In [1]:
```
from pylab import *
import imageio
imageio.plugins.freeimage.download()
import copy
import numpy as np
from skimage.color import rgb2gray
from skimage.color import gray2rgb
imageio.plugins.freeimage.download()
from skimage.exposure import *
from scipy import ndimage
from ctypes import *
from numpy.ctypeslib import ndpointer
from skimage.restoration import denoise_bilateral
import time
plt.style.use('dark_background')
```

In [2]:
```
f = imageio.imread('../sekvence/sea.hdr', format = 'HDR-FI')
f = f/max(f.flatten())
```

1. a) A = 3, B = 11, C = 37;

In [3]:
```
a = 3;
mul = np.full(shape(f),a); #maska sa zeljenom vrenosti skaliranja
tmp = np.multiply(mul,f); #skaliranje svih piksela
tmp[tmp>1] = 1; #zasicivanje piksela koji su izasli iz opsega
tmp[tmp<0] = 0; #zasicivanje piksela koji su izasli iz opsega
res = np.power(tmp,np.full(shape(f),1/2.2)); #primena gama korekcije
imshow(res);
axis('off');
```

In [4]:
```
b = 11;
mul = np.full(shape(f),b);
tmp = np.multiply(mul,f);
tmp[tmp>1] = 1;
tmp[tmp<0] = 0;
res = np.power(tmp,np.full(shape(f),1/2.2));

imshow(res);
axis('off');
```

In [5]:
```
c = 37;
mul = np.full(shape(f),c);
tmp = np.multiply(mul,f);
tmp[tmp>1] = 1;
tmp[tmp<0] = 0;
res = np.power(tmp,np.full(shape(f),1/2.2));

imshow(res);
axis('off');
```

1. b) Za logaritamsku funkciju je odabran parametar a= 150, dok je za stepenu funkciju odabran parametar a=0.65. Povecavanjem parametra kod stepene funcije dobice se tamnija slika, dok ce se smanjivanjem dobiti svetlija slika. Kod logaritamske funkcije smanjivanjem parametra će se dobiti tamnija slika, dok će povećavanje vrednosti parametra slka biti svetlija.

In [6]:
```
a = 150;
tmp = log(1 + a*f)/log(1+a); #primena logaritaske funkcije, vrednosti su odmah skaliran
tmp[tmp>1] = 1;
tmp[tmp<0] = 0;
res = np.power(tmp,np.full(shape(f),1/2.2));
imshow(res);
axis('off');
```

In [7]:
```
a = 0.65;
mul = np.full(shape(f),a); #maska za primenu stepene funkcije
tmp = np.power(f,mul); #primena stepene funkcije
tmp[tmp>1] = 1;
tmp[tmp<0] = 0;
res = np.power(tmp,np.full(shape(f),1/2.2));
imshow(res);
axis('off');
```

1. c) Ideja algoritma je da se delovi slike koji su različito osvetljeni na slici(ali i u stvarnosti) zasebno obrađuju. Konkretno na ovoj slici su to trava i kući koji su tamniji, nebo i more koji su svetliji i sunce koje je najsvetlije. Predloženi algoritam je na jednostavan način pretpostavljao koji su delovi slike u svarnosti svetli, a koji tamni na osnovi grayscale slike. Osim na osnovu osvetljenosti konkretne grupe piksela osvetljenost nekog dela slike bi se mogle pretpostavljati i na osnovu osvetljenosti piksela u njenoj okolini. Za svaki od 3 grupe piksela je urađeno skaliranje histograma, dok je za piksele koji čine more, nebo i sunce pre skaliranja primenjena stepena funkcija. Pri primeni ovog algoritma dolazi do problema pri obradi delova slike koji čine brda i čamci. Oni su svrstani u tamne delove slike, ali pošto su okruženi svetlim delovima oni su neprirodno prikazani.

In [8]:
```
ma = rgb2gray(f);
```

```
ma1 = np.zeros(shape(f));
ma1[:,:,0] = ma;
ma1[:,:,1] = ma;
ma1[:,:,2] = ma;
ma1 = ma1/max(ma1.flatten());

g = copy.deepcopy(f)
g[ma1>0.004] = 0; #odabir zeljenih piksela

g1 = np.zeros(shape(g))
g1[:,:,0] = rescale_intensity(g[:,:,0],in_range=(0.0,0.004),out_range=(0,0.15)) #skalir
g1[:,:,1] = rescale_intensity(g[:,:,1],in_range=(0.0,0.004),out_range=(0,0.15)) #skalir
g1[:,:,2] = rescale_intensity(g[:,:,2],in_range=(0.0,0.004),out_range=(0,0.15)) #skalir

g = copy.deepcopy(f)
g[ma1>0.3] = 0;
g[ma1<0.004] = 0;

y = 1/1.6;
g = np.power(g,np.full(shape(f),y)); #gama korekcija

g2 = np.zeros(shape(g))
g2[:,:,0] = rescale_intensity(g[:,:,0],in_range=(np.power(0.004,y),0.3),out_range=(0,0.
g2[:,:,1] = rescale_intensity(g[:,:,1],in_range=(np.power(0.004,y),0.3),out_range=(0,0.
g2[:,:,2] = rescale_intensity(g[:,:,2],in_range=(np.power(0.004,y),0.3),out_range=(0,0.

g = copy.deepcopy(f)
g[ma1<0.3] = 0;

y = 1/1.5;
g = np.power(g,np.full(shape(f),y));

g3 = np.zeros(shape(g))
g3[:,:,0] = rescale_intensity(g[:,:,0],in_range=(np.power(0.3,y),1),out_range=(0.55,1))
g3[:,:,1] = rescale_intensity(g[:,:,1],in_range=(np.power(0.3,y),1),out_range=(0.55,1))
g3[:,:,2] = rescale_intensity(g[:,:,2],in_range=(np.power(0.3,y),1),out_range=(0.55,1))


f1 = np.zeros(shape(f))
f1[ma1>=0.3] = g3[ma1>=0.3];
f1[ma1<=0.3] = g2[ma1<=0.3];
f1[ma1<=0.004] = g1[ma1<=0.004];

f2 = np.power(f1,np.full(shape(f),1/2.2));

fig,axes = plt.subplots(figsize=(32,8),dpi=200)
axis('off');
imshow(f2);
```

1. Pri izdvajanju traženih delova slike korišćen je lab sistem boja. Pri izradi oba dela zadatka prvo je filtrirana slika da bi se izdvojili delovi slike u kojima se nalaze tražene boje. Zatim je urađena operacija erozije binarizovane slike kojom su izdvojeni samo delovi slike u kojima se nalaze traženi delovi slike, koji čine velike povezane oblasti u okviru binarizovane slike. Takođe je u okviru prvog dela zadatka primenjena i operacija vetrikalog popunjavanja praznina da bi se obojio ceo traženi deo slike.

In [9]:
```
from skimage.color import rgb2lab
img = imread('../sekvence/marlyn.jpg')

la = rgb2lab(img);
l = np.multiply(la[:,:,0]>5,la[:,:,0]<45); #filtracija komponenti lab sistema boja
a = np.multiply(la[:,:,1]>20,la[:,:,1]<101);
b = np.multiply(la[:,:,2]>20,la[:,:,2]<195);
s = (np.multiply(l,a));

s = s*255.0;
filter_mask = ones(shape=(15,30), dtype='uint8')
filter_mask_norm = filter_mask/sum(filter_mask.flatten())
img_blured = ndimage.correlate(s,filter_mask_norm) #primena maske da bi se utvrdilo kol
img_blured[img_blured<130]=0; #binarizacije slike
img_blured[img_blured>=130]=1;

for i in range(0,shape(img_blured)[1]-1): # pounjavanje rupa u okviru binariyovane slik
    img_blured[:,i] = ndimage.binary_fill_holes(img_blured[:,i])

gray = gray2rgb(rgb2gray(img))
final = gray*255.0

img_bl = np.zeros(shape(img))
img_bl[:,:,0] = img_blured
img_bl[:,:,1] = img_blured
img_bl[:,:,2] = img_blured
```

```
final[img_bl.astype(bool)] = img[img_bl.astype(bool)]

fig,axes = plt.subplots(figsize=(32,8),dpi=200);
axis('off');
imshow(final.astype(np.uint8));
```

In [10]:
```
img = imread('../sekvence/street.jpg')

la = rgb2lab(img);

l = np.multiply(la[:,:,0]>65,la[:,:,0]<256);
a = np.multiply(la[:,:,1]>-100,la[:,:,1]<-5);
b = np.multiply(la[:,:,2]>20,la[:,:,2]<195);
s = np.multiply(np.multiply(l,a),b);

s = s*255.0
filter_mask = ones(shape=(8,8), dtype='uint8')
filter_mask_norm = filter_mask/sum(filter_mask.flatten())
img_blured = ndimage.correlate(s,filter_mask_norm)
img_blured[img_blured<100]=0;
img_blured[img_blured>=100]=1;
```

```
gray = gray2rgb(rgb2gray(img))
final = np.zeros(shape(img))

for i in range(0,shape(img)[0]-1):
    for ii in range(0,shape(img)[1]-1):
        if(img_blured[i,ii]==1):
            final[i,ii,:] = img[i,ii,:];
        else:
            final[i,ii,:] = gray[i,ii,:]*255.0;

fig,axes = plt.subplots(figsize=(32,8),dpi=200);
imshow(final.astype(np.uint8));
axis('off');
```

1. U okviru ovog zadatka su implementirane dve funkcije za Bilateralni filter. Jedna u jeziku Python i druga u jeziku C. Funkcija u jeziku C se pokreće pomoću bilateral.dll fajla. Funkciju je moguće pokrenuti samo na Windows operativnom sistemu.

In [11]:
```
#wrapper funkcija za funkciju pisanu u C-u
def bilateral_filter_c(x,radius,sigma_s,sigma_r):

    bil_dll = CDLL("./bilateral.dll")#učitavanje dll-a sa funkcijom
    _doublepp = ndpointer(dtype=np.uintp, ndim=1, flags='C') #definisanje pokazivača ko
    bil_dll.bilateral_filter.argtypes = [_doublepp, _doublepp, c_int, c_int, c_int, c_f
    bil_dll.bilateral_filter.restype = None

    img = rgb2gray(x) #dobijanje grayscale slike

    gr = np.zeros((shape(img)[0],shape(img)[1])) #formiranje placeholder matrice koja c
    xpp = (img.__array_interface__['data'][0] + np.arange(img.shape[0])*img.strides[0])
    ypp = (gr.__array_interface__['data'][0] + np.arange(gr.shape[0])*gr.strides[0]).as
```

```
    bil_dll.bilateral_filter(xpp,ypp,c_int(shape(img)[0]),c_int(shape(img)[1]),radius,s

    return gr
```

In [12]:

```
def bilateral_filter_p(x, r, sigma_s, sigma_r):
    if type(x[0,0])==np.uint8:
        x = x/255.0;
    elif (type(x[0,0])!=np.float64):
        print("Nije odgovarajuci tip podataka matrice!")
        return x;
    if sigma_r<0:
        print("Nije uneta dobra vrednost za sigma_r")
        return x;
    if sigma_r>1:
        print("Nije uneta dobra vrednost za sigma_r")
        return x;
    res = np.zeros(shape(x));
    sh = shape(x)
    tmp = np.zeros((sh[0]+2*r,sh[1]+2*r))
    tmp[r:r+sh[0],r:r+sh[1]] = x;
    for i in range(0,r):
        tmp[i,r:r+sh[1]] = x[0,:];
        tmp[sh[0]+2*r-1-i,r:r+sh[1]] = x[sh[0]-1,:];
        tmp[r:r+sh[0],i] = x[:,0];
        tmp[r:r+sh[0],sh[1]+2*r-1-i] = x[:,sh[1]-1];

    # posto prostorna komponenta ne zavisi od vrednosti piksela ona se racuna samo jeda
    e1 = np.zeros((2*r+1,2*r+1))
    for k in range(-r,r+1):
        for l in range(-r,r+1):
            e1[k+r,l+r] = -(np.power(k,2)+np.power(l,2))/2/np.power(sigma_s,2);

    #racunanje izlaza filtra
    for n in range(r,r+sh[1]):
        for m in range(r,r+sh[0]):
            tmp2 = np.exp(-((tmp[m-r:m+r+1,n-r:n+r+1]-tmp[m,n])**2)/2/(sigma_r)+e1);
            tmp1 = np.multiply(tmp2,tmp[m-r:m+r+1,n-r:n+r+1]);
            res[m-r,n-r] = sum(tmp1)/sum(tmp2);#normaizacija vrednosti
    return res;
```

In [15]:

```
def test_bilateral(img, radius, sigma_s, sigma_r):
    if type(img[0,0])==np.uint8:
        img = img/255.0;
    elif (type(x[0,0])!=np.float64):
        print("Nije odgovarajuci tip podataka matrice!")
        return x;
    if sigma_r<0:
        print("Nije uneta dobra vrednost za sigma_r")
        return x;
    if sigma_r>1:
        print("Nije uneta dobra vrednost za sigma_r")
        return x;

    gr = rgb2gray(img)

    start = time.time()
    res1 = bilateral_filter_p(gr,radius, sigma_s, sigma_r)
    end = time.time()

    print("Radius: "+ str(radius)+ "pix\n")
```

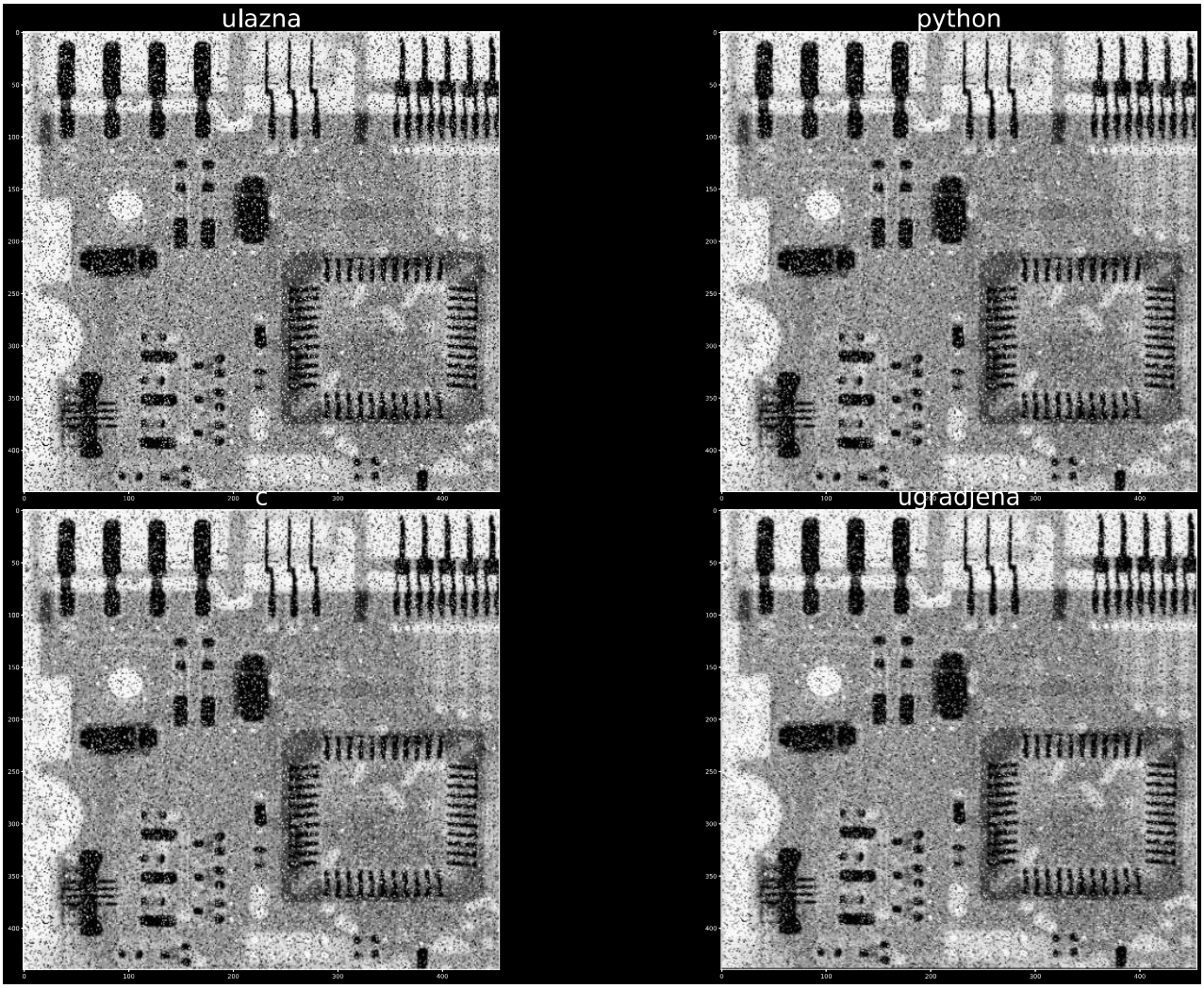
```
    print("Funkcija implemetirana u Pajtonu:")
    execution_time = (end - start);
    print("Vreme izvrsavanja: "+ str(round(execution_time,3))+ "s")

    execution_time_norm = execution_time/np.size(gr)
    print("Vreme izvrsavanja:"+ str(round(execution_time*1e6,3))+ " us/pix \n")

    start = time.time()
    res2 = bilateral_filter_c(gr,radius, sigma_s, sigma_r)
    end = time.time()

    print("Funkcija implementirana u c-u:")
    execution_time = (end - start);
    print("Vreme izvrsavanja: "+ str(round(execution_time,3))+ "s")

    execution_time_norm = execution_time/np.size(img)
    print("Vreme izvrsavanja:"+ str(round(execution_time*1e6,3))+ " us/pix \n")


    start = time.time()
    res3 = denoise_bilateral(gr, radius*2+1, sigma_r, sigma_s)
    end = time.time()

    print("Ugradjena funkcija:")
    execution_time = (end - start);
    print("Vreme izvrsavanja: "+ str(round(execution_time,3))+ "s")

    execution_time_norm = execution_time/np.size(gr)
    print("Vreme izvrsavanja:"+ str(round(execution_time*1e6,3))+ " us/pix \n")

    fig, ax = subplots(2, 2, figsize=(32,22), dpi=40);
    tight_layout();
    ax[0,0].imshow(gr, cmap='gray'); ax[0,0].set_title('ulazna', fontsize=40);
    ax[0,1].imshow(res1,cmap='gray'); ax[0,1].set_title('python', fontsize=40);
    ax[1,0].imshow(res2, cmap='gray'); ax[1,0].set_title('c', fontsize=40);
    ax[1,1].imshow(res3, cmap='gray'); ax[1,1].set_title('ugradjena', fontsize=40);
```

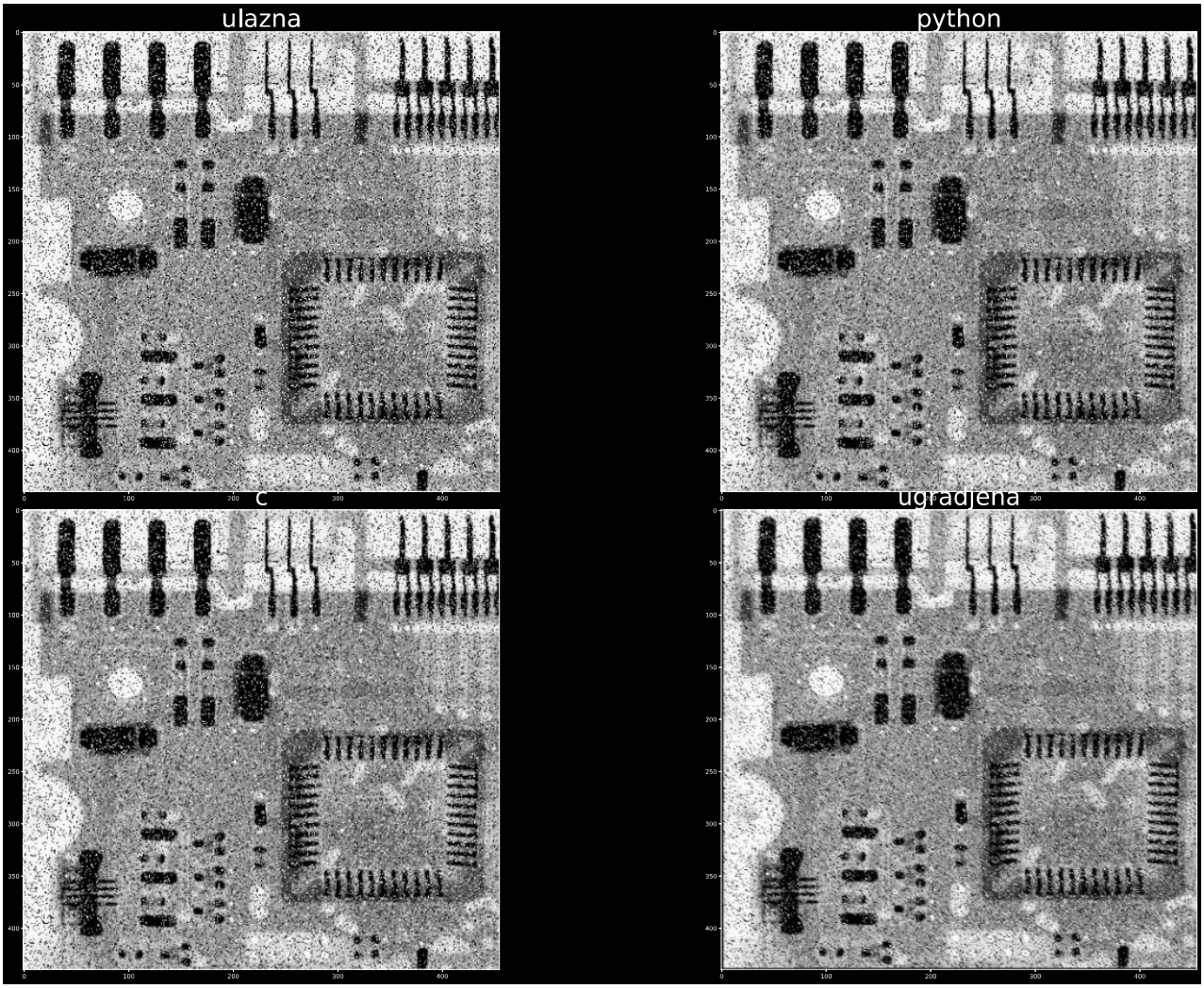
In [16]:

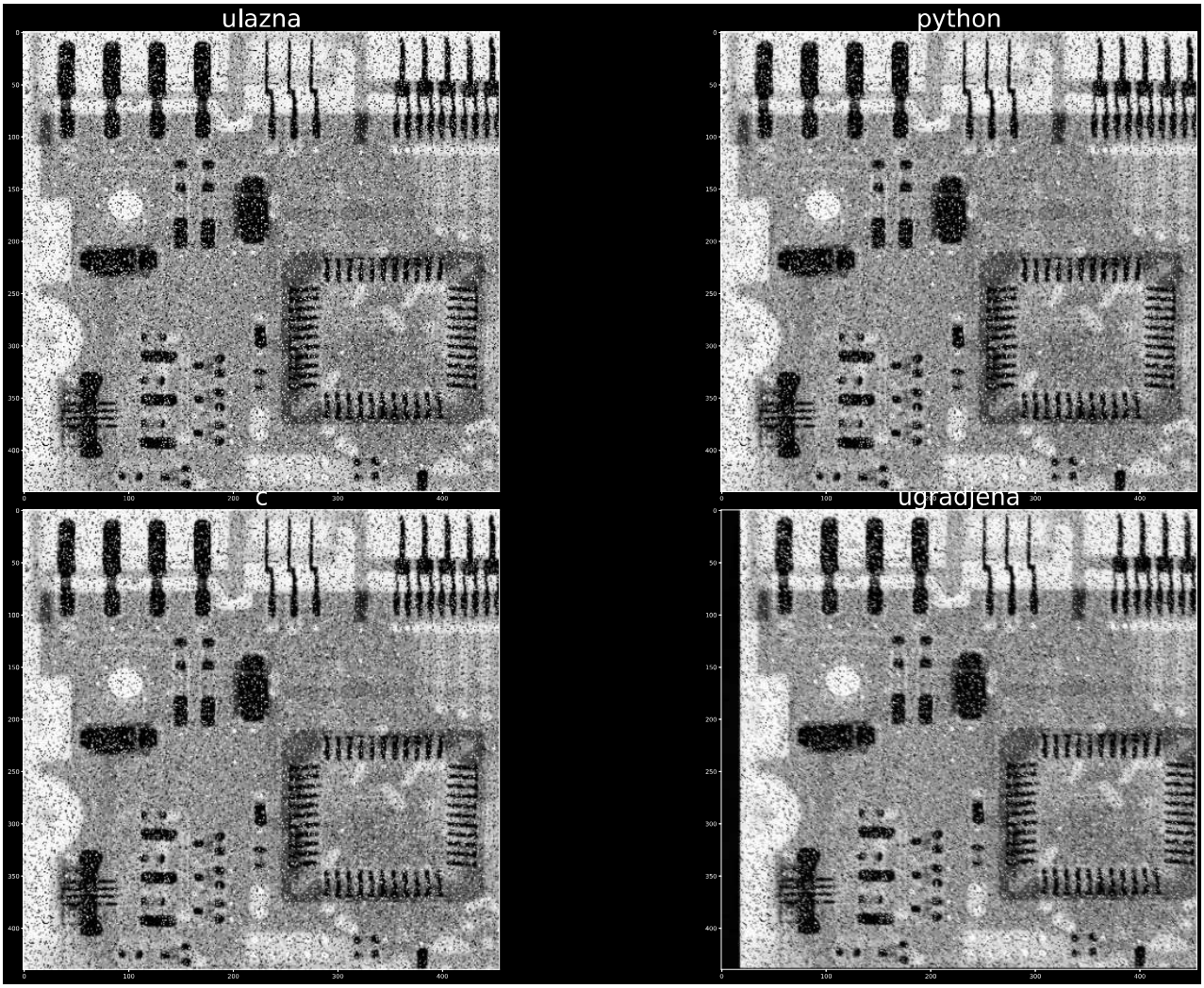
```
img = imread('../sekvence/ckt_board.tif')
test_bilateral(img,2,0.5,0.9)
test_bilateral(img,4,0.5,0.9)
test_bilateral(img,20,0.5,0.9)
test_bilateral(img,40,0.5,0.9)
```

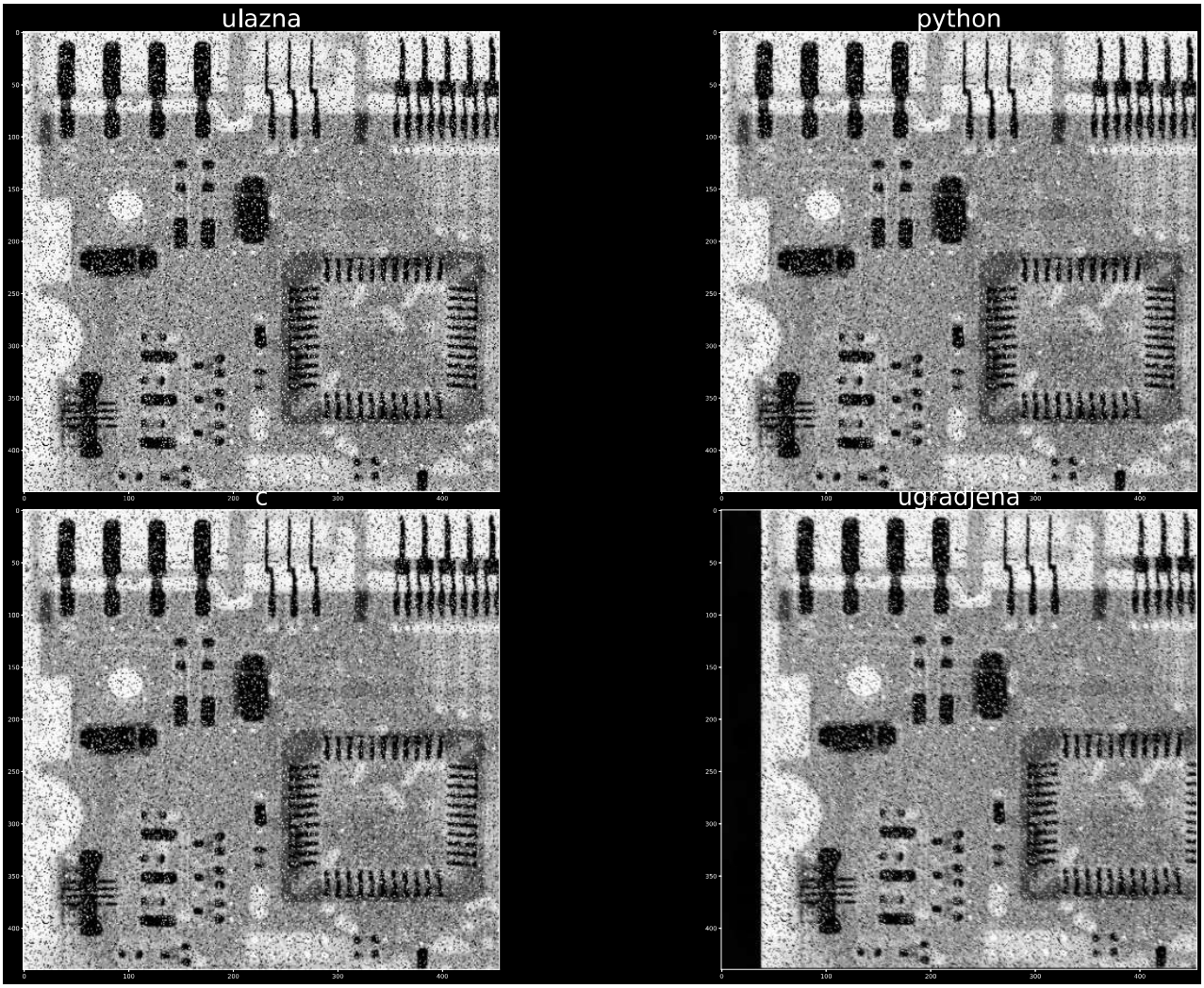
```
Radius: 2pix

Funkcija implemetirana u Pajtonu:
Vreme izvrsavanja: 9.938s
Vreme izvrsavanja:9937711.477 us/pix

Funkcija implementirana u c-u:
Vreme izvrsavanja: 0.722s
Vreme izvrsavanja:721828.222 us/pix

Ugradjena funkcija:
Vreme izvrsavanja: 0.17s
Vreme izvrsavanja:169922.59 us/pix

Radius: 4pix

Funkcija implemetirana u Pajtonu:
Vreme izvrsavanja: 10.347s
```

```
Vreme izvrsavanja:10346924.782 us/pix

Funkcija implementirana u c-u:
Vreme izvrsavanja: 1.889s
Vreme izvrsavanja:1888877.63 us/pix

Ugradjena funkcija:
Vreme izvrsavanja: 0.369s
Vreme izvrsavanja:369236.231 us/pix

Radius: 20pix

Funkcija implemetirana u Pajtonu:
Vreme izvrsavanja: 52.654s
Vreme izvrsavanja:52654133.558 us/pix

Funkcija implementirana u c-u:
Vreme izvrsavanja: 40.872s
Vreme izvrsavanja:40871960.878 us/pix

Ugradjena funkcija:
Vreme izvrsavanja: 8.257s
Vreme izvrsavanja:8256636.62 us/pix

Radius: 40pix

Funkcija implemetirana u Pajtonu:
Vreme izvrsavanja: 115.187s
Vreme izvrsavanja:115186847.925 us/pix

Funkcija implementirana u c-u:
Vreme izvrsavanja: 152.644s
Vreme izvrsavanja:152644173.384 us/pix

Ugradjena funkcija:
Vreme izvrsavanja: 26.956s
Vreme izvrsavanja:26956059.456 us/pix
```


ulazna
python
c
ugradjena


ulazna
python
c
ugradjena


ulazna
python
c
ugradjena


ulazna
python
c
ugradjena