremenite v drug tip, npr. np.float32. To lahko naredimo z ukazom im = im.astype(np.float32)

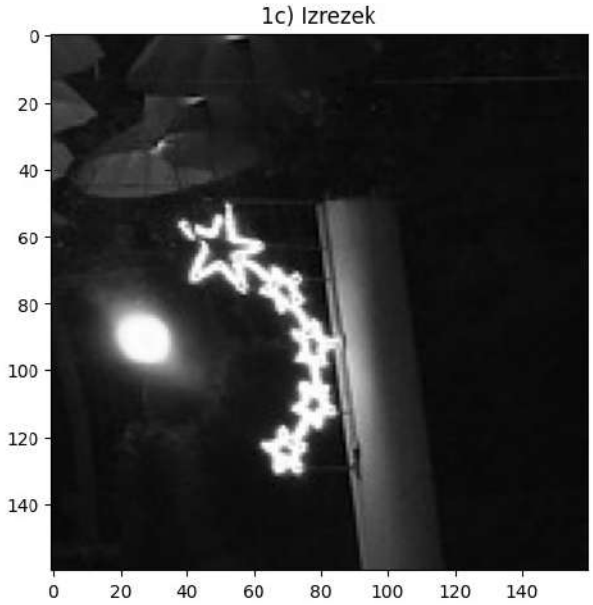
```
im = im.astype(np.float32)
for i in range(im.shape[0]):
    for j in range(im.shape[1]):
        pixel_value = im[i, j]
        siva = (pixel_value[0] + pixel_value[1] + pixel_value[2]) / 3
        im[i, j] = siva

im = im.astype(np.uint8)

# ali im = np.mean(im, axis=2)

plt.clf()
plt.imshow(im, cmap='gray')
plt.title('1b) Sivinska slika')
plt.show()
```

1c) Izrežite pravokotno regijo iz slike in jo izrišite kot novo sliko (lahko z uporabo plt.subplot()). Pravokotno regijo lahko določite s sintakso im[top:bottom, left:right, :]. Isto regijo v originalni sliki označite tako, da modri kanal postavite na nič.

```
print(f'{im.shape=}')
cut = im[200:360, 480:640, :]  # im[top:bottom, left:right, :]
plt.subplot(1, 2, 1)  # leva - plt.subplot(nrows, ncols, index)
plt.imshow(im, cmap="gray")
plt.title('1c) Original')
plt.subplot(1, 2, 2)  # desna
plt.imshow(cut, cmap="gray")
plt.title('1c) Izrezek')
plt.show()
```

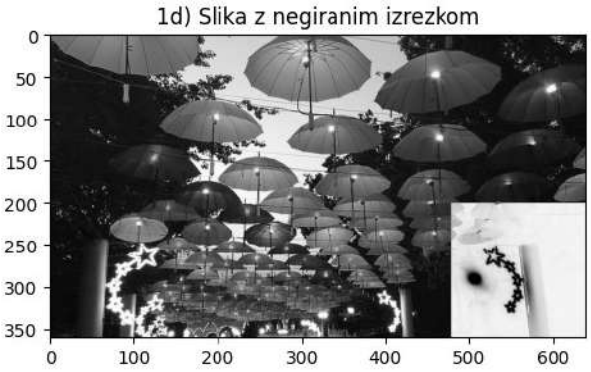
```
im.shape=(360, 640, 3)
```

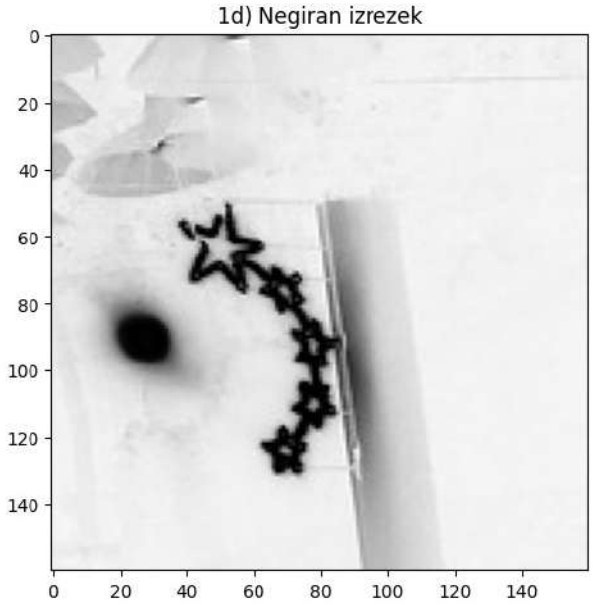




1d) Izrišite sivinsko sliko, kjer del sivinske verzije slike negirate. Razmislite, kako deluje negacija slike, če je ta zapisana v različnih podatkovnih tipih (uint8, float, int16).

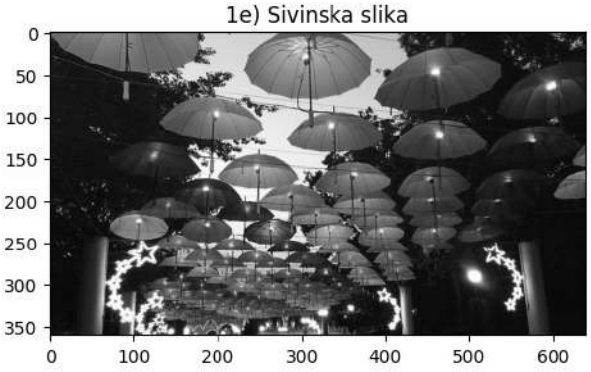
```
im2 = im.copy()
cut = 255 - im2[200:360, 480:640]
im2[200:360, 480:640] = cut

plt.subplot(1, 2, 1)
plt.imshow(im2, cmap="gray")
plt.title('1d) Slika z negiranim izrezkom')
plt.subplot(1, 2, 2)
plt.imshow(cut, cmap="gray")
plt.title('1d) Negiran izrezek')
plt.show()
```

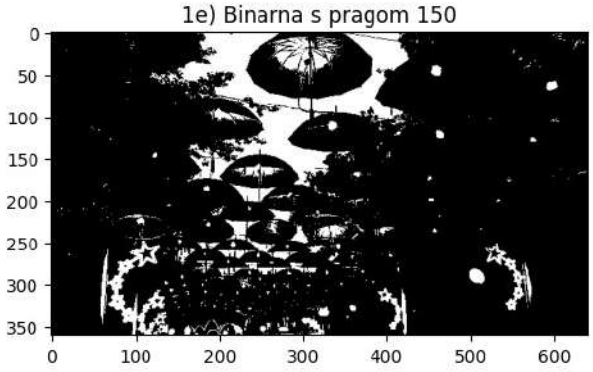




1e) Izrišite upragovano binarno sliko, v kateri vrednost 1 označuje elemente, ki imajo v izhodiščni sliki sivinski nivo večji od 150 in spremenite barvno lestvico v črnobelo.

```
im3 = (im > 150).astype(float)

plt.subplot(1, 2, 1)
plt.imshow(im, cmap="gray")
plt.title("1e) Sivinska slika")
plt.subplot(1, 2, 2)
plt.imshow(im3, cmap="gray")
plt.title("1e) Binarna s pragom 150")
plt.show()
```





2. Kamera

2a) Kockasta škatla z velikostjo stranice 10cm z majhno odprtino na prednji strani deluje kot kamera z luknjico. Usmerimo jo proti drevesu, ki je od kamere oddaljeno 14m. Kako velika je slika drevesa, ki nastane na zadnji strani škatle, če je drevo visoko 5m?

X=500cm, Z=1400cm, f=10cm
x= X*f*/*Z*
*x= 500*10/1400 = 3.57cm

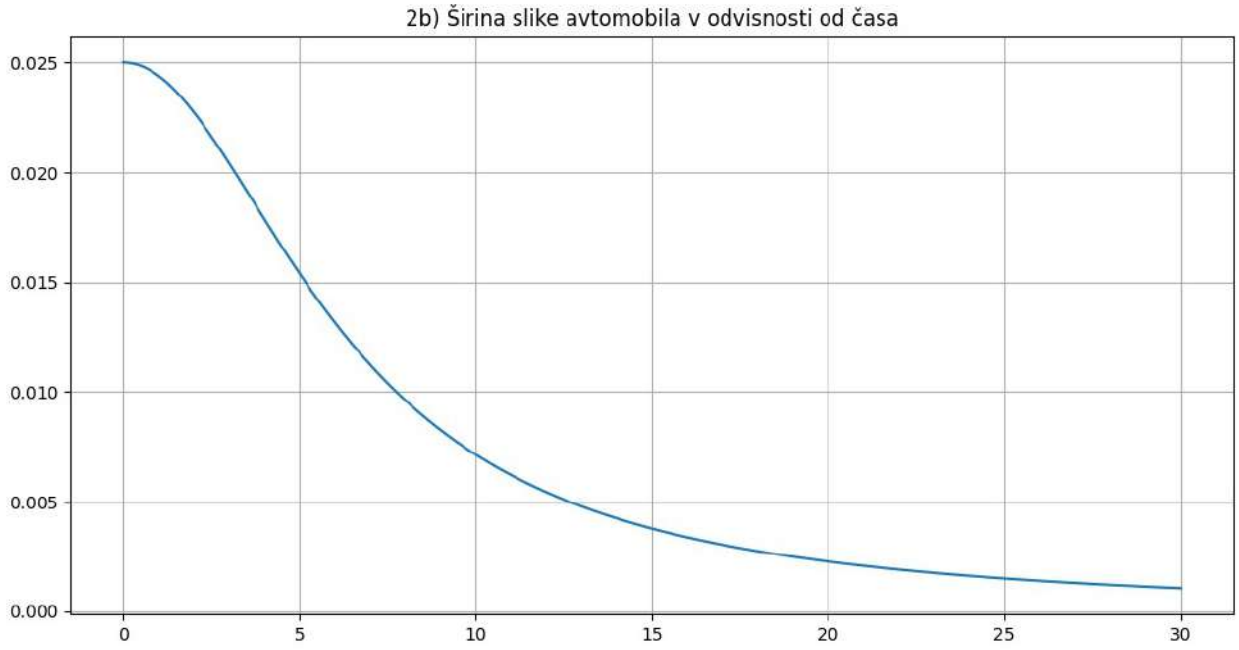
```
p = [500, 0, 1400]
f = 10
proj_fn = lambda f, p: (-f * p[0] / p[2], -f * p[1] / p[2])
print("2a) Projekcija drevesa:", proj_fn(f, p))
```

```
2a) Projekcija drevesa: (-3.5714285714285716, 0.0)
```

2b) Z enako kamero, kot v prejšnji nalogi, opazujemo avtomobil, širok 2.5m, ki je na začetku od kamere oddaljen 10m, nato pa se z enakomernim pospeškom 0.5 m/s^2 oddaljuje od kamere. S pomočjo python skripte in knjižnice MatPlotLib narišite graf, kako se širina slike avtomobila spreminja s časom. Izračunajte vrednosti za prvih 30s v intervalu 10 meritev na sekundo. Za izris grafa uporabite funkcijo plt.plot().

f = 10cm = 0.1m
Y = 5.5m
Z0 = 10m
a = 0.5 m/s^2

```
f = 0.1
Y = 2.5
Z0 = 10
a = 0.5

t = np.linspace(0, 30, 30 * 10)
Z = Z0 + 0.5 * a * t ** 2
y = f * Y / Z

# Izris grafa
plt.figure()
plt.plot(t, y)
plt.title("2b) Širina slike avtomobila v odvisnosti od časa")
plt.grid(True)
plt.show()
```



2c) Zakaj se kamere z luknjico uporabljajo bolj kot teoretičen model in ne tudi v praksi? Naštejte prednosti in slabosti kamer z lečami. (https://ksimek.github.io/2013/08/13/intrinsic/)

Kamera z luknjico se uporablja kot teoretičen model, ker omogoča poenostavljeno razlago delovanja kamere brez vpliva kompleksne optike. Takšna kamera temelji na idealnem principu perspektivne projekcije, kjer svetloba skozi majhno odprtino projicira sliko na ravno površino. Model je matematično enostaven in zato zelo uporaben za razumevanje osnovnih načel računalniškega vida in geometrije kamere. V praksi pa bi bila uporaba kamere z luknjico neprimerna, saj bi zaradi zelo majhne odprtine na senzor prišlo premalo svetlobe, kar bi zahtevalo dolge čase osvetlitve. Poleg tega bi se pri premajhnih odprtinah pojavili učinki difrakcije, ki bi povzročili zameglitev slike, zato takšna kamera ne omogoča dovolj kakovostne slike za večino praktičnih namenov.

Kamera z lečo te težave odpravlja, saj omogoča zbiranje večje količine svetlobe in posledično krajše čase osvetlitve. Leče omogočajo tudi nastavitev fokusa, kar pomeni, da lahko kamera ostri predmete na različnih razdaljah. Poleg tega je slika zaradi uporabe leč ostrejša in omogoča nadzor nad goriščno razdaljo, kar omogoča zumiranje in različne perspektive. Vendar imajo kamere z lečami tudi slabosti. Zaradi optičnih lastnosti leč lahko pride do različnih popačenj slike, kot so sferične in kromatične aberacije. Za natančne meritve je treba kamero kalibrirati, da se določijo parametri leče in odpravi vpliv popačenj. Izdelava takšnih kamer je bolj kompleksna in dražja v primerjavi s preprosto kamero z luknjico, poleg tega pa so bolj občutljive na mehanske premike, ki lahko vplivajo na kakovost slike.

2d) S kamero z goriščno razdaljo f = 60 mm posnamemo sliko vertikalnega valja, ki je od kamere oddaljen 95m. Določi višino valja, če v digitalizirani obliki slika valja po višini zavzame 200 slikovnih elementov. Ločljivost tipala je 2500 DPI.

```
f = 60
Z = 95000
dpi = 2500
```

```
pixels = 200

dots_per_mm = dpi / 25.4
y = pixels / dots_per_mm
Y = y * Z / f  # mm
print(f"2d) Višina valja: {Y / 1000:.3f} m")
```

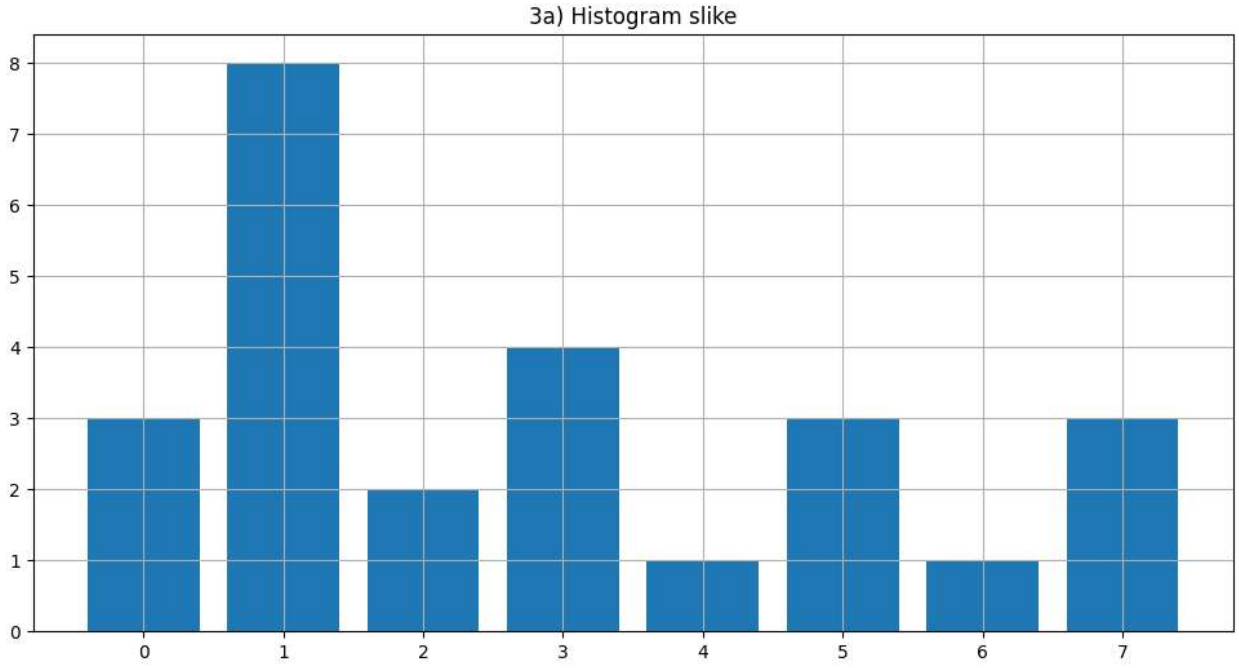
```
2d) Višina valja: 3.217 m
```

3. Histogrami

3a) Določite histogram za podano sliko (v obliki tabele in z grafično predstavitvijo).

```
# vrednost 0-7 => 3-bitna slika
bins = np.zeros(8)
I = np.array([
    [5, 3, 2, 7, 1],
    [7, 1, 0, 0, 0],
    [4, 5, 7, 1, 1],
    [1, 3, 2, 1, 1],
    [5, 3, 1, 6, 3]
])
for i in range(I.shape[0]):
    for j in range(I.shape[1]):
        b = I[i, j]
        bins[b] += 1

print(bins)
x = np.arange(len(bins))
plt.bar(x, bins)
plt.grid(True)
plt.title("3a) Histogram slike")
plt.show()
```

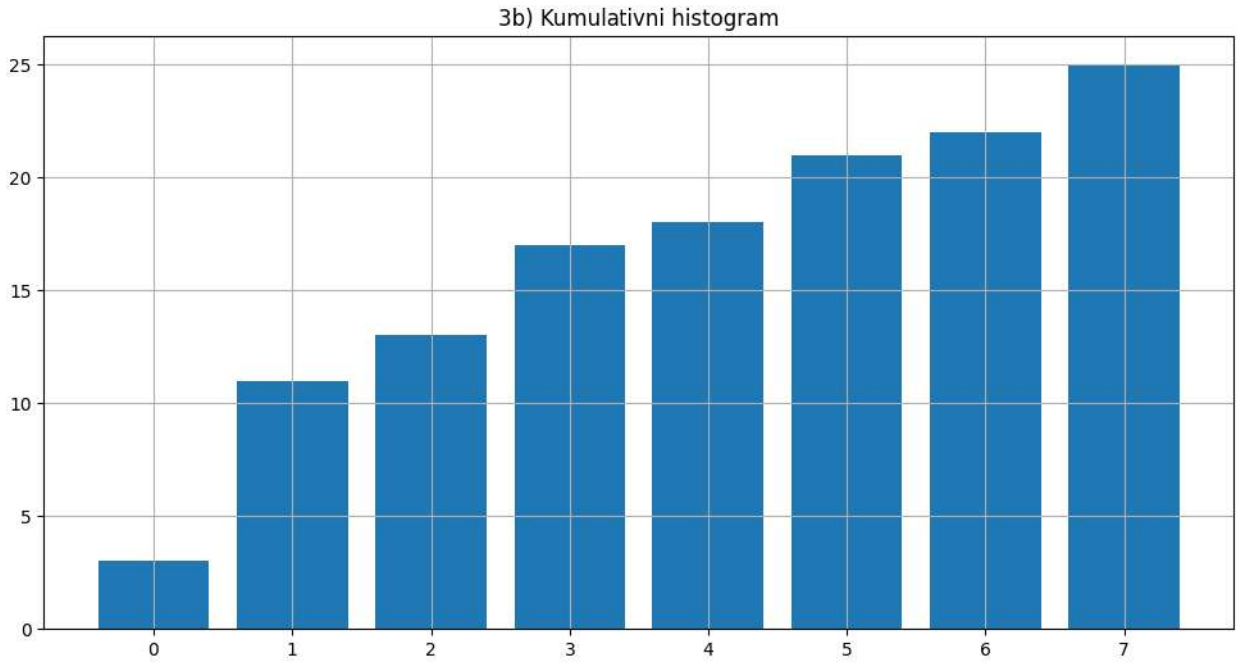
```
[3. 8. 2. 4. 1. 3. 1. 3.]
```



3b) Na podlagi izračunanega histograma določite kumulativni histogram slike.

```
cumsum = np.zeros(8)
sum = 0
for i, b in enumerate(bins):
    sum += b
    cumsum[i] += sum

# ali cumsum = np.cumsum(bins)

print(cumsum)
x = np.arange(len(cumsum))
plt.bar(x, cumsum)
plt.grid(True)
plt.title("3b) Kumulativni histogram")
plt.show()
```

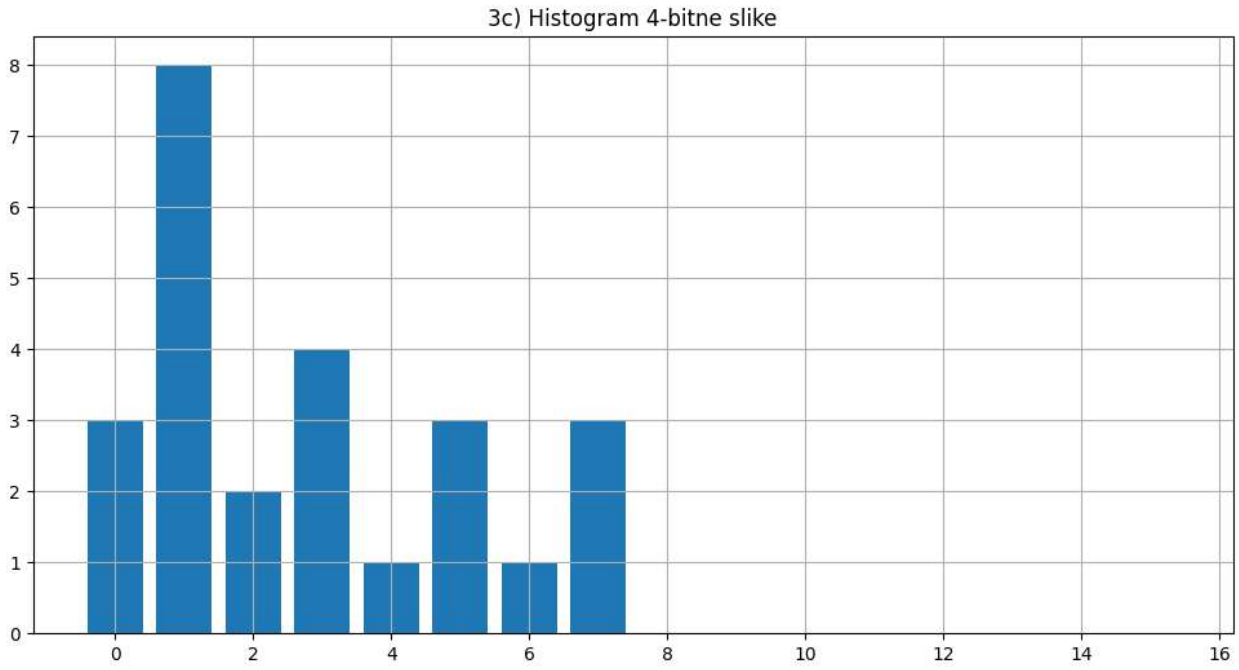
```
[ 3. 11. 13. 17. 18. 21. 22. 25.]
```



3c) Kakšen bi bil histogram, če bi bila slika 4-bitna?

```
# 4 bitna slika: 2^4 = 16 moznih vrednosti, damo jih v 8 košev
bins = np.zeros(16)
for i in range(I.shape[0]):
    for j in range(I.shape[1]):
        b = I[i, j]
        bins[b] += 1

print(bins)
x = np.arange(len(bins))
plt.bar(x, bins)
plt.grid(True)
plt.title("3c) Histogram 4-bitne slike")
plt.show()
```

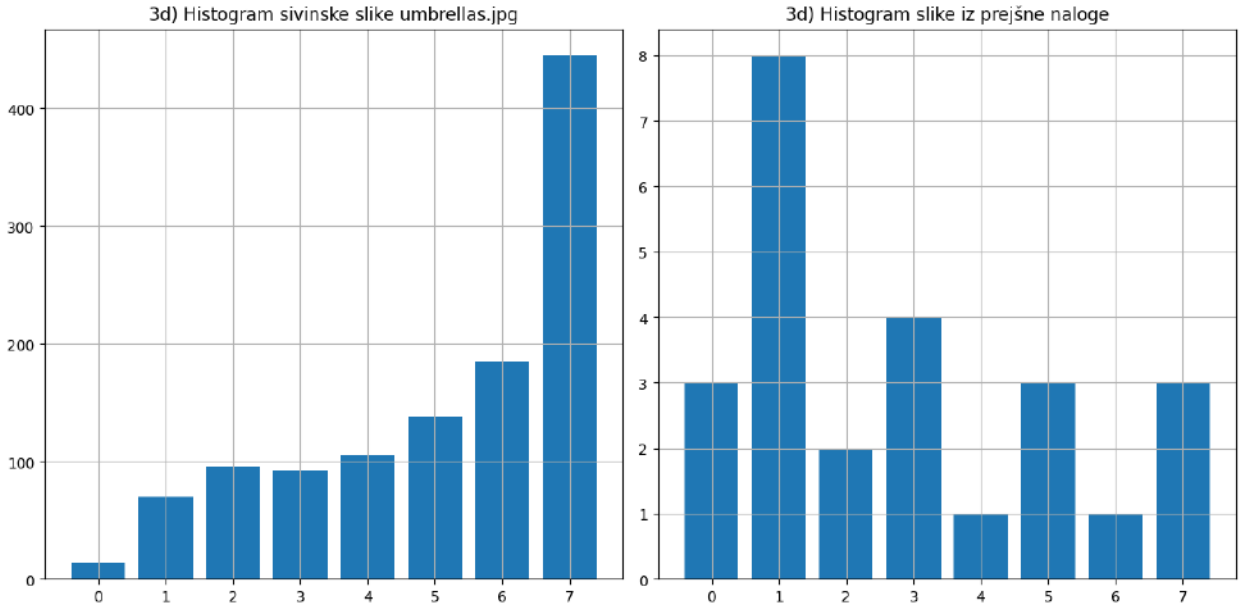
```
[3. 8. 2. 4. 1. 3. 1. 3. 0. 0. 0. 0. 0. 0. 0. 0.]
```



3d) Računanje histograma je v numpy implementirano v okviru funkcije np.histogram(). Naložite sliko umbrellas.jpg in jo spremenite v sivinsko. Uporabite funkcijo np.histogram, da dobite histogram slike in ga prikažite s funkcijo plt.bar(). Privzeto funkcija np.histogram uporabi 10 košev, zato nastavite vrednost parametra bins=8, kar ustreza vrednostim v vaši sliki. Ločeno izrišite še histogram iz prejšnje točke. To lahko naredite s parametrom range v funkciji np.histogram.

```
im = cv2.imread('umbrellas.jpg')
im = cv2.cvtColor(im, cv2.COLOR_BGR2RGB)
im_gray = np.mean(im, axis=2)
hist_umbrellas, bin_edges_umbrellas = np.histogram(im_gray, bins=8,
range=(0, 8))

# Histogram iz prejšnje naloge
I_example = np.array([
    [5, 3, 2, 7, 1],
    [7, 1, 0, 0, 0],
    [4, 5, 7, 1, 1],
    [1, 3, 2, 1, 1],
    [5, 3, 1, 6, 3]
])
hist_example, bin_edges_example = np.histogram(I_example, bins=8,
range=(0, 8))

plt.subplot(1, 2, 1)
x = np.arange(len(hist_umbrellas))
plt.bar(x, hist_umbrellas)
plt.title("3d) Histogram sivinske slike umbrellas.jpg")
plt.grid(True)
```

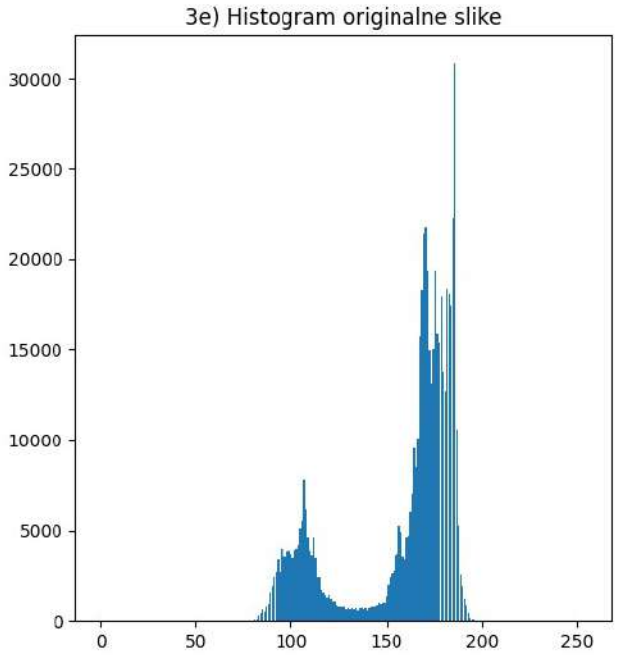
```
plt.subplot(1, 2, 2)
x2 = np.arange(len(hist_example))
plt.bar(x2, hist_example)
plt.title("3d) Histogram slike iz prejšne naloge")
plt.grid(True)
plt.tight_layout()
plt.show()
```

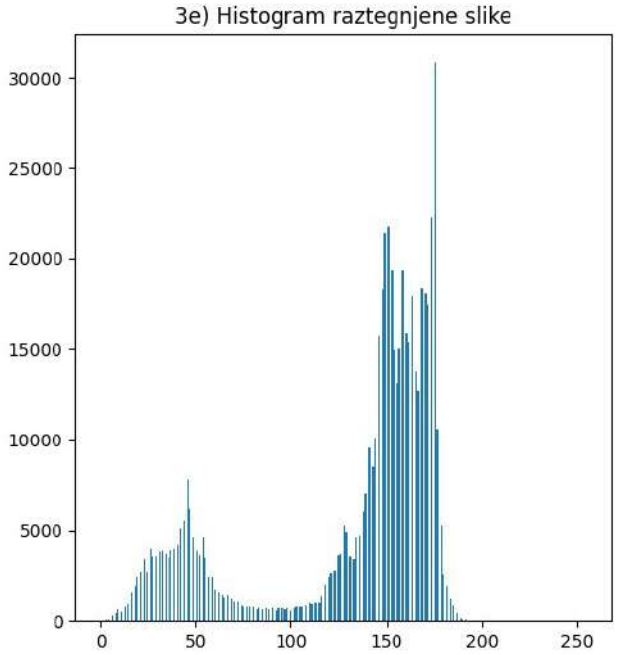


3e) Implementirajte funkcijo histogram_stretch(), ki vhodno sliko popravi tako, da njene vrednosti raztegne preko celotnega spektra sivinskih nivojev. Za okvirni potek algoritma se zgledujte po prosojnicah s predavanj. Ker funkcija izvede enako operacijo na vseh slikovnih elementih slike, implementirajte celotno funkcijo brez zank z uporabo matričnih operacij.
Namig: Za vhodno sliko lahko določite najvišjo vmax in najnižjo vmin sivinsko vrednost z np.max(I) in np.min(I).
Pomembno: Pri nalogi ni dovoljena uporaba integriranih funkcij, izračun novih vrednosti morate implementirati sami. Za preizkus funkcije preberite z diska datoteko phone.jpg (ki je že sivinska slika) in zanjo izrišite histogram z 256 celicami. S histograma lahko opazite, da najnižja in najvišja sivinska vrednost v sliki nista 0 in 255. Nato na sliki opravite razteg histograma ter prikažite rezultat v obliki slike in njenega histograma (histogrami naj vsebujejo 256 celic).

```
def histogram_stretch(I):
    I = I.astype(np.float32)
    vmin = np.min(I)
    vmax = np.max(I)
    I_stretched = (I - vmin) / (vmax - vmin) * 255.0
    return np.clip(I_stretched, 0, 255).astype(np.uint8)


im = cv2.imread('phone.jpg')
```

```
im = cv2.cvtColor(im, cv2.COLOR_BGR2RGB)
im_stretched = histogram_stretch(im)

hist_orig, _ = np.histogram(im, bins=256, range=(0, 256))
hist_stretched, _ = np.histogram(im_stretched, bins=256, range=(0,
256))

plt.subplot(1, 2, 1)
plt.imshow(im, cmap='gray')
plt.title("3e) Originalna sivinska slika")
plt.subplot(1, 2, 2)
plt.bar(np.arange(256), hist_orig)
plt.title("3e) Histogram originalne slike")
plt.show()

plt.subplot(1, 2, 1)
plt.imshow(im_stretched, cmap='gray')
plt.title("3e) Raztegnjena sivinska slika")
plt.subplot(1, 2, 2)
plt.bar(np.arange(256), hist_stretched)
plt.title("3e) Histogram raztegnjene slike")
plt.show()
```





3f) Upragovanje slike je zelo uporabno pri implementaciji preproste detekcije objektov, vendar je določitev praga pogosto problematična. Preizkusite OpenCV funkcijo cv2.threshold(), ki zna določiti prag samodejno z uporabo različnih metod, vključno z Otsujevo metodo (angl. Otsu's method), ki temelji na analizi histograma slike.

```
im = cv2.imread('phone.jpg', cv2.IMREAD_GRAYSCALE)

#ret1, th1 = cv2.threshold(im, n, 255, cv2.THRESH_BINARY) #Prag je
fiksno ročno določen
ret2, th2 = cv2.threshold(im, 0, 255, cv2.THRESH_BINARY +
cv2.THRESH_OTSU)  # Prag se samodejno določi s Otsujevo metodo

plt.subplot(1, 2, 1)
plt.imshow(im, cmap='gray')
plt.title("3f) Originalna slika")
plt.axis('off')
plt.subplot(1, 2, 2)
plt.imshow(th2, cmap='gray')
plt.title(f"3f) Otsujev prag {ret2:.0f}")
plt.axis('off')
plt.tight_layout()
plt.show()
```

3g) Kakšen histogram je idealen za Otsujevo metodo, ali, povedano drugače, na kakšnih slikah ta metoda deluje dobro?
(https://docs.opencv.org/4.x/d7/d4d/tutorial_py_thresholding.html, https://en.wikipedia.org/wiki/Otsu%27s_method)

Otsujeva metoda najbolje deluje na slikah, kjer je histogram dvodomni, to pomeni, da ima dva jasno ločena vrhova, ki predstavljata objekt in ozadje. Prag se določi tako, da kar najbolje loči ti dve skupini, s čimer se zmanjša varianca znotraj skupin in poveča razlika med njima. Metoda je še posebej učinkovita pri slikah s kontrastnimi predmeti na ozadju, kjer so temna in svetla področja jasno ločena. Pri slikah z več kot dvema glavnima skupinama ali pri enakomerno porazdeljenih sivinskih vrednostih pa Otsujeva metoda pogosto ne deluje optimalno. Tudi prisoten šum lahko oteži določitev praga, saj zameša vrhove v histogramu.

4. Barvni prostori

4a) Barvo, zapisano v RGB barvnem prostoru z (255, 34, 126) bi radi preslikali v barvni prostor HSV. Ročno izvedite postopek preslikave in izračunajte rezultat pretvorbe. Rešitev lahko preverite npr. s funckijo matplotlib.colors.rgb_to_hsv(). Pri uporabi takšnih funkcij vedno preglejte dokumentacijo, saj so lahko vhodni in izhodni formati med implementacijami različni. Kanal H je lahko naprimer zapisan na intervalu [0, 100], intervalu [0, 1] ali celo v stopinjah [0, 360].

```
R, G, B = 255 / 255, 34 / 255, 126 / 255
Cmax = max(R, G, B)
Cmin = min(R, G, B)
delta = Cmax - Cmin

if delta == 0:
    H = 0
elif Cmax == R:
    H = 60 * (((G - B) / delta) % 6)
elif Cmax == G:
```

```
    H = 60 * (((B - R) / delta) + 2)
else:
    H = 60 * (((R - G) / delta) + 4)

S = 0 if Cmax == 0 else delta / Cmax
V = Cmax
H = H / 360

print("4a) Ročno izračunano HSV:", H, S, V)
```

```
4a) Ročno izračunano HSV: 0.9306184012066364 0.8666666666666667 1.0
```

```
rgb = np.array([[[255 / 255, 34 / 255, 126 / 255]]])
hsv = colors.rgb_to_hsv(rgb)
H, S, V = hsv[0][0] * [1, 1, 1]
print("4a) Matplotlib HSV pretvorba:", H, S, V)
```

```
4a) Matplotlib HSV pretvorba: 0.9306184012066365 0.8666666666666667
1.0
```

4b) Barvo, zapisano v HSV barvnem prostoru z (0.65, 0.7, 0.15) bi radi preslikali v barvni prostor RGB. Ročno izvedite postopek preslikave in izračunajte rezultat pretvorbe.

```
H, S, V = 0.65, 0.7, 0.15
H_deg = H * 360
C = V * S
X = C * (1 - abs((H_deg / 60) % 2 - 1))
m = V - C

if 0 <= H_deg < 60:
    r1, g1, b1 = C, X, 0
elif 60 <= H_deg < 120:
    r1, g1, b1 = X, C, 0
elif 120 <= H_deg < 180:
    r1, g1, b1 = 0, C, X
elif 180 <= H_deg < 240:
    r1, g1, b1 = 0, X, C
elif 240 <= H_deg < 300:
    r1, g1, b1 = X, 0, C
else:
    r1, g1, b1 = C, 0, X

R = (r1 + m) * 255
G = (g1 + m) * 255
B = (b1 + m) * 255

print("4b) Ročno izračunano RGB:", R, G, B)
```

```
4b) Ročno izračunano RGB: 11.475 14.152500000000002 38.25
```

```
H, S, V = 0.65, 0.7, 0.15
hsv = np.array([[[H, S, V]]])
rgb = colors.hsv_to_rgb(hsv)
rgb_255 = rgb[0][0] * 255
print("4b) Matplotlib RGB pretvorba:", R, G, B)
```

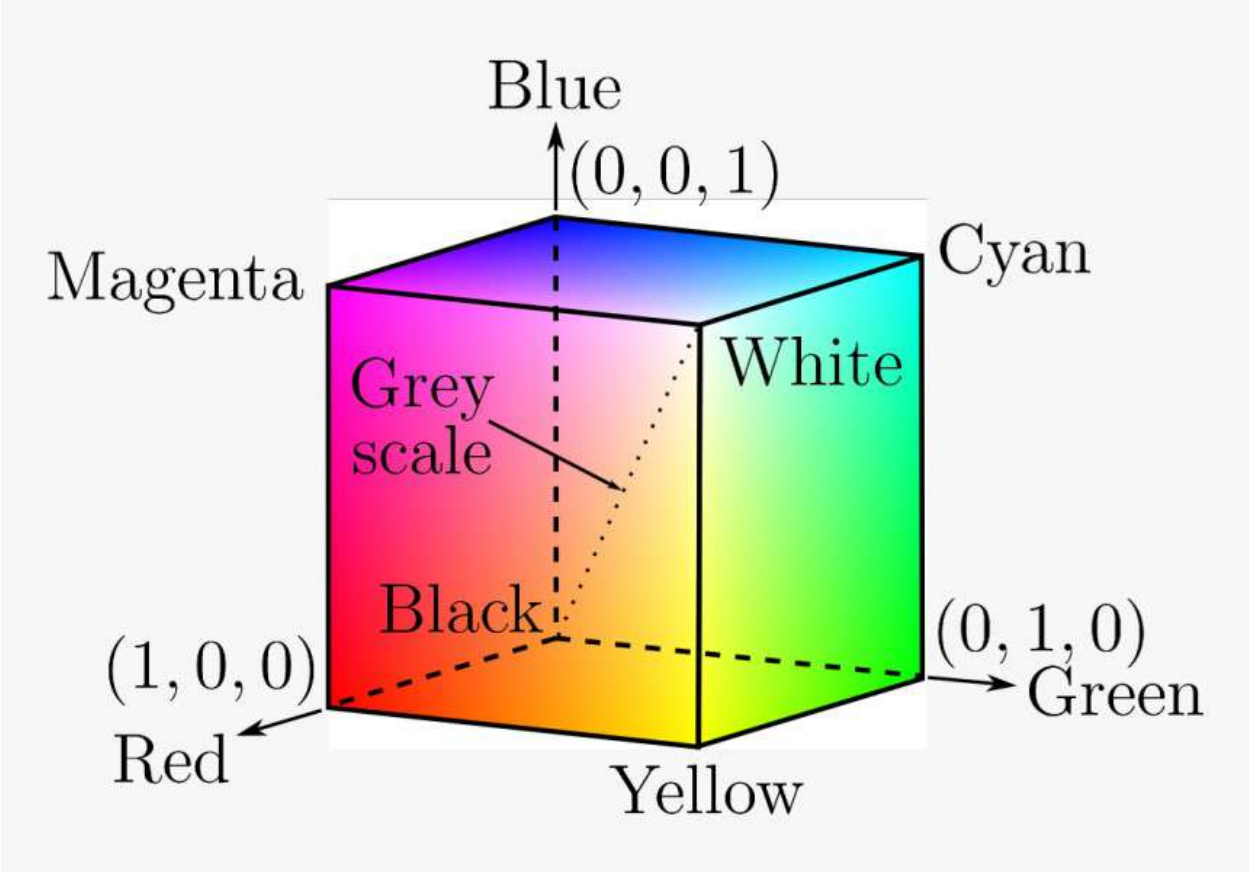
```
4b) Matplotlib RGB pretvorba: 11.475 14.152500000000002 38.25
```

4c) Izvedite pretvorbo med barvnimi prostori na konkretnem primeru z uporabo OpenCV funkcije cv2.cvtColor(). Naložite sliko trucks.jpg ter jo prikažite kot RGB sliko ter vsako komponento posebej kot sivinsko sliko. Nato pretvorite prebrano sliko v HSV barvni prostor z uporabo funkcije cv2.cvtColor() ter spet prikažite vsako komponento posebej kot sivinsko sliko. Potrebno je paziti na pravilno obravnavo tipov matrik, saj je RGB slika privzeto shranjena v matriki tipa uint8} (cela števila od 0 do 255), slika v barvnem prostoru HSV pa v matriki tipa double (realna števila od 0 do 1). Razmislite, kaj pomenijo posamezne sivinske vrednosti v vsakem od kanalov, relativno na barve v izhodiščni sliki.

```
im = cv2.imread('trucks.jpg')
im = cv2.cvtColor(im, cv2.COLOR_BGR2RGB)

im_hsv = cv2.cvtColor(im, cv2.COLOR_RGB2HSV)
im_hsv = im_hsv.astype('float32') / 255.0

plt.subplot(3, 3, 7)
plt.imshow(im)
plt.title('4c) RGB slika')
plt.axis('off')

plt.subplot(3, 3, 1)
plt.imshow(im[:, :, 0], cmap='gray')
plt.title('4c) R kanal')
plt.axis('off')

plt.subplot(3, 3, 2)
plt.imshow(im[:, :, 1], cmap='gray')
plt.title('4c) G kanal')
plt.axis('off')

plt.subplot(3, 3, 3)
plt.imshow(im[:, :, 2], cmap='gray')
plt.title('4c) B kanal')
plt.axis('off')

plt.subplot(3, 3, 4)
plt.imshow(im_hsv[:, :, 0], cmap='gray')
plt.title('4c) H kanal')
plt.axis('off')

plt.subplot(3, 3, 5)
plt.imshow(im_hsv[:, :, 1], cmap='gray')
```

```
plt.title('4c) S kanal')
plt.axis('off')

plt.subplot(3, 3, 6)
plt.imshow(im_hsv[:, :, 2], cmap='gray')
plt.title('4c) V kanal')
plt.axis('off')

plt.tight_layout()
plt.show()
```

V RGB prostoru sivinske vrednosti predstavljajo količino posamezne barve: svetlejše pomeni več rdeče, zelene ali modre.



V HSV prostoru pa kanal H določa odtenek (vrsto barve), S nasičenost (živost barve) in V svetlost. Svetlejše vrednosti v S in V pomenijo bolj nasičene in svetlejše barve, kanal H pa označuje, kateri barvni ton prevladuje.

4d) Različni barvni prostori so koristni tudi za upragovanje. V RGB barvnem prostoru je recimo težko na preprost način določiti področja, ki pripadajo določenemu barvnemu odtenku. Napišite

skripto, ki sliko trucks.jpg upraguje po modrem kanalu za vrednost praga 150. Prikažite originalno sliko ter binarno sliko eno ob drugi (uporabite plt.subplot().

```
modri_kanal = im[:, :, 2]
_, binarna = cv2.threshold(modri_kanal, 150, 255, cv2.THRESH_BINARY)

plt.subplot(1, 2, 1)
plt.imshow(im)
plt.title('4d) Originalna slika')
plt.axis('off')

plt.subplot(1, 2, 2)
plt.imshow(binarna, cmap='gray')
plt.title('4d) Upragovana po modrem kanalu')
plt.axis('off')

plt.show()
```

4e) Za osnovne komponene (rdeča, zelena, modra) se upragovanje poenostavi, če sliko preslikamo v normaliziran RGB prostor, kjer je vrednost vsake barve deljena z vsoto vrednosti vseh treh komponent (takemu prostoru rečemo tudi normaliziran RGB). Napišite kodo, ki modro komponento slike po elementih deli z vsoto soležnih treh barvnih komponent (uporabite funkcijo np.sum). Na tako sliko aplicirajte upragovanje (ker so normalizirane vrednosti definirane na razponu vrednosti od 0 do 1 je treba prag prilagoditi. Eksperimentirajte z vrednostmi okoli 0.5 ter upragovano sliko prikažite.

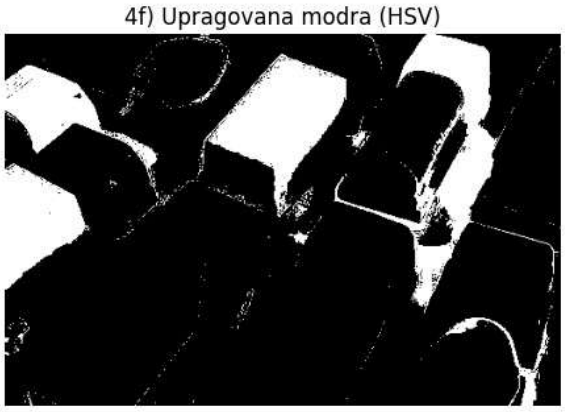
```
im = im.astype('float32')
vsota = np.sum(im, axis=2)
vsota[vsota == 0] = 1
norm_b = im[:, :, 2] / vsota
_, binarna = cv2.threshold(norm_b, 0.45, 1.0, cv2.THRESH_BINARY)

plt.subplot(1, 2, 1)
plt.imshow(norm_b, cmap='gray')
plt.title('4e) Normalizirana modra komponenta')
plt.axis('off')
```

```
plt.subplot(1, 2, 2)
plt.imshow(binarna, cmap='gray')
plt.title('4e) Upragovana slika')
plt.axis('off')

plt.show()
```

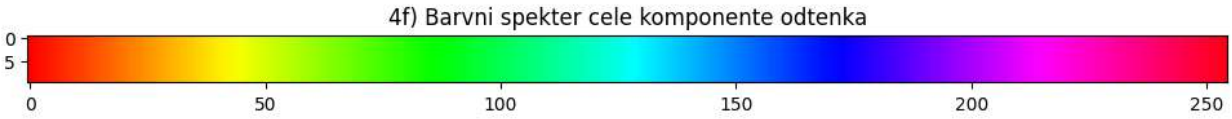
4f) Z uporabo normaliziranih RGB vrednosti sicer lahko določimo regije rdeče, zelene in modre barve. Enostavnejši način za določitev barvnih regij pa je uporaba HSV barvnega prostora. Dodajte kodo, ki sliko iz RGB barvnega prostora preslika v HSV prostor ter upragujte po komponenti odtenka (Hue). Ker zavzema modra barva samo del vrednostnega območja, je potrebno sliko upragovati z dvema pragoma. To se najhitreje reši kot logična funkcija dveh mask (vsako dobimo z uporabo enega praga). Primer: AB = A&B.

```
im = cv2.imread('trucks.jpg')
im = cv2.cvtColor(im, cv2.COLOR_BGR2RGB)
im_hsv = cv2.cvtColor(im, cv2.COLOR_RGB2HSV)
im_hsv = im_hsv.astype('float32') / 255.0

H = im_hsv[:, :, 0]

# Pragovi za modro barvo (H v intervalu 0-1)
mask1 = H >= 0.55
mask2 = H <= 0.70
mask_modra = mask1 & mask2

plt.subplot(1, 2, 1)
plt.imshow(im)
plt.title('4f) Originalna slika')
plt.axis('off')

plt.subplot(1, 2, 2)
plt.imshow(mask_modra, cmap='gray')
plt.title('4f) Upragovana modra (HSV)')
```

```
plt.axis('off')
plt.show()
```



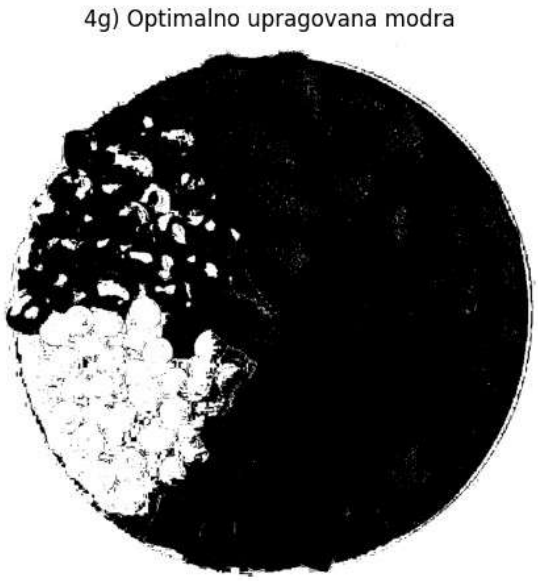

```
plt.figure()
plt.imshow(np.meshgrid(np.linspace(0, 1, 255), np.ones((10, 1)))[0],
cmap=plt.cm.hsv)
plt.title('4f) Barvni spekter cele komponente odtenka')
plt.show()
```



4g) Eksperimentirajte z robnimi vrednostmi območja ter optimalno upragovano sliko prikažite. Postopek ponovite za poljubno izbrano sliko (sliko si izberite sami, prav tako si izberite barvo, ki bi jo radi izluščili iz slike).

```
ime_slike = 'fruit.jpg'
im = cv2.imread(ime_slike)
im = cv2.cvtColor(im, cv2.COLOR_BGR2RGB)
im_hsv = cv2.cvtColor(im, cv2.COLOR_RGB2HSV)
im_hsv = im_hsv.astype('float32') / 255.0
H = im_hsv[:, :, 0]

prag_spodnji = 0.40
prag_zgornji = 0.60
mask_modra = (H >= prag_spodnji) & (H <= prag_zgornji)

plt.subplot(1, 2, 1)
plt.imshow(im)
plt.title('4g) Originalna slika')
plt.axis('off')

plt.subplot(1, 2, 2)
plt.imshow(mask_modra, cmap='gray')
```

```
plt.title('4g) Optimalno upragovana modra')
plt.axis('off')

plt.show()
```

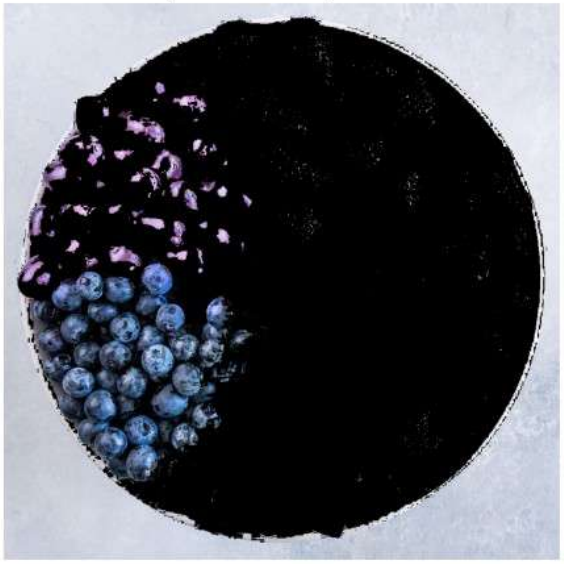


4h) Implementirajte funkcijo im_mask, ki ji podate sliko v RGB barvnem prostoru ter binarno sliko iste velikosti, funkcija pa vam vrne barvno sliko, kjer je barva slikovnih elementov postavljena na črno, če je vrednost istoležnega elementa v maski 0. Ostali slikovni elementi morajo ostati nespremenjeni. Funkcijo implementirajte brez eksplicitnih zank.

```
def im_mask(slika_rgb, maska):
    maska3 = np.stack([maska] * 3, axis=2)
    rezultat = slika_rgb * maska3
    return rezultat.astype(slika_rgb.dtype)

im_filtrirana = im_mask(im, mask_modra.astype(np.uint8))

plt.subplot(1, 2, 1)
plt.imshow(im)
plt.title('4h) Originalna slika')
plt.axis('off')

plt.subplot(1, 2, 2)
plt.imshow(im_filtrirana)
plt.title('4h) Barvna slika po maski')
plt.axis('off')

plt.show()
```

4h) Originalna slika

4h) Barvna slika po maski



5. Regije in morfološke operacije

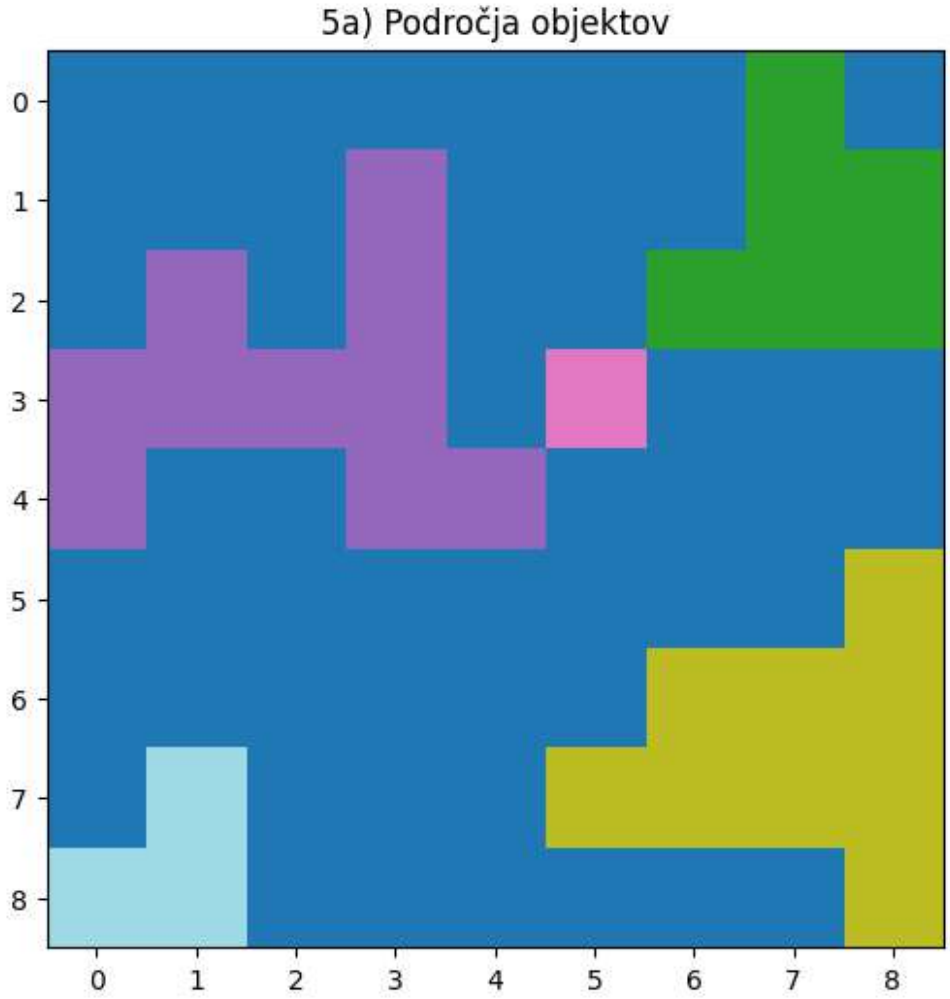
Glavne operacije:

- Krčenje (Erosion): Odstrani piksle z robov objektov, zmanjšanje hrupa, ločevanje povezanih objektov.
- Širjenje (Dilation): Dodaja piksle na robove objektov, zapolnjevanje lukenj, povezovanje razbitih delov.
- Odpiranje (Opening): Erozija, nato dilatacija, odstranjevanje manjših objektov ali šuma.
- Zapiranje (Closing): Dilatacija, nato erozija, zapolnjevanje majhnih lukenj.

5a) Podana je binarna slika, za katero določite rezultat algoritma barvanja regij po prvem in drugem sprehodu po sliki ter povezave med oznakami za okolico N4. Na sliki so z 1 označena področja objektov v sliki, celice z vrednostjo 0 pa so ozadje.

```
A = np.array([
    [0, 0, 0, 0, 0, 0, 0, 1, 0],
    [0, 0, 0, 1, 0, 0, 0, 1, 1],
    [0, 1, 0, 1, 0, 0, 1, 1, 1],
    [1, 1, 1, 1, 0, 1, 0, 0, 0],
    [1, 0, 0, 1, 1, 0, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 0, 0, 1],
    [0, 0, 0, 0, 0, 0, 1, 1, 1],
    [0, 1, 0, 0, 0, 1, 1, 1, 1],
    [1, 1, 0, 0, 0, 0, 0, 0, 1]
```

```
], dtype=np.uint8)

num_labels, labels = cv2.connectedComponents(A, connectivity=4)
plt.imshow(labels, cmap='tab20')
plt.title("5a) Področja objektov")
plt.show()
```

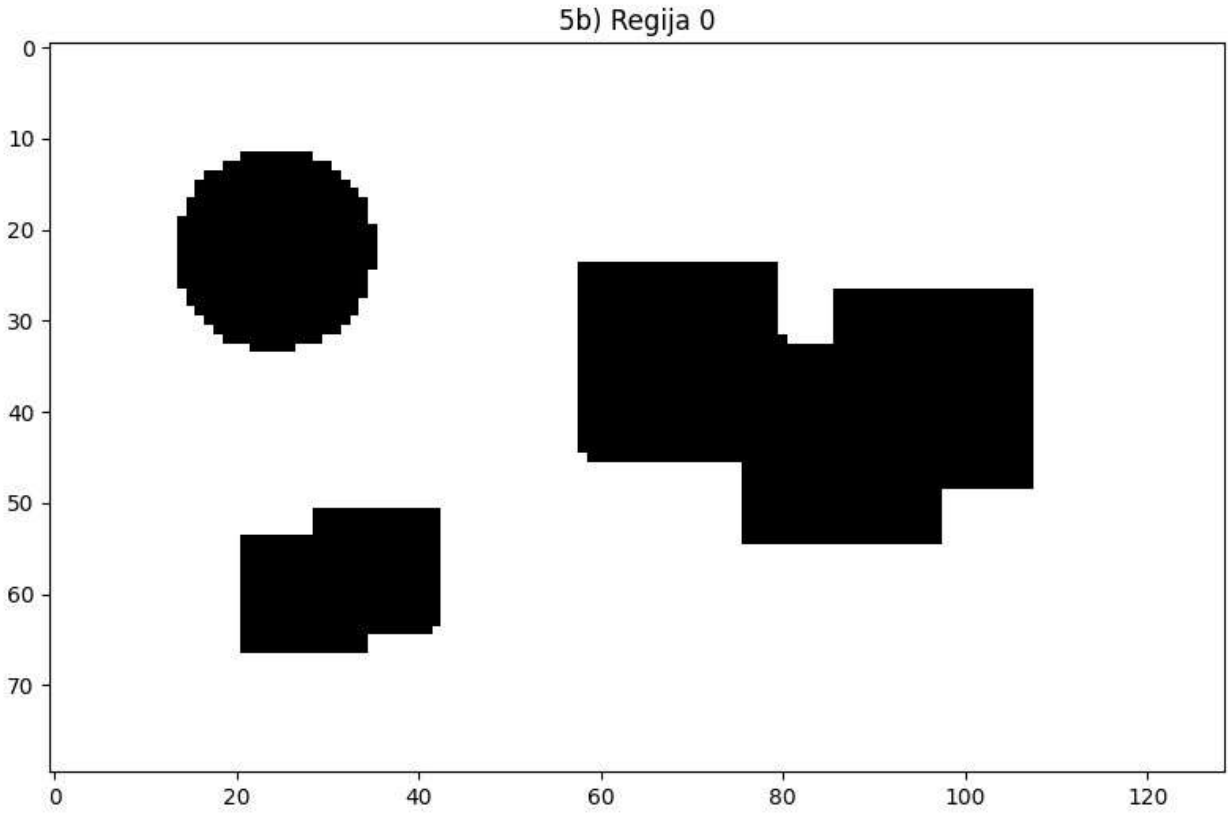


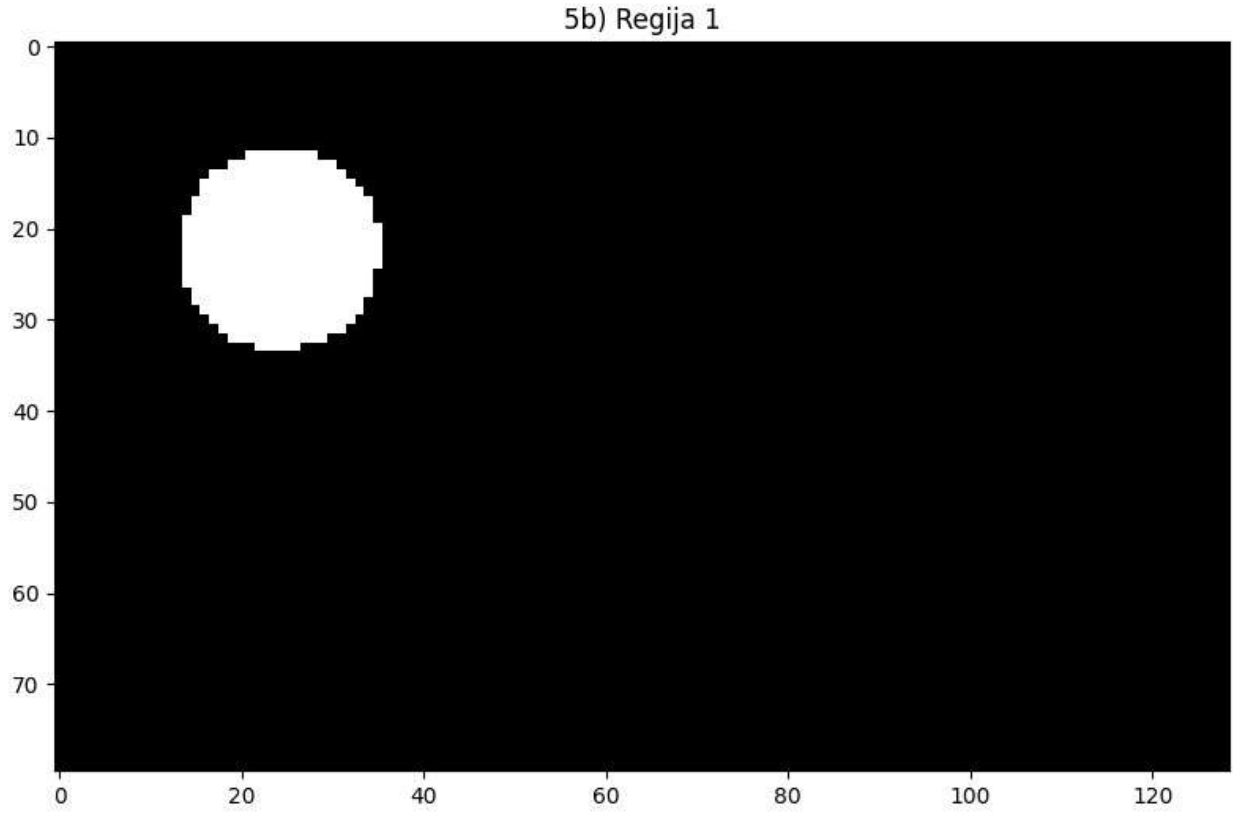
5b) Preberite sliko regions.png. Kljub temu, da slika vsebuje zgolj dve vrednosti, ni shranjena v tipu binarnih logičnih vrednosti, zato jo najprej spremenite v binarno z uporabo funkcije cv2.threshold() in uporabo ustreznega praga (recimo 127). Uporabite funkcijo cv2.connectedComponents(), ki implementira algoritem označevanja povezanih regij. Izpišite število regij (vključno z ozadjem) in prikažite maske, ki označujejo posamezne regije.

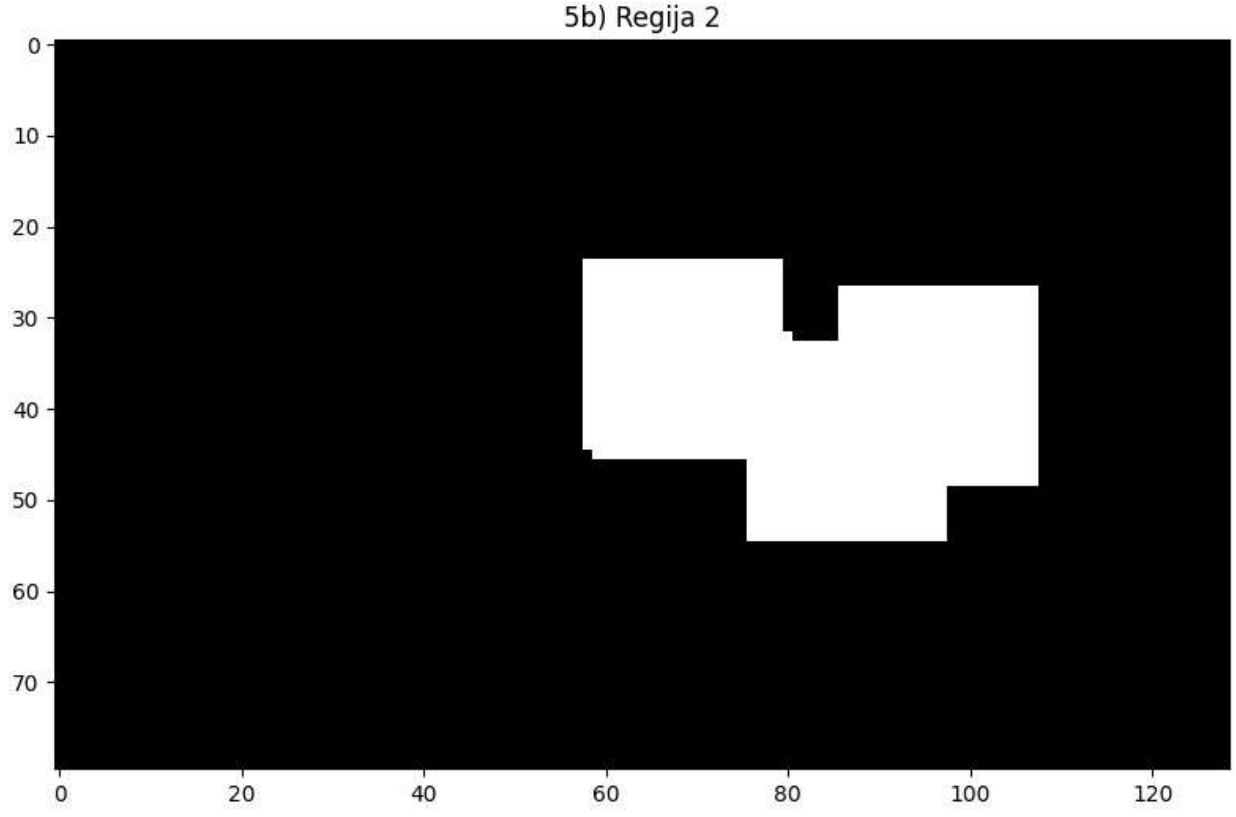
```
im = cv2.imread('regions.png', cv2.IMREAD_GRAYSCALE)
_, bin = cv2.threshold(im, 127, 1, cv2.THRESH_BINARY)
num_labels, labels = cv2.connectedComponents(bin, connectivity=4)
print(num_labels)
for i in range(num_labels):
```

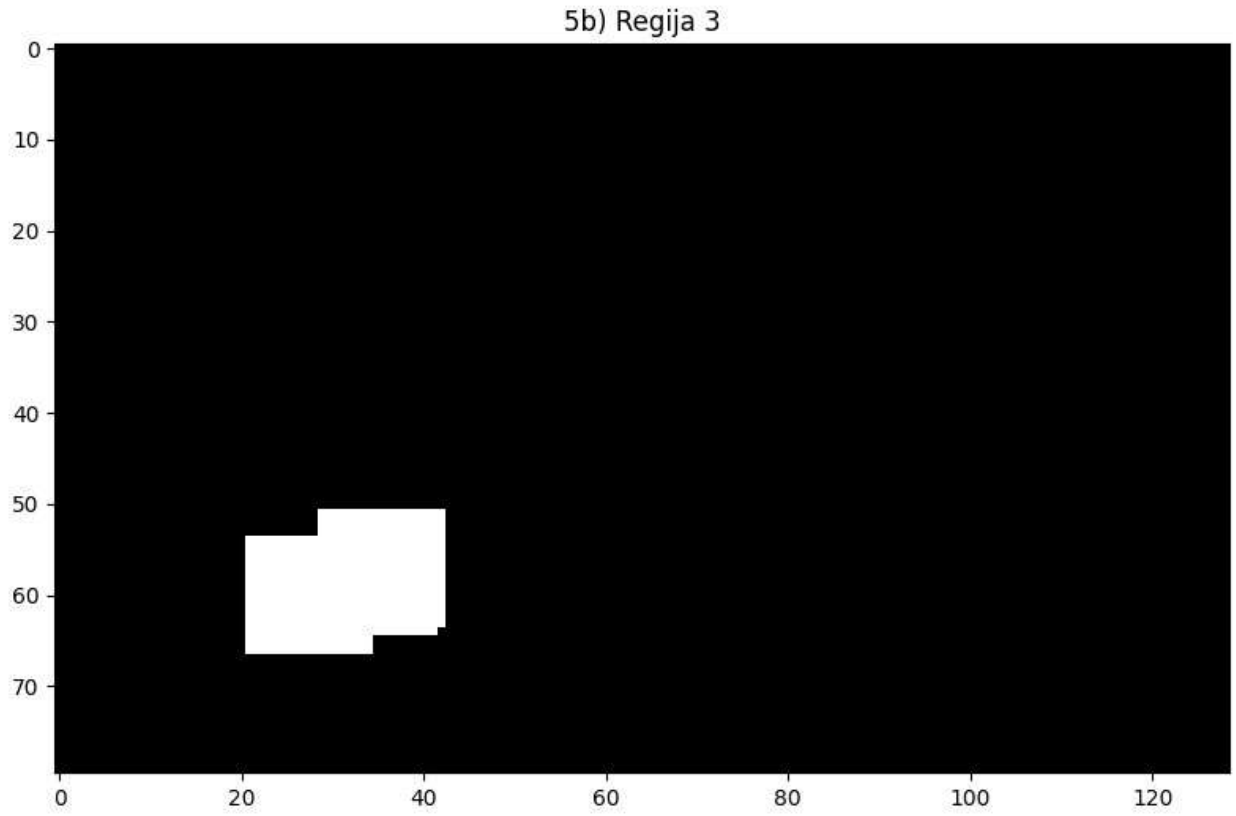
```
    mask = (labels == i).astype(np.uint8)
    plt.imshow(mask, cmap='gray')
    plt.title(f'5b) Regija {i}')
    plt.show()
```

4



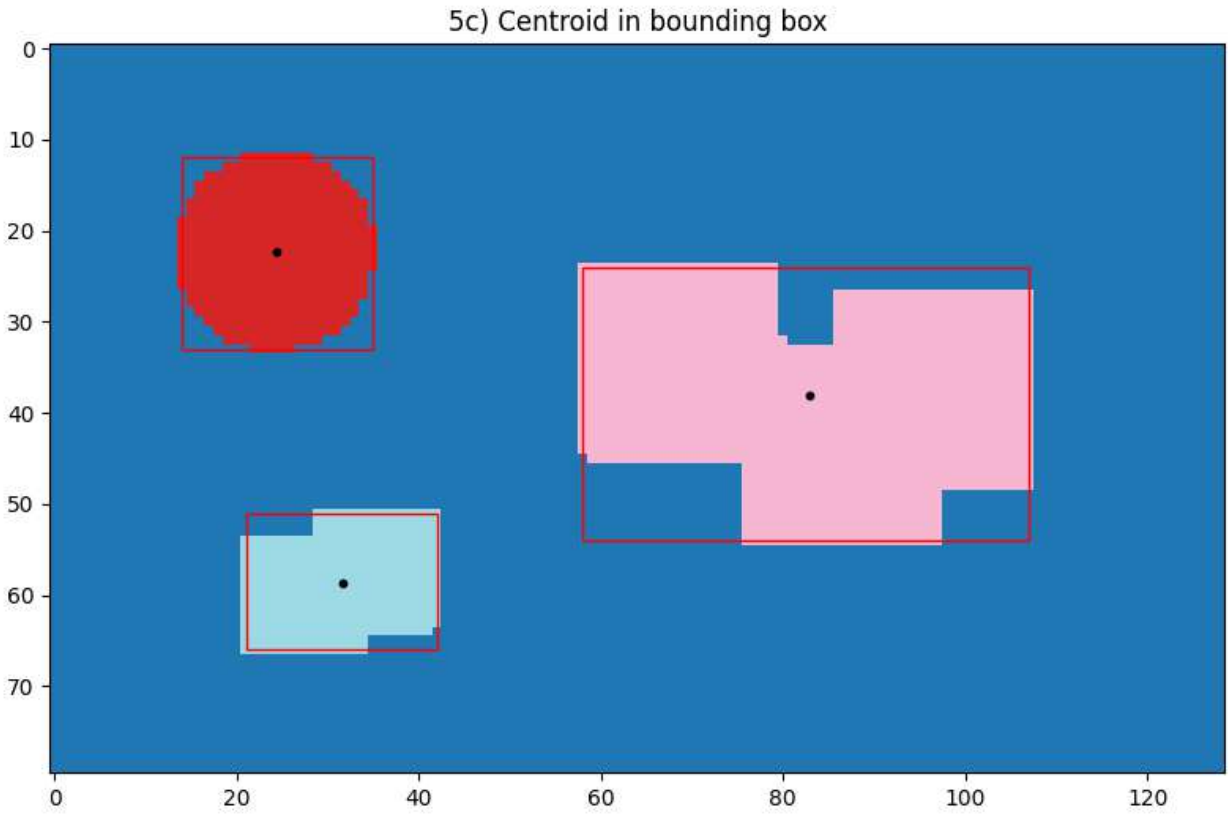

5b) Regija 1
0
10
20
30
40
50
60
70
0
20
40
60
80
100
120


5b) Regija 2
0
10
20
30
40
50
60
70
0
20
40
60
80
100
120



5c) Za posamezne regije iz prejšnje naloge določite centroid ter očrtani pravokotnik regije (angl. bounding box). Navedene lastnosti prikažite nad sliko maske. Namig: za izris centroidov uporabite PyPlot funkcijo plt.scatter(), za izris pravokotnikov pa PyPlot funkcijo matplotlib.patches.Rectangle().

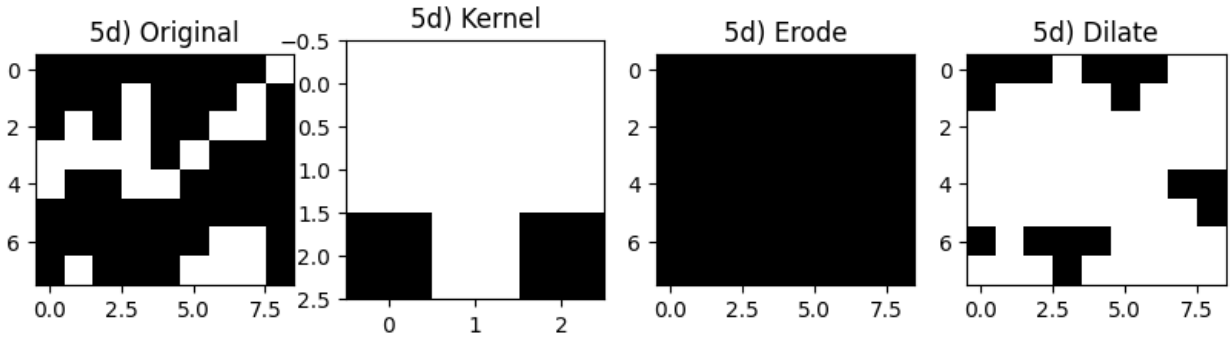
```
im = cv2.imread('regions.png', cv2.IMREAD_GRAYSCALE)
_, bin = cv2.threshold(im, 127, 1, cv2.THRESH_BINARY)
num_labels, labels = cv2.connectedComponents(bin, connectivity=4)

plt.imshow(labels, cmap='tab20')
ax = plt.gca()

for i in range(1, num_labels):
    y, x = np.where(labels == i)
    cx, cy = np.mean(x), np.mean(y)
    x_min, x_max = np.min(x), np.max(x)
    y_min, y_max = np.min(y), np.max(y)
    rect = patches.Rectangle((x_min, y_min), x_max - x_min, y_max -
y_min, fill=False, edgecolor='red')
    ax.add_patch(rect)
    plt.scatter(cx, cy, c='black', s=10)

plt.title("5c) Centroid in bounding box")
plt.show()
```



5d) Podana je binarna slika, za katero določite rezultat algoritma filtriranja s filtroma razširi ter skrči za jedro K.

```
A = np.array([
    [0, 0, 0, 0, 0, 0, 0, 0, 1],
    [0, 0, 0, 1, 0, 0, 0, 1, 0],
    [0, 1, 0, 1, 0, 0, 1, 1, 0],
    [1, 1, 1, 1, 0, 1, 0, 0, 0],
    [1, 0, 0, 1, 1, 0, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 1, 1, 0],
    [0, 1, 0, 0, 0, 1, 1, 1, 0]
], dtype=np.uint8)

K = np.array([
    [1, 1, 1],
    [1, 1, 1],
    [0, 1, 0]
], dtype=np.uint8)

er = cv2.erode(A, K, iterations=1)
dil = cv2.dilate(A, K, iterations=1)

plt.figure(figsize=(10, 3))
plt.subplot(1, 4, 1)
```

```
plt.imshow(A, cmap='gray')
plt.title('5d) Original')

plt.subplot(1, 4, 2)
plt.imshow(K, cmap='gray')
plt.title('5d) Kernel')

plt.subplot(1, 4, 3)
plt.imshow(er, cmap='gray')
plt.title('5d) Erode')

plt.subplot(1, 4, 4)
plt.imshow(dil, cmap='gray')
plt.title('5d) Dilate')

plt.show()
```

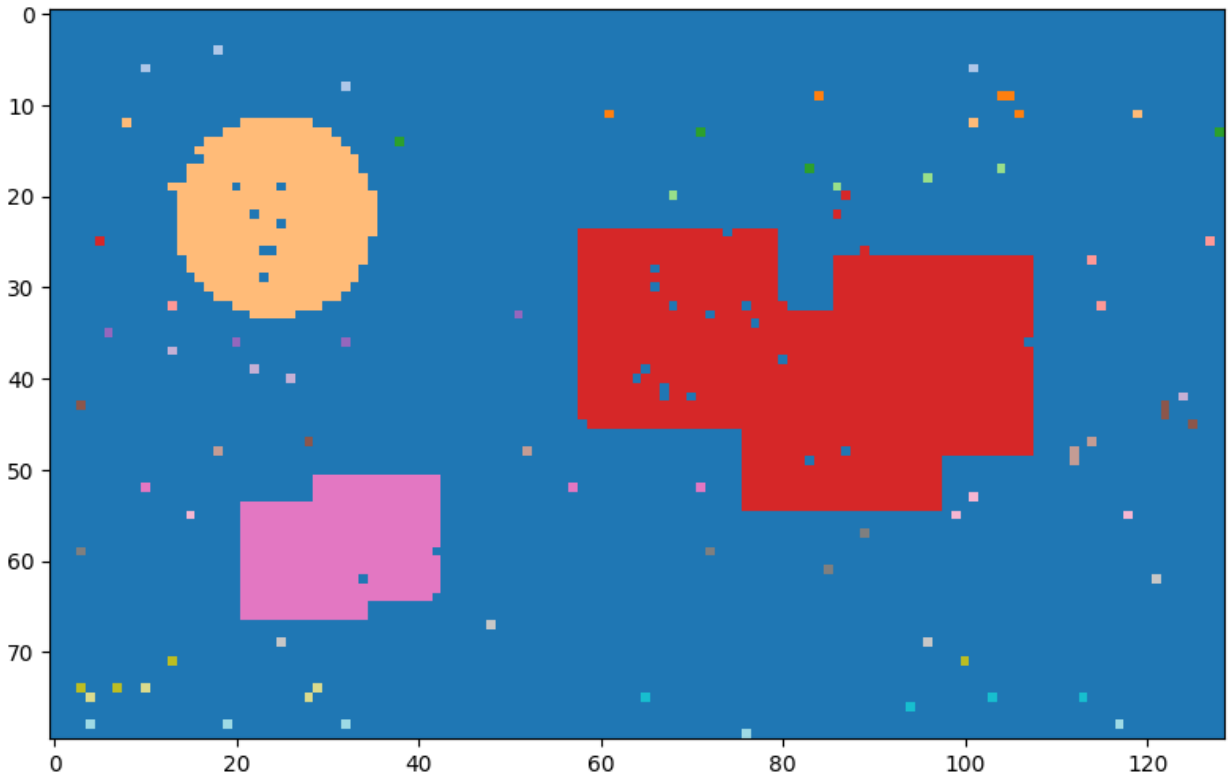


5e) Naložite sliko regions_noise.png in jo binarizirajte z uporabo ustreznega praga. Uporabite funkcijo cv2.connectedComponents() za določitev regij in regije preštejte. Kaj opazite? Kako velike so posamezne regije?

```
im = cv2.imread('regions_noise.png', cv2.IMREAD_GRAYSCALE)
_, bin = cv2.threshold(im, 127, 1, cv2.THRESH_BINARY)
num_labels, labels = cv2.connectedComponents(bin, connectivity=4)
print('5e) Število regij:', num_labels)

sizes = []
for i in range(1, num_labels):
    size = np.sum(labels == i)
    sizes.append(size)
print('Velikosti regij:', sizes)

plt.imshow(labels, cmap='tab20')
plt.show()
```

```
5e) Število regij: 81
Velikosti regij: [np.int64(2), np.int64(1), np.int64(1), np.int64(1),
np.int64(1), np.int64(1), np.int64(1), np.int64(1), np.int64(2),
```

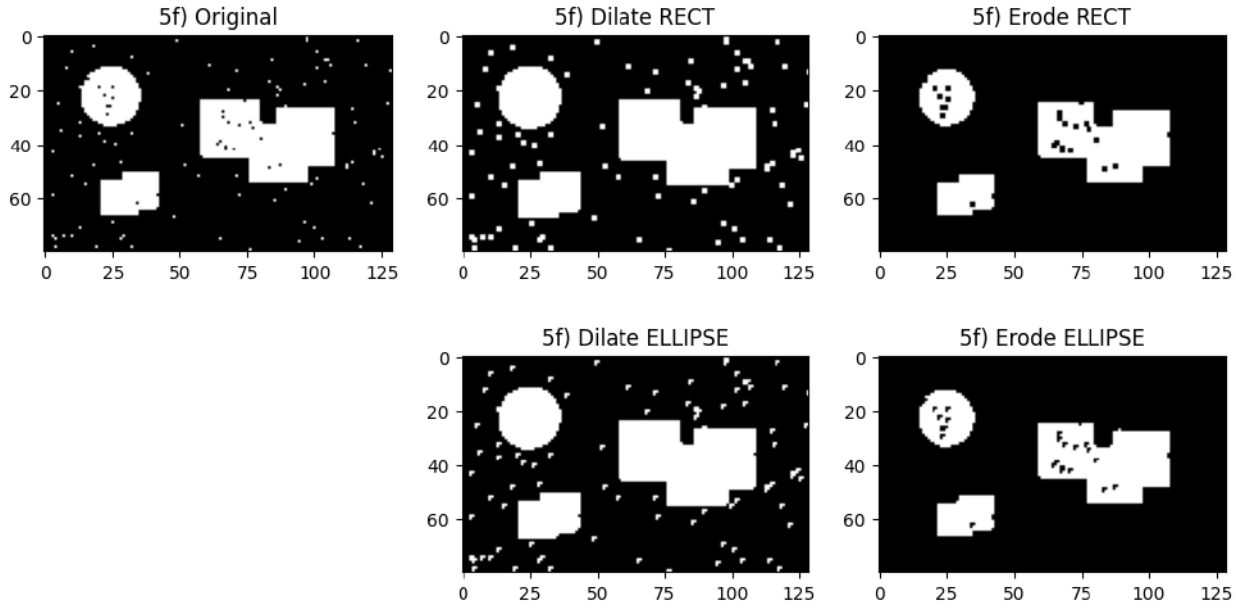
```
np.int64(1), np.int64(1), np.int64(1), np.int64(1), np.int64(377),
np.int64(1), np.int64(1), np.int64(1), np.int64(1), np.int64(1),
np.int64(1), np.int64(1), np.int64(1), np.int64(1), np.int64(1),
np.int64(1), np.int64(1193), np.int64(1), np.int64(1), np.int64(1),
np.int64(1), np.int64(1), np.int64(1), np.int64(1), np.int64(1),
np.int64(1), np.int64(1), np.int64(1), np.int64(1), np.int64(1),
np.int64(1), np.int64(2), np.int64(1), np.int64(1), np.int64(1),
np.int64(1), np.int64(1), np.int64(2), np.int64(309), np.int64(1),
np.int64(1), np.int64(1), np.int64(1), np.int64(1), np.int64(1),
np.int64(1), np.int64(1), np.int64(1), np.int64(1), np.int64(1),
np.int64(1), np.int64(1), np.int64(1), np.int64(1), np.int64(1),
np.int64(1), np.int64(1), np.int64(1), np.int64(1), np.int64(1),
np.int64(1), np.int64(1), np.int64(1), np.int64(1), np.int64(1),
np.int64(1), np.int64(1), np.int64(1), np.int64(1), np.int64(1),
np.int64(1)]
```



Večina regij je zelo majhnih, kar kaže na prisotnost šuma v binarni sliki. Na sliki so le trije objekti, ostale pa so izolirani posamezni piksli ali drobne regije, ki nastanejo zaradi šumnih motenj pri binarizaciji.

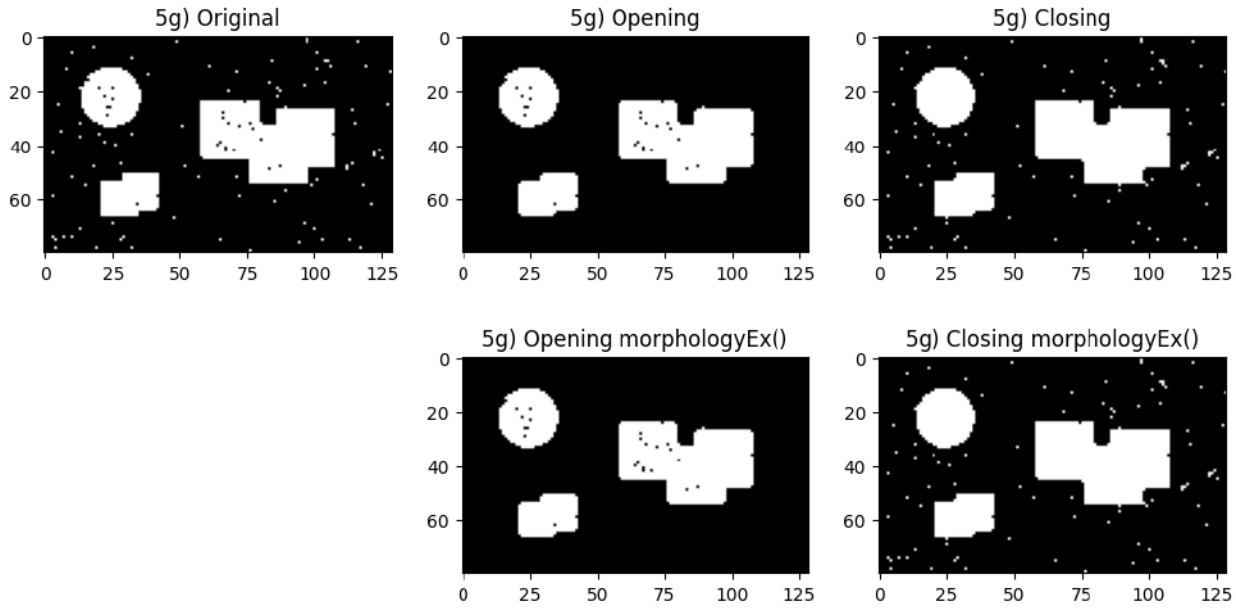
5f) Na sliki preizkusite OpenCV funkciji cv2.dilate() in cv2.erode() ter vizualizirajte njune rezultate. Funkcijama lahko podate tudi poljuben morfološki operator, ki se uporabi za izvedbo operacij. Preizkusite različne operatorje, ki jih lahko sestavite s funkcijo cv2.getStructuringElement(). Ti operatorji lahko zavzamejo različne oblike, ki jih določimo npr. s parametrom shape=cv2.MORPH_RECT ali shape=cv2.MORPH_ELLIPSE.

```
K_rect = cv2.getStructuringElement(cv2.MORPH_RECT, (2, 2))
K_ellipse = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (2, 2))

dil_rect = cv2.dilate(bin, K_rect, iterations=1)
er_rect = cv2.erode(bin, K_rect, iterations=1)
dil_ellipse = cv2.dilate(bin, K_ellipse, iterations=1)
er_ellipse = cv2.erode(bin, K_ellipse, iterations=1)

plt.subplot(2, 3, 1)
plt.imshow(bin, cmap='gray')
plt.title('5f) Original')

plt.subplot(2, 3, 2)
plt.imshow(dil_rect, cmap='gray')
plt.title('5f) Dilate RECT')

plt.subplot(2, 3, 3)
plt.imshow(er_rect, cmap='gray')
plt.title('5f) Erode RECT')

plt.subplot(2, 3, 5)
plt.imshow(dil_ellipse, cmap='gray')
plt.title('5f) Dilate ELLIPSE')

plt.subplot(2, 3, 6)
plt.imshow(er_ellipse, cmap='gray')
plt.title('5f) Erode ELLIPSE')

plt.show()
```

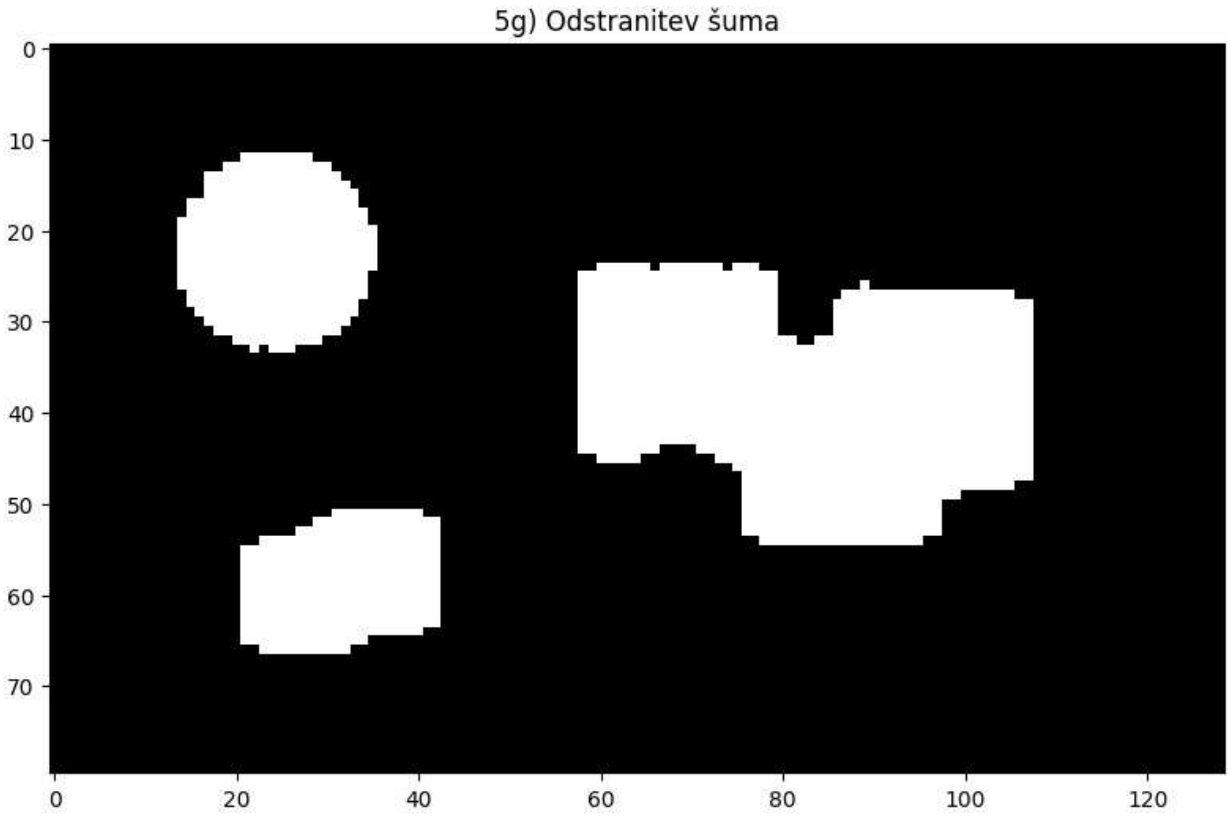


5g) S pomočjo kombinacije funkcij cv2.erode() in cv2.dilate() implementirajte še operaciji odpiranja (angl. opening) in zapiranja (angl. closing) ter rezultat primerjajte z vgrajenima funkcijama implementiranima v sklopu OpenCV funkcije cv2.morphologyEx(). Prikažite rezultat obeh operacij na upragovani sliki regions_noise.png. Glede na rezultate premislite, kako bi odpravi celoten šum, ki je prisoten v izvorni binarni sliki? Rešitev implementirajte in preizkusite.

```
K = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (3, 3))

opening_manual = cv2.dilate(cv2.erode(bin, K, iterations=1), K,
iterations=1)
closing_manual = cv2.erode(cv2.dilate(bin, K, iterations=1), K,
iterations=1)

# morphologyEx()
opening_cv = cv2.morphologyEx(bin, cv2.MORPH_OPEN, K)
closing_cv = cv2.morphologyEx(bin, cv2.MORPH_CLOSE, K)

plt.subplot(2, 3, 1)
plt.imshow(bin, cmap='gray')
plt.title('5g) Original')

plt.subplot(2, 3, 2)
plt.imshow(opening_manual, cmap='gray')
plt.title('5g) Opening')

plt.subplot(2, 3, 3)
plt.imshow(closing_manual, cmap='gray')
plt.title('5g) Closing')

plt.subplot(2, 3, 5)
plt.imshow(opening_cv, cmap='gray')
plt.title('5g) Opening morphologyEx()')

plt.subplot(2, 3, 6)
plt.imshow(closing_cv, cmap='gray')
plt.title('5g) Closing morphologyEx()')

plt.show()
```




```
# Za popolno odstranitev šuma: uporabimo kombinacijo opening +
closing, večje jedro, iteracije >1
K2 = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (5, 5))
clean = cv2.morphologyEx(bin, cv2.MORPH_OPEN, K2)
clean = cv2.morphologyEx(clean, cv2.MORPH_CLOSE, K2)

plt.imshow(clean, cmap='gray')
plt.title("5g) Odstranitev šuma")
plt.show()
```



Za odstranjevanje šuma v binarni sliki sem uporabila kombinacijo morfoloških operacij odpiranja in zapiranja z večjim eliptičnim jedrom velikosti 5×5. Z operacijo odpiranja sem najprej odstranila drobne osamljene pike in manjše šumne regije, nato pa sem z zapiranjem zapolnila luknje in povezala prekinjene dele večjih objektov. Na ta način sem ohranila glavne oblike na sliki, hkrati pa sem učinkovito odpravila šum in dobila čistejši rezultat za nadaljnjo obdelavo.

6. Delo z živimi slikami

6a) V okviru te naloge si bomo ogledali delo z živimi slikami. Za reševanje naloge boste potrebovali spletno kamero. V primeru, da le-te nimate na voljo, lahko nalogo rešite tudi z uporabo poljubnega video izseka. Oglejmo si kodo za pridobitev in prikaz slike s spletne kamere. Za prekinitev izvajanja funkcijo bomo definirali tipko q.

```
cap = cv2.VideoCapture(0)

while True:
    ret, frame = cap.read()
    if not ret:
        break
    cv2.imshow('frame', frame)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
```

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

6b) Izrišite trenutno sliko spletne kamere poleg katere vizualizirate tudi pripadajočo sivinsko sliko. Izrisani sliki zrcalite z uporabo OpenCV funkcije cv2.flip().

```
cap = cv2.VideoCapture(0)

while True:
    ret, frame = cap.read()
    if not ret:
        break
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    frame = cv2.flip(frame, 1)
    gray = cv2.flip(gray, 1)
    cv2.imshow('6b) Barvna slika', frame)
    cv2.imshow('6b) Sivinska slika', gray)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

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