

domaci11_17_0017

December 6, 2020

1

Prvi domaci zadatak

1.0.1

Nikolina Bunijevac, 2017/0017

1.1 Prvi zadatak

```
[1]: from __future__ import print_function

from pylab import *
%matplotlib inline

import skimage

from skimage import *
from skimage.color import *
from skimage.exposure import *
from skimage.filters import *
from skimage.morphology import disk
from skimage import img_as_float

import math
from math import exp

import scipy
from scipy import ndimage

import time

import numpy as np
import imageio
```

```
[12]: f255 = imageio.imread('sekvence/sea.hdr', format='HDR-FI')
f = f255/max(f255.flatten()) #skaliramo sliku na opseg od 0 do 1
```

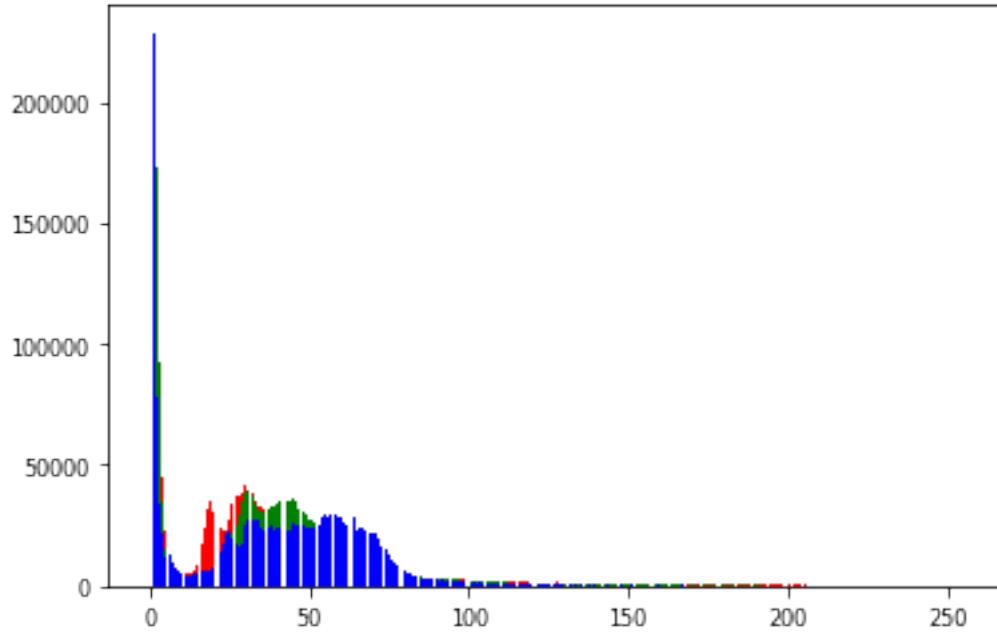
U prvom zadatku potrebno je popraviti kontrast slike "sea.hdr". Mozemo videti na histogramu da su pikseli velikom vecinom slabo osvetljeni (sto se slaze sa prikazom slike kasnije), te stoga zelimo da povecamo osvetljenje slike.

```
[13]: f_red = f255[:, :, 0].flatten()
f_green = f255[:, :, 1].flatten()
f_blue = f255[:, :, 2].flatten()

hist_f_red, bin_edges = np.histogram(f_red, bins=256, range=(0,255))
hist_f_green, bin_edges = np.histogram(f_green, bins=256, range=(0,255))
hist_f_blue, bin_edges = np.histogram(f_blue, bins=256, range=(0,255))

plt.bar(bin_edges[0:-1], hist_f_red, color='r')
plt.bar(bin_edges[0:-1], hist_f_green, color='g')
plt.bar(bin_edges[0:-1], hist_f_blue, color='b')

plt.show()
```



1.a) Potrebno je koriscenjem linearog mapiranja naci odgovarajuce parametre A, B i C tako da redom izlazne slike budu slabo osvetljenje, dobro osvetljene i previse osvetljene. Na sliku pre prikazivanja treba primeniti gama korekciju. Dobijeni parametri su $A = 2$, $B = 20$, $C = 45$. Logicno je da budu veci od 1 jer zelimo da povecamo osvetljenje piksela. Naravno, sa povecanjem skala faktora, sve vise se pojavljuje trava u cosku, medjutim, predeo oko sunca koji je najsvetlij i postepeno "puca", odnosno prelazi u zasicenje (to se desava kad sve piksele vece od 1 setujemo na 1 nakon skaliranja).

```
[17]: # linearne mapiranje

[A, B, C] = [2, 20, 45]
fA = f*A
fB = f*B
fC = f*C

# ako je neki piksel nakon mnozenja postao veci od 1, treba ga setovati na 1
fA[fA>1] = 1;
fB[fB>1] = 1;
fC[fC>1] = 1;

# gama korekcija
fA = exposure.adjust_gamma(fA, 1/2.2)
fB = exposure.adjust_gamma(fB, 1/2.2)
fC = exposure.adjust_gamma(fC, 1/2.2)

# iscrtavanje
fig, axes = plt.subplots(ncols=2, nrows = 2, figsize=(32,16), dpi=80)
ax = axes.ravel()

ax[0].imshow(f); ax[0].set_title('Ulagana slika');
ax[1].imshow(fA); ax[1].set_title('Lin_skalirana slika param. A = ' + str(A));
ax[2].imshow(fB); ax[2].set_title('Lin_skalirana slika param. B = ' + str(B));
ax[3].imshow(fC); ax[3].set_title('Lin_skalirana slika param. C = ' + str(C));
ax[0].axis('off'); ax[1].axis('off'); ax[2].axis('off'); ax[3].axis('off');
plt.rc('font', size=20)

plt.show()
```

Ulagana slika



Lin_skalirana slika param. A = 2



Lin_skalirana slika param. B = 20



Lin_skalirana slika param. C = 45



1.b) Potrebno je ovog puta popraviti kontrast koriscenjem logaritamske, potom i stepene funkcije. Da bismo dobili rezultate koji se vise razlikuju, potrebno je da parametri logaritamskoj funkciji ne zadajemo u ekvidistantnim rastojanjima (kao sto smo priblizno kod linearног skaliranja radili), vec u, na primer, logaritamskoj skali (zbog samih priroda funkcija transformacije).

Prva diskusija: mozemo primetiti da je razlika u osvetljenosti izmedju slike sa parametrom $c = 5$ i slike sa parametrom $c = 50$ slicna kao razlika u osvetljenosti izmedju slike sa parametrom $c = 50$ i slike sa parametrom $c = 500$. Licni utisak: najbolje izgleda slika sa parametrom $c = 50$. Takodje, mozemo primetiti da ovde ne dolazi do zasicanja najsvetlijih piksela oko sunca jer logaritamska funkcija slika opseg $(0,1)$ u isti taj opseg - ne dolazi do prekoracenja (isto vazi i za stepenu funkciju).

```
[6]: def log_transform(img,c):

    # funkcija koja vrši logaritamsku transformaciju slike

    if (c<=0):
        # potrebno je dodati ovaj uslov kako bi argument logaritma bio validan,
        # tj. kako bi se izbeglo deljenje nulom ili negativnim logaritmom
        print('Parametar c mora biti pozitivan!')
        return img

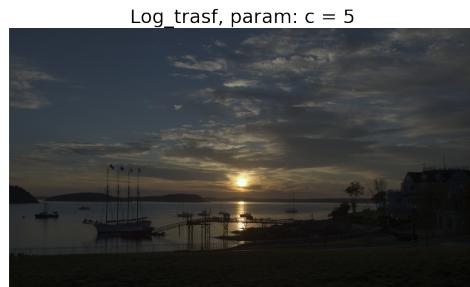
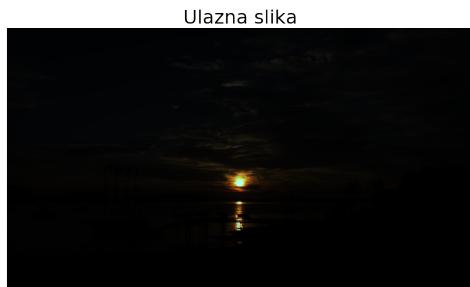
    f_log = log(1 + c*f)/log(1 + c)
    # delimo sa lod(1+c) da bi izlaz bio u opsegu (0,1)

    f_log = exposure.adjust_gamma(f_log, 1/2.2)
```

```
# dodajemo gama transformaciju, iako je mogla biti kasnije uradjena  
return f_log
```

[7]: # prva diskusija

```
[c1,c2,c3] = [5, 50, 500]  
f_log1 = log_transform(f,c1)  
f_log2 = log_transform(f,c2)  
f_log3 = log_transform(f,c3)  
  
# iscrtavanje  
fig, axes = plt.subplots(ncols=2, nrows = 2, figsize=(32,16), dpi=80)  
ax = axes.ravel()  
  
ax[0].imshow(f); ax[0].set_title('Ulagana slika');  
ax[1].imshow(f_log1); ax[1].set_title('Log_trasf, param: c = ' + str(c1));  
ax[2].imshow(f_log2); ax[2].set_title('Log_trasf, param: c = ' + str(c2));  
ax[3].imshow(f_log3); ax[3].set_title('Log_trasf, param: c = ' + str(c3));  
ax[0].axis('off'); ax[1].axis('off'); ax[2].axis('off'); ax[3].axis('off');  
plt.rc('font', size=24)  
  
plt.show()
```



Druga diskusija: mozemo na graficima ispod primetiti da je razlika izmedju dobijenih slika vrlo mala, jer je red velicine argumenta logaritamske transforamcije - c - otprilike isti. Licni utisak: opet $c = 50$ daje najlepsu sliku, iako su razlike minimalne.

```
[8]: # Druga diskusija
```

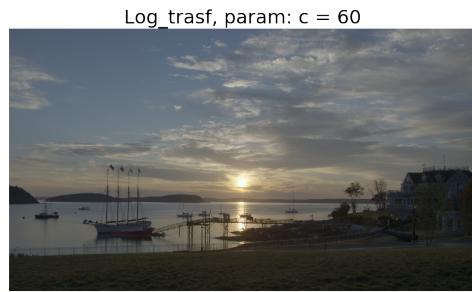
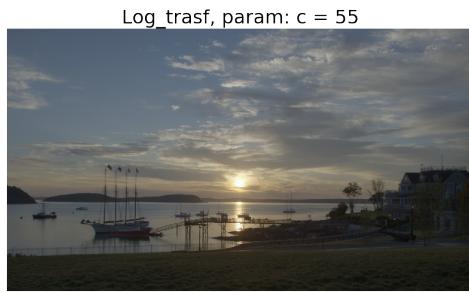
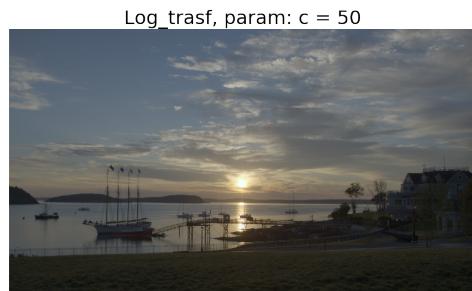
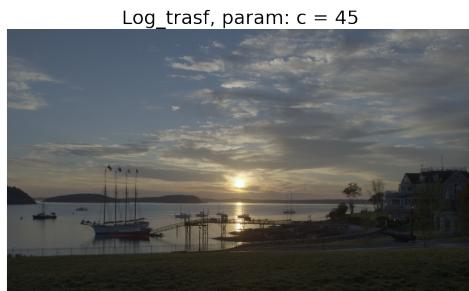
```
[c0,c4,c5,c6] = [45, 50, 55, 60]
f_log0 = log_transform(f,c0)
f_log4 = log_transform(f,c4)
f_log5 = log_transform(f,c5)
f_log6 = log_transform(f,c6)

# iscrtavanje
fig, axes = plt.subplots(ncols=2, nrows=2, figsize=(32,16), dpi=80)
ax = axes.ravel()

ax[0].imshow(f_log0); ax[0].set_title('Log_trasf, param: c = ' + str(c0));
ax[1].imshow(f_log4); ax[1].set_title('Log_trasf, param: c = ' + str(c4));
ax[2].imshow(f_log5); ax[2].set_title('Log_trasf, param: c = ' + str(c5));
ax[3].imshow(f_log6); ax[3].set_title('Log_trasf, param: c = ' + str(c6));
ax[0].axis('off'); ax[1].axis('off'); ax[2].axis('off'); ax[3].axis('off');

plt.rc('font', size=24)

plt.show()
```



1.b) Sada koristimo stepenu funkciju. S obzirom na to da su pikseli slike vrednosti izmedju 0 i 1, samim stepenovanjem nekim brojem vecim od 1 ce se oni smanjivati, dakle potrebno je da eksponent stepene funkcije bude neki broj od 0 do 1. Licni utisak, najbolje ispada slika sa parametrom 0.55, iako su sve generalno jakog loseg izgleda.

```
[9]: def step_transform(img,step):

    # funkcija za stepenu transformaciju slike

    f_step = f**step;
    f_step = exposure.adjust_gamma(f_step, 1/2.2)
    # nije potrebno ni ovde ni kod logaritamske funkcije sredjivati piksele
    # vece od 1 jer ih nece biti

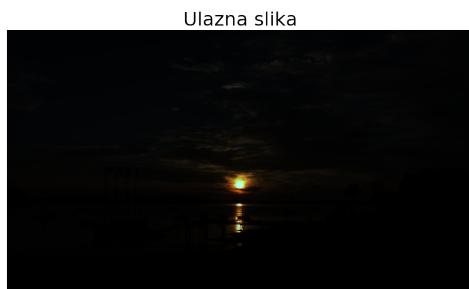
    return f_step
```

```
[10]: [s1,s2,s3] = [0.95, 0.55, 0.3]
f_step1 = step_transform(f,s1)
f_step2 = step_transform(f,s2)
f_step3 = step_transform(f,s3)

# iscrtavanje
fig, axes = plt.subplots(ncols=2, nrows = 2, figsize=(32,16), dpi=80)
ax = axes.ravel()

ax[0].imshow(f); ax[0].set_title('Ulagana slika');
ax[1].imshow(f_step1); ax[1].set_title('Stepena_trasf, param: ' + str(s1));
ax[2].imshow(f_step2); ax[2].set_title('Stepena_trasf, param: ' + str(s2));
ax[3].imshow(f_step3); ax[3].set_title('Stepena_trasf, param: ' + str(s3));
ax[0].axis('off'); ax[1].axis('off'); ax[2].axis('off'); ax[3].axis('off');
plt.rc('font', size=24)

plt.show()
```



1.c) Neki od nacina koje smo pominjali za popravku intenziteta bio je ekvalizacija histograma, medjutim, kao sto se ispod moze videti, on nije dao dobre rezultate, a i sam postupak je skup i zahtevan. Zato ovo nije dobar predlog.

```
[18]: f_hist_eq = skimage.exposure.equalize_hist(f, nbins=256)

fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(32,16), dpi=120)
ax = axes.ravel()

ax[0].imshow(f); ax[0].set_title('Ulaz');
ax[1].imshow(f_hist_eq); ax[1].set_title('Ekvalizacija histograma');
ax[0].axis('off'); ax[1].axis('off')
plt.rcParams['font', size=36)
```



S obzirom na to da mi se najvise svideo izgled slike nakon logaritamske finkcije - nije bilo zasicenja u okolini sunca - iako sam proces jeste zahtevniji od, na primer, linearne mapiranja, ali i manje zahtevan od ekvalizacije histograma, izabrala bih logaritamsku transformaciju kao postupak poboljsanja kvaliteta ove slike.

1.2 Drugi zadatak

Potrebno je izvrsiti selektivnu dekolorizaciju dveju slika. Prva je "marlyn.jpg". Zadatak je radjen u rgb kolor sistemu. Da bismo znali sta da postavimo kao granice po r, g, b slojevima i po intenzitetu, potrebno je bilo prvo izdvojiti te slojeve i plotovati.

```
[3]: img = img_as_float(imread('sekvence/marlyn.jpg'))
```

```
[4]: #izdvajamo segment gde su usta
```

```
lips_xmin = 1200
lips_xmax = 1350
lips_ymin = 900
lips_ymax = 1200
```

```
lips = img[lips_xmin:lips_xmax,lips_ymin:lips_ymax,:]
```

```
[7]: f_r = img[:, :, 0] # izdvajanje r sloja (crvena)
f_g = img[:, :, 1] # izdvajanje g sloja (zelena)
f_b = img[:, :, 2] # izdvajanje b sloja (plava)
img_gray = rgb2gray(img); # crno-bela slika
```

```
fig, ax=plt.subplots(nrows = 2, ncols=3, figsize=(42,32), dpi=240)
#tight_layout()
```

```
ax[0,0].imshow(f_b, cmap='gray'); ax[0,0].set_title('blue');
ax[0,1].imshow(f_g, cmap='gray'); ax[0,1].set_title('green');
ax[0,2].imshow(f_r, cmap='gray'); ax[0,2].set_title('red');
ax[1,0].imshow(img); ax[1,0].set_title('original');
ax[1,1].imshow(img_gray, cmap='gray'); ax[1,1].set_title('gray');
ax[1,2].imshow(lips); ax[1,2].set_title('Usne');
ax[0,0].axis('off'); ax[0,1].axis('off'); ax[0,2].axis('off')
ax[1,0].axis('off'); ax[1,1].axis('off'); ax[1,2].axis('off')
plt.rc('font', size=50)
```

```
plt.show()
```



Nova slika koju pravimo mora da bude sa r,g i b slojem, tacnije, ne sme biti jednoslojna. Ideja je prvo da u sve slojeve upisemo vrednosti crno-bele slike, koju dobijamo pomocu funkcije `rgb2gray`, a da zatim usne izdvojimo pomocu poredjenja svakog sloja sa nekim pragom. Vidimo da je osvetljenost usana mala, dok najvise ima crvene boje, a plave dosta manje, a zelene najmanje, pa na osnovu toga isprobavamo razlicite pragove (korisceni su 0.3 za crvenu, 0.17 za zelenu, 0.35 za plavu i 0.35 za osvetljenje). Nakon ovoga dobijamo usta, ali sa rupicama jer je i sama maska imala rupice, pa isfiltriramo masku i ponovo je primenimo. Videcemo da su usta popunjena, crni delovi nisu crni, a usne su pune.

```
[19]: img_new = ones(shape(img));
img_new[:, :, 0] = img_gray;
img_new[:, :, 1] = img_gray;
img_new[:, :, 2] = img_gray;

mask = (img[:, :, 0] > 0.3) & (img[:, :, 1] < 0.17) & (img[:, :, 2] < 0.35) &  
    ~ (img_gray < 0.35);
img_new[mask, :] = img[mask, :];

#ok ali ima rupica, pa cemo da popunimo masku u sledecoj celiji
fig, ax=plt.subplots(ncols = 2, figsize=(32,16), dpi=240)
tight_layout()

ax[0].imshow(img); ax[0].set_title('Ulazna slika');
ax[1].imshow(img_new); ax[1].set_title('Slika nakon primene prve maske');
ax[0].axis('off'); ax[1].axis('off')

plt.rc('font', size=24)
```



```
[52]: img_new_full = ones(shape(img));
img_gray = rgb2gray(img);
img_new_full[:, :, 0] = img_gray;
img_new_full[:, :, 1] = img_gray;
img_new_full[:, :, 2] = img_gray;

selem = disk(1)
mask_full_1 = mask
# nova maska ce da bude filtrirana prethodna maska
mask_full_1[:] = rank.mean(mask[:, :], selem=selem)

prag = 0.99999999
mask_true = mask_full_1>prag;
mask_false = mask_full_1<=prag;
mask_full_1[mask_true] = True;
mask_full_1[mask_false] = False;

# moramo jos jedno filtriranje da izvrsimo, i posto ce ovaj filter da izduoji
# sredinu usana, moramo da kao konacnu masku uzmemu uniju ove i prethodne
selem = disk(40)
mask_full_2 = median(mask_full_1, selem=selem)
prag = 0.99
mask_true = mask_full_2>prag;
# kao konacnu masku uzimamo uniju ove i prethodne
mask_true = mask_true + mask_full_1
mask_true = mask_true >= 1
```

```

# samo tamo gde je vrednost maske True, upisi vrednosti iz obojene slike
img_new_full[mask_true,:,:] = img[mask_true,:,:]

fig, ax=plt.subplots(ncols = 2, figsize=(32,16), dpi=240)
tight_layout()

ax[0].imshow(img); ax[0].set_title('Ulagna slika'); ax[0].axis('off')
ax[1].imshow(img_new_full); ax[1].set_title('Slika nakon primene popunjene maske'); ax[1].axis('off')

plt.rc('font', size=40)

```

Ulagna slika



Slika nakon primene popunjene maske



1.2.1 2. Rad sa slikom "street.jpg". U ovom slučaju, prva maska je prilicno dobro izdvajala kisobran, pa nije bilo potrebe za filtriranjem maske.

```
[85]: img2_2 = img_as_float(imread('sekvenca/street.jpg'))
```

```
[86]: # izdvajanje kisobrana
[umbr_xmin, umbr_xmax, umbr_ymin, umrb_max] = [170, 250, 225, 330];
umbr = img2_2[umbr_xmin:umbr_xmax, umbr_ymin:umrb_max]
```

```
[87]: f2_r = img2_2[:, :, 0]
f2_g = img2_2[:, :, 1]
f2_b = img2_2[:, :, 2]
img2_gray = rgb2gray(img2_2);
```

```

fig, ax=plt.subplots(nrows = 2, ncols=3, figsize=(32,16), dpi=180)
tight_layout()

ax[0,0].imshow(f2_r, cmap='gray'); ax[0,0].set_title('red');
ax[0,1].imshow(f2_g, cmap='gray'); ax[0,1].set_title('green');
ax[0,2].imshow(f2_b, cmap='gray'); ax[0,2].set_title('blue');
ax[1,0].imshow(img2_2); ax[1,0].set_title('original');
ax[1,1].imshow(img2_gray, cmap='gray'); ax[1,1].set_title('gray');
ax[1,2].imshow(umbr); ax[1,2].set_title('kisobran');

ax[0,0].axis('off'); ax[0,1].axis('off');
ax[1,0].axis('off'); ax[1,1].axis('off');

plt.rc('font', size=40)

plt.show()

```



```

[91]: img2_new = ones(shape(img2_2));
img2_new[:, :, 0] = img2_gray;
img2_new[:, :, 1] = img2_gray;
img2_new[:, :, 2] = img2_gray;

mask = (img2_2[:, :, 0] > 0.5) & (img2_2[:, :, 1] > 0.7) & (img2_2[:, :, 2] < 0.6) &
       (img2_gray > 0.5) & (img2_gray < 0.99) | (img2_2[:, :, 0] > 0.7) & (img2_2[:, :, 1] > 0.8) & (img2_2[:, :, 2] < 0.8) & (img2_gray > 0.7) & (img2_gray < 0.99);
img2_new[mask, :] = img2_2[mask, :];
# sasvim je ok ispalo, nema potreba za filtriranjem maske

```

```

fig, ax=plt.subplots(ncols = 2, figsize=(32,16), dpi=180)
tight_layout()

ax[0].imshow(img2_2); ax[0].set_title('ulazna slika'); ax[0].axis('off')
ax[1].imshow(img2_new); ax[1].set_title('izlazna slika'); ax[1].axis('off')

plt.rc('font', size=40)

plt.show()

```



1.3 Treci zadatak

- 1.3.1 Potrebno je bilo napraviti funkciju bilateral_filter. Detaljniji opis same funkcije nalaze se u kodu. Funkcija je dosta sporija od ugradjene ako radimo sa slikama srednjih velicina, za male radijuse, dok je za radijuse 20 i 40 sporija manje oko 1.5 put. Za slike veoma malih dimenzija oko 5 puta je sporija funkcija od ugradjene (poslednje slike). Uporedjivanjem izlaza ugradjene i implementirane funkcije filtriranja moze se videti daju priblizno iste rezultate.

```
[2]: img3 = img_as_float(imread('sekvence/ckt_board.tif'))
img3_gray = skimage.color.rgb2gray(img3)
min_3 = min(img3_gray.flatten())
max_3 = max(img3_gray.flatten())
img3_gray = exposure.rescale_intensity(img3_gray, in_range=(min_3, max_3),
                                         out_range=(0,1))
```

```
[3]: def bilateral_filter(x, radius, sigma_s, sigma_r):
    # funkcija koja vrsti filtriranje ocuvanjem ivica
    # uzimamo min i max vrednost ulazne slike kako bismo znali da li je
    # uravnotezenog osvetljaja
```

```

xmin = min(x.flatten())
xmax = max(x.flatten())

# proveravamo da li je slika u opsegu (0,1), ako nije, prebacimo je
# u taj opseg
if (xmax>1):
    x = img_as_float(x)

# flag b ce biti true ako je sve u redu sa slikom, pa cemo da vrsimo obradu
# ako je false, znaci da neki od ulaznih parametara ne ispunjava trazene
# uslove o cemu ce govoriti izlazna poruka koju printujemo za svaku gresku
b = True

if ((xmax-xmin < 0.98)):
    # ne moramo da zahtevamo da bude celokupan opseg iskoriscen, vec oko 98%
    # ako nije, onda se trazi preskaliranje
    print('Izvrsiti preskaliranje slike')
    b = False

if (sigma_r>1) | (sigma_r<0):
    # provera da li stdev intenziteta pripada odgovarajucom opsegu
    print('Standardna devijacija intenziteta van okvira [0,1]')
    b = False

if(b):
    # ako nije bilo gresaka (if(b))

        [xdimx, xdimy] = shape(x) # dimenzije slike x
        r = radius                 # skraceni zapis radi preglednosti

        # PRAVLJENJE PROSIRENE SLIKE y
        y = np.empty((xdimx+2*r, xdimy+2*r))

        [ydimx, ydimy] = shape(y) # dimenzije prosirene slike

        # kopiranje centralnih piksela
        y[r:xdimx+r, r:xdimy+r] = x;

        # popunjavanje ,,coskova"
        y[0:r,0:r] = x[0,0]*ones((r,r))
        y[0:r,xdimy+r:ydimy] = x[0,xdimy-1]*ones((r,r))
        y[xdimx+r:ydimx,0:r] = x[xdimx-1,0]*ones((r,r))
        y[xdimx+r:ydimx,xdimy+r:ydimy] = x[xdimx-1,xdimy-1]*ones((r,r))

        # popunjavanje ivicnih redova i kolona
        y[0:r, r:xdimy+r] = x[0,:]*ones((r,xdimy))

```

```

y[xdimx + r:xdimx, r:xdimy+r] = x[xdimx-1,:]*ones((r,xdimy))
# iz nekog razloga, nije moglo na slican nacin da se izvrsi kopiranje
# pa mora kroz petlju
for i in range(0,r):
    y[r:xdimx+r,i] = x[:,0];
    y[r:xdimx+r,xdimy+r+i]=x[:,xdimy-1];

# izlaz
x_out = zeros(shape(x))

# prvi cinilac u maski je isti bez obzira na redni broj piksela
# (zavisi samo od k i l) pa mozemo pre for petlje napraviti prvi
# deo maske, da ne bismo ponavljaljali stalno u petlji
maska_1 = ones((2*radius+1,2*radius+1))
for k in range(-radius,radius+1):
    # range treba da obuhvati u radius, pa moramo da dodamo 1
    for l in range(-radius,radius+1):
        maska_1[k+r,l+r] = exp(-(k**2+l**2)/(2*sigma_s**2))

# FILTRIRANJE
for m in range(radius,xdimx+radius):
    for n in range(radius,xdimy+radius):
        # prolazimo kroz sve piksele u originalnoj slici, izracunavamo
        # masku i nove vrednosti za piksel na poziciji [m,n]

        # ovim linijama kod prethodile su dve zapetljanje for petlje,
        # cije je izvrsavanje sveukupno trajalo 20x vise, ali nakon
        # malo istrazivanja rada sa matricama u pythonu,
        # kod se suo na par linija)

        # pravljenje drugog dela maske po formuli
        maska_2 = np.exp(-((y[m-r:m+r+1,n-r:n+r+1]-y[m,n])**2)/
→(2*sigma_r**2));
        # masku nakon proizvoda prvog i drugog dela dobijemo u
        # „razvijenoj”, „flatten” formi, medjutim to je pogodan
        # oblik za kasniju korelaciju, pa nema potrebe da prepravljamo
        # dimenzije
        maska = maska_1.flatten()*maska_2.flatten()

        # normiranje maske
        maska = maska/sum(maska)

# KORELACIJA
x_out[m-r,n-r] = np.dot(maska, y[m-r:m+r+1,n-r:n+r+1].flatten())

```

```
# nakon prolaska kroz sve piksele, vracamo rezultat - isfiltriranu sliku
return x_out
```

```
[4]: r = [2,4,20,40]      # preporucene vrednosti za radijus
execution_time_my = [0.0, 0.0, 0.0, 0.0];
execution_time_norm_my = [0.0, 0.0, 0.0, 0.0];
execution_time_default = [0.0, 0.0, 0.0, 0.0];
execution_time_norm_default = [0.0, 0.0, 0.0, 0.0];
# inicializacija liste, da ne bi slicajno bilo problema sa dimenzijama
img_out = [img3_gray,img3_gray,img3_gray,img3_gray]
# inicializacija liste
img_out_default = [img3_gray,img3_gray,img3_gray,img3_gray]
sigma_s = 1
sigma_r = 0.7

# prolazimo u petlji kroz sve vrednosti radijusa
for i in range(0, len(r)):

    # pozivanje funkcije bilateral_filter i merenje vremena za svaki od radijusa
    start = time.time()
    img_out[i] = bilateral_filter(img3_gray,r[i],sigma_s,sigma_r)
    end = time.time()
    execution_time_my[i] = end-start
    execution_time_norm_my[i] = execution_time_my[i]/np.size(img3_gray);

    # pozivanje ugradjene funkcije i merenje vremena za svaki od radijusa
    start = time.time()
    img_out_default[i] = restoration.
    ↪denoise_bilateral(img3_gray,2*r[i]+1,sigma_r,sigma_s,mode='edge')
    end = time.time()
    execution_time_default[i] = end-start
    execution_time_norm_default[i] = execution_time_default[i]/np.
    ↪size(img3_gray);
```

```
[6]: # plotovanje

fig, ax=plt.subplots(nrows = 5, ncols = 2,figsize=(15,24), dpi=72)

ax[0,0].imshow(img3_gray,cmap='gray'); ax[0,0].set_title('ulaz');
ax[0,1].imshow(img3_gray,cmap='gray'); ax[0,1].set_title('ulaz');
ax[0,0].axis('off'); ax[0,1].axis('off')

for i in range(0,4):
```

```

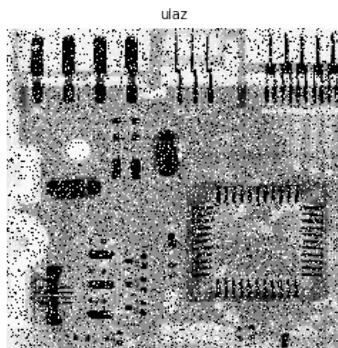
    ax[(i+1),0].imshow(img_out[i],cmap='gray'); ax[(i+1),0].set_title('moja - r\u
\u21d3=' + str(r[i]) + ', vreme: ' + str(round(execution_time_my[i],3)) + ',\u
\u21d3normirano vreme: ' + str(round(execution_time_norm_my[i]*1e6,3)) + 'us/pix');
    ax[(i+1),1].imshow(img_out_default[i],cmap='gray'); ax[(i+1),1].
\u21d3set_title('ugradjena - r = ' + str(r[i]) + ', vreme: ' +
\u21d3str(round(execution_time_default[i],3)) + ', normirano vreme: ' +
\u21d3str(round(execution_time_norm_default[i]*1e6,3)) + 'us/pix');

    ax[(i+1),0].axis('off'); ax[(i+1),1].axis('off')

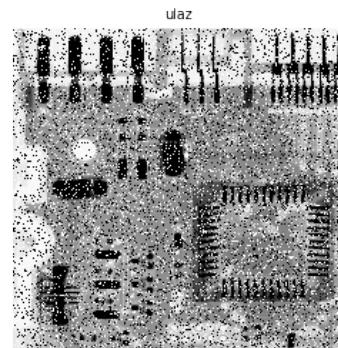
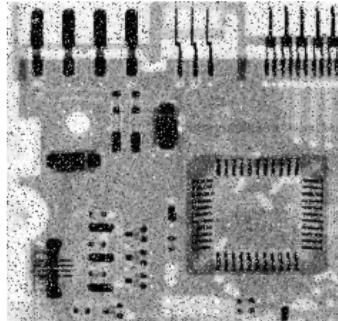
plt.rc('font', size=8)

plt.show()

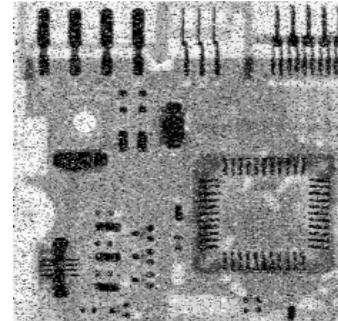
```



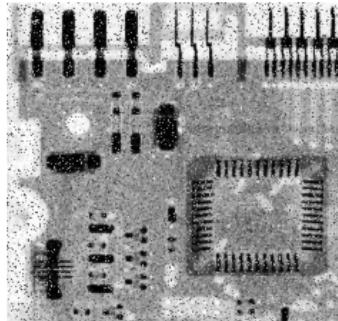
moja - r =2, vreme: 5.179, normirano vreme: 25.868us/pix



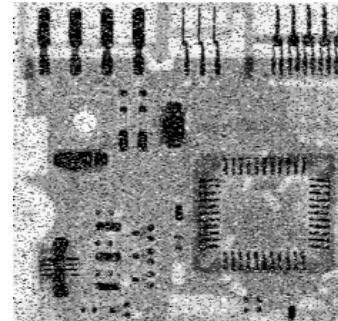
ugradjena - r =2, vreme: 0.109, normirano vreme: 0.546us/pix



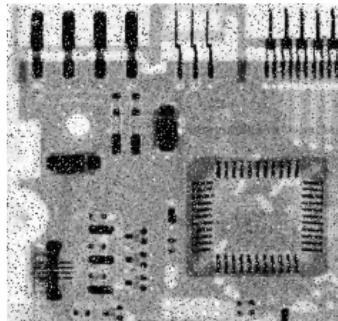
moja - r =4, vreme: 6.014, normirano vreme: 30.04us/pix



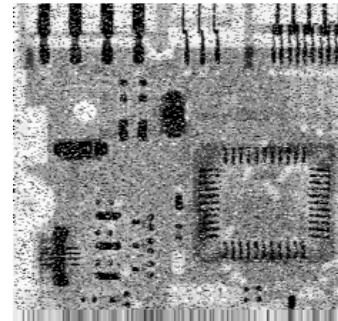
ugradjena - r =4, vreme: 0.414, normirano vreme: 2.067us/pix



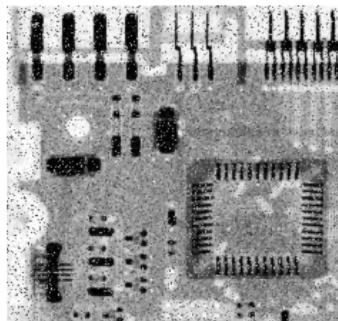
moja - r =20, vreme: 12.192, normirano vreme: 60.898us/pix



ugradjena - r =20, vreme: 8.195, normirano vreme: 40.936us/pix



moja - r =40, vreme: 39.853, normirano vreme: 199.068us/pix



ugradjena - r =40, vreme: 24.976, normirano vreme: 124.756us/pix

```
[24]: # veoma mala slika
```

```
a = np.zeros((7,7))
a[0:3,0:3] = 1

r = 4;
sigma_s = 1
sigma_r = 1

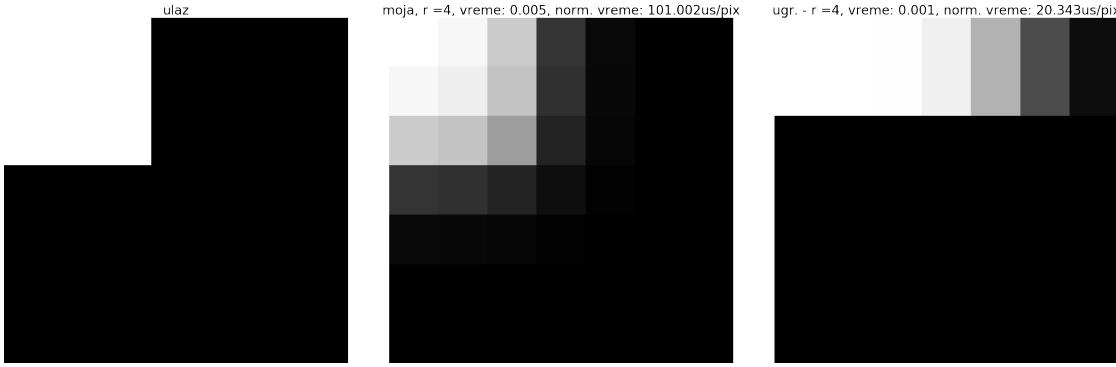
start = time.time()
a_out = bilateral_filter(a,r,sigma_s,sigma_r)
end = time.time()
execution_time_my = end-start
execution_time_norm_my = execution_time_my/np.size(a);

start = time.time()
a_out_default = restoration.
    ↪denoise_bilateral(a,2*r+1,sigma_r,sigma_s,mode='edge')
end = time.time()
execution_time_default = end-start
execution_time_norm_default = execution_time_default/np.size(a);

fig, ax=plt.subplots(nrows = 1, ncols = 3,figsize=(30,90), dpi=80)
tight_layout()

ax[0].imshow(a, cmap='gray'); ax[0].set_title('ulaz'); ax[0].axis('off')
ax[1].imshow(a_out,cmap='gray'); ax[1].set_title('moja, r =' + str(r) + ','
    ↪vreme: ' + str(round(execution_time_my,3)) + ', norm. vreme: ' +
    ↪str(round(execution_time_norm_my*1e6,3)) + 'us/pix'); ax[1].axis('off')
ax[2].imshow(a_out_default,cmap='gray'); ax[2].set_title('ugr. - r =' + str(r) +
    ↪+ ', vreme: ' + str(round(execution_time_default,3)) + ', norm. vreme: ' +
    ↪str(round(execution_time_norm_default*1e6,3)) + 'us/pix'); ax[2].axis('off')

plt.rc('font', size=20)
plt.show()
```



- 1.3.2 Ono sto jos mozemo ovde prokomentarisati jeste da ugradjena funkcija ima odredjenih bagova, ponekad nece da se izvrsi uopste, kao sto je u primeru za radius 40. Vrednosti slike koje vraca su nan. Takodje, mozemo primetiti da su za nijansu lepse isfiltrirane slike nasom funkcijom, u odnosu na ugradjenu. Vreme izvrsavanja nije linerano raslo povecavanjem radijusa - za $r = 4$ priblizno je isto kao za filtriranje od $r = 2$. Takodje, za nize vrednosti radijusa ugradjena funkcija zavrsava se i do 50 puta brze, dok za vece i manje od 2 puta. Za manju sliku takodje nasa funkcija daje brze rezultate i za manje velicine filtra (ovog puta 5 puta sporije od ugradjene, 101.002us/px, za razliku od 30.04us/px za istu velicinu radijusa za prvu sliku), a rezultat koji daje je neuporedivo bolji od rezultata dobijenog ugradjenom funkcijom.
- 1.3.3 Ako nije presudno da filtriranje bude za red velicine brze, izabrala bih uvek rucno napravljenu funkciju jer se nikad nije desilo da se ne izvrsi kako treba ili da otkaze.