LABORATORIJSKE VAJE PRI PREDMETU BIOMEDICINSKE SLIKOVNE TEHNOLOGIJE

Univerza *v Ljubljani*
Fakulteta *za elektrotehniko*

Založba FE

MIRAN BÜRMEN

Kataložni zapis o publikaciji (CIP) pripravili v Narodni in univerzitetni knjižnici v Ljubljani

COBISS.SI-ID 77872643

ISBN 978-961-243-421-2 (PDF)

URL: https://lipa.fe.uni-lj.si/courses/bst/prirocnik/prirocnik.pdf

Recenzenta: prof. dr. Boštjan Likar, prof. dr. Tomaž Vrtovec
Založnik: Založba FE, Ljubljana
Izdajatelj: Fakuleta za elektrotehniko, Ljubljana
Urednik: prof. dr. Sašo Tomažič
1. elektronska izdaja

Univerza v Ljubljani
Fakulteta za elektrotehniko

LABORATORIJSKE VAJE PRI PREDMETU BIOMEDICINSKE SLIKOVNE TEHNOLOGIJE

Miran Bürmen

Ljubljana, 2021

Kazalo

Poglavje 1

Uvod v programski jezik Python

Celoviti uvod v programski jezik Python, ki vključuje tudi navodila za namestitev izbranih grafičnih razvojnih okolij ter številne ilustrativne primere uporabe najpomembnejših knjižnic programkega jezika, najdete v [1], obsežnejšo obravnavo številnih drugih Python knjižnic pa v [2].

V programskem jeziku Python lahko sivinske slike enostavno predstavimo z dvorazsežnimi podatkovnimi polji knjižnice numpy, ki so primerki razreda `ndarray`. Knjižnico običajno uvozimo pod krajšim imenom `np`.

```
>>> import numpy as np
>>>
```

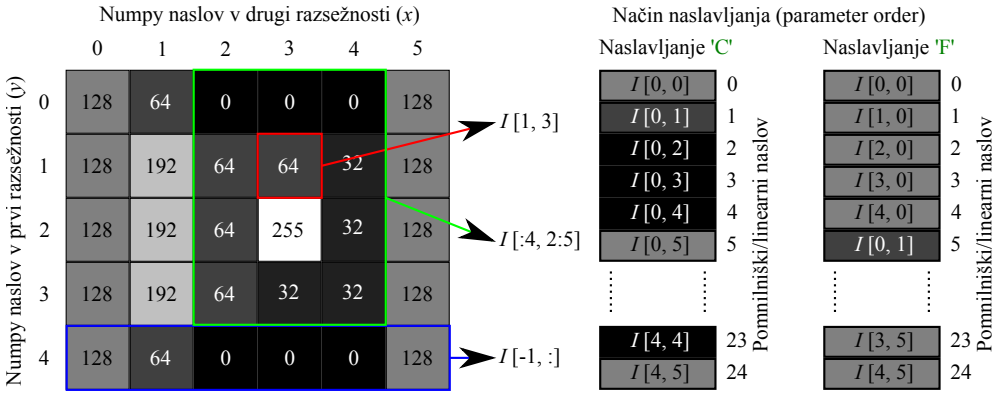
Slikovni elementi slik standardnih formatov [3] se najpogosteje hranijo kot nepredznačena 8-, 16- ali 32-bitna cela števila, včasih pa tudi v zapisu s plavajočo vejico. Pripadajoči podatkovni tipi knjižnice numpy so povzeti v tabeli 1.1. Ko izvajamo računske operacije nad posame-

Tabela 1.1: Zapis sivinskih vrednosti s podatkovnimi tipi knjižnice numpy.

Zapis sivinkih vrednosti	Podatkovni tip	Zaloga vrednosti
binarna slika	`np.bool`	{`False`, `True`}
8-bitni nepredznačeni	`np.uint8`	$[0, 255]$
16-bitni nepredznačeni	`np.uint16`	$[0, 65535]$
32-bitni predznačeni	`np.int32`	$[-2^{31}, 2^{31} - 1]$
32-bitni s plavajočo vejico	`np.float32` ali `np.single`	$[0.0, 1.0]$
64-bitni s plavajočo vejico	`np.float64` ali `np.float` ali `np.double`	$[0.0, 1.0]$

znimi slikovnimi elementi, običajno uporabimo zapis sivinskih vrednosti s plavajočo vejico, saj v nasprotnem primeru hitro pride do zaokrožitvenih napak ali pa presežemo zalogo vrednosti celoštevilskega podatkovnega tipa. Po končanem delu zapis sivinskih vrednosti pretvorimo v prvotni podatkovni tip in pri tem uporabimo ustrezno zaokrožitev.

```
>>> np.array([253, 254, 255], dtype=np.uint8) + 1
array([254, 255,   0], dtype=uint8)

```



Slika 1.1: Primer 8-bitne nepredznačene sivinske slike velikosti 6×4, predstavljene z dvorazsežnim podatkovnim poljem numpy velikosti $(4, 6)$, ter naslavljanje elementov podatkovnega polja.

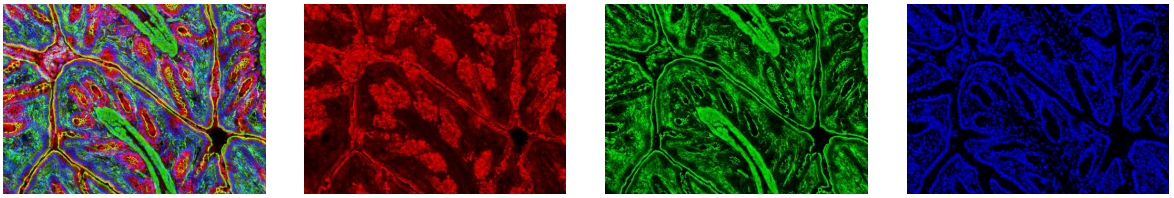
```
>>> np.array([253, 254, 255], dtype=np.float64) + 1
array([ 254.,  255.,  256.])
>>>
```

Podatki v večrazsežnih poljih so običajno shranjeni v enem izmed dveh najbolj razširjenih zapisov (glej sliko 1.1), in sicer `'C'` (po programskem jeziku C) ali `'F'` (po programskem jeziku Fortran). Bistvena razlika med obema načinoma naslavljanja izhaja iz vrstnega reda zapisa elementov večrazsežnega polja v linearnem računalniškem pomnilniku. Pri prvem načinu naslavljanja teče linearni naslov po prvi razsežnosti, pri drugem pa po zadnji razsežnosti podatkovnega polja. Knjižnica numpy sicer omogoča oba načina naslavljanja, a bomo v okviru tega priročnika uporabljali izključno privzeti način naslavljanja `'C'`. Dvorazsežne slike bomo predstavili z dvorazsežnimi numpy podatkovnimi polji, in sicer bo prva razsežnost podatkovnega polja predstavljala število vrstic (y koordinata), druga razsežnost podatkovnega polja pa število stolpcev slike (x koordinata). Na ta način lahko sliko širine W in višine H slikovnih elementov predstavimo z numpy podatkovnim poljem velikosti (H, W). Na podoben način lahko trirazsežne slike predstavimo s trirazsežnimi numpy podatkovnimi polji velikosti (D, H, W), kjer D predstavlja število rezin slike v smeri z koordinatne osi. Pri izbiri koordinatnega sistema, s katerim določimo prostorske koordinate slikovnih elementov, sledimo naslavljanju knjižnice numpy. Koordinatno izhodišče slike I se nahaja v levem zgornjem krajišču slike, torej na naslovu $I[0, 0]$, spodnje desno krajišče slike pa na naslovu $I[H-1, W-1]$. Vsebino podatkovnega polja numpy vzdolž izbrane koordinatne osi naslovimo kot `[...,start:stop:korak,...]`, kjer `start` in `stop` predstavljata začetni in končni naslov naslovljenih elementov, `step` pa korak vzorčenja vzdolž izbrane koordinatne osi. Pri tem naslovimo vse slikovne elemente od vključno prvega naslova (`start`) do vključno predzadnjega naslova (`stop-1`). Privzeta vrednost parametra `start` je enaka 0, privzeta vrednost parametra `stop` je enaka velikosti podatkovnega polja v naslovljeni razsežnosti, privzeta vrednost parametra `step` pa je enaka 1.

```
>>> polje = np.array([[ 1,  2,  3,  4,  5],
                      [11, 12, 13, 14, 15],
```

```
3                          [21, 22, 23, 24, 25],
4                          [31, 32, 33, 34, 35]])
5 >>> # korak 3 vzdolž prve (y) in korak 2 vzdolž druge (x) koordinatne osi
6 >>> polje[::3, ::2]
7 array([[ 1,  3,  5],
8        [21, 23, 25]])
9 >>>
```

Barvne RGB slike predstavimo s trirazsežnimi numpy podatkovnimi polji velikosti $(H, W, 3)$, kjer posamezni prerezi polja predstavljajo barvne komponente slike. Pri tem se držimo pravila, da prvi prerez polja $I[:, :, 0]$ predstavlja rdečo komponento, drugi prerez polja $I[:, :, 1]$ zeleno komponento, tretji prerez polja $I[:, :, 2]$ pa modro komponento slike (slika 1.2). Slike



(a) RGB barvna slika I (b) Rdeča komponenta (c) Zelena komponenta (d) Modra komponenta

Slika 1.2: 8-bitna RGB mikroskopska slika razdeljena na posamezne barvne komponente.

zapisane v PNG formatu lahko naložimo in shranimo s funkcijama `imread` in `imsave` modula `matplotlib.pyplot`. Pri praktičnem delu se pogosto srečujemo tudi z drugimi formati slik. Slike velike večine standardnih formatov (BMP, PNG, GIF, EPS, JPEG, itn.) lahko naložimo ali shranimo s pomočjo knjižnice pillow. Sliko naložimo s funkcijo `Image.open`, ki nam ustvari primerek razreda `Image`. Slednjega nato s funkcijo `np.array` pretvorimo v numpy podatkovno polje. Ko zaključimo z obdelavo slike in jo želimo shraniti, uporabimo funkcijo `Image.fromarray`, ki iz podatkovnega polja numpy ustvari primerek razreda `Image`. Pri tem mora tip podatkovnega polja numpy ustrezati enemu izmed podatkovnih tipov iz tabele 1.1, in sicer logičnemu podatkovnemu tipu `np.bool`, enemu izmed celoštevilskih tipov `np.uint8`, `np.uint16` ter `np.int32` ali zapisu s plavajočo vejico `np.float32` ter `np.float64`. Sivinske in RGB barvne slike funkcija `Image.fromarray` razpozna samodejno, in sicer na podlagi velikosti in tipa podatkovnega polja, zapise, kot so CMYK, RGBA ter YCbCr, pa je potrebno definirati z vrednostjo parametra `mode`:

- `'1'` - binarna slika (numpy tip `np.bool`, velikost (H, W)),
- `'L'` - sivinska slika (numpy tip `np.uint8`, velikost (H, W)),
- `'P'` - barvna slika z barvno mapo (numpy tip `np.uint8`, velikost (H, W)),
- `'RGB'` - barvna slika (numpy tip `np.uint8`, velikost $(H, W, 3)$),
- `'RGBA'` - barvna slika (numpy tip `np.uint8`, velikost $(H, W, 4)$),
- `'CMYK'` - barvna slika (numpy tip `np.uint8`, velikost $(H, W, 4)$),
- `'YCbCr'` - barvna slika (numpy tip `np.uint8`, velikost $(H, W, 3)$),

- `'I'` - sivinska slika (numpy tip `np.int32`, velikost (H, W)),
- `'F'` - sivinska slika (numpy tip `np.float32`, velikost (H, W)).

Zapis slikovnih elementov primerka razreda `Image` lahko ugotovimo z metodo `getbands`, pretvorbo zapisa pa sprožimo z metodo `convert`. Opisane lastnosti knjižnice pillow so povzete v sledečem primeru.

```
from PIL import Image as im
import numpy as np

# Nalaganje slike s knjižnico pillow.
# Naloži sliko 'slika.bmp' in ustvari primerek razreda Image.
pilSlika = im.open('slika.bmp')
# Shrani sliko v tif zapisu.
pilSlika.save('slika.tif')
# Pretvori primerek razreda Image v numpy podatkovno polje.
polje = np.array(pilSlika)

pilSlika.getbands() # Vrne oznako trenutnega zapis slike, recimo 'L'.
pilBinarnaSlika = pilSlika.convert('1') # Pretvori zapis slike v binarnega.

# Shranjevanje slike primerka razreda Image.
# Pretvori numpy podatkovno polje v primerek razreda Image.
pilSlika = im.fromarray(polje)
# Shrani sliko v gif zapisu.
pilSlika.save('slika.gif')
```

Priročen most med knjižnicama pillow in numpy najdemo v modulu `scipy.misc`, in sicer v funkcijah `toimage` in `imread`. S funkcijo `toimage` lahko po potrebi razpon sivinskih vrednosti omejimo na želeno območje, ki ga definiramo s parametroma `low` in `high`. Hkrati lahko z vrednostjo parametra `mode` eksplicitno določimo podatkovni tip zapisa sivinskih vrednosti.

```
import scipy.misc as spm

# Naloži sliko tipa tif neposredno v novo numpy podatkovno polje.
podatkovnoPolje = spm.imRead('nekaSlika.tif')

# Shrani sliko v dveh korakih.
# 1.  Pretvori sliko iz numpy polja v primerek razreda Image.
pilSlika = spm.toimage(podatkovnoPolje)
# 2.  Shrani sliko v jpg formatu.
pilSlika.save('nekaDrugaSlika.jpg')

# Vsili 32-bitni nepredznačeni zapis sivin z zalogo vrednosti [0,65535].
pillSlika = spm.toimage(podatkovnoPolje, low=0, high=65535, mode='I')
pillSlika.save('testnaSlika.tif')
```

Slike, zapisane s podatkovnim poljem numpy, lahko prikažemo s pomočjo modula `pyplot` knjižnice `matplotlib`, ki ga običajno uvozimo pod krajšim imenom `pp`.

```
1 from matplotlib import pyplot as pp
2 import numpy as np
3
4 npslika = np.asarray(im.open('slika.png'))
5 pp.figure()         # ustvari novo grafično okno
6 pp.imshow(npslika) # izriše sliko
7 pp.show()          # prikaže grafično okno
```

1.1 Naloge in vprašanja

1. S pomočjo funkcije `open` modula `PIL.Image` ali funkcije `imread` modula `scipy.misc` naložite 8-bitno nepredznačeno sivinsko sliko mrBrainSlice.png širine $W = 217$ in višine $H = 181$ slikovnih elementov. S funkcijo `figure` ustvarite novo grafično okno in v njem s funkcijo `imshow` prikažite sliko. Kot že rečeno, sta omenjeni funkciji del modula `matplotlib.pyplot`.

2. S pomočjo metode `tofile` shranite numpy podatkovno polje slike mrBrainSlice.png v surovo binarno datoteko mrBrainSlice_217x181_uint8.raw.

3. Sliko mrBrainSlice.png pretvorite v 16-bitni nepredznačeni zapis. Pri tem območje sivinskih vrednosti ustrezno prilagodite, tako da uporabite celotno zalogo vrednosti 16-bitnih nepredznačenih števil. Sliko shranite v surovem zapisu (mrBrainSlice_217x181_uint16-.raw), in sicer pri tem spet uporabite metodo `tofile`. Ne pozabite, da je potrebno tip podatkovnega polja pred klicem metode `tofile` pretvoriti v `np.uint16`, kar dosežete z metodo `astype`. Shranite 16-bitno sivinsko sliko še v mrBrainSlice.tif datoteko. Pri tem uporabite funkciji `fromarray` in `save` modula `PIL.Image`. Pri tem mora podatkovno polje numpy ustrezati zapisu `'I'` ali `'F'`, tj. podatkovni tip `np.int32` ali `np.float32`.

4. S pomočjo funkcije `fromfile` modula `numpy` naložite shranjeni surovi 8- in 16-bitni nepredznačeni sliki mrBrainSlice_217x181_uint8.raw ter mrBrainSlice_217x181_uint16.raw in ju prikažite.

5. Ustvarite funkciji, ki bosta primerni za nalaganje in shranjevanje surovih dvorazsežnih binarnih slik iz ali v datoteko `'fid'`. Podatkovni tip slike naj bo določen s parametrom `dtype`, širina slike s parametrom (`width`) in višina slike s parametrom (`height`). Pri tem uporabite imena podatkovnih tipov knjižnice numpy (`np.uint8, np.uint16, np.float32, np.float64`), vrstni red zapisa v pomnilniku (`order`) pa naj bo `'xy'` ali `'yx'`. Funkcija `imLoadRaw2d` za nalaganje binarne slike naj vedno vrne podatkovno polje velikosti `(height, width)`, ki sledi privzetemu načinu naslavljanja `[y, x]`.

   (a) `def imLoadRaw2d(fid, width, height, dtype=np.uint8, order='xy')`

   (b) `def imSaveRaw2d(fid, data)`

6. Prikažite del slike mrBrainSlice.png, ki ga omejuje pravokotno področje s krajiščema v slikovnih elementih `[60, 80]` ter `[120, 160]`.

7. Zamenjajte vrednosti vseh slikovnih elementov slike mrBrainSlice.png, ki imajo vrednost med 160 in 200, z vrednostjo 255. Sivinske vrednosti ostalih slikovnih elementov postavite na vrednost 0. Dobljeno sliko prikažite. Ustvarite in prikažite še sliko, kjer omenjene slikovne elemente predstavite z rdečo barvo (vrednost rdeče komponente postavite na 255, vrednosti zelene in modre komponente na 0), vse ostale slikovne elemente pa s črno barvo (vrednosti vseh treh barvnih komponent postavite na 0). Posamezna podatkovna polja treh barvnih komponent boste najenostavneje združili v trirazsežno podatkovno polje s funkcijo `dstack` modula `numpy`.

8. Dopolnite funkciji `imLoadRaw2d` in `imSaveRaw2d` tako, da bo mogoče z njima naložiti ali shraniti tudi trirazsežne slike, zapisane v surovih raw datotekah. Pri tem naj bo velikost slike v smeri z koordinatne osi določena s parametrom `depth`. Vrstni red zapisa v pomnilniku naj bo eden izmed `('xyz', 'xzy', 'yxz', 'yzx', 'zxy', 'zyx')`. Funkcija `imLoadRaw3d` naj vedno vrne podatkovno polje velikosti `(depth, height, width)`, ki sledi privzetemu načinu naslavljanja `[z, y, x]`.

   (a) `def imLoadRaw3d(fid, width, height, depth, dtype=np.uint8, order='xyz')`

   (b) `def imSaveRaw3d(fid, data)`

9. Ustvarite funkcijo `imGrid2d`, ki vrne koordinatno mrežo točk slikovnih elementov poljubne 2D slike. Pri tem naj bodo x koordinate slikovnih elementov shranjene v podatkovnem polju `oX`, pripadajoče y koordinate slikovnih elementov pa v podatkovnem polju `oY`. Velikost slike, izraženo s številom slikovnih elementov, v smeri x koordinatne osi naj določa parameter `width`, v smeri y koordinatne osi pa parameter `height`. Pripadajoči velikosti slikovnega elementa naj določata parametra `dx` in `dy`. Parametra `xoffset` in `yoffset` pa naj določata koordinati (x, y) slikovnega elementa v levem zgornjem krajišču slike, ki se nahaja na naslovu `[0, 0]`. Če je vrednost parametra `xoffset` ali `yoffset` enaka `'center'`, naj geometrično središče slike sovpada s koordinatnim izhodiščem pripadajoče koordinatne osi. Koordinatno mrežo točk boste najenostavneje ustvarili s funkcijama `meshgrid` ter `linspace` ali `arange` modula `numpy`.

```
1 def imGrid2d(width, height, dx=1.0, dy=1.0,
2              xoffset='center', yoffset='center'):
3   ...
4   return oY, oX
```

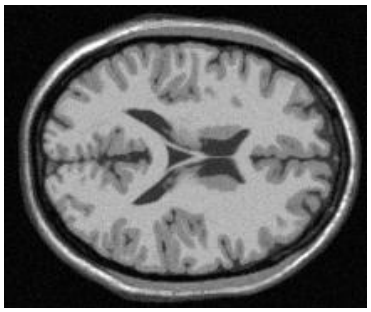
10. Po vzoru funkcije `imGrid2d` ustvarite še funkcijo `imGrid3d`, ki vrne koordinatno mrežo točk slikovnih elementov poljubne 3D slike. Pri tem naj bodo x koordinate shranjene v podatkovnem polju `oX`, y koordinate v podatkovnem polju `oY` ter z koordinate v podatkovnem polju `oZ`. Velikost slike, izraženo s številom slikovnih elementov, v smeri x koordinatne osi določa parameter `width`, v smeri y koordinatne osi parameter `height` ter v smeri z koordinatne osi parameter `depth`. Pripadajočo velikost slikovnega elementa določajo parametri `dx`, `dy` in `dz`. Parameteri `xoffset`, `yoffset` in `zoffset` naj določajo koordinate (x, y, z) slikovnega elementa v levem zgornjem krajišču slike, ki se nahaja na naslovu `[0, 0, 0]`. Če je vrednost katerega izmed parametrov `xoffset`, `yoffset` ali `zoffset` enaka `'center'`, naj

geometrično središče slike sovpada s koordinatnim izhodiščem pripadajoče koordinatne osi.

```
def imGrid3d(width, height, depth,
             dx=1.0, dy=1.0, dz=1.0,
             xoffset='center', yoffset='center',
             zoffset='center'):
  ...
  return oZ, oY, oX
```

1.2 Rešitve in odgovori na vprašanja

Pri odgovorih privzamemo, da so funkcije zbrane v modulu `funkcije`, slikovno gradivo pa se nahaja v podmapi `poglavje_1`.



Slika 1.3: Rezina magnetnoresonančne slike možganov mrBrainSlice.png.

Najprej uvozimo potrebne module in funkcije.

```
import numpy as np
from matplotlib import pyplot as pp
from PIL import Image as im
import funkcije
```

1. Naložimo in prikažemo PNG sliko (slika 1.3).

```
pilSlika = im.open('./poglavje_1/mrBrainSlice.png')
npSlika = np.array(pilSlika)

pp.figure()
pp.imshow(npSlika, cmap='gray')
pp.show()
```

2. Shranimo naloženo sliko v surovo binarno datoteko.

```
npSlika.tofile(
  './poglavje_1/rezultati/mrBrainSlice_217x181_uint8.raw')
```

3. Pretvorimo in shranimo naloženo sliko v zahtevane formate.

```
# prtetvorimo v 16-bitni nepredznačeni zapis
npSlika16 = np.round(
  npSlika.astype(np.float)*(65535.0/255.0)).astype(np.uint16)
# shranimo v surovo binarno datoteko
npSlika16.tofile(
  './poglavje_1/rezultati/mrBrainSlice_217x181_uint16.raw')
# shranimo v 16-bitno tif datoteko
pilSlika32 = im.fromarray(npSlika16.astype(np.int32))
pilSlika32.save('./poglavje_1/rezultati/mrBrainSlice.tif')
```

4. Naložimo in prikažemo zahtevani sliki.

```
npRaw8 = np.fromfile(
  './poglavje_1/rezultati/mrBrainSlice_217x181_uint8.raw',
  dtype=np.uint8, count=217*181)
npRaw8.shape = (181, 217)

npRaw16 = np.fromfile(
  './poglavje_1/rezultati/mrBrainSlice_217x181_uint16.raw',
  dtype=np.uint16, count=217*181)
npRaw16.shape = (181, 217)

pp.figure()

pp.subplot(1, 2, 1)
pp.imshow(npRaw8, cmap='gray')
pp.title('Slika mrBrainSlice_217x181_uint8.raw')

pp.subplot(1, 2, 2)
pp.imshow(npRaw16, cmap='gray')
pp.title('Slika mrBrainSlice_217x181_uint16.raw')

pp.show()
```

5. (a) V modulu `funkcije` ustvarimo funkcijo `imLoadRaw2d`

```
def imLoadRaw2d(fid, width, height,
                dtype=np.uint8, order='xy'):
  order = str(order).lower()
  data = np.fromfile(filename, dtype=dtype)
  if order == 'xy':
    data.shape = (height, width)
  elif order == 'yx':
    data.shape = (width, height)
    data = data.transpose()
  else:
    raise ValueError(
      'Vrednost vhodnega parametra "order" je lahko '
      '"xy" ali "yx"!')

  return data
```

(b) V modulu `funkcije` ustvarimo funkcijo `imSaveRaw2d`

```
def imSaveRaw2d(fid, data):
  data.tofile(fid)
```

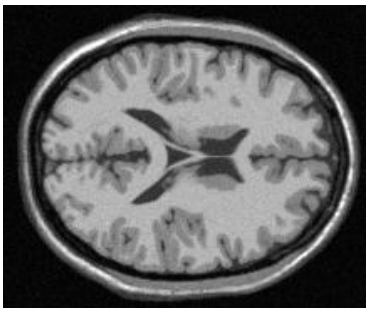
Kratek test funkcij `imLoadRaw2d` ter `imSaveRaw2d`.

```
mr8 = funkcije.imLoadRaw2d(
  './poglavje_1/rezultati/mrBrainSlice_217x181_uint8.raw',
```

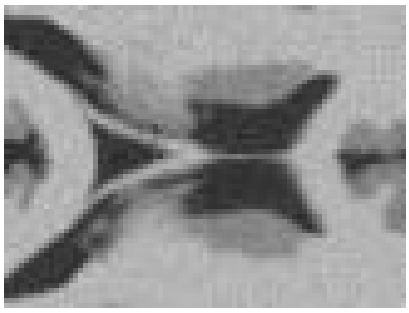
```
  217, 181)
mr16 = funkcije.imLoadRaw2d(
  './poglavje_1/rezultati/mrBrainSlice_217x181_uint16.raw',
  217, 181, np.uint16)

pp.figure()

pp.subplot(1, 2, 1)
pp.imshow(mr8, cmap='gray')
pp.title('Slika mrBrainSlice_217x181_uint8.raw')

pp.subplot(1, 2, 2)
pp.imshow(mr16, cmap='gray')
pp.title('Slika mrBrainSlice_217x181_uint16.raw')

pp.show()
```

6. Prikažemo zahtevani del slike (slika 1.4b).

```
pp.figure()
pp.imshow(mr8[60:120+1, 80:160+1], cmap='gray')
pp.title('Podokno [60:121, 80:161]')
pp.show()
```



(a) Izvirna slika



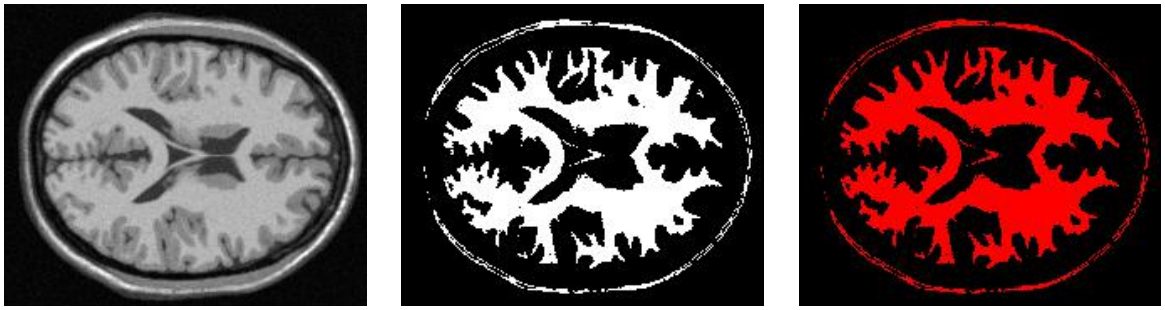
(b) Osrednji del slike

Slika 1.4: Rezina magnetnoresonančne slike možganov s povečanim osrednjim delom.

7. Ustvarimo zahtevano binarno sivinsko in RGB barvno sliko ter ju prikažemo (slika 1.5).

```
maska = np.logical_and(mr8 >= 160, mr8 <= 200)
mrRgb = np.zeros([181, 217, 3], dtype=np.uint8)
mrRgb[maska, 0] = 255

pp.figure()

pp.subplot(1, 3, 1)
pp.imshow(mr8, cmap='gray')
pp.title('Izvirna')
```

```
10
11 pp.subplot(1,3,2)
12 pp.imshow(maska, cmap='gray')
13 pp.title('Binarna sivinska')
14
15 pp.subplot(1, 3, 3)
16 pp.imshow(mrRgb)
17 pp.title('Binarna barvna')
```



(a) Izvirna slika (b) Binarna sivinska slika (c) Binarna barvna slika

Slika 1.5: Rezina magnetnoresonančne slike možganov, pripadajoča binarna sivinska slika slikovnih elementov s sivinsko vrednostjo na intervalu [160, 200] ter z rdečo barvo na črni podlagi prikazani slikovni elementi binarne slike.

8. (a) V modulu `funkcije` ustvarimo funkcijo `imLoadRaw3d`.

```
1  def imLoadRaw3d(
2      fid, width, height, depth, dtype=np.uint8, order='xyz'):
3    order = str(order).lower()
4    data = np.fromfile(filename, dtype=dtype)
5    if order == 'xyz':
6      data.shape = (depth, height, width)
7    elif order == 'xzy':
8      data.shape = (height, depth, width)
9      data = data.transpose((1,0,2))
10   elif order == 'yxz':
11     data.shape = (depth, width, height)
12     data = data.transpose((0,2,1))
13   elif order == 'yzx':
14     data.shape = (width, depth, height)
15     data = data.transpose((1,2,0))
16   elif order == 'zxy':
17     data.shape = (height, width, depth)
18     data = data.transpose((2,0,1))
19   elif order == 'zyx':
20     data.shape = (width, height, depth)
21     data = data.transpose((2,1,0))
22   else:
```

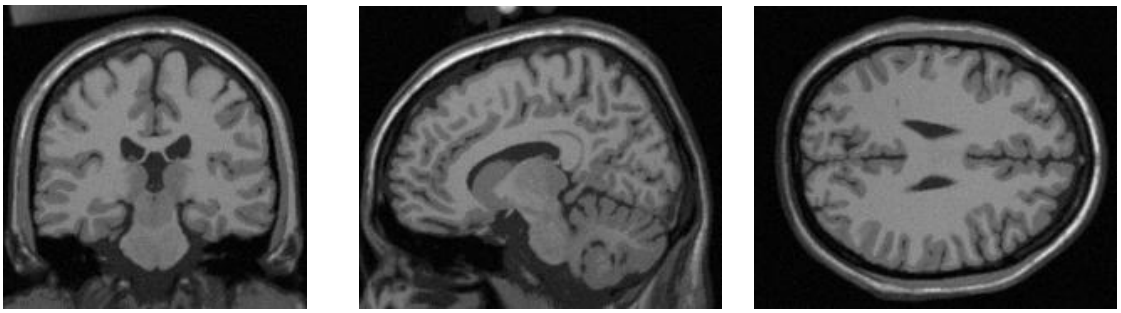
```
23      raise ValueError(
24        'Vrednost vhodnega parametra "order" je lahko '
25        '"xyz", "xzy", "yxz", "yzx", "zxy" ali "zyx"!')
26    return data
```

(b) V modulu `funkcije` ustvarimo funkcijo `imSaveRaw3d`.

```
1 def imSaveRaw3d(fid, data):
2   data.tofile(fid)
```

Kratek test funkcije `imLoadRaw3d` in izris nekaj primerov osnovnih prerezov magnetnoresonančne slike možganov (slika 1.6).

```
1  mr3d = funkcije.imLoadRaw3d(
2    './poglavje_1/mr_217x181x181_uint8.raw', 217, 181, 181)
3
4  pp.figure()
5
6  pp.subplot(1, 3, 1)
7  pp.imshow(mr3d[:, :, 100], cmap='gray')
8  pp.title('Prerez slike pri [:, :, 100]')
9
10 pp.subplot(1, 3, 2)
11 pp.imshow(mr3d[:, 100, :], cmap='gray')
12 pp.title('Prerez slike pri [:, 100, :]')
13
14 pp.subplot(1, 3, 3)
15 pp.imshow(mr3d[100, :, :], cmap='gray')
16 pp.title('Prerez [100, :, :]')
17
18 pp.show()
```



(a) Prerez [:, :, 100] (b) Prerez [:, 100, :] (c) Prerez [100, :, :]

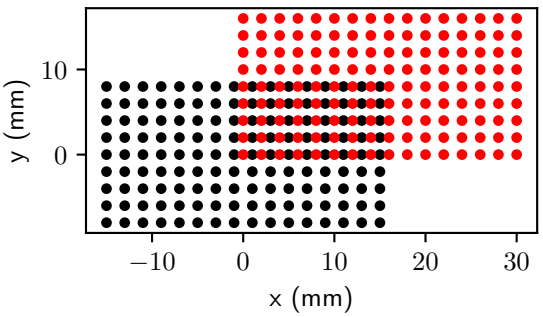
Slika 1.6: Primeri osnovnih prerezov 3D magnetnoresonančne slike možganov mr_217x181x181-_uint8.raw.

9. V modulu `funkcije` ustvarimo funkcijo `imGrid2d`.

```
def imGrid2d(width, height, dx=1, dy=1,
             xoffset=0, yoffset=0):
  if yoffset == 'center':
    yoffset = -0.5*(height - 1)*dy
  if xoffset == 'center':
    xoffset = -0.5*(width - 1)*dx

  x = np.arange(xoffset, xoffset + width*dx, dx, dtype=np.float)
  y = np.arange(yoffset, yoffset + height*dy, dy, dtype=np.float)

  return np.meshgrid(y, x, indexing='ij')
```

Preverimo delovanje funkcije `imGrid2d` na sliki velikosti 16×9 slikovnih elementov z velikostjo slikovnega elementa 2×2 mm (slika 1.7).

```
# koordinatno izhodišče v geometričnem središču slike
Y2d_c, X2d_c = funkcije.imGrid2d(
  16, 9, 2.0, 2.0, 'center', 'center')
# koordinatno izhodišče v krajišču slike
Y2d, X2d = funkcije.imGrid2d(16, 9, 2.0, 2.0)

pp.figure()
pp.plot(X2d_c, Y2d_c, '.k')
pp.plot(X2d, Y2d, '.r')
pp.title('16 x 9 sl. el. vel. 2 x 2 mm')
pp.xlabel('x (mm)')
pp.ylabel('y (mm)')
pp.show()
```



Slika 1.7: Koordinatni sistem slike velikosti 16×9 slikovnih elementov z velikostjo slikovnega elementa 2×2 mm. (črno) Koordinatno izhodišče v geometričnem središču slike. (rdeče) Koordinatno izhodišče v središču slikovnega elementa na naslovu $[0, 0]$ (krajišče slike).

10. V modulu `funkcije` ustvarimo funkcijo `imGrid3d`.

```
def imGrid3d(width, height, depth, dx=1, dy=1, dz=1,
```

```
2                xoffset=0, yoffset=0, zoffset=0):
3    if zoffset == 'center':
4      zoffset = -0.5*(depth - 1)*dz
5    if yoffset == 'center':
6      yoffset = -0.5*(height - 1)*dy
7    if xoffset == 'center':
8      xoffset = -0.5*(width - 1)*dx
9
10   x = np.arange(xoffset, xoffset + width*dx, dx, dtype=np.float)
11   y = np.arange(yoffset, yoffset + height*dy, dy, dtype=np.float)
12   z = np.arange(zoffset, zoffset + depth*dz, dz, dtype=np.float)
13
14   return np.meshgrid(z, y, x, indexing='ij')
```

Poglavje 2

Interpolacija in decimacija slik

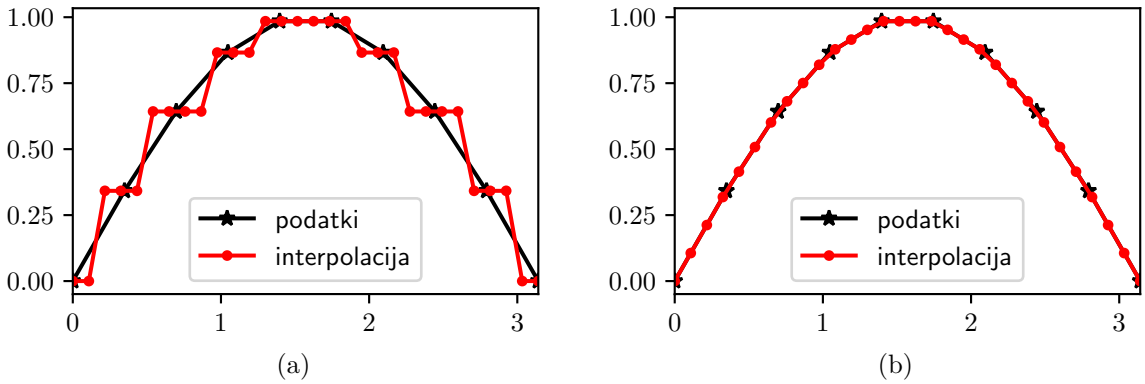
Poglavje je namenjena spoznavanju in razumevanju osnovnih postopkov interpolacije in decimacije slik, s katerimi lahko povečamo ali zmanjšamo vzorčno vrekvenco in s tem velikost slik.

2.1 Interpolacija slik

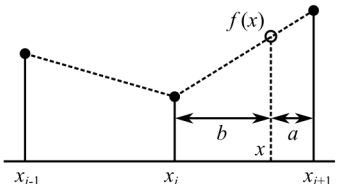
S postopkom interpolacije slik lahko priredimo sivinsko vrednost poljubni točki v slikovni ravnini. Na ta način lahko povečamo vzorčno frekvenco ter s tem velikost slik in tako zmanjšamo velikost slikovnih elementov. Glede na to, koliko sosednjih slikovnih elementov upoštevamo pri izračunu sivinske vrednosti v dani točki, delimo postopke interpolacije na:

- ničti red ali interpolacija najbližjega soseda - upoštevamo le najbližji slikovni element,
- prvi red ali (bi)linearna interpolacija - upoštevamo le štiri sosednje slikovne elemente,
- višji red, npr. (bi)kubična interpolacija (drugi red), ki upošteva 16 sosednjih slikovnih elementov.

Primerjavo postopkov interpolacije z najbližjim sosedom in linearne interpolacije za sinusni polval prikazuje slika 2.1. Matematični zapis postopka linearne interpolacije enorazsežnega signala prikazuje slika 2.2. S posplošitvijo interpolacije na dve razsežnosti je postopek mogoče uporabiti za sivinske slike ter posamezne komponente barvnih slik. Bilinearno interpolacijo v točki (x, y) izračunamo kot uteženo vsoto funkcijskih vrednosti, ki obdajajo interpolacijsko točko (slika 2.3). Računska zahtevnost interpolacijskih postopkov 2D slik v grobem narašča s kvadratom reda interpolacije, kar pomeni, da je bikubična interpolacija (drugi red) približno štiri krat bolj zahtevna (počasnejša) od bilinearne (prvi red) interpolacije. Interpolacijske postopke je mogoče posplošiti, tako da omogočajo interpolacijo večrazsežnih slik. Kot primer si oglejmo trilinearno interpolacijo, ki jo je mogoče razčleniti na dve bilinearni in eno linearno interpolacijo, kot to prikazuje slika 2.4. Računska zahtevnost interpolacije 3D slik raste s tretjo potenco reda interpolacijskega postopka.



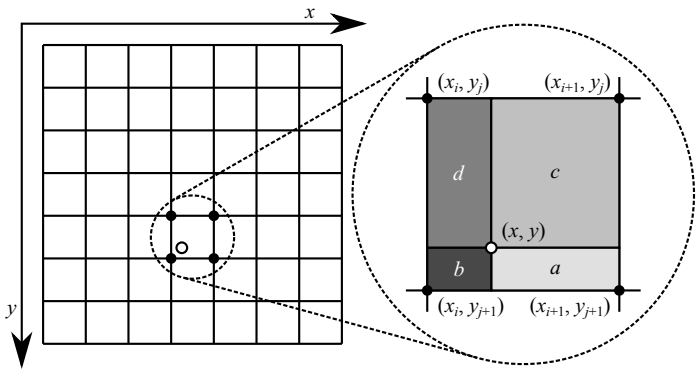
Slika 2.1: Primerjava interpolacije (a) z najbližjim sosedom in (b) linearne interpolacije.



$$dx = x_{i+1} - x_i$$
$$a = x_{i+1} - x$$
$$b = x - x_i$$

$$f(x) = f(x_i)\frac{a}{dx} + f(x_{i+1})\frac{b}{dx} \tag{2.1}$$

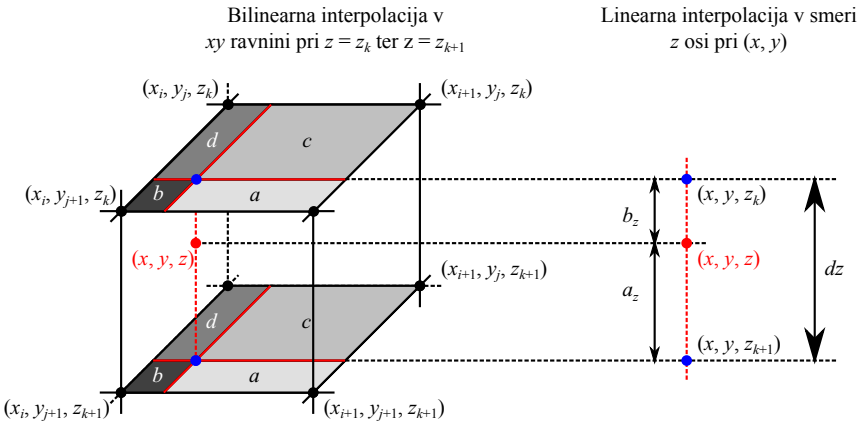
Slika 2.2: Linearna interpolacija



$$dx = x_{i+1} - x_i$$
$$dy = y_{j+1} - y_j$$
$$s = dx \cdot dy$$
$$a = (x_{i+1} - x)(y_{j+1} - y)$$
$$b = (x - x_i)(y_{j+1} - y)$$
$$c = (x_{i+1} - x)(y - y_j)$$
$$d = (x - x_i)(y - y_j)$$

$$f(x,y) = f(x_i, y_i)\frac{a}{s} + f(x_{i+1}, y_i)\frac{b}{s} + f(x_i, y_{i+i})\frac{c}{s} + f(x_{i+1}, y_{i+1})\frac{d}{s} \tag{2.2}$$

Slika 2.3: Bilinearna interpolacija.



Bilinearna interpolacija v xy ravnini pri $z = z_k$ ter $z = z_{k+1}$:

$$f(x, y, z_k) = f(x_i, y_i, z_k)\frac{a}{s} + f(x_{i+1}, y_i, z_k)\frac{b}{s} + f(x_i, y_{i+i}, z_k)\frac{c}{s} + f(x_{i+1}, y_{i+1}, z_k)\frac{d}{s}$$
$$f(x, y, z_{k+1}) = f(x_i, y_i, z_{k+1})\frac{a}{s} + f(x_{i+1}, y_i, z_{k+1})\frac{b}{s} + f(x_i, y_{i+i}, z_{k+1})\frac{c}{s} + f(x_{i+1}, y_{i+1}, z_{k+1})\frac{d}{s}$$

Linearna interpolacija v smeri z osi pri (x, y):

$$f(x, y, z) = f(x, y, z_k)\frac{a_z}{dz} + f(x, y, z_{k+1})\frac{b_z}{dz}. \tag{2.3}$$

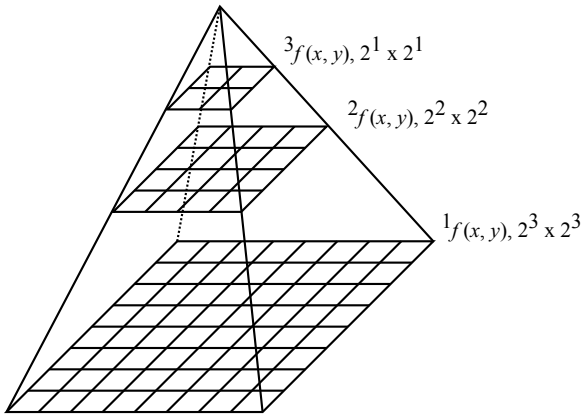
Slika 2.4: Trilinearna interpolacija kot zaporedje dveh bilinearnih in ene linearne interpolacije.

2.2 Decimacija slik

S postopkom decimacije slik zmanjšamo vzorčno frekvenco ter s tem velikost slik. Skladno z Nyquistovim vzorčnim teoremom je pred postopkom decimacije sliko potrebno filtrirati z nizko prepustnim sitom in na ta način odstraniti visoko frekvenčno informacijo. Pri decimaciji se pogosto uporablja piramidna shema, kjer se vzorčna frekvenca izvirne slike zaporedoma zmanjšuje s celoštevilskim faktorjem, običajno dva. Filtriranje slike lahko izvedemo s postopkom 2D diskretne konvolucije med podano sliko S velikosti (H, W) in konvolucijskim jedrom K velikosti (A, B). Skladno z naslavljanjem elementov podatkovnega polja numpy lahko 2D diskretno konvolucijo zapišemo kot:

$$S_c[i, j] = \sum_{k=0}^{A-1}\sum_{l=0}^{B-1} S[i - (k - c_1), j - (l - c_2)] \cdot K[k, l]. \tag{2.4}$$

Na področju, kjer slika ni definirana, bomo predpostavili sivinsko vrednost 0. Konstanti c_1 in c_2, ki določata središče konvolucijskega jedra, naj bosta definirani kot $c_1 = \lfloor \frac{A}{2} \rfloor$ ter $c_2 = \lfloor \frac{B}{2} \rfloor$. V programskem jeziku Python je konvolucijo mogoče neposredno izvesti s štirimi `for` zankami. Dve zunanji zanki uporabimo za sprehajanje po slikovnih elementih, dve notranji zanki pa za



(a)

1/16	1/8	1/16
1/8	1/4	1/8
1/16	1/8	1/16

1/400	1/80	1/50	1/80	1/400
1/80	1/16	1/10	1/16	1/80
1/50	1/10	4/25	1/10	1/50
1/80	1/16	1/10	1/16	1/80
1/400	1/80	1/50	1/80	1/400

(b)

Slika 2.5: (a) Piramidna decimacijska shema. (b) Primer dveh jeder nizkoprepustnega decimacijskega sita.

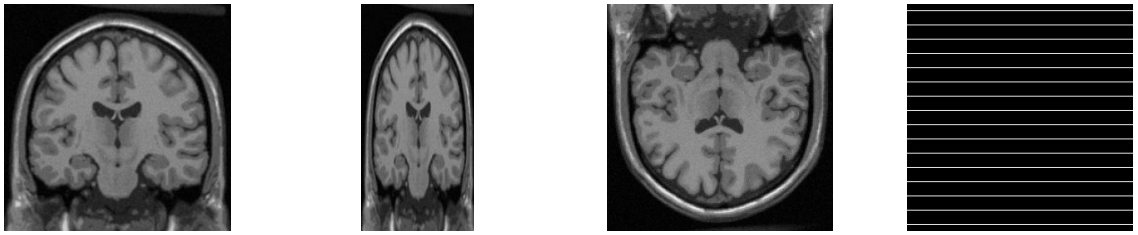
sprehajanje po konvolucijskem jedru:

```
c = np.floor(np.array(K.shape)/2.0) # Naslov središča konvolucijskega jedra.
H, W = S.shape  # Velikost vhodne slike.
A, B = K.shape  # Velikost konvolucijskega jedra.
Sc = np.zeros([H, W])     # Podatkovno polje izhodne (zglajene) slike.
for i in range(H):        # Vrstica slikovnega elementa.
  for j in range(W):      # Stolpec slikovnega elementa.
    for k in range(A):    # Vrstica konvolucijskega jedra.
      for l in range(B): # Stolpec konvolucijskega jedra.
        # Premik naslovov.
        ic = i - (k - c[0])
        jc = j - (l - c[1])
        # Pred izračunom preverimo veljavnost naslova slikovnega elementa.
        if ic > 0 and ic < H and jc > 0 and jc < W:
          Sc[i, j] += K[k, l]*S[ic, jc]
```

2.3 Naloge in vprašanja

1. Ustvarite funkcijo `interp1d`, ki bo interpolirala vrednosti v točkah `x`. Predpostavite, da so podatki `fp` podani za monotono naraščajoče zaporedje ekvidistantnih točk `xp`. Parameter `method` določa tip interpolacije, in sicer naj `'nearest'` označuje ničti red interpolacije z najbližjim sosedom, `'linear'` pa linearno interpolacijo. Parameter `x` lahko v splošnem vsebuje poljubno število elementov. Izhodno podatkovno polje `f` naj bo tipa `np.float` in enake velikosti kot podatkovno polje `x`.

```
def interp1d(x, xp, fp, method='linear'):
  ...
```



Slika 2.6: Od leve proti desni: slika mr_256x256_uint8.raw, slika mr_128x256_uint8.raw, slika mr_217x181x181_uint8.raw v yz ravnini pri $x = 110$ ter slika decimacija_256x256_uint8.raw.

```
3 |   return f
```

Kaj so prednosti in slabosti interpolacije z najbližjim sosedom?

2. Ustvarite funkcijo `interp2d`, ki bo interpolirala sivinske vrednosti v točkah (x, y). Predpostavite, da definicijsko območje vhodne slike `fp` napenjata vektorja monotono naraščajočih ekvidistantno razporejenih točk `xp` in `yp`. Parameter `method` naj ima enak pomen kot pri funkciji `interp1d`. Vhodna parametra `x` in `y` lahko v splošnem vsebujeta poljubno, a enako število elementov, podatkovno polje izhodne slike `f` naj bo tipa `np.float` in enake velikosti kot podatkovni polji `x` ter `y`.

```
1 | def interp2d(x, y, xp, yp, fp, method='linear'):
2 |   ...
3 |   return f
```

   (a) Interpolirajte sliko mr_256x256_uint8.raw tako, da podvojite število slikovnih elementov v obeh razsežnostih (velikost interpolirane slike bo tako 512×512 slikovnih elementov). Mrežo interpolacijskih točk boste najenostavneje ustvarili s funkcijama `meshgrid` ter `linspace` modula `numpy`.

   (b) Slika mr_128x256_uint8.raw je nastala z neenako frekvenco vzorčenjem v smeri x in y osi. Slikovni element v smeri x osi je dvakrat večji kot v smeri y osi. Sliko interpolirajte tako, da bo velikost slikovnega elementa v smeri obeh koordinatnih osi enaka. Pri tem prilagodite vzorčno frekvenco zgolj v smeri ene koordinatne osi slike! Uporabite funkcijo `interp2d` z bilinearno interpolacijo.

   (c) Interpolirajte osrednji del slike mr_128x256_uint8.raw velikosti $x \times y = 32 \times 64$ slikovnih elementov. Pri tem naj bo velikost slikovnega elementa interpolirane slike $x \times y = 0{,}25 \times 0{,}25$ mm. Predpostavite, da je velikost slikovnega elementa izhodiščne slike $x \times y = 2 \times 1$ mm.

3. Ustvarite funkcijo `interp3d`, ki bo interpolirala sivinske vrednosti v točkah (x, y, z). Predpostavite, da definicijsko območje vhodne slike `fp` napenjajo vektorji monotono naraščajočih ekvidistantno razporejenih točk `xp`, `yp` in `zp`. Parameter `method` naj ima enak pomen kot pri funkciji `interp1d`. Vhodni parametri `x`, `y` in `z` lahko v splošnem vsebujejo poljubno, a enako število elementov, podatkovno polje izhodne slike `f` naj bo tipa `np.float` in enake velikosti kot podatkovna polja `x`, `y` ter `z`.

```
def interp3d(x, y, z, xp, yp, zp, fp, method='linear'):
  ...
  return f
```

Interpolirajte sliko mr_217x181x181_uint8.raw v yz ravnini pri $x = 110{,}3$. Število slikovnih elementov v smeri y in z osi naj se podvoji.

4. Ustvarite funkcijo `conv2d`, ki izračuna konvolucijo vhodne slike `data` s konvolucijskim jedrom `kernel` ter vrne podatkovno polje izhodne slike `odata` tipa `np.float`.

```
def conv2d(data, kernel):
  ...
  return odata
```

5. Ustvarite še funkcijo `imDecimate2d`, ki bo zaporedoma (`level` krat) decimirala vhodno sliko s faktorjem dva (glej piramidno shemo na sliki 2.5a). Ko je vrednost parametra `kernel` enaka `None`, uporabite konvolucijsko jedro velikosti 3×3 iz navodil.

```
def imDecimate2d(img, kernel=None, level=1):
  ...
  return oimg
```

Decimirajte sliki mr_256x256_uint8.raw ter decimacija_256x256_uint8.raw tako, da bo velikost decimirane slike 64×64 slikovnih elementov. Uporabite funkcijo `imDecimate2d` s konvolucijskim jedrom nizkoprepustnega sita velikosti 3×3 (slika 2.5b). Izvedite decimacijo tudi brez filtriranja ter primerjajte dobljeni sliki. Kaj opazite?

2.4 Rešitve in odgovori na vprašanja

Pri odgovorih privzamemo, da se funkcije iz rešitev nahajajo v modulu `funkcije`, slikovno gradivo pa se nahaja v podmapi `poglavje_2`. Naprej uvozimo potrebne module in naložimo slike.

```
import numpy as np
from matplotlib import pyplot as pp
import funkcije

im1 = funkcije.imLoadRaw2d(
  './poglavje_2/mr_256x256_uint8.raw', 256, 256)
im2 = funkcije.imLoadRaw2d(
  './poglavje_2/mr_128x256_uint8.raw', 128, 256)
im3 = funkcije.imLoadRaw3d(
  './poglavje_2/mr_217x181x181_uint8.raw', 217, 181, 181)
im4 = funkcije.imLoadRaw2d(
  './poglavje_2/decimacija_256x256_uint8.raw', 256, 256)
```

1. V modulu `funkcije` ustvarimo funkcijo `interp1d`.

```
def interp1d(x, xp, fp, method='linear'):

  x = np.asarray(x, np.float)
  xp = np.asarray(xp, np.float)
  fp = np.asarray(fp, np.float)

  xshape = x.shape
  x = x.flatten()
  f = np.zeros_like(x)
  dx = float(xp[1] - xp[0])
  Nxp = xp.size
  Nx= x.size

  if method == 'linear':
    for i in range(Nx):
      indf = (x[i] - x[0])/dx
      d = indf - int(indf)
      ind1 = max(int(indf), 0)
      ind2 = min(ind1 + 1, Nxp - 1)
      f[i] = (1.0 - d)*fp[ind1] + d*fp[ind2]
    ''' hitra izvedba brez for zanke
    indf = (x - xp[0])/dx
    ind1 = indf.astype('int')
    f = np.zeros([Nx,1])
    ind1 = np.maximum(ind1, 0)
    ind2 = np.minimum(ind1 + 1, Nxp - 1)
    f[indok] = (1.0 - d)*fp[ind1] + d*fp[ind2 + 1]
    '''
```

```
29
30    elif method == 'nearest':
31      for i in range(Nx):
32        indf = np.round((x[i] - xp[0])/dx)
33        ind = min(max(indf, 0), Nxp - 1)
34        f[i] = fp[ind]
35      ''' hitra izvedba brez for zanke
36      ind = np.round((x - xp[0])/dx)
37      ind = np.minimum(np.maximum(ind, 0), Nxp - 1)
38      f = fp[ind]
39      '''
40
41    else:
42      raise ValueError(
43        'Vrednost parametra "method" je lahko '
44        '"linear" ali "nearest"!')
45
46    f.shape = xshape
47
48    return f
```

Glavna prednost interpolacije z metodo najbližjega soseda leži v računski nezahtevnosti postopka, slabost pa v nizki kakovosti interpolirane slike. Linearna interpolacija je približno 4-krat zahtevnejša od interpolacije z metodo najbližjega soseda. Kakovost interpoliranih slik je bistveno boljša, saj postopek upošteva uteženo vsoto sivinskih vrednost 4-ih sosednjih slikovnih elementov.

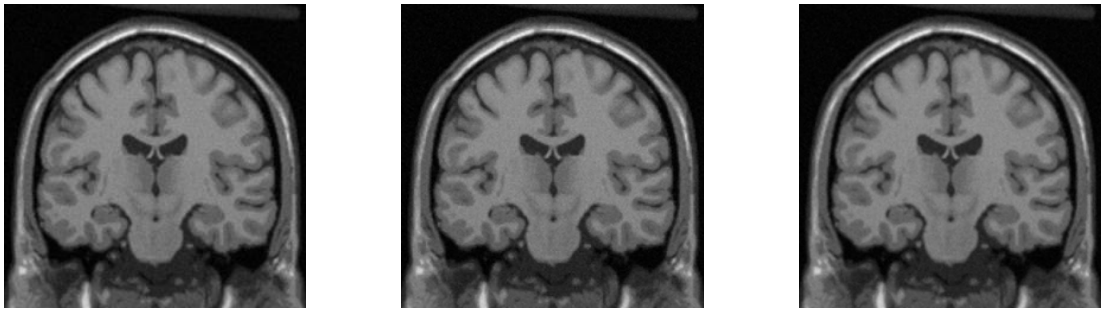
2. V modulu `funkcije` ustvarimo funkcijo `interp2d`.

```
1  def interp2d(x, y, xp, yp, fp, method='linear'):
2
3    x = np.asarray(x, np.float)
4    y = np.asarray(y, np.float)
5    xp = np.asarray(xp, np.float)
6    yp =  np.asarray(yp, np.float)
7    fp = np.asarray(fp, np.float)
8
9    xshape = x.shape
10   x, y = x.flatten(), y.flatten()
11   dx, dy = float(xp[1] - xp[0]), float(yp[1] - yp[0])
12   Nx, Ny = x.size, y.size
13   Nxp, Nyp = xp.size, yp.size
14   if Nx != Ny:
15     raise ValueError(
16       'Število elementov v x in y mora biti enako!')
17   f = np.zeros([Nx])
18   if method == 'linear':
19     for i in range(Nx):
20       indxf = (x[i] - xp[0])/dx
```

```
      indyf = (y[i] - yp[0])/dy
      xf = indxf - int(indxf)
      yf = indyf - int(indyf)
      indx = max(min(int(indxf), Nxp - 1), 0)
      indy = max(min(int(indyf), Nyp - 1), 0)
      a = (1.0 - xf)*(1.0 - yf)
      b = xf*(1.0 - yf)
      c = (1.0 - xf)*yf
      d = xf*yf
      s = 1.0
      f[i] = a/s*fp[indy, indx] + \
             b/s*fp[indy, min(indx + 1, Nxp - 1)] + \
             c/s*fp[min(indy + 1, Nyp - 1), indx] + \
             d/s*fp[
               min(indy + 1, Nyp - 1), min(indx + 1, Nxp - 1)
             ]

  elif method == 'nearest':
    for i in range(Nx):
      indxf = np.round((x[i] - xp[0])/dx)
      indyf = np.round((y[i] - yp[0])/dy)
      indx = min(max(indxf, 0), Nxp - 1)
      indy = min(max(indyf, 0), Nyp - 1)
      f[i] = fp[indy, indx]

  else:
    raise ValueError(
      'Vrednost parametra "method" je lahko '
      '"linear" ali "nearest"!')

  f.shape = xshape

  return f
```

(a) Interpolacija s podvojenim vzorčenjem v smeri obeh koordinatnih osi (slika 2.7).

```
x = y = np.linspace(0, 255, 512)
xp = yp = np.arange(256)
Yi, Xi = np.meshgrid(y, x, indexing='ij')
im1i_linear = funkcije.interp2d(
  Xi, Yi, xp, yp, im1, 'linear')
im1i_nearest = funkcije.interp2d(
  Xi, Yi, xp, yp, im1, 'nearest')

pp.figure('Odgovor 2 a)')

pp.subplot(1, 2, 1)
pp.imshow(im1i_linear, cmap='gray')
pp.title('Linearna')
```

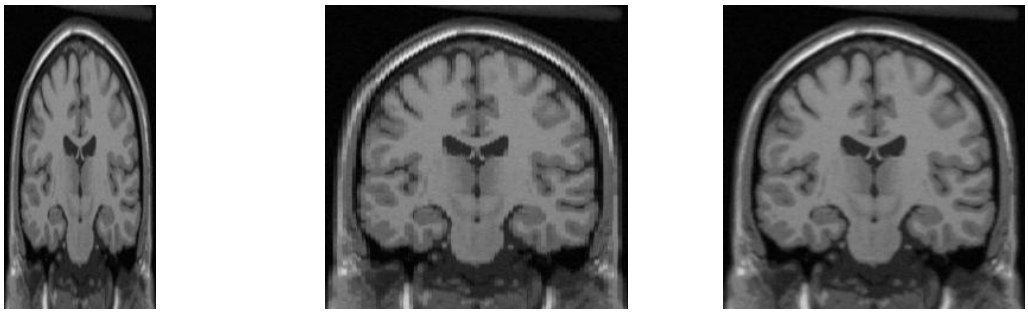


(a) Izvirna slika (b) Interp. z najbližjim sosedom (c) Bilinearna interpolacija

Slika 2.7: Interpolacija s podvojenim vzorčenjem v smeri obeh koordinatnih osi.

```
14
15 pp.subplot(1, 2, 2)
16 pp.imshow(im1i_nearest, cmap='gray')
17 pp.title('Najbližji sosed')
18
19 pp.show()
```

(b) Interpolacija s podvojenim vzorčenjem v smeri x osi (slika 2.8).



(a) Izvirna slika (b) Interp. z najbližjim sosedom (c) Bilinearna interpolacija

Slika 2.8: Interpolacija s podvojenim vzorčenjem v smeri x osi.

```
1  xp = np.arange(0, 256, 2)
2  yp = np.arange(256)
3  x = np.arange(0, 256)
4  y = yp
5  Yi, Xi = np.meshgrid(y, x,  indexing='ij')
6  im2i_linear = funkcije.interp2d(
7    Xi, Yi, xp, yp, im2, 'linear')
8  im2i_nearest = funkcije.interp2d(
9    Xi, Yi, xp, yp, im2, 'nearest')
10
11 pp.figure('Odgovor 2 b)')
```

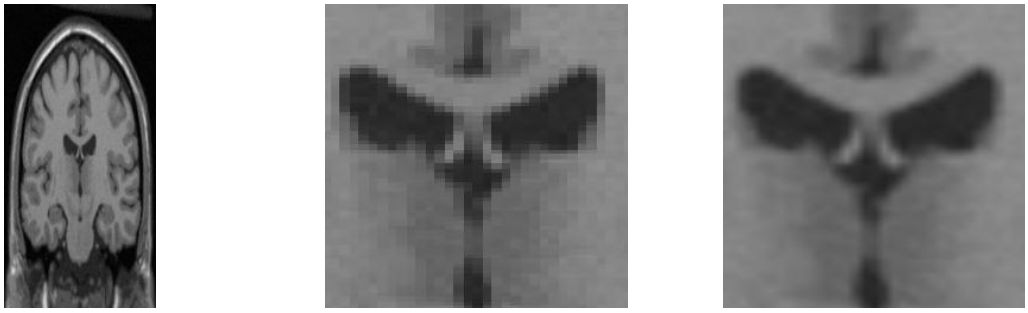
```
12
13 pp.subplot(1, 2, 1)
14 pp.imshow(im2i_linear, cmap='gray')
15 pp.title('Linearna')
16
17 pp.subplot(1, 2, 2)
18 pp.imshow(im2i_nearest, cmap='gray')
19 pp.title('Najbližji sosed')
20
21 pp.show()
```

(c) Interpolacija osrednjega dela slike.

```
1  xi = np.arange(128 - 16*2, 128 + 16*2, 0.25)
2  yi = np.arange(128 - 32, 128 + 32, 0.25)
3  Yi, Xi = np.meshgrid(yi, xi, indexing='ij')
4
5  im2i_box_nearest = funkcije.interp2d(
6    Xi, Yi, xp, yp, im2, 'nearest')
7  im2i_box_linear = funkcije.interp2d(
8    Xi, Yi, xp, yp, im2, 'linear')
9
10 from matplotlib.patches import Rectangle
11 region = Rectangle((64 - 16, 128 - 32), 32, 64,
12                     edgecolor='b', facecolor='none')
13
14 pp.figure('Odgovor 2 c)')
15 pp.suptitle('Interpolacija osrednjega dela slike')
16
17 pp.subplot(1, 3, 1)
18 pp.imshow(im2, cmap='gray', vmin=0, vmax=255)
19 pp.gca().add_patch(region)
20 pp.title('Izhodiščna slika')
21
22 pp.subplot(1, 3, 2)
23 pp.title('Najbližji sosed')
24 pp.imshow(im2i_box_nearest, cmap='gray', vmin=0, vmax=255)
25
26 pp.subplot(1, 3, 3)
27 pp.title('Linearna')
28 pp.imshow(im2i_box_linear, cmap='gray', vmin=0, vmax=255)
29
30 pp.show()
```

3. V modulu `funkcije` ustvarimo funkcijo `interp3d`.

```
1 def interp3d(x, y, z, xp, yp, zp, fp, method='linear'):
2   x = np.asarray(x, np.float)
3   y = np.asarray(y, np.float)
4   z = np.asarray(z, np.float)
```



(a) Izhodiščna slika (b) Interp. z najbližjim sosedom (c) Bilinearna interpolacija

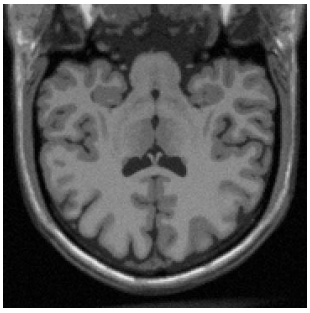
Slika 2.9: Interpolacija osrednjega dela slike. Velikost slikovnega elementa interpolirane slike znaša $x \times y = 0{,}25 \times 0{,}25$ mm.

```
5   xp = np.asarray(xp, np.float)
6   yp = np.asarray(yp, np.float)
7   zp = np.asarray(zp, np.float)
8   fp = np.asarray(fp, np.float)
9
10  xshape = x.shape
11  x = x.flatten()
12  y = y.flatten()
13  z = z.flatten()
14  dx = float(xp[1] - xp[0])
15  dy = float(yp[1] - yp[0])
16  dz = float(zp[1] - zp[0])
17  Nxp, Nyp, Nzp = xp.size, yp.size, zp.size
18  Nx, Ny, Nz = x.size, y.size, z.size
19  if Nx != Ny or Nx != Nz:
20    raise ValueError(
21      'Število elementov v x, y in z mora biti enako!')
22
23  f = np.zeros([Nx])
24  if method == 'linear':
25    for i in range(Nx):
26      indxf = (x[i] - xp[0])/dx
27      indyf = (y[i] - yp[0])/dy
28      indzf = (z[i] - zp[0])/dz
29      xf = indxf - int(indxf)
30      yf = indyf - int(indyf)
31      zf = indzf - int(indzf)
32      indx = max(min(int(indxf), Nxp - 1), 0)
33      indy = max(min(int(indyf), Nyp - 1), 0)
34      indz = max(min(int(indzf), Nzp - 1), 0)
35      a = (1.0 - xf)*(1.0 - yf)
36      b = xf*(1.0 - yf)
37      c = (1.0 - xf)*yf
```

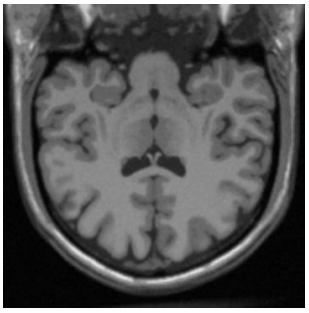
```
      d = xf*yf
      s = 1.0
      bz = zf
      az = 1.0 - zf
      fzk = a/s*fp[indz, indy, indx] + \
        b/s*fp[indz, indy, min(indx + 1, Nxp - 1)] + \
        c/s*fp[indz, min(indy + 1, Nyp - 1), indx] + \
        d/s*fp[indz,
               min(indy + 1, Nyp - 1), min(indx + 1, Nxp - 1)]
      indz1 = min(indz + 1, Nzp - 1)
      fzk1 = a/s*fp[indz1, indy, indx] + \
        b/s*fp[indz1, indy, min(indx + 1, Nxp - 1)] + \
        c/s*fp[indz1, min(indy + 1, Nyp - 1), indx] + \
        d/s*fp[indz1, min(indy + 1, Nyp - 1),
               min(indx + 1, Nxp - 1)]
      f[i] = fzk*az + fzk1*bz

  elif method == 'nearest':
    for i in range(Nx):
      indxf = np.round((x[i] - xp[0])/dx)
      indyf = np.round((y[i] - yp[0])/dy)
      indzf = np.round((z[i] - zp[0])/dz)
      indx = min(max(indxf, 0), Nxp - 1)
      indy = min(max(indyf, 0), Nyp - 1)
      indz = min(max(indzf, 0), Nzp - 1)
      f[i] = fp[indz, indy, indx]

  else:
    raise ValueError(
      'Vrednost parametra "method" je lahko '
      '"linear" ali "nearest"!')

  f.shape = xshape

  return f
```

Interpolacija v yz ravnini pri $x = 110{,}3$ (slika 2.10).

```
xp = np.arange(217)
yp = np.arange(181)
zp = np.arange(181)
y = z = np.linspace(0, 180, 181*2)
Zi, Yi, Xi = np.meshgrid(z, y, 110.3, indexing='ij')
im3i_nearest = funkcije.interp3d(
  Xi, Yi, Zi, xp, yp, zp, im3, 'nearest')
im3i_linear = funkcije.interp3d(
  Xi, Yi, Zi, xp, yp, zp, im3, 'linear')

pp.figure('Odgovor 3')
```



(a) Interp. z najbližjim sosedom



(b) Trilinearna interpolacija

Slika 2.10: Interpolacija v yz ravnini pri $x = 110.3$ s podvojenim vzorčenjem v smeri y in z osi.

```
12
13 pp.subplot(1, 2, 1)
14 pp.imshow(im3i_linear.squeeze(), cmap='gray')
15 pp.title('Linearna')
16
17 pp.subplot(1, 2, 2)
18 pp.imshow(im3i_nearest.squeeze(), cmap='gray')
19 pp.title('Najbližji sosed')
20
21 pp.show()
```

4. V modulu `funkcije` ustvarimo funkcijo `conv2d`.

```
1  def conv2d(data, kernel):
2    K = np.asarray(kernel, np.float)
3    S = np.asarray(data, np.float)
4    c = np.floor(np.array(K.shape)/2.0) # Središče konvolucijskega jedra.
5    H, W = S.shape     # Velikost vhodne slike.
6    A, B = K.shape     # Velikost konvolucijskega jedra.
7    odata = np.zeros([H, W])  # Podatkovno polje izhodne (zglajene) slike.
8    for i in range(H):        # Vrstica slikovnega elementa.
9      for j in range(W):  # Stolpec slikovnega elementa.
10       for k in range(A):  # Vrstica konvolucijskega jedra.
11         for l in range(B): # Stolpec konvolucijskega jedra.
12           # Premik naslovov.
13           ic = i - (k - c[0])
14           jc = j - (l - c[1])
15           # Pred izračunom preverimo veljavnost naslova slikovnega elementa.
16           if ic > 0 and ic < H and jc > 0 and jc < W:
17             odata[i, j] += K[k, l]*S[ic, jc]
18
19   return odata
```

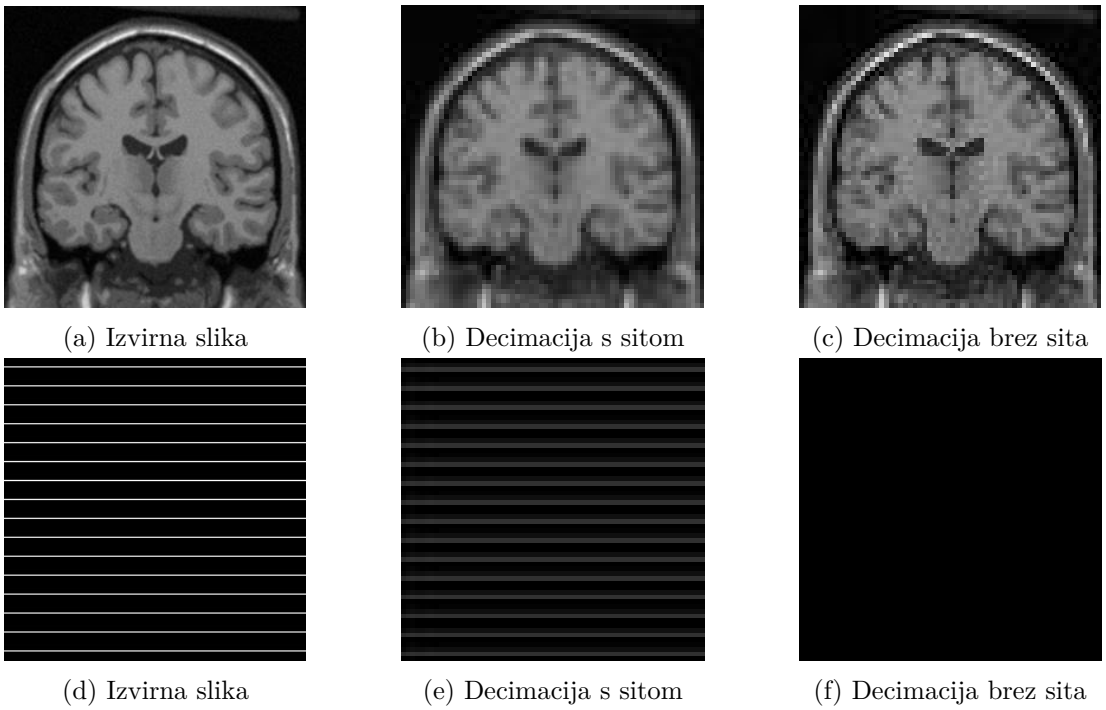
5. V modulu `funkcije` ustvarimo funkcijo `imDecimate2d`.

```
def imDecimate2d(img, kernel=None, level=1):
  if kernel is None:
    kernel = np.array([[1.0/16, 1/8, 1/16],
                       [1/8,    1/4, 1/8],
                       [1.0/16, 1/8, 1/16]])
  oimg = np.array(img, 'float')
  for i in range(level):
    oimg = conv2d(oimg, kernel)
    oimg = oimg[::2, ::2]

  return oimg
```

Rezultat decimacija slike z in brez uporabe nizkoprepustnega sita velikosti 3×3 je prikazan na sliki 2.11. Pri decimaciji brez predhodnega filtriranja slike z nizkoprepustnim sitom, lahko pride do popačenja ali izgube informacije, saj predhodno ne omejimo frekvenčne informacije slike na polovico vzorčne frekvence. Omenjeni pojav je še posebej poudarjen pri decimaciji sintetične slike decimacija_256x256_uint8.raw.

```
K3 = np.array([[1.0/16,  1/8,   1/16],
               [1/8,     1/4,   1/8 ],
               [1.0/16,  1/8,   1/16]])
K5 = np.array([[1.0/400, 1/80,  1/50,  1/80, 1/400],
               [1/80,    1/16,  1/10,  1/16, 1/80 ],
               [1/50,    1/10,  4/25,  1/10, 1/50 ],
               [1/80,    1/16,  1/10,  1/16, 1/80 ],
               [1.0/400, 1/80,  1/50,  1/80, 1/400]])

im4d_k3 = funkcije.imDecimate2d(im4, K3 ,2)
im4d = im4[::4,::4]
im1d_k3 = funkcije.imDecimate2d(im1, K3 ,2)
im1d = im1[::4,::4]

pp.figure('Odgovor 5')

pp.subplot(2, 3, 1)
pp.imshow(im1, cmap='gray')
pp.title('Izvirna slika')

pp.subplot(2, 3, 2)
pp.imshow(im1d_k3, cmap='gray')
pp.title('Decimimacija s filtranjem')

pp.subplot(2, 3, 3)
pp.imshow(im1d, cmap='gray')
pp.title('Decimirana brez filtriranja')

pp.subplot(2, 3, 4)
pp.imshow(im4, cmap='gray')
pp.title('Izvirna slika')
```

```
32
33 pp.subplot(2, 3, 5)
34 pp.imshow(im4d_k3, cmap='gray')
35 pp.title('Decimimacija s filtranjem')
36
37 pp.(2, 3, 6)
38 pp.imshow(im4d, cmap='gray')
39 pp.title('Decimirana brez filtranja')
40
41 pp.show()
```



(a) Izvirna slika (b) Decimacija s sitom (c) Decimacija brez sita

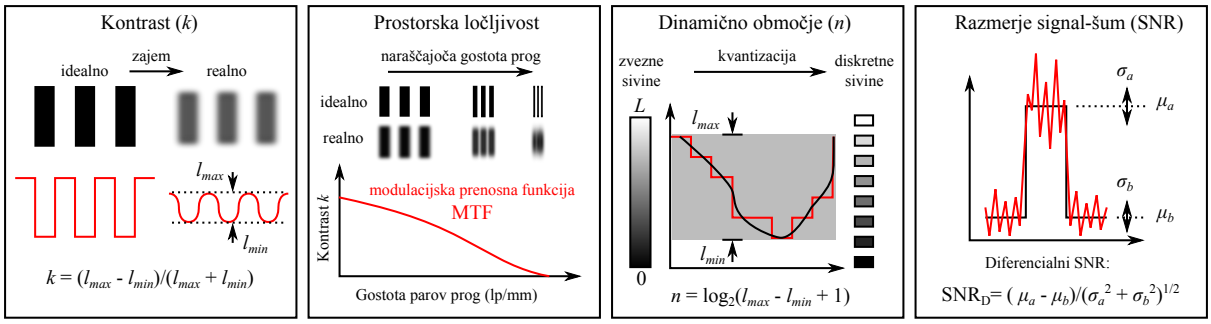
(d) Izvirna slika (e) Decimacija s sitom (f) Decimacija brez sita

Slika 2.11: Dve zaporedni decimaciji s faktorjem 2 z in brez uporabe nizkoprepustnega sita velikosti 3×3.

Poglavje 3

Parametri in kakovost slik

Poglavje je namenjena spoznavanju in razumevanju osnovnih lastnosti realnih slik kot so svetlost, dinamično območje, histogram, kvantizacija, šum in barva ter spoznavanju načinov za določanje značilnih parametrov kakovosti realnih slik kot je razmerje signal-šum (slika 3.1).



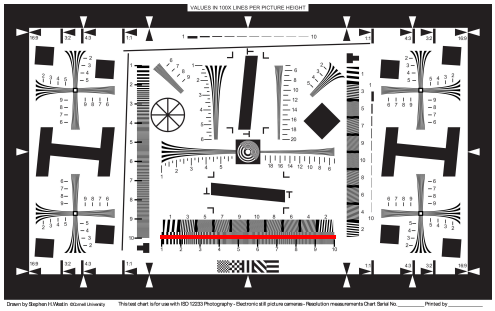
Slika 3.1: Parametri kakovosti slik.

3.1 Naloge in vprašanja

1. S pomočjo fotoaparata zajemite sliko kalibra ColorChecker in sliko kalibra ISO-12233, ki ju prikazuje slika 3.2. Bliskavica naj bo pri tem izklopljena. Sliki poizkušajte zajeti tako, da bosta zunanja robova kalibra čim bolje poravnana s tipalom. Sliki prenesite na računalnik in ju uvozite s pomočjo funkcije `open` modula `PIL.Image` in funkcije `array` modula `numpy` ter ju prikažite s funkcijo `imshow` modula `matplotlib.pyplot`. Iz zajetih slik izluščite pravokotna področja oziroma podslike tako, da bo vsako področje vsebovalo le eno barvo oziroma sivino. Področja lahko shranite v seznam ali večrazsežno polje v enakem vrstnem redu kot so označena na sliki 3.2a. Točke na sliki boste najenostavneje izbrali s funkcijo `ginput` modula `matplotlib.pyplot`. Za hitro in enostavno določanje pravokotnih področij na sliki lahko približno določite zgolj središči področij 1 in 24, lege preostalih pravokotnih podpodročij pa izračunate. Na podoben način lahko določite sivinski prerez kalibra ISO-12233, ki ga prikazuje rdeča črta na sliki 3.2b. Števila na kalibru ISO-12233

označujejo število prog v enotah 100, ki ustrezajo višini uporabnega dela kalibra (200 mm). Na primer, število 3 označuje proge širine $\frac{200}{100 \cdot 3}$ mm.

(a) Kaliber ColorChecker



(b) Kaliber ISO-12233

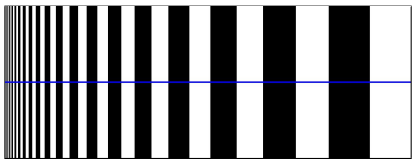
Slika 3.2: Slike kalibrov z oznakami.

2. Posamezna področja pretvorite v sivinske slike po enačbi $S = 0.299 \cdot R + 0.587 \cdot G, 0.144 \cdot B$, kjer R, G in B predstavljajo sivinske vrednosti rdeče, zelene in modre barvne komponente. Za vsako področje izračunajte povprečje in standardno deviacijo sivinskih vrednosti. Razmislite, katero področje bi moralo imeti najmanjšo in katero največjo povprečno 8-bitno sivinsko vrednost. Navedite zaporedni številki pripadajočih področij.

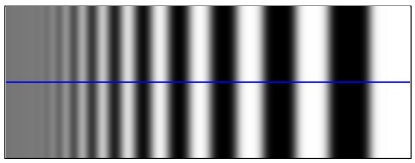
3. Prostorska ločljivost optičnega sistema nam pove najmanjšo razdaljo med točkastima objektoma, pri kateri lahko objekta še dobro razločimo. Slednjo lahko ocenimo z modulacijsko prenosno funkcijo sistema (MTF) [4], ki meri sivinski kontrast v odvisnosti od prostorske gostote svetlo-temnih prog. Gostoto prog kalibracijske tarče MTF najpogosteje podamo kot število parov prog na milimeter (lp/mm), ločljivost sistema pa pogosto podamo kot število parov prog, pri katerem sivinski kontrast pade na 10 % začetne vrednosti (ilustracija na sliki 3.3).

   Prostorsko ločljivost slikovnega sistema lahko določimo tudi tako, da zajamemo sliko odziva na enotino stopnico. V ta namen uporabimo kalibracijsko tarčo z ostrim prehod sivinskih vrednosti (slika 3.4). Zajeti odziv nato odvajamo v smeri, ki je pravokotna na prehod. Na ta način dobimo linijski odziv slikovnega sistema (ang. Line Spread Function ali LSF), tj. odziv slikovnega sistema na neskončno tanko in svetlo črto. Za prostorsko ločljivost običajno proglasimo širino linijskega odziva pri polovični amplitudi (ang. Full Width at Half Maximum ali FWHM). Ta postopek je zelo enostaven, saj je treba zajeti zgolj sliko kalibracijske tarče z ostrim prehodom sivinskih vrednosti. Zaradi numeričnega odvajanja pa je postopek zelo občutljiv na prisotnost šuma v zajeti sliki.

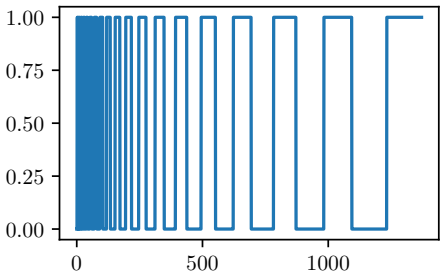
   Izrišite z rdečo črto označeni sivinski prerez kalibra ISO-12233 ter približno določite indeks območja (na kalibru ISO-12233 je označen z 1 do 10), kjer sivinski kontrast k s slike 3.1 pade na 10 % začetne vrednosti. Ob smiselni uporabi lastnosti kalibra ISO-12233 (višina 200 mm) približno ocenite širino črte v milimetrih, ki ustreza dobljeni vrednosti. Kakovost prereza lahko izboljšate tako, da povprečite nekaj zaporednih vrstic slike.



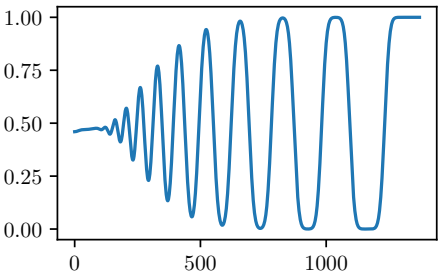
(a) MTF tarča



(b) Odziv na MTF tarčo

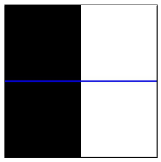


(c) Prerez MTF tarče

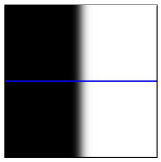


(d) Prerez odziva na MTF tarčo

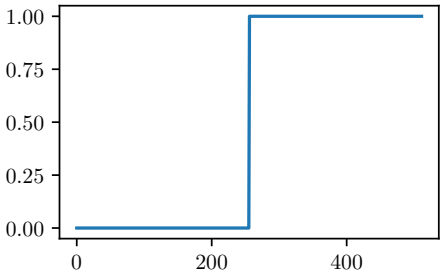
Slika 3.3: Primer MTF kalibracijske tarče in pripadajoči odziv slikovnega sistema.



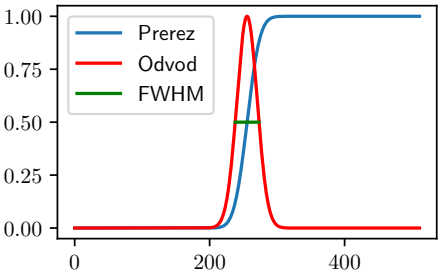
(a) LSF tarča



(b) Odziv na LSF tarčo



(c) Prerez LSF tarče



(d) Prerez in pripadajoči odvod odziva na LSF tarčo

Slika 3.4: Primer LSF kalibracijske tarče in pripadajoči odziv slikovnega sistema ter FWHM ločljivost.

4. Narišite graf, ki prikazuje povprečne sivinske vrednosti področij od 19 do 24 slike kalibra ColorChecker v odvisnosti od pripadajoče zaporedne številke področja. Na podlagi prikazanih vrednosti ocenite uporabno dinamično območje sivinskih vrednosti (izrazite s številom bitov).

5. Histogram slike je grafično orodje za prikazovanje frekvenčne porazdelitve sivinskih vre-

dnosti slikovnih elementov. Abscisna os histograma predstavlja sivinske vrednosti, ordinatna os histograma pa podaja število slikovnih elementov na izbranem intervalu sivinskih vrednosti. Ustvarite funkcijo za izračun in prikaz histograma slike.

```
def imHistogram(img, bins=256, span=None,
                density=False, title=None):
  ...
  return hist, edges
```

Pri tem parameter `img` predstavlja vhodno sliko, parameter `bins` pa število razredov histograma na razponu sivinskih vrednosti `span`. Če je vrednost parametra `span` enaka `None`, potem naj razpon sivinskih vrednosti določata najmanjša in največja sivinska vrednost v sliki. Parameter `title` predstavlja naslovno vrstico grafičnega prikaza histograma, ki ga ustvarimo le, ko je vrednost različna od `None`. Razredi naj vključujejo spodnjo, ne pa tudi zgornjo mejo razpona sivinskih vrednosti. Če je vrednost parametra `density` enaka `True`, potem histogram normalizirajte tako, da predstavlja oceno gostote verjetnosti sivinskih vrednosti. Funkcija naj vrne vrednosti v obliki vektorjev frekvenc `hist` in robnih točk razredov `edges`, katerih število elementov znaša `bins` ter `bins + 1`. Histogram izračunajte brez uporabe naprednih Python funkcij, za prikazovanje pa uporabite funkcijo `bar` modula `matplotlib.pyplot`. Pravilnost delovanja programske kode preverite s funkcijo `histogram` modula `numpy`.

Na področjih 19 in 24 kalibra ColorChecker kvalitativno preverite ali šum ustreza normalni porazdelitvi. To storite tako, da v skupno grafično okno izrišete normalizirani histogram sivin in pripadajočo Gaussovo porazdelitev:

$$\mathcal{N}(\mu, \sigma^2) = \frac{1}{\sigma\sqrt{2\pi}} \exp^{-\frac{(x-\mu)^2}{2\sigma^2}}. \tag{3.1}$$

6. Razmerje signal-šum (SNR) je pomemben kriterij za vrednotenje relativne jakosti oziroma moči signala glede na prisoten šum. SNR je namreč merilo zanesljivosti oziroma sposobnosti zaznavanja prisotnosti sprememb v opazovanem signalu. Pogosto se uporabljajo trije osnovni načini podajanja SNR, in sicer amplitudni ($\mathrm{SNR_A}$), diferencialni ($\mathrm{SNR_D}$) ter močnostni ($\mathrm{SNR_M}$) način. Podajanje SNR ni standardizirano in je predvsem odvisno od vrste signala in šuma ter od namena podajanja in uporabe. V primeru dveh nivojev signala zapišemo $\mathrm{SNR_D}$ tako kot prikazuje slika 3.1.

   Izračunajte diferencialno razmerje signal-šum ($\mathrm{SNR_D}$) med področjema 19 in 24. Na podoben način izračunajte še diferencialno razmerje signal-šum med področjema 23 in 24. Katero izmed obeh razmerij je večje?

7. Barvo slikovnega elementa običajno definiramo s tremi, lahko pa tudi le z dvema komponentama oziroma vrednostima. Zaradi načina pretvorbe svetlobe v digitalni zapis se najpogosteje uporablja zapis barve slikovnega elementa s komponentami RGB, ki ustrezajo odzivom treh različnih tipal svetlobe. Slednja so selektivno občutljiva na valovnih območjih okoli 700 nm (R), 550 nm (G) in 450 nm (B). Obstajajo tudi drugi barvni

prostori, ki so bolj primerni za analizo digitalnih slik, na primer XYZ in Lab, kjer ena izmed treh komponenta predstavlja svetlost, preostali dve komponenti pa barvni odtenek (poglobljeno obravnavo različnih barvnih prostorov najdemo v [5, 6]). Sliko, zapisano v RGB barvnem prostoru, lahko pretvorimo v drug barvni prostor z (ne)linearno preslikavo RGB komponent. Preslikava iz RGB v XYZ barvni prostor pri referenčni beli osvetlitvi D65 je definirana kot:

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} 0,4124564 & 0,3575761 & 0,1804375 \\ 0,2126729 & 0,7151522 & 0,0721750 \\ 0,0193339 & 0,1191920 & 0,9503041 \end{bmatrix} \cdot \begin{bmatrix} R \\ G \\ B \end{bmatrix} \tag{3.2}$$
$$x = \frac{X}{X+Y+Z}, \quad y = \frac{Y}{X+Y+Z}, \quad z = \frac{Z}{X+Y+Z},$$

pri čemer morajo vrednosti komponent R, G in B ležati na intervalu [0, 1]. Matrično množenje, s katerim lahko zapišemo preslikavo iz RGB v XZY barvni prostor, najlažje udejanimo s funkcijo `dot` modula `numpy`. Preslikava iz XYZ v Lab barvni prostor je za belo referenco (X_w, Y_w, Z_w) definirana kot:

$$\begin{aligned} L &= 116 \cdot f\left(\frac{Y}{Y_w}\right) - 16 \\ a &= 500 \cdot \left(f\left(\frac{X}{X_w}\right) - f\left(\frac{Y}{Y_w}\right)\right) \\ b &= 200 \cdot \left(f\left(\frac{Y}{Y_w}\right) - f\left(\frac{Z}{Z_w}\right)\right) \end{aligned} \tag{3.3}$$
$$f(t) = \begin{cases} t^{1/3}, & \text{ko } t > \left(\frac{6}{29}\right)^3 \\ \frac{1}{3}\left(\frac{29}{6}\right)^2 t + \frac{4}{29}, & \text{drugod.} \end{cases}$$

Lab barvni prostor je zaradi linearne metrike med različnimi barvami še posebej primeren za kvantitativno primerjavo barv in barvnih odtenkov.

(a) Pretvorite sliko kalibra ColorChecker iz RGB v XYZ barvni prostor in prikažite komponento Y ter normalizirani komponenti x, y kot sivinske slike. Katera izmed komponent barvnega prostora xyY predstavlja svetlost?

(b) Pretvorite dobljeno sliko iz XYZ še v Lab barvni prostor in prikažite L, a in b komponente kot sivinske slike. Pri pretvorbi naj bela referenca (X_w, Y_w, Z_w) ustreza preslikanim normaliziranim RGB koordinatam $(1.0, 1.0, 1.0)$. Katera komponenta Lab barvnega prostora predstavlja svetlost?

8. Pri izgubnem zgoščevanju slik (ang. compression) nas pogosto zanima, kakšna je stopnja degradacije kakovosti zgoščene slike. Slednjo lahko številsko ovrednotimo z metriko strukturne podobnosti (ang. Structural Similarity Index - SSIM), ki oceni kakovost testne (zgoščene) slike Y glede na referenčno sliko X.

$$\text{SSIM} = \frac{(2\mu_x\mu_y + c_1)(2\sigma_{xy} + c_2)}{(\mu_x^2 + \mu_y^2 + c_1)(\sigma_x^2 + \sigma_y^2 + c_2)} \tag{3.4}$$

Pomen posameznih členov v enačbi je sledeč:

- μ_x predstavlja povprečno vrednost sivin referenčne slike,
- μ_y predstavlja povprečno vrednost sivin testne slike,
- σ_x^2 predstavlja varianco sivin referenčne slike,
- σ_y^2 predstavlja varianco sivin testne slike,
- σ_{xy} predstavlja kovarianco sivin referenčne in testne slike,
- $c_1 = (k_1 L)^2$ in $c_2 = (k_2 L)^2$ sta regularizacijska člena,
- L dinamično območje sivinskih vrednosti (običajno $[0, 2^{\text{št. bitov}} - 1]$),
- $k_1 = 0,01$ in $k_2 = 0.03$.

Vrednost SSIM lahko izračunamo za vsak slikovni element testne slike in s tem pridobimo informacijo o kakovosti posameznih področij slike. Pri tem običajno uporabimo okolico velikosti 11×11 slikovnih elementov, sivinske vrednosti v izbrani okolici pa pred izračunom SSIM utežimo z Gaussovo funkcijo (enačba 3.1).

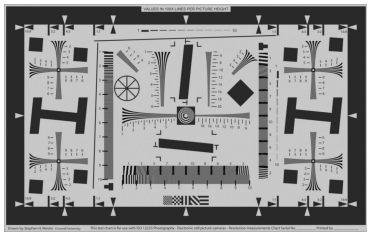
(a) Ustvarite funkcijo `ssim`, ki izračuna vrednosti SSIM za okolice vseh slikovnih elementov vhodnih slik `imgx` in `imgy`. Parameter `l` predstavlja dinamično območje sivinskih vrednosti vhodnih slik, `n` velikost okolice, `sigma` parameter σ Gaussove funkcije, parametra `k1` ter `k2` pa konstanti k_1 ter k_2 metrike SSIM.

```
def ssim(imgx, imgy, l=255, n=11, sigma=1.5,
         k1=0.01, k2=0.03):
  ...
  return ossim
```

(b) Raziščite, kako se vrednost metrike SSIM spreminja v odvisnosti od kakovosti izgubnega zgoščevanja JPEG za sliko mrBrainSlice.png iz poglavja 1. Zgoščevanje slik lahko enostavno izvedete z metodo `save` modula `PIL.Image`. Kakovost zgoščevanja določite s parametrom `quality`, in sicer tako, da spreminjajte njegovo vrednost od 1 (najnižja kakovost) do 100 (najvišja kakovost).

3.2 Rešitve in odgovori na vprašanja

Pri odgovorih privzamemo, da se funkcije iz rešitev nahajajo v modulu `funkcije`, slikovno gradivo pa se nahaja v podmapi `poglavje_3`. Funkcija za pretvorbo sRGB barvne slike v sivinsko

Slika 3.5: Sliki kalibra ColorChecker in ISO-12233.

sliko, ki je del modula `funkcije`.

```
def srgb2gs(img):
  return 0.299*img[:,:,0] + 0.587*img[:,:,1] + 0.144*img[:,:,2]
```

Najprej ustvarimo nekaj spremenljivk in pomožnih funkcij za določanje področij in pretvorbo med barvnimi prostori.

```
# Uvozimo potrebne module.
import numpy as np
from matplotlib import pyplot as pp
from PIL import Image as im
import funkcije

# Naložimo sliki kalibrov ColoCheker in ISO-12233.
CC = np.asarray(im.open('./poglavje_3/ColorChecker.jpg'))
ISO = np.asarray(im.open('./poglavje_3/ISO-12233.jpg'))

# Pretvorimo sliki v sivinski.
CCg, ISOg = funkcije.srgb2gs(CC), funkcije.srgb2gs(ISO)

# Število področij v sliki kalibra ColorChecker.
Nx, Ny = 6, 4
N = Nx*Ny
dd = 0.5     # delež izrezanega področja

# Velikost slike kalibra ColorChecker.
H, W = CC.shape[0], CC.shape[1]

# Preslikava iz sRGB v XYZ pri beli referenci D65.
RGB2XYZ = np.array([[0.4124564, 0.3575761, 0.1804375],
                    [0.2126729, 0.7151522, 0.0721750],
                    [0.0193339, 0.1191920, 0.9503041]])

# Funkcija za določanje področij na sliki.
```

```
28 def extractSubImages(img, rect=None):
29   if rect is None:
30     pp.imshow(img)
31     pp.title('Označi središče levega zgornjega, '\
32              'nato desnega spodnjega področja.')
33     t1, t2 = pp.ginput(2)
34   else:
35     t1, t2 = rect
36 
37   h = (t2[1] - t1[1])/float(Ny - 1) # višina področja
38   w = (t2[0] - t1[0])/float(Nx - 1) # širina področja
39 
40   x0 = t1[0] # x koordinata središča področja 1
41   y0 = t1[1] # y koordinata središča področja 1
42 
43   h2 = round(dd*h/2)
44   w2 = round(dd*w/2)
45   podslike = []
46   for j in range(Ny): # naslov področja v navpični (y) smeri
47     for i in range(Nx): # naslov področja v vodoravni (x) smeri
48       xcenter = round(x0 + w*i) # odmik središča v vodoravne smeri
49       ycenter = round(y0 + h*j) # odmik središča v navpični smeri
50       podslike.append( # sliko področja pripnemo na koncu seznama
51         img[ycenter - h2 : ycenter + h2,
52             xcenter - w2 : xcenter + w2])
53 
54       if rect is None:
55         pp.subplot(Ny, Nx, i + j*Nx + 1)
56         pp.imshow(podslike[-1])
57         pp.axis('off')
58 
59   if rect is None:
60     pp.suptitle('Izbrana področja.')
61 
62   return podslike, rect
63 
64 # Pomožna funkcija za pretvorbo iz XYZ v Lab barvni prostor.
65 def funXYZ2Lab(t):
66   out = np.zeros_like(t)
67   ind = t > (6.0/29.0)**3
68   out[ind] = t[ind]**(1.0/3.0)
69   nind = np.logical_not(ind)
70   out[nind] = (1.0/3.0*(29.0/6.0)**2)*t[nind] + 4.0/29.0
71 
72   return out
73 
74 # Funkcija za pretvorbo iz XYZ v Lab barvni prostor.
75 def Xyz2Lab(X, Y, Z, Xw, Yw, Zw):
76   tx = X/Xw
```

```
77    ty = Y/Yw
78    tz = Z/Zw
79
80    L = 116.0*funXYZ2Lab(tx) - 16.0
81    a = 500*(funXYZ2Lab(tx) - funXYZ2Lab(ty))
82    b = 200*(funXYZ2Lab(ty) - funXYZ2Lab(tz))
83
84    return L, a, b
```

Sedaj lahko začnemo z obravnavo zastavljenih vprašanj.

1. Določimo področja na sliki kalibra ColorChecker.

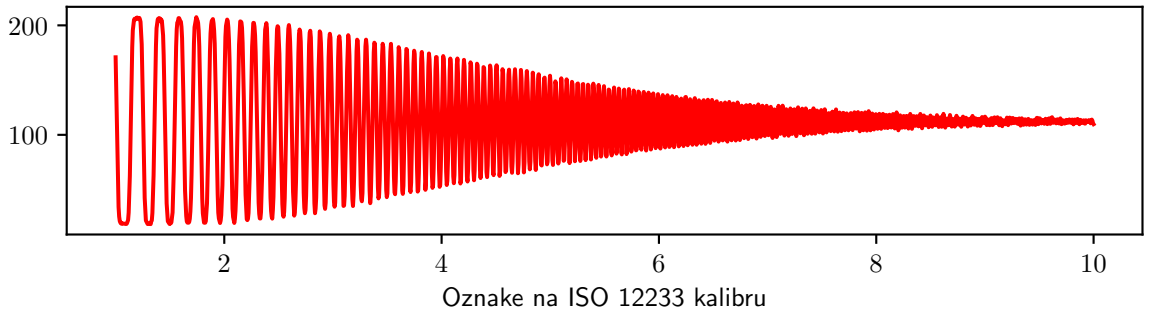
```
1  pp.figure()
2  sCC, rectCC = extractSubImages(CC)
```

2. Največjo povprečno sivinsko vrednost ima področje 19, najmanjšo pa področje 24. Sledi izračun statistike sivinskih vrednosti področij.

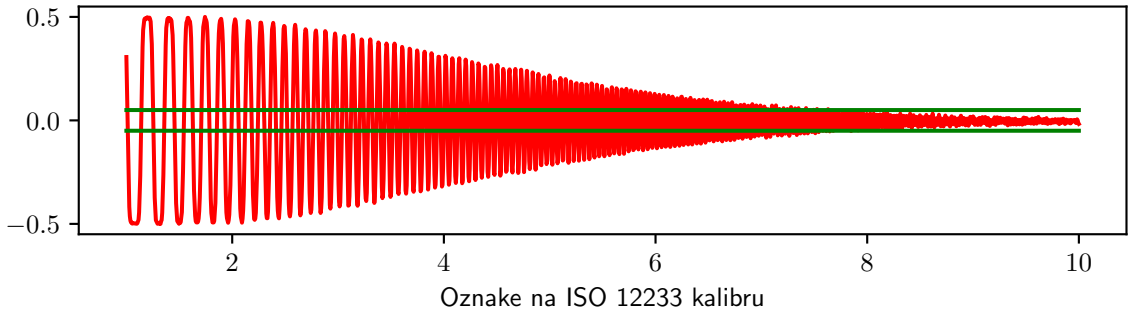
```
1  sCCg = []
2  sCCmean = np.zeros(N)
3  sCCstd = np.zeros(N)
4  for i in range(len(sCC)):
5    cc = sCC[i]
6    sCCg.append(funkcije.srgb2gs(cc))
7    sCCmean[i] = sCCg[-1].mean()
8    sCCstd[i] = sCCg[-1].std()
```

3. Za lažje vrednotenje izrišemo izvirni in normalizirani sivinski prerez slike kalibra ISO-12233 (normaliziramo na interval od $-0{,}5$ do $0{,}5$). Slednjega opremimo še z mejami kontrasta. Šum lahko izdatno zmanjšamo tako, da povprečimo več vrstic slike.

```
1  pp.figure()
2
3  pp.subplot(3, 1, 1)
4  pp.imshow(ISO, cmap='gray')
5  pp.title('Označi krajišča prereza - '
6           'levo zgoraj nato desno spodaj')
7  if t1iso is None or t2iso is None:
8    t1iso, t2iso = pp.ginput(n=2, timeout=120)
9    t1iso = [int(t1iso[0]), int(t1iso[1])]
10   t2iso = [int(t2iso[0]), int(t2iso[1])]
11 pp.plot([t1iso[0], t2iso[0]], [t1iso[1], t2iso[1]], '-r')
12
13 pp.subplot(3, 1, 2)
14 mtfprerez = ISOg[t1iso[1]:max(t1iso[1] + 1, t2iso[1]),
15                 t1iso[0]:t2iso[0]].mean(0)
16 isolp = np.linspace(1, 10, mtfprerez.size)
17 pp.plot(isolp, mtfprerez)
18 pp.title('Izvirni prerez')
```



(a) Izvirni prerez



(b) Normalizirani prerez

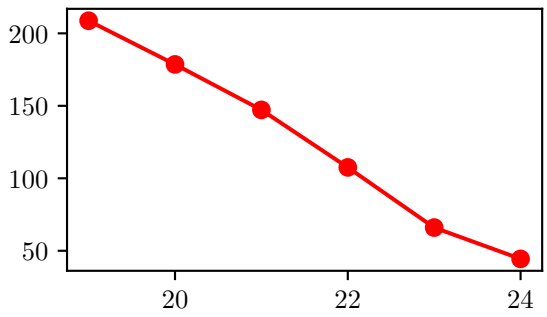
Slika 3.6: Izvirni in normalizirani sivinski prerez slike kalibra ISO-12233. Vodoravni črti zelene barve označujeta mejo kontrasta, ki znaša 10 % začetne vrednosti.

```
19
20 pp.subplot(3, 1, 3)
21 pp.title('Normalizirani prerez')
22 l, h = mtfprerez.min(), mtfprerez.max()
23 pp.plot(isolp, (mtfprerez - l)/(h - l) - 0.5, '-r')
24 pp.plot([1, 10], [0.05, 0.05], '-g')
25 pp.plot([1, 10], [-0.05, -0.05], '-g')
26
27 pp.show()
```

Z metodo ostrega očesa ocenimo, da kontrast pade na 10 % začetne vrednosti pri približno 750 progah na višino (200 mm) kalibra ISO-12233. Iz tega izračunamo širino proge, ki znaša približno 0,266 mm. Ločljivost slikovnega sistema torej znaša približno 0,266 mm.

4. Izrišemo potek povprečne sivinske vrednosti za izbrana področja (slika 3.7) in določimo dinamično območje sivinskih vrednosti.

```
1 pp.figure()
```



Slika 3.7: Povprečna sivinska vrednost področij od 19 do 24.

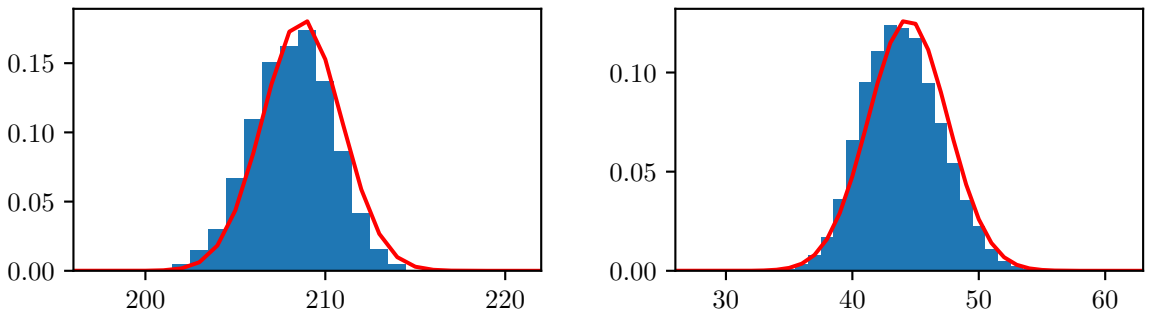
```
pp.plot(np.arange(19, 25), sCCmean[18:24], '-or')
pp.xlabel('Področje slike ColorChecker')
pp.ylabel('Povprečna sivina')
pp.show()
print('Dinamično območje znaša {:.1f} bitov.'.format(
  np.log2(sCCmean[18] - sCCmean[23])))
pp.show()
```

Dinamično območje znaša 7,4 bitov.

5. V modulu `funkcije` ustvarimo funkcijo `imHistogram`.

```
def imHistogram(img, bins=256, span=None,
                density=False, title=None):
  img = np.asarray(img)
  if span is None:
    span = (img.min(), img.max())
  span = np.asarray(span, np.float)
  edges = np.linspace(span[0], span[1], bins)
  hist = np.zeros([bins])
  for i in range(bins - 1):
    hist[i] = np.count_nonzero(np.logical_and(
      img >= edges[i], img < edges[i + 1]))
  hist[-1] = np.count_nonzero(img >= edges[-1])

  if density:
    hist /= hist.sum()

  if title is not None:
    pp.bar(edges[:-1], hist)
    pp.title(title)

  return hist, edges
```

Preverimo, ali porazdelitev šuma ustreza normalni (slika 3.8).



(a) Področje 19 (b) Področje 24

Slika 3.8: Primerjava porazdelitve sivinskih vrednosti z normalno porazdelitvijo.

```
pp.figure()

hA, eA = funkcije.imHistogram(
  sCCg[18], span=[0,255], density=True)
hB, eB = funkcije.imHistogram(
  sCCg[23], span=[0,255], density=True)

pp.subplot(1, 2, 1)
pp.bar(eA, hA, width=eA[1]-eA[0])
pp.plot(eA, 1.0/(sCCstd[18]*np.sqrt(2.0*np.pi))* \
  np.exp(-(eA - sCCmean[18])**2/2.0/sCCstd[18]**2), '-r')
pp.title('Področje 19')

pp.subplot(1, 2, 2)
pp.bar(eB, hB, width=eB[1]-eB[0])
pp.plot(eB, 1.0/(sCCstd[23]*np.sqrt(2.0*np.pi))* \
  np.exp(-(eB - sCCmean[23])**2/2.0/sCCstd[23]**2), '-r')
pp.title('Področje 24')

pp.show()
```

6. Določimo diferencialni razmerji signal-šum za izbrana področja.

```
SNRda = (sCCmean[18] - sCCmean[23])/ \
        (sCCstd[18]**2 + sCCstd[23]**2)**0.5

SNRdb = (sCCmean[22] - sCCmean[23])/ \
        (sCCstd[22]**2 + sCCstd[23]**2)**0.5

print('SNRd za področji 19-24: {:.1f}'.format(SNRda))
print('SNRd za področji 23-24: {:.1f}'.format(SNRdb))
```

Diferencialno razmerje signal-šum za področji 19 in 24 znaša 42,5, za področji 23 in 24 pa 5,3.

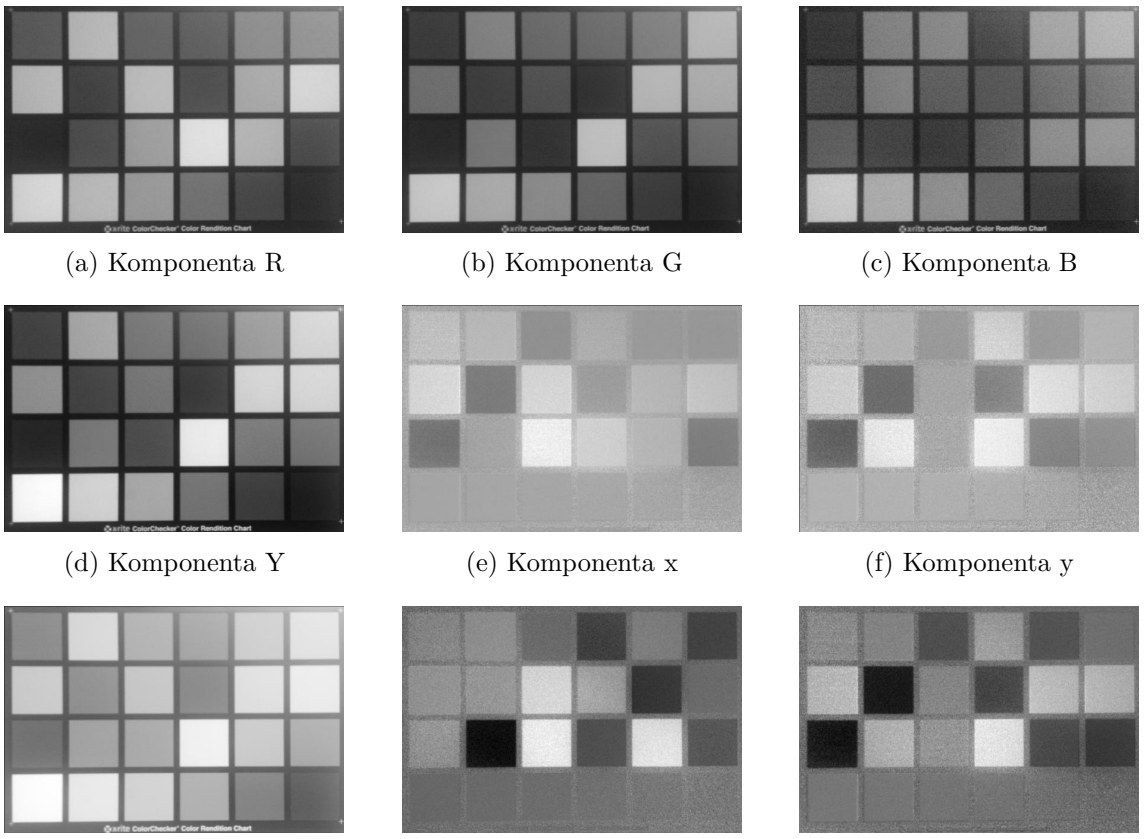
7. (a) Izvedemo preslikavo iz RGB v XYZ barvnimi prostor (slika 3.10).

```
R, G, B = CC[:,:,0], CC[:,:,1], CC[:,:,2]
rgb = np.vstack((R.flatten(),
                 G.flatten(),
                 B.flatten()))

xyz = np.dot(RGB2XYZ, rgb/255.0)
Xw, Yw, Zw = np.dot(RGB2XYZ, np.ones([3, 1]))
X = xyz[0].reshape(H, W)
Y = xyz[1].reshape(H, W)
Z = xyz[2].reshape(H, W)
x, y, z = X/(X + Y + Z), Y/(X + Y + Z), Z/(X + Y + Z)
XYZ = np.dstack((X, Y, Z))

# (a) Pretvorba iz RGB v XYZ.
pp.figure()

pp.subplot(1, 3, 1)
pp.imshow(Y, cmap='gray')
pp.title('Y')
pp.axis('off')

pp.subplot(1, 3, 2)
pp.imshow(x, cmap='gray')
pp.title('x')
pp.axis('off')

pp.subplot(1, 3, 3)
pp.imshow(y, cmap='gray')
pp.title('y')
pp.axis('off')

pp.show()
```

(b) Izvedemo preslikavo iz XYZ v Lab barvnimi prostor (slika 3.10).

```
L, a, b = Xyz2Lab(X, Y, Z, Xw, Yw, Zw)
Lab = np.dstack((L, a, b))

pp.figure()

pp.subplot(1, 3, 1)
pp.imshow(L, cmap='gray')
pp.title('L')
pp.axis('off')

```

```
11 pp.subplot(1, 3, 2)
12 pp.imshow(a, cmap='gray')
13 pp.title('a')
14 pp.axis('off')
15
16 pp.subplot(1, 3, 3)
17 pp.imshow(b, cmap='gray')
18 pp.title('b')
19 pp.axis('off')
20
21 pp.show()
```



(a) Komponenta R (b) Komponenta G (c) Komponenta B

(d) Komponenta Y (e) Komponenta x (f) Komponenta y

(g) Komponenta L (h) Komponenta a (i) Komponenta b

Slika 3.9: Komponente barvnih prostorov RGB, xyY in Lab.

8. (a) V modulu `funkcije` ustvarimo funkcijo `ssim`.

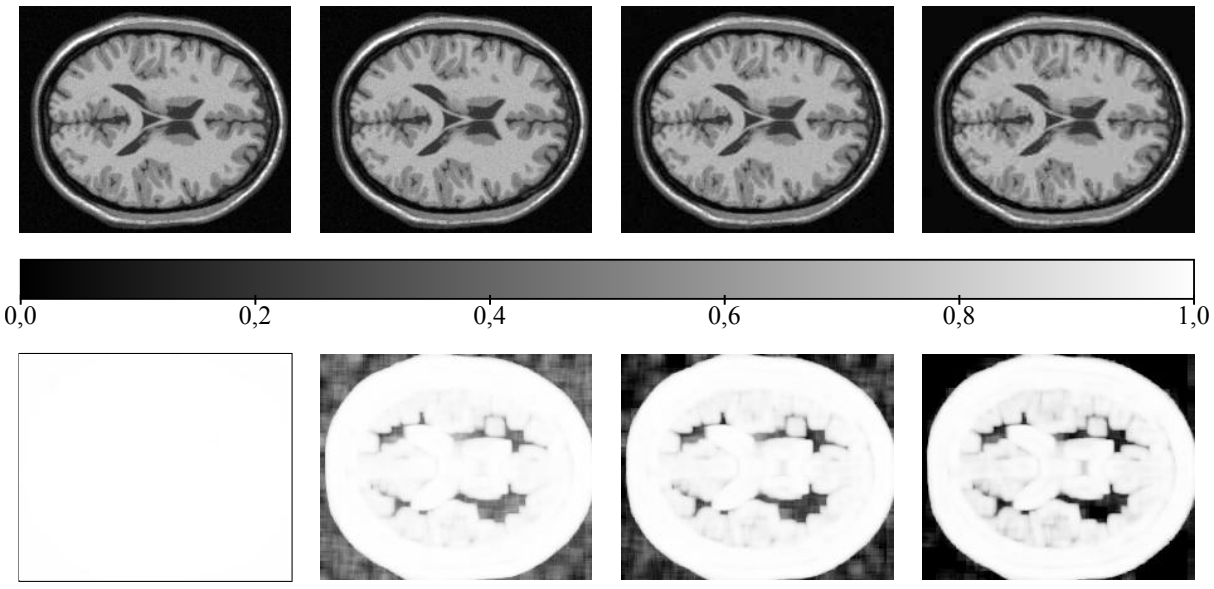
```
1 def ssim(imgx, imgy, l=255, n=11, sigma=1.5,
2          k1=0.01, k2=0.03):
3   imgx = np.asarray(imgx, dtype=np.float)
4   imgy = np.asarray(imgy, dtype=np.float)
```

```
5
6   n_half = int(n//2)
7   n = n_half*2 + 1
8   if imgx.shape != imgy.shape:
9     raise ValueError('Velikost vhodnih slik "imgx" in "imgy" '
10                    'morata biti enaki!')
11
12  x = y = np.arange(-n_half, n_half + 1, dtype=np.float)
13  Y, X = np.meshgrid(y, x, indexing='ij')
14  W = 1.0/(2.0*np.pi*sigma**2)* \
15      np.exp(-(X**2 + Y**2)/(2.0*sigma**2))
16  W *= 1.0/W.sum()
17
18  imgxp = imPad2d(imgx, n_half, boundary='reflect')
19  imgyp = imPad2d(imgy, n_half, boundary='reflect')
20
21  c1 = (k1*l)**2
22  c2 = (k2*l)**2
23
24  H, W = imgx.shape
25  ossim = np.zeros(imgx.shape)
26  for i in range(W):
27    for j in range(H):
28      xr = W*imgxp[j: j + n, i: i + n]
29      yr = W*imgyp[j: j + n, i: i + n]
30
31      mxr = xr.mean()
32      myr = yr.mean()
33      sxr = xr.std()
34      syr = yr.std()
35      sxyr = (xr*yr).mean() - mxr*myr
36
37      ossim[j, i] = (2.0*mxr*myr + c1)*(2.0*sxyr + c2)/ \
38                    ((mxr**2 + myr**2 + c1)*(sxr**2 + syr**2
                      +c2))
39
40  return ossim
```

(b) Izračunamo SSIM slike pri izgubnem JPEG zgoščevanju za vrednosti parametra kakovosti 25, 50, 75 in 100 (izvirna slika). Vrednost SSIM se nahaja na intervalu $[0, 1]$. Vrednost 1 dobimo zgolj, ko se sliki ali izbrani okolici slikovnega elementa slik povsem ujemata (brezizgubno zgoščevanje, ko je vrednost parametra kakovosti 100). V splošnem vrednost SSIM sledi kakovosti zgoščevanja.

```
1 imga = im.open('./poglavje_1/mrBrainSlice.png')
2
3 quality = [25, 50, 75, 100]
4
5 pp.figure()
```

```
6
7  jpegs = []
8  ssims = []
9  tmpfile = './poglavje_3/rezultati/tmp.jpeg'
10 for index, q in enumerate(quality):
11   imga.save(tmpfile, quality=q)
12   jpegs.append(np.array(im.open(tmpfile)))
13
14   pp.subplot(1, len(quality) + 1, 1 + index)
15   ssims.append(funkcije.ssim(imga, jpegs[-1]))
16   pp.imshow(ssims[-1], cmap='gray', vmin=0, vmax=1)
17   pp.title('SSIM pri kakovosti {:d} %'.format(q))
18
19 pp.subplot(1, len(quality) + 1, len(quality) + 1)
20 pp.colorbar()
21 pp.axis('off')
22 pp.show()
```



(a) Kakovost 100 (b) Kakovost 75 (c) Kakovost 50 (d) Kakovost 25

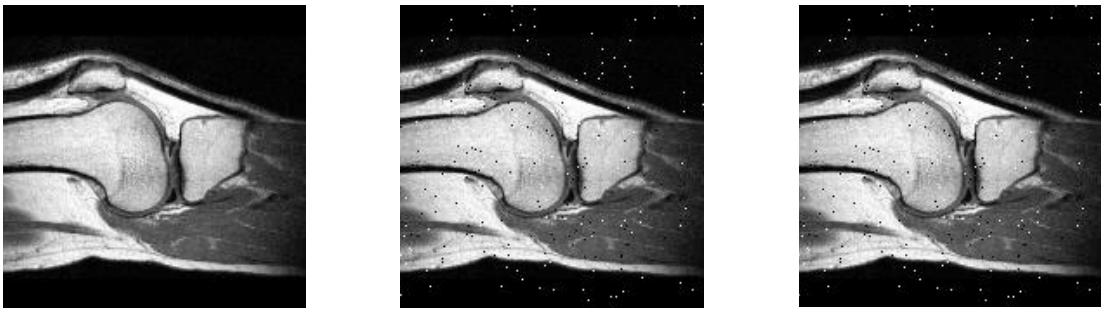
Slika 3.10: Izgubno JPEG zgoščevanje slike mrBrainSlice.png za vrednosti parametra kakovosti 100 (izvirna slika), 75, 50 in 25 (prva vrstica) ter pripadajoče vrednosti metrike SSIM med izvirno in zgoščeno sliko (druga vrstica).

Poglavje 4

Filtriranje slik

Poglavje je namenjena spoznavanju in razumevanju osnovnih postopkov filtriranja slik. Postopke linearnega filtriranja 2D slik je mogoče predstaviti s konvolucijo slike I velikosti (H, W) z izbranim konvolucijskim jedrom K velikosti (A, B):

$$S[i,j] = I * K = \sum_{k=0}^{A-1} \sum_{l=0}^{B-1} I[i-(k-c_1, j-(l-c_2)] \cdot K[k,l]. \tag{4.1}$$



Slika 4.1: Slike, ki jih boste uporabljali tekom te vaje. Od leve proti desni: slika ct_175x175-_uint8.raw, slika ct_sp_175x175_uint8.raw ter slika mr_217x181x181_uint8 v xy ravnini pri $z = 90$.

4.1 Naloge in vprašanja

1. Konstanti c_1 in c_2, ki določata središče konvolucijskega jedra, naj bosta definirani kot $\lfloor \frac{A}{2} \rfloor$ ter $\lfloor \frac{B}{2} \rfloor$. Z opisano konvolucijo smo se že srečali v poglavju 2, kjer smo postopek 2D konvolucije udejanili v obliki funkcije `conv2d`. S pomočjo funkcije `conv2d` izvedite glajenje slik z navadnim povprečenjem, z uteženim povprečenjem ter z Gaussovim jedrom.

   Primerjajte in komentirajte rezultate glajenja slike ct_175x175_uint8.raw s podanimi jedri.

$$\frac{1}{9}\cdot\begin{bmatrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{bmatrix} \qquad \frac{1}{16}\cdot\begin{bmatrix}1 & 2 & 1\\ 2 & 4 & 2\\ 1 & 2 & 1\end{bmatrix} \qquad \begin{bmatrix}0{,}01 & 0{,}08 & 0{,}01\\ 0{,}08 & 0{,}64 & 0{,}08\\ 0{,}01 & 0{,}08 & 0{,}01\end{bmatrix}$$

(a) Navadno povprečje (b) Uteženo povprečenje (c) Gaussovo sito

Slika 4.2: Primeri konvolucijskih jeder velikosti 3×3 slikovne elemente.

2. Ustvarite funkcijo `gaussianKernel2d` za izračun konvolucijskega jedra v obliki 2D simetrične Gaussove funkcije, kjer je `sigma` standardna deviacija σ 2D simetrične Gaussove funkcije, ki je definirana kot:

$$K(u,v) = \frac{1}{2\pi\sigma^2}\exp^{-\frac{(u^2+v^2)}{2\sigma^2}}. \tag{4.2}$$

```
1 def gaussianKernel2d(sigma, truncate=4):
2   ...
3   return kernel
```

Velikost konvolucijskega jedra $K(u,v)$ naj bo določena z vrednostmi parametrov `sigma` in `truncate` kot $2\cdot\lceil truncate\cdot\sigma\rceil+1$, vrednost $K(0,0)$ pa naj se nahaja v središčnem elementu izhodnega 2D polja `kernel`. Zagotovite, da bo vsota vseh elementov konvolucijskega jedra `kernel` enaka 1.

(a) Preizkusite delovanje funkcije za poljubne nenegativne vrednosti `sigma` in prikažite konvolucijsko jedro kot sliko. Obrazložite vpliv vrednosti `sigma` na obliko konvolucijskega jedra `kernel`.

(b) Kaj se tekom glajenja zgodi z zunanjim robom slike in kako širok je ta rob? Kako bi se lahko izognili ali ublažili opažene spremembe?

(c) Dopolnite funkcijo `conv2d` tako, da bo tretji parameter `boundary`, določal način obravnave sivinskih vrednosti izven definicijskega območja slike, četrti parameter `fillvalue` pa določal sivinsko vrednost izven definicijskega območja slike, ko je vrednost parametra `boundary` enaka `'constant'`. Sledeči primer ilustrira učinke različnih vrednosti parametra `boundary`:

`boundary`	Razširitev			Definicijsko območje								Razširitev		
`'mirror'`	4	3	2	1	2	3	4	5	6	7	8	7	6	5
`'reflect'`	3	2	1	1	2	3	4	5	6	7	8	8	7	6
`'nearest'`	1	1	1	1	2	3	4	5	6	7	8	8	8	8
`'constant'`	0	0	0	1	2	3	4	5	6	7	8	0	0	0
`'wrap'`	6	7	8	1	2	3	4	5	6	7	8	1	2	3

V ta namen ustvarite funkcijo `imPad2d`, ki na zahtevani način razširi definicijsko območje slike v navpični smeri za `n[0]` in v vodoravni smeri za `n[1]` slikovnih elementov. V pomoč vam bo funkcija `pad` modula `numpy`.

```
def imPad2d(img, n, boundary='constant', fillvalue=0):
  ...
  return oimg
```

```
def conv2d(img, kernel, boundary='reflect', fillvalue=0):
  ...
  return oimg
```

(d) Kako širok mora biti razširjeni rob slike, da pri filtriranju ublažimo neželene učinke, ki ste jih opazili pod točko (b)?

3. 2D konvolucijo slike z Gassovim jedrom je mogoče razbiti na dve zaporedni 1D konvoluciji, ki potekata vzdolž vrstic in stolpcev slike. Ustvarite funkcijo `conv1d`, ki bo izračunala 1D konvolucijo signala I s konvolucijskim jedrom K dolžine a. Parametra `boundary` in `fillvalue` naj imata enak pomen kot pri funkciji `conv2d`:

$$S(i) = \sum_{k=0}^{a-1} I(i - (k - c)) \cdot K(k). \tag{4.3}$$

```
def conv1d(data, kernel, boundary='constant', fillvalue=0):
  ...
  return odata
```

Predpostavite, da je središče c konvolucijskega jedra K pri $\lfloor \frac{a}{2} \rfloor$.

4. Ustvarite še funkcijo `gaussianKernel1d` za izračun konvolucijskega jedra v obliki 1D Gaussove funkcije, kjer je `sigma` standardna deviacija σ Gaussove funkcije, ki je definirana kot:

$$K(u) = \frac{1}{\sigma\sqrt{2\pi}} \exp^{-\frac{u^2}{2\sigma^2}}. \tag{4.4}$$

Velikost konvolucijskega jedra $K(u)$ naj bo določena z vrednostmi parametrov `sigma` in `truncate` kot $2 \cdot \lceil truncate\sigma \rceil + 1$, vrednost $K(0)$ pa naj se nahaja v središčnem elementu izhodnega 2D polja `kernel`. Zagotovite, da bo vsota vseh elementov konvolucijskega jedra `kernel` enaka 1.

```
def gaussianKernel1d(sigma, truncate=4):
  ...
  return kernel
```

Kaj je glavna prednost filtriranja slik s postopkom 1D konvolucije?

5. Ustvarite funkcijo `imGaussFilt2d`, ki bo filtrirala vhodno sliko `img` z Gaussovim jedrom standardne deviacije `sigma`. Jedro filtra izračunajte s funkcijo `gaussianKernel1d`, konvolucijo pa izvedite z uporabo funkcije `conv1D`. Primerjajte rezultate filtriranja slike s funkcijama `conv1d` ter `conv2d`. Parametra `boundary` in `fillvalue` naj imata enak pomen kot pri funkciji `conv2d`.

```
def imGaussFilt2d(img, sigma, boundary='constant', fillvalue=0):
  ...
  return oimg
```

6. Ostrenje slike je analogno prostorskemu odvajanju sivinskih vrednosti. Odvajanje lahko izvedemo s pomočjo Laplaceovega operatorja (drugi odvod):

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}.$$

Ostrenje slik pa izvedemo tako, da od vhodne slike $I(x, y)$ odštejemo uteženo sliko drugega odvoda $\nabla^2 I(x, y)$, ki ga izračunamo z Laplaceovim operatorjem:

$$S(x, y) = I(x, y) - c \cdot \left(\nabla^2 I(x, y)\right), \tag{4.5}$$

kjer konstanta c določa stopnjo ostrenja. Pogosto se uporablja tudi maskiranje neostrih področij, pri čemer od vhodne slike $I(x, y)$ najprej odštejemo njeno zglajeno različico $I(x, y) * K$ ter tako dobimo sliko maske $M(x, y)$, ki jo prištejemo vhodni sliki skladno s stopnjo ostrenja c:

$$\begin{aligned} M(x, y) &= I(x, y) - I(x, y) * K, \\ S(x, y) &= I(x, y) + c \cdot M(x, y). \end{aligned} \tag{4.6}$$

Ustvarite funkcijo `imSharpen2D`, ki bo glede na vrednost parametra `kind` sliko izostrila z Laplaceovim operatorjem (`'laplace'`) ali z maskiranjem (`'mask'`). Pri ostrenju z maskiranjem uporabite glajenje z Gaussovim jedrom standardne deviacije `sigma`. Za glajenje uporabite funkcijo `imGaussFilt2d`, za izračun Laplaceovega operatorja pa uporabite funkcijo `conv2d`. Definicijsko območje slike pred izvajanjem filtriranja ustrezno razširite s funkcijo `imPad2d`.

```
def imSharpen2D(img, kind='mask', c=1.0, sigma=1.0):
  ...
  return oimg
```

(a) Izostrite sivinsko sliko ct_175x175_uint8.raw s postopkom na podlagi Laplaceovega operatorja in z maskiranjem neostrih področij. Za stopnjo ostrenja izberite vrednost $c = 1$ ter uporabite glajenje z Gaussovim filtrom $\sigma = 1$.

(b) Preizkusite različne vrednosti c in obrazložite njihov vpliv na izostreno sivinsko sliko.

7. Statistično filtriranje na podlagi mediane se uporablja pri nesimetričnih porazdelitvah sivinskih vrednosti, še posebej kadar imamo opravka z bipolarnim šumom tipa sol in poper. Mediana urejenega niza n vrednosti je definirana kot:

$$\text{median}(z_1, z_2, z_3, ..., z_n) = \begin{cases} z_{(n+1)/2}, & n \text{ je liho število}, \\ \frac{1}{2}(z_{n/2} + z_{n/2+1}), & n \text{ je sodo število}. \end{cases} \tag{4.7}$$

Na podoben način je mogoče definirati še filter maksimalne in minimalne vrednosti. Ustvarite funkcijo `imStatFilt2d`, ki bo filtrirala vhodno sivinsko sliko `img` z nelinearnim filtrom tipa `kind` (`'median'`, `'min'` ali `'max'`). Predpostavite kvadratno jedro lihe velikosti. Za iskanje največje in najmanjše vrednosti niza števil uporabite funkciji `min` in `max`, za iskanje mediane pa funkcijo `median` modula `numpy`. Predpostavite, da je vrednost sivin na področjih, kjer slika ni definirana, enaka sivini najbližjega slikovnega elementa. Namig: sliko `img` pred filtriranjem ustrezno razširite s funkcijo `imPad2d` tako, da postavite vrednost parametra `pad` na `'nearest'`.

```
def imStatFilt2d(img, n=3, kind='median'):
  ...
  return oimg
```

Primerjajte statistično filtriranje z mediano `n=3` ter Gassovim jedrom `sigma=0.5` na sliki ct_sp_175x175_uint8.raw ter komentirajte rezultate.

8. Ustvarite konvolucijsko jedro velikosti 3×3, ki vrednost slikovnega elementa nadomesti s povprečno vrednostjo 8-ih sosednjih slikovnih elementom. Uporabite konvolucijsko jedro na sliki ct_sp_175x175_uint8.raw. Kakšna je razlika med filtriranjem z opisanim konvolucijskim jedrom in statističnim filtriranjem na podlagi mediane?

9. Filtriranje 3D slik z Gaussovim jedrom je mogoče izvesti s postopkom 3D konvolucije ali z zaporednim filtriranjem po prvi, drugi in tretji razsežnosti slike z 1D Gaussovim jedrom. Ustvarite funkcijo `imGaussFilt3d`, ki bo filtrirala vhodno 3D sliko `img` z Gaussovim filtrom standardne deviacije `sigma`. Jedro filtra ustvarite s funkcijo `gaussianKernel1d`, filtriranje pa izvedite s funkcijo `conv1d`. Parametra `boundary` in `fillvalue` naj imata enak pomen kot pri funkciji `conv2d`.

```
def imGaussFilt3d(img, sigma, boundary='constant', fillvalue=0):
  ...
  return oimg
```

Filtrirajte 3D sliko mr_217x181x181_uint8.raw iz poglavja 2 s konvolucijskim jedrom `sigma=0.5` ter prikažite prereza $z = 90$ ter $x = 90$ pred in po filtriranju.

10. Računsko učinkovite funkcije za filtriranje večrazsežnih slik najdemo v knjižnici `scipy.ndimage`. Raziščite in uporabite funkcije `convolve`, `gaussian_filter`, `median_filter` ter `laplace`.

4.2 Rešitve in odgovori na vprašanja

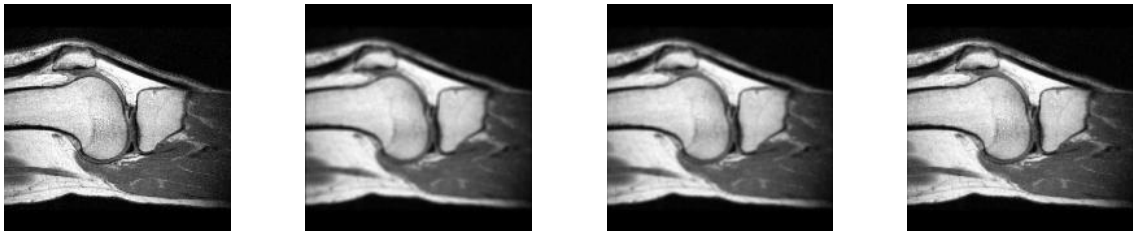
Pri odgovorih privzamemo, da se funkcije iz rešitev nahajajo v modulu `funkcije`, slikovno gradivo pa se nahaja v podmapi `poglavje_4`. Naprej uvozimo potrebne module, naložimo slike in definiramo pomožne spremenljivke.

```
import numpy as np
from matplotlib import pyplot as pp
from PIL import Image as im
import funkcije

Knp = 1.0/9.0*np.array([[1, 1, 1],
                        [1, 1, 1],
                        [1, 1, 1]])
Kup = 1.0/16.0*np.array([[1, 2, 1],
                         [2, 4, 2],
                         [1, 2, 1]])
Kg = np.array([[0.01, 0.08, 0.01],
               [0.08, 0.64, 0.08],
               [0.01, 0.08, 0.01]])
I1 = funkcije.imLoadRaw2d(
  './poglavje_4/ct_175x175_uint8.raw', 175, 175)
I2 = funkcije.imLoadRaw2d(
  './poglavje_4/ct_sp_175x175_uint8.raw', 175, 175)
I3 = funkcije.imLoadRaw3d(
  './poglavje_4/mr_217x181x181_uint8.raw', 217, 181, 181)
pp.ioff()
```

1. Primerjava glajenja s tremi konvolucijskimi jedri (slika 4.3).

```
# Izvedemo glajenje s predlaganimi jedri.
I1np = np.round(funkcije.conv2d(I1, Knp)).astype(np.uint8)
I1up = np.round(funkcije.conv2d(I1, Kup)).astype(np.uint8)
I1g = np.round(funkcije.conv2d(I1, Kg)).astype(np.uint8)

# Prikažemo izvirno in vse tri zglajene slike.
pp.figure()
pp.suptitle('Primerjava treh konvolucijskih jeder sit.')

pp.subplot(1, 4, 1)
pp.title('Izvirna slika')
pp.imshow(I1, cmap='gray')
pp.axis('off')

pp.subplot(1, 4, 2)
pp.imshow(I1np, cmap='gray')
pp.title('Navadno povprecje')
pp.axis('off')

```

```
20 pp.subplot(1, 4, 3)
21 pp.imshow(I1up, cmap='gray')
22 pp.title('Uteženo povprečje')
23 pp.axis('off')
24
25 pp.subplot(1, 4, 4)
26 pp.imshow(I1g, cmap='gray')
27 pp.title('Gaussovo jedro')
28 pp.axis('off')
29
30 pp.show()
```



(a) Izvirna slika (b) Navadno povprečje (c) Uteženo povprečje (d) Gaussovo jedro

Slika 4.3: Glajenje z različnimi konvolucijskimi jedri.

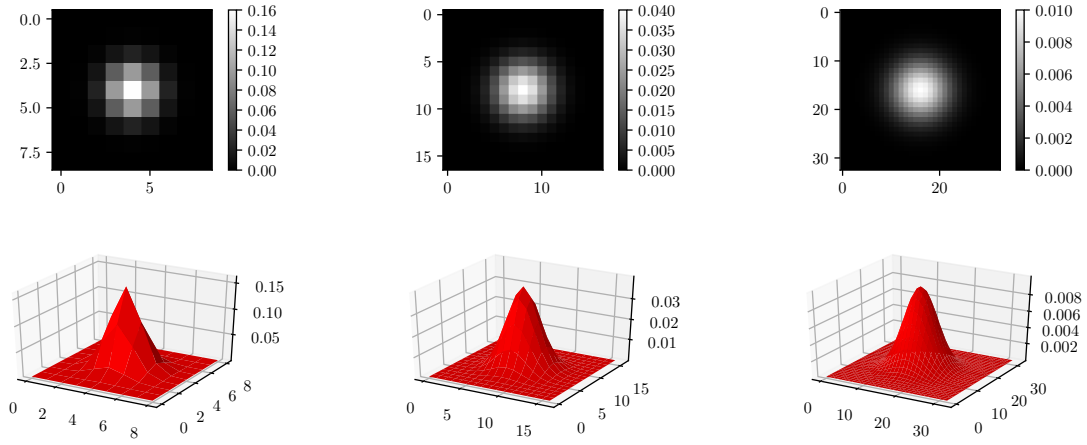
2. V modulu `funkcije` ustvarimo funkcijo `gaussianKernel2d`.

```
1  def gaussianKernel2d(sigma, truncate=4):
2    n = int(2*np.ceil(truncate*sigma) + 1)
3    xy = np.arange(n)
4    xy -= xy.mean()
5    Y, X = np.meshgrid(xy, xy, indexing='ij')
6    kernel = 1.0/(2.0*np.pi*sigma**2)* \
7      np.exp(-(X**2 + Y**2)*(1.0/(2.0*sigma**2)))
8    kernel /= kernel.sum()
9
10   return kernel
```

(a) Ustvarimo in primerjamo dve Gaussovi konvolucijski jedri (slika 4.4) s `sigma=0.5` ter `sigma=1`. Z večanjem vrednosti parametra `sigma` se veča velikost jedra in stopnja glajenja.

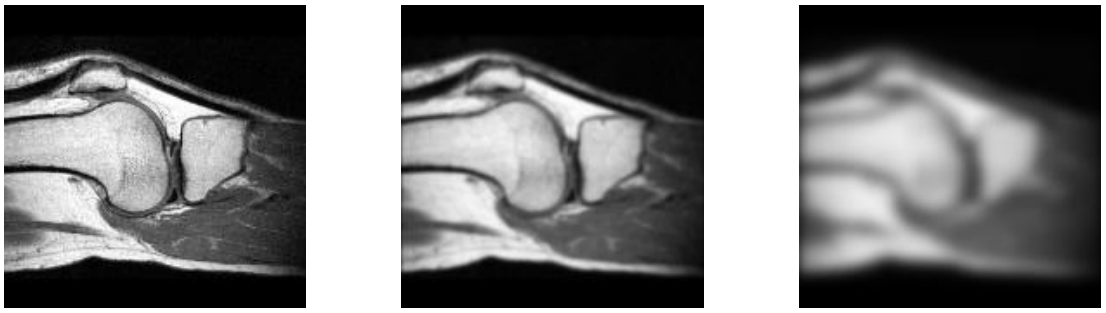
```
1 Kg1 = funkcije.gaussianKernel2d(1)
2 Kg2 = funkcije.gaussianKernel2d(2)
3 I1Kg1 = funkcije.conv2d(I1, Kg1)
4 I1Kg2 = funkcije.conv2d(I1, Kg2)
5
6 # izriše 3D površino jedra v izbrano podokno
7 def surf(
8     gk, subplot,
```

```
9       color='r', rstride=1, cstride=1, shade=True):
10      from mpl_toolkits.mplot3d import Axes3D
11    ax = pp.gcf().add_subplot(subplot, projection='3d')
12    x, y = np.arange(gk.shape[0]), np.arange(gk.shape[1])
13    x = x - x.mean()
14    y = y - x.mean()
15    Yk, Xk = np.meshgrid(y, x, indexing='ij')
16    ax.plot_surface(
17      Xk, Yk, gk,
18      color=color,
19      rstride=rstride, cstride=cstride, shade=shade)
20
21  pp.figure()
22  pp.suptitle('Gaussova jedra za vrednosti sigma 1, 2 in 4')
23
24  pp.subplot(2, 3, 1)
25  pp.imshow(Kg1, cmap='gray')
26  pp.title('sigma = 1, velikost={}'.format(Kg1.shape))
27
28  pp.subplot(2, 3, 2)
29  pp.imshow(Kg2, cmap='gray')
30  pp.title('sigma = 2, velikost={}'.format(Kg2.shape))
31
32  pp.subplot(2, 3, 3)
33  pp.imshow(Kg4, cmap='gray')
34  pp.title('sigma = 4, velikost={}'.format(Kg4.shape))
35
36  surf(Kg1, 234)
37  surf(Kg2, 235)
38  surf(Kg4, 236)
39
40  pp.figure()
41  pp.suptitle(
42    'Nezveznosti na robu slike '
43    'po konvoluciji z Gaussovim jedrom.')
44
45  pp.subplot(1, 3, 1)
46  pp.imshow(I1, cmap='gray')
47  pp.title('Izvirna slika')
48
49  pp.subplot(1, 3, 2)
50  pp.imshow(I1Kg1, cmap='gray')
51  pp.title('Filtrirana  z sigma=1')
52
53  pp.subplot(1, 3, 3)
54  pp.imshow(I1Kg2, cmap='gray')
55  pp.title('Filtrirana z sigma=2')
56
57  pp.show()
```



(a) $\sigma = 1$ (b) $\sigma = 2$ (c) $\sigma = 4$

Slika 4.4: Primerjava treh 2D Gaussovih konvolucijskih jeder.



(a) Izvirna slika (b) Jedro za $\sigma = 1$ (c) Jedro za $\sigma = 4$

Slika 4.5: Nezveznosti na robu slike po konvoluciji z Gaussovim jedrom.

(b) S tem, ko postavimo vrednosti sivinskih elementov izven definicijskega območja slike na 0, vnašamo nezveznosti v zunanji rob slike, in sicer v širini polovice konvolucijskega jedra (slika 4.5). Izboljšanje bi lahko dosegli z uporabo sivinske vrednosti najbližjega slikovnega elementa iz definicijskega območja slike namesto vrednosti 0.

(c) V modulu `funkcije` ustvarimo funkcijo `imPad2d`.

```
1  def imPad2d(img, n, boundary='constant', fillvalue=0):
2                boundary = str(boundary).lower()
3    # primer kode za razširitev s konstantno ali najbližjo vrednostjo
4    '''
5    if isinstance(n, int)
6      py, px = n, n
7    else:
8      py, px = n[0], n[1]
9
```

```
  H, W = img.shape[0], img.shape[1]
  Hp, Wp = H + 2*py, W + 2*px
  if boundary == 'nearest':
    oimg = np.zeros([Hp, Wp], dtype=img.dtype)
    oimg[-py:, px:-px] = img[-1, :]
    oimg[py:-py, :px] = img[:,0].reshape(H, 1)
    oimg[py:-py:, -px:] = img[:,-1].reshape(H, 1)

    oimg[:py, :px]=img[0,0]
    oimg[:py, -px:]=img[0, -1]
    oimg[-py:, :px]=img[-1,0]
    oimg[-py:, -px:]=img[-1,-1]
  elif boundary == 'constant':
    oimg = np.tile(img.dtype.type(value), [Hp, Wp])

  oimg[py:-py, px:-px] = img
  return oimg
  '''
  # učinkovitejša rešitev s funkcijo pad modula numpy
  if boundary == 'constant':
      return np.pad(
        img, n, mode=boundary, constant_values=fillvalue)

  elif boundary in ['reflect', 'wrap']:
      return np.pad(img, n, mode=boundary)

  elif boundary == 'nearest':
      return np.pad(img, n, mode='edge')

  elif boundary == 'mirror':
      return np.pad(img, n, mode='symmetric')

  else:
    raise ValueError(
      'Vrednost parametra "mode" je lahko '
      '"constant", "reflect", "nearest" ali "mirror"!')
```

V modulu `funkcije` preimenujemo funkcijo `conv2d` v `_conv2d` in ustvarimo novo funkcijo `conv2d`, ki ustrezno uporabi `_conv2d` in `imPad2d`.

```
def conv2d(data, kernel, boundary='constant', fillvalue=0):
  n = np.floor(np.array(kernel.shape)/2.0).astype(np.int)

  pdata = imPad2d(data, n, boundary, fillvalue)
  odata = _conv2d(pdata, kernel)

  return odata[n[0]:-n[0], n[1]:-n[1]]
```

(d) Razširjeni del slike mora znašati polovico velikosti konvolucijskega jedra n, in sicer $\left\lfloor \frac{n-1}{2} \right\rfloor$.

3. V modulu `funkcije` ustvarimo funkcijo `conv1d`.

```
def conv1d(data, kernel, boundary='constant', fillvalue=0):
  a = len(kernel)
  c = p = int(np.floor(a/2))
  N = data.size
  Np = N + 2*p
  pdata = np.zeros([Np])
  pdata[p:-p] = data

  if boundary == 'nearest':
    pdata[:p] = data[0]
    pdata[-p:] = data[-1]

  else:
    pdata[:p] = fillvalue
    pdata[-p:] = fillvalue

  odata = np.zeros([N])
  for i in range(N):
    for j in range(a):
      odata[i] += kernel[j]*pdata[p + i - (j - c)]

  return odata.reshape(data.shape)
```

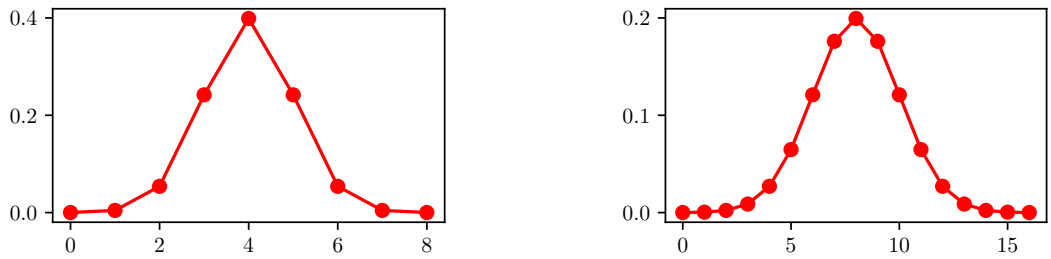
4. V modulu `funkcije` ustvarimo funkcijo `gaussianKernel1d`.

```
def gaussianKernel1d(sigma, truncate=4):
  n = int(2*np.ceil(truncate*sigma) + 1)
  x = np.arange(n)
  x -= x.mean()
  kernel = 1.0/(sigma*np.sqrt(2.0*np.pi))* \
    np.exp(-x**2*(1.0/(2.0*sigma**2)))
  kernel /= kernel.sum()

  return kernel
```

Glavna prednost filtriranja slik z dvema zaporednima 1D konvolucijama pred enim filtriranjem z 2D konvolucijo je v numerični zahtevnosti. V prvem primeru zahtevnost raste linearno z velikostjo konvolucijskega jedra, v drugem primeru pa s kvadratom velikosti konvolucijskega jedra.

5. V modulu `funkcije` ustvarimo funkcijo `imGaussFilt2D`.

```
def imGaussFilt2d(img, sigma, boundary='constant', fillvalue=0):
  H, W = img.shape[0], img.shape[1]
  K = gaussianKernel1d(sigma)
  oimg = np.zeros(img.shape)

```



(a) Jedro dolžine 7, $\sigma = 1$

(b) Jedro dolžine 13, $\sigma = 2$

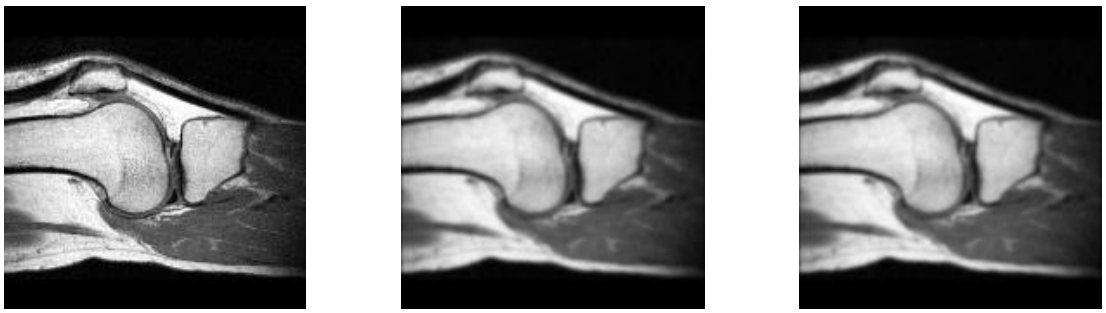
Slika 4.6: Primerjava dveh 1D Gaussovih konvolucijskih jeder.

```
6   # Filtriramo po vrsticah slike.
7   for i in range(H):
8     oimg[i, :] = conv1d(img[i, :], K, boundary, fillvalue)
9
10  # Filtriramo po stolpcih slike.
11  for i in range(W):
12    oimg[:, i] = conv1d(oimg[:, i], K, boundary, fillvalue)
13
14  return oimg
```

Primerjavo rezultatov filtriranja s funkcijama `conv1d` in `conv2d` prikazuje slika 4.7.

```
1  I1f = I1.astype(np.float)
2  I1C2d = funkcije.conv2d(I1f, Kg1, boundary='constant')
3  I1C1d = np.zeros(I1.shape)
4  k = funkcije.gaussianKernel1d(1.0)
5  for i in range(I1.shape[0]):
6    I1C1d[i, :] = funkcije.conv1d(
7      I1f[i, :], k, boundary='constant')
8  for i in range(I1.shape[1]):
9    I1C1d[:, i] = funkcije.conv1d(
10     I1C1d[:, i], k, boundary='constant')
11
12 pp.figure()
13 pp.suptitle('Primerjava 2D in 2 x 1D konvolucije')
14
15 pp.subplot(1, 3, 1)
16 pp.imshow(I1C1d, cmap='gray')
17 pp.title('2 x 1D konvolucija')
18 pp.axis('off')
19
20 pp.subplot(1, 3, 2)
21 pp.imshow(I1C2d, cmap='gray')
22 pp.title('2D konvolucija')
23 pp.axis('off')
```

```
24
25 pp.subplot(1, 3, 3)
26 pp.imshow(I1C2d - I1C1d, cmap='gray')
27 pp.title('Razlika slik')
28 pp.axis('off')
29
30 pp.show()
```



(a) Izvirna slika (b) 2D konvolucija, $\sigma = 1$ (c) 2 x 1D konvolucija, $\sigma = 1$

Slika 4.7: Primerjava filtriranja med 2D konvolucijo in dvakratno zaporedno 1D konvolucijo.

6. V modulu `funkcije` ustvarimo funkcijo `imSharpen2d`.

```
1  def imSharpen2d(img, kind='mask', c=1.0, sigma=1.0):
2    img = img.astype(np.float64)
3
4    if kind == 'mask':
5      M = img - imGaussFilt2d(img, sigma, 'nearest')
6      oimg = img + c*M
7
8    elif kind == 'laplace':
9      Lk = np.array([[0, 1, 0],[1, -4, 1],[0, 1, 0]], np.float64)
10     oimg = img - c*conv2d(img, Lk, 'nearest')
11
12   else:
13     raise ValueError(
14       'Vrednost parametra kind je lahko'
15       ' "mask" ali "laplace"!')
16   return oimg
```

(a) Primerjavo med ostrenjem slike z maskiranjem neostrih področij in Laplaceovim operatorjem prikazuje slika 4.8.

```
1 I1sm = funkcije.imSharpen2d(I1, 'mask', c=1.0, sigma=1.0)
2 I1sm = np.clip(I1sm, 0, 255)
3 I1sl = funkcije.imSharpen2d(I1, 'laplace', c=1.0)
4 I1sl = np.clip(I1sl, 0, 255)
5
```

(a) Izvirna slika (b) Ostrenje z maskiranjem (c) Ostrenje z Laplaceom

Slika 4.8: Primerjava med ostrenjem slike z maskiranjem neostrih področij in Laplaceovim operatorjem pri $\sigma = 1$ ter $c = 1$.

```
6  pp.figure()
7  pp.suptitle(
8    'Primerjava ostrenja z masko in '
9    'Laplaceovim operatorjem')
10
11 pp.subplot(1, 3, 1)
12 pp.imshow(I1, cmap='gray')
13 pp.title('Izvirna slika')
14 pp.axis('off')
15
16 pp.subplot(1, 3, 2)
17 pp.imshow(I1sm, cmap='gray')
18 pp.title('Ostrenje z masko')
19 pp.axis('off')
20
21 pp.subplot(1, 3, 3)
22 pp.imshow(I1sl, cmap='gray')
23 pp.title('Ostrenje z Laplaceom')
24 pp.axis('off')
25
26 pp.show()
```

(b) Izvedemo ostrenje slike za različne vrednosti parametra c.

```
1  pp.figure()
2  pp.suptitle('Ostrenje z masko')
3  pp.subplot(1, 4, 1)
4  pp.imshow(I1, cmap='gray')
5  for index, c in enumerate((0.5, 1, 2.0)):
6    img = funkcije.imSharpen2d(I1, 'mask', c=c, sigma=1.0)
7    img = np.clip(img, 0, 255)
8    pp.subplot(1, 4, index + 2)
9    pp.imshow(img, cmap='gray')
10   pp.title('c = {:.1f}'.format(c))
```

```
11
12 pp.figure()
13 pp.suptitle('Ostrenje z Laplaceom')
14 pp.subplot(1, 4, 1)
15 pp.imshow(I1, cmap='gray')
16 for index, c in enumerate((0.5, 1, 2.0)):
17   img = funkcije.imSharpen2d(I1, 'laplace', c=c)
18   img = clip(img, 0, 255)
19   pp.subplot(1, 4, index + 2)
20   pp.imshow(img, cmap='gray')
21   pp.title('c = {:.1f}'.format(c))
```

(a) $c = 0.5$ (b) $c = 1$ (c) $c = 2$

Slika 4.9: Ostrenje z Laplaceovim operatorjem za nekaj različnih vrednosti parametra c.

(a) $c = 0.5$ (b) $c = 1$ (c) $c = 2$

Slika 4.10: Ostrenje z masko za nekaj različnih vrednosti parametra c pri $\sigma = 1$.

7. V modulu `funkcije` ustvarimo funkcijo `imStatFilt2d`.

```
1 def imStatFilt2d(img, n=3, kind='median'):
2   H, W = img.shape
3
4   if kind == 'min':
5     f = np.min
6
```

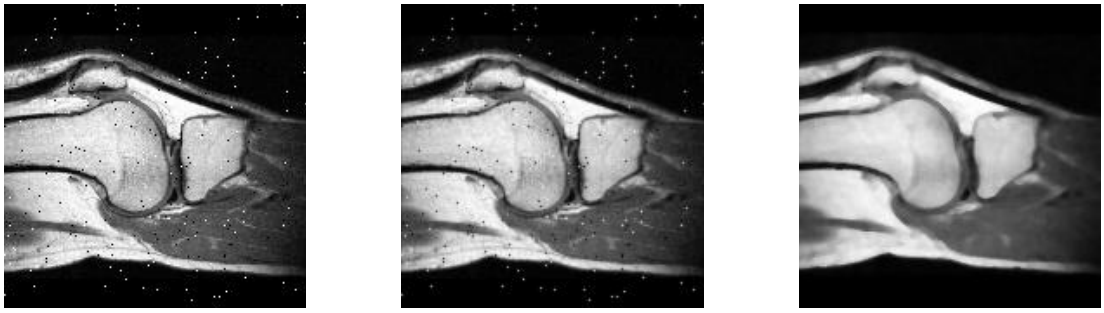
```
  elif kind == 'max':
    f = np.min

  elif kind == 'median':
    f = np.median

  else:
    raise ValueError(
      'Vrednost parametra kind je lahko'
      ' "min", "max" ali "median"!')

  p = int(np.floor(n/2.0))
  pimg = imPad2d(img, [p, p], 'nearest')
  oimg = np.zeros(img.shape)
  for i in range(H):
    for j in range(W):
      oimg[i, j] = f(pimg[p + i:p + i + n + 1,
                          p + j:p + j + n + 1])
  return oimg
```

Izvedemo primerjavo med Gaussovim in medianinim filtrom (slika 4.11).

```
I2m = funkcije.imStatFilt2d(I2, 3, 'median')
I2g = funkcije.imGaussFilt2d(I2, 0.5)

pp.figure()
pp.suptitle('Primerjava medianinega in Gaussovega filtra.')

pp.subplot(1, 3, 1)
pp.imshow(I2, cmap='gray')
pp.title('Izvirna slika')
pp.axis('off')

pp.subplot(1, 3, 2)
pp.imshow(I2m, cmap='gray')
pp.title('Medianin filter')
pp.axis('off')

pp.subplot(1, 3, 3)
pp.imshow(I2g, cmap='gray')
pp.title('Gaussov filter')
pp.axis('off')pp.show()
```

8. Ustvarimo in preizkusimo konvolucijsko jedro velikosti 3×3, ki vrednost slikovnega elementa nadomesti s povprečno vrednostjo 8-ih sosednjih slikovnih elementom (slika 4.12).

```
Ka = np.array([[1, 1, 1],
               [1, 0, 1],
               [1, 1, 1]], dtype=np.float)/8.0
```



(a) Izvirna slika (b) Gaussov filter (c) Medianin filter

Slika 4.11: Primerjava med Gaussovi $\sigma = 0{,}5$ in medianinim $n = 3$ filtrom.

```
4
5  I2a = funkcije.conv2d(I2, Ka, boundary='reflect')
6
7  pp.figure()
8
9  pp.subplot(1, 3, 1)
10 pp.title('Izvirna slika')
11 pp.imshow(I2, cmap='gray')
12
13 pp.subplot(1, 3, 2)
14 pp.title('Mediana 3 x 3')
15 pp.imshow(I2m, cmap='gray')
16
17 pp.subplot(1, 3, 3)
18 pp.title('Povprečje sosedov')
19 pp.imshow(I2a,cmap='gray')
20
21 pp.show()
```

Filtriranje z mediano bo uspešno odstranilo šum tipa sol in poper tudi v primeru, ko je več sosednjih slikovnih elementov podvrženo šumu (največ polovica sosednjih slikovnih elementov). Filtriranje s povprečjem sosednjih slikovnih elementov bo uspešno le, če sosednji slikovni elementi niso podvrženi šumu tipa sol in poper.

9. V modulu `funkcije` ustvarimo funkcijo `imGaussFilt3d`.

```
1  def imGaussFilt3d(img, sigma, boundary='constant', fillvalue=0):
2    H, W, D = img.shape
3    K = gaussianKernel1d(sigma)
4    oimg = np.zeros(img.shape)
5
6    for i in range(D):
7      for j in range(H):
8        oimg[j, : ,i] = conv1d(
9          img[j, :, i], K, boundary, fillvalue)
```

(a) Izvirna slika (b) Povprečje sosedov (c) Medianin filter

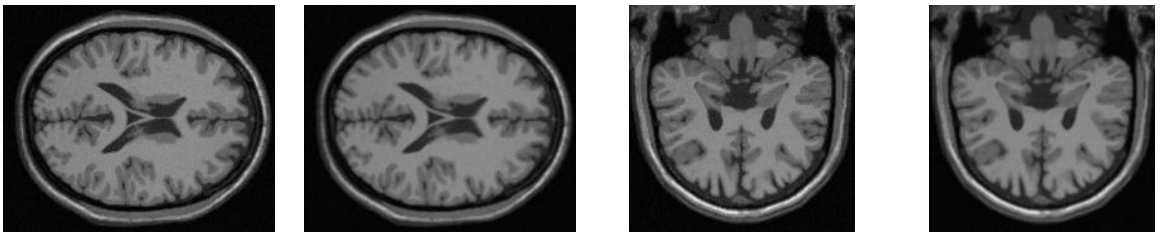
Slika 4.12: Primerjava med filtriranjem s konvolucijskim jedrom velikosti 3×3, ki vrednost slikovnega elementa nadomesti s povprečno vrednostjo 8-ih sosednjih slikovnih elementom, in medianinim filtrom enake velikosti.

```
10
11    for i in range(D):
12      for j in range(W):
13        oimg[:, j, i] = conv1d(
14          oimg[:, j, i], K, boundary, fillvalue)
15
16    for i in range(H):
17      for j in range(W):
18        oimg[i, j, :] = conv1d(
19          oimg[i, j, :], K, boundary, fillvalue)
20
21    return oimg
```

Glajenje 3D slike z Gaussovim konvolucijskim jedrom (slika 4.13).

```
1  I3f = funkcije.imGaussFilt3d(I3, 0.5)
2
3  pp.figure()
4  pp.suptitle('Filtriranje 3D slike, z=90 in x=90')
5
6  pp.subplot(2, 2, 1)
7  pp.imshow(I3[:,:,90], cmap='gray')
8  pp.title('Izvirna slika x=90')
9  pp.axis('off')
10
11 pp.subplot(2, 2, 2)
12 pp.imshow(I3f[:,:,90], cmap='gray')
13 pp.title('Filtrirana slika x=90')
14 pp.axis('off')
15
16 pp.subplot(2, 2, 3)
17 pp.imshow(I3[90,:,:].squeeze(), cmap='gray')
```

```
18 pp.title('Izvirna slika z=90')
19 pp.axis('off')
20
21 pp.subplot(2, 2, 4)
22 pp.imshow(I3f[90,:,:].squeeze(), cmap='gray')
23 pp.title('Filtrirana slika z=90')
24 pp.axis('off')
25
26 pp.show()
```



(a) Izvirna slika $z = 90$ (b) Filtrirana slika $z = 90$ (c) Izvirna slika $x = 90$ (d) Filtrirana slika $x = 90$

Slika 4.13: Primer glajenja 3D slike z Gaussovim konvolucijskim jedrom $\sigma = 0{,}5$.

10. Preizkusimo funkcije modula `scipy.ndimage`.

```
1  from scipy.ndimage import convolve, gaussian_filter,
2                            median_filter, laplace
3
4  # konvolucija z gaussovim jedrom
5  oimg = convolve(I1, Kg1)
6
7  # gaussov filter sigma=1.0, 2D slika
8  oimg = gaussian_filter(I1, sigma=1.0)
9
10 # gaussov filter sigma=1.0, 3D slika
11 oimg = gaussian_filter(I3, sigma=1.0)
12
13 # medianin filter velikosti 3 x 3
14 oimg = median_filter(I1, [3, 3])
15
16 # laplace
17 oimg = laplace(I1)
```

Poglavje 5

Prikazovanje in preslikovanje slik

Prvi del poglavja je namenjen spoznavanju orodij za prikazovanje standardnih prerezov 3D sivinskih slik (slika 5.1 in slika 5.2) ter izračunu projekcij sivinskih vrednosti vzdolž glavnih koordinatnih osi slike. V drugem delu poglavja obravnavamo sivinske preslikave in izračun sivinskih prerezov slik. Sivinske preslikave so v splošnem poljubne preslikave, ki vsakemu slikovnemu elementu referenčne sivinske slike r z dinamičnim območjem sivinskih vrednosti $[0, Lr]$ priredijo vrednost iz dinamičnega območja preslikane slike $[0, Ls]$. Glavni namen sivinskih preslikav je povečanje kontrasta struktur zanimanja na sliki in prilagoditev sivinskih vrednosti za potrebe prikazovanja.

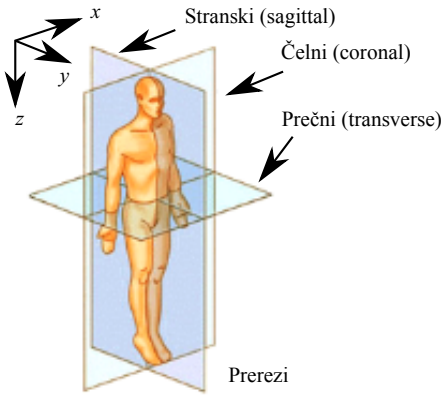
5.1 Naloge in vprašanja

1. Naložite 3D sliko telesa ct_287x165x194_uint8.raw, ki je bila zajeta s slikovno tehniko računalniške tomografije. Velikost slike je $x \times y \times z = 287 \times 165 \times 194$ slikovnih elementov, velikost slikovnega elementa pa $dx \times dy \times dz = 2 \times 2 \times 4$ mm. Slikovni elementi so shranjeni v vrstnem redu `'xyz'`. Pri izrisu slike upoštevajte velikost slikovnega elementa. To vam omogoča parameter `extent=[xmin, xmax, ymin, ymax]` funkcije `imshow` modula `matplotlib.pyplot`. Koordinatno izhodišče slike naj bo v središču slikovnega elementa z naslovom `[0, 0, 0]`.

   (a) Ustvarite funkcijo `imShowEx`, ki izriše sivinsko ali barvno sliko `img`, definirano na poljubnem pravokotnem področju `extent`. V ta namen uporabite funkcijo `imshow` modula `matplotlib.pyplot`. Prikaz koordinatnih osi prilagodite vrednosti parametra `axis`, prikazovalno okno pa opremite z naslovno vrstico, ki jo določa vrednost parametra `title`. Če je vrednost parametra `subplot` različna od `None`, izrišite sliko v podanem podoknu. Sivine slik prikažite z barvno mapo `cmap='gray'`.

```
1 def imShowEx(img, extent=None,
2               title='', axis='off', subplot=None):
```

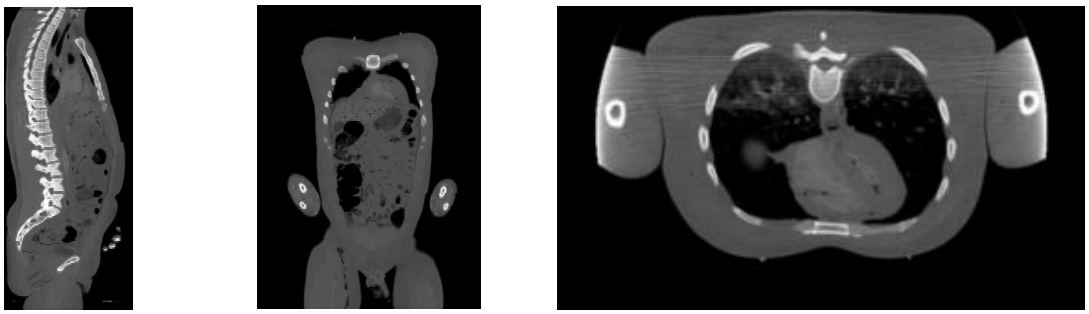
   (b) Določite stranske ($x = konst.$), čelne ($y = konst.$) in prečne ($z = konst.$) prereze slike pri $x = 280$ mm, $y = 220$ mm in $z = 200$ mm ter jih prikažite. Sliko stranskega



Slika 5.1: Standardni prerezi 3D slik.

prereza I_x boste uporabili pri reševanju nalog pod točkami 2-5. Za enostavno izločitev razsežnosti podatkovnega polja, ki so enake 1, uporabite člansko funkcijo `squeeze`.

(c) Izračunajte projekcije maksimalne in povprečne sivinske vrednosti 3D slike vzdolž koordinatnih osi x, y in z ter jih prikažite. Povprečno in maksimalno vrednost vzdolž poljubne razsežnosti podatkovnega polja lahko določite s članskima funkcijama `mean` in `max`.



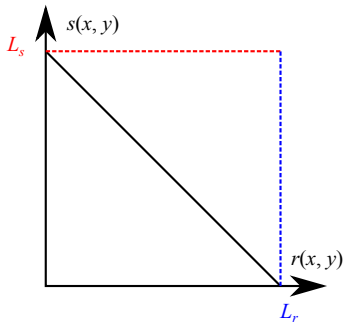
(a) Stranski (b) Čelni (c) Prečni

Slika 5.2: Stranski, čelni in prečni prerez slike ct_287x165x194_uint8.raw.

2. Linearno sivinsko preslikavo (slika 5.3) izvedemo tako, da sivinsko vrednost slikovnega elementa referenčne slike r preslikamo z linearno funkcijo. Rezultat preslikave je slika s, ki ima linearno preslikane sivinske vrednosti. Pri izbiri vrednosti parametrov preslikave a in b je potrebno upoštevati dinamično območje preslikane slike s:

$$s(x, y) = a \cdot r(x, y) + b. \tag{5.1}$$

(a) Ustvarite funkcijo `imScale`, ki linearno preslika sivinske vrednosti slike `img`. Parametra linearne preslikave naj bosta določena kot $a =$`slope` in $b =$`intersection`.



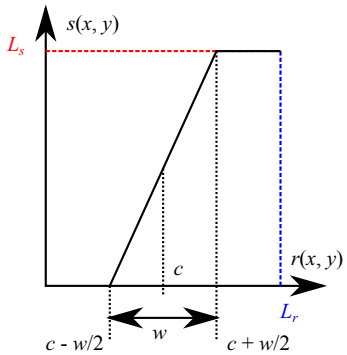
Slika 5.3: Linearna preslikava.

```
1 def imScale(img, slope, intersection):
2   ...
3   return oImage
```

(b) S pomočjo funkcije `imScale` invertirajte sliko I_x. Dinamično območje slike $[0, 255]$ naj ostane nespremenjeno. Izrišite tudi pripadajoča histograma in linijska prereza ($y = 175$) obeh slik.

3. Linearno oknjenje (slika 5.4) izvedemo tako, da na dinamičnem območju referenčne slike r definiramo poljubno okno s središčem c in širino w. Sivinskim vrednostim referenčne slike, ki so manjše od $c - \frac{w}{2}$, priredimo vrednost 0, sivinskim vrednostim, ki so večje od $c + \frac{w}{2}$, priredimo vrednost L_s, sivinske vrednosti na intervalu $[c - \frac{w}{2}, c + \frac{w}{2}]$ pa preslikamo z linearno preslikavo:

$$s(x,y) = \begin{cases} 0, & r(x,y) < c - \frac{w}{2}, \\ \frac{L_s}{w}\left(r(x,y) - \left(c - \frac{w}{2}\right)\right), & c - \frac{w}{2} <= r(x,y) < c + \frac{w}{2}, \\ L_s, & r(x,y) >= c + \frac{w}{2}. \end{cases} \tag{5.2}$$



Slika 5.4: Linearno oknjenje.

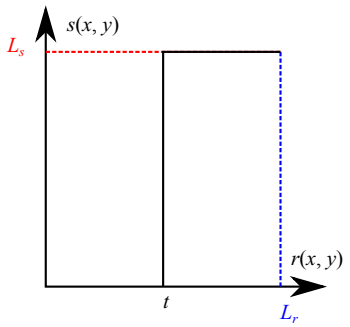
(a) Ustvarite funkcijo `imWindow`, ki izvede linearno oknjenje slike `img`. Središče okna c naj bo določeno s parametrom `center`, širina okna w s parametrom `width`, dinamično območje L_s izhodne slike pa naj določa vrednost parametra `ls`.

```
1 def imWindow(img, center, width, ls=255):
2   ...
3   return oImage
```

(b) Oknite sliko I_x tako, da iz linearnega območja preslikave izločite 5 % najtemnejših in 5 % najsvetlejših sivinskih vrednosti v sliki. Dinamično območje slike naj ostane nespremenjeno. Izrišite tudi pripadajoča histograma in sivinska prereza ($y = 175$) obeh slik.

4. Upragovljanje slike (slika 5.5) izvedemo tako, da vsaki sivinski vrednosti referenčne slike r, ki je manjša od praga t, priredimo vrednost 0, sicer pa največjo možno sivinsko vrednost L_s:

$$s(x,y) = \begin{cases} 0, & r(x,y) < t, \\ L_s, & \text{drugod.} \end{cases} \tag{5.3}$$



Slika 5.5: Upragovljanje.

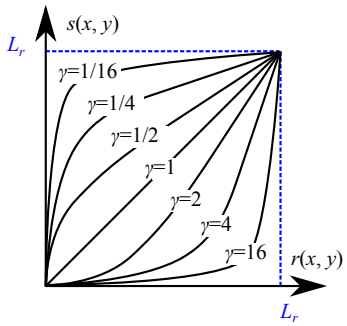
(a) Ustvarite funkcijo za upragovljanje sivinskih slik `imThreshold`, kjer je `img` referenčna slika, `threshold` izbrani prag, `ls` pa dinamično območje L_s upragovljene slike.

```
1 def imThreshold(img, threshold, ls=255):
2   ...
3   return oImage
```

(b) Preizkusite delovanje funkcije na sliki I_x. Vrednost praga naj bo $t = 127$.

5. Gama preslikava (slika 5.6) je nelinearna zvezna preslikava. Pri preslikavi običajno predpostavimo, da sta dinamični območji referenčne in preslikane slike enaki:

$$s(x,y) = L_r^{1-\gamma} r^{\gamma}(x,y). \tag{5.4}$$



Slika 5.6: Gama preslikava.

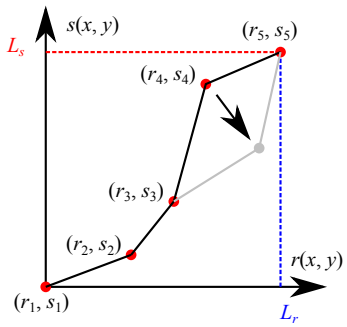
(a) Ustvarite funkcijo `imGamma`, ki preslika sivinske vrednosti referenčne slike `img` z gama preslikavo $\gamma =$ `gamma`.

```
1 def imGamma(img, gamma):
2     ...
3     return oImage
```

(b) Preizkusite delovanje funkcije na sliki I_x z vrednostjo $\gamma = 2$. Izrišite tudi pripadajoča histograma in linijska prereza ($y = 175$) obeh slik.

6. Odsekoma linearna preslikava (slika 5.7) je popolnoma določena z množico urejenih parov kontrolnih točk $(r_i, s_i), i = 1...N$. Zaporedni pari točk določajo linearno preslikavo iz dinamičnega območja $[r_i, r_{i+1}]$ referenčne slike v dinamično območje $[s_i, s_{i+1}]$ preslikane slike. Na odsekih, kjer so multiplikativni koeficienti linearne preslikave večji od 1, kontrast povečujemo, sicer ga zmanjšujemo. V splošnem je z odsekoma linearno preslikavo mogoče izraziti ali aproksimirati vse v predhodnih točkah obravnavane sivinske preslikave:

$$s(x,y) = \frac{s_{i+1} - s_i}{r_{i+1} - r_i} \left(r(x,y) - r_i\right) + s_i. \tag{5.5}$$



Slika 5.7: Odsekoma linearna preslikava.

(a) Ustvarite funkcijo `imMultiScale`, ki odsekoma linearno preslika sivinske vrednosti vhodne slike `img`. Parameter `r` naj bo vektor kontrolnih točk $[r_1, ..., r_N]$ na dinamičnem območju referenčne slike, `s` pa vektor kontrolnih točk $[s_1, ..., s_N]$ na dinamičnem območju preslikane slike.

```
def imMultiScale(img, r, s):
  ...
  return oImage
```

(b) S funkcijo `imMultiscale` aproksimirajte nelinearno Gama preslikavo za $\gamma = 2$. Določite število odsekov, ki so potrebni, da napaka aproksimacije ne presega 1 sivine.

7. Ustvarite funkcijo `imProfile2d`, ki izračuna linijski prerez slike v ekvidistantnih točkah `ox`, `oy` vzdolž poljubne daljice s krajiščema v točkah `t1` in `t2`. Koordinatni sistem slike naj bo definiran z vektorjema `x` in `y`. Korak vzorčenja vzdolž izbrane daljice naj določa parameter `step`. Za interpolacijo sivinskih vrednosti vzdolž daljice uporabite funkcijo `interp2` priloženega modula `interp`, red interpolacije pa naj določa parameter `order` (0 - interpolacija z najbližjim sosedom, 1 - bilinearna interpolacija).

```
def imProfile2d(img, x, y, t1, t2, step=1, order=1):
  ...
  return profile, ox, oy
```

(a) Izrišite diagonalna linijska prereza slike I_x. Uporabite korak `step=2` ter bilinearno interpolacijo (`order=1`).

(b) Ustvarite še funkcijo `imProfile3d`, ki izračuna linijski prerez 3D slike vzdolž daljice s krajiščema v točkah `t1` in `t2`. Interpolacijo sivinskih vrednosti vzdolž daljice izvedite s funkcijo `interp3` priloženega modula `interp`. Izrišite diagonalni linijski prerez 3D slike z enim krajiščem v koordinatnem izhodišču. Uporabite korak `step=2` ter trilinearno interpolacijo (`order=1`).

```
def imProfile3d(img, x, y, z, t1, t2, step=1, order=1):
  ...
  return profile, ox, oy, oz
```

5.2 Rešitve in odgovori na vprašanja

Pri odgovorih privzamemo, da se funkcije iz rešitev nahajajo v modulu `funkcije`, slikovno gradivo pa se nahaja v podmapi `poglavje_5`. Naprej uvozimo potrebne module, naložimo slike in definiramo pomožne spremenljivke.

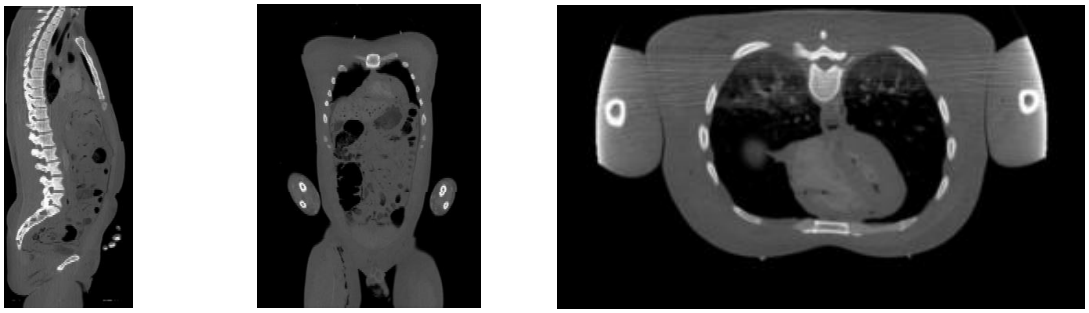
```
import numpy as np
from matplotlib import pyplot as pp
import funkcije

I = funkcije.imLoadRaw3d(
  './polgavje_5/ct_287x165x194_uint8.raw',
  287, 165, 194, dtype=np.uint8, order='xyz')
D, H, W = I.shape

dx, dy, dz = 2.0, 2.0, 4.0
Hmm, Wmm, Dmm = H*dy, W*dx, D*dz
IxExtent = [0, Hmm, 0, Dmm]
x, y, z = np.arange(W)*dx, np.arange(H)*dy, np.arange(D)*dz
zi = np.arange(0, Dmm, 2.0)
yi = np.tile(175.0, zi.shape)
nBins = 64
```

1. (a) V modulu `funkcije` ustvarimo funkcijo `imShowEx`.

```
def imShowEx(img, extent=None,
             title='', axis='off', subplot=None, **kwargs):
  if subplot is not None:
    if isinstance(subplot, (list, tuple)):
      pp.subplot(*subplot)
    else:
      pp.subplot(subplot)

  if len(img.shape) <= 2:
    cmap='gray'

  pp.imshow(img.squeeze(), extent=extent, cmap=cmap, **kwargs)
  pp.title(title)
  pp.axis(axis)
```

(b) Izrišemo standardne prereze pri $x = 280$, $y = 220$ in $z = 200$ mm (slika 5.8).

```
pp.figure()

Iprecni = I[int(200/dz), :, :].squeeze()
funkcije.imShowEx(
  Iprecni, extent=[0, Wmm, 0, Hmm],
  title='Prečni prerez z=200 mm', subplot=131)
  Icelni = I[:, int(220/dy), :].squeeze()

```

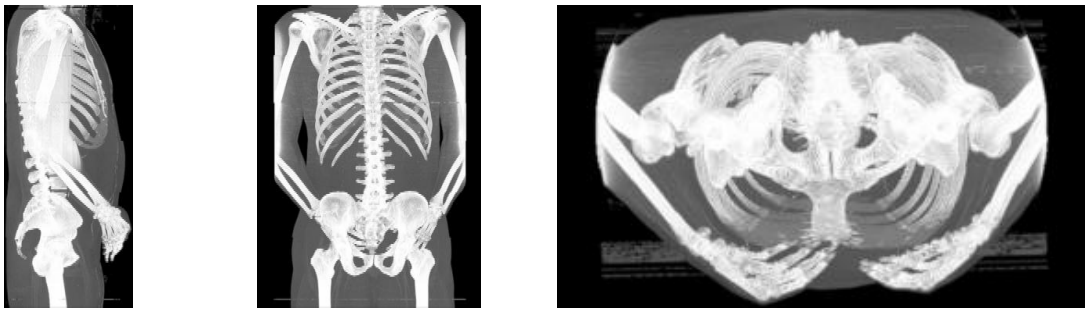


(a) Stranski (b) Čelni (c) Prečni

Slika 5.8: Stranski, čelni in prečni prerez slike ct_287x165x194_uint8.raw.

```
9  funkcije.imShowEx(
10   Icelni, extent=[0, Wmm, 0, Dmm],
11   title='Čelni prerez y=220 mm', subplot=132)
12
13 Ix = Istranski = I[:, :, int(280/dx)].squeeze()
14 funkcije.imShowEx(
15   Istranski, extent=[0, Hmm, 0, Dmm],
16   title='Stranski prerez x=280 mm', subplot=133)
17
18 pp.show()
```

(c) Izračunamo in izrišemo še projekcije maksimalne sivinske vrednosti vzdolž koordinatnih osi (slika 5.9).



(a) Vzdolž x osi (b) Vzdolž y osi (c) Vzdolž z osi

Slika 5.9: Projekcije maksimalnih sivinskih vrednosti vzdolž koordinatnih osi.

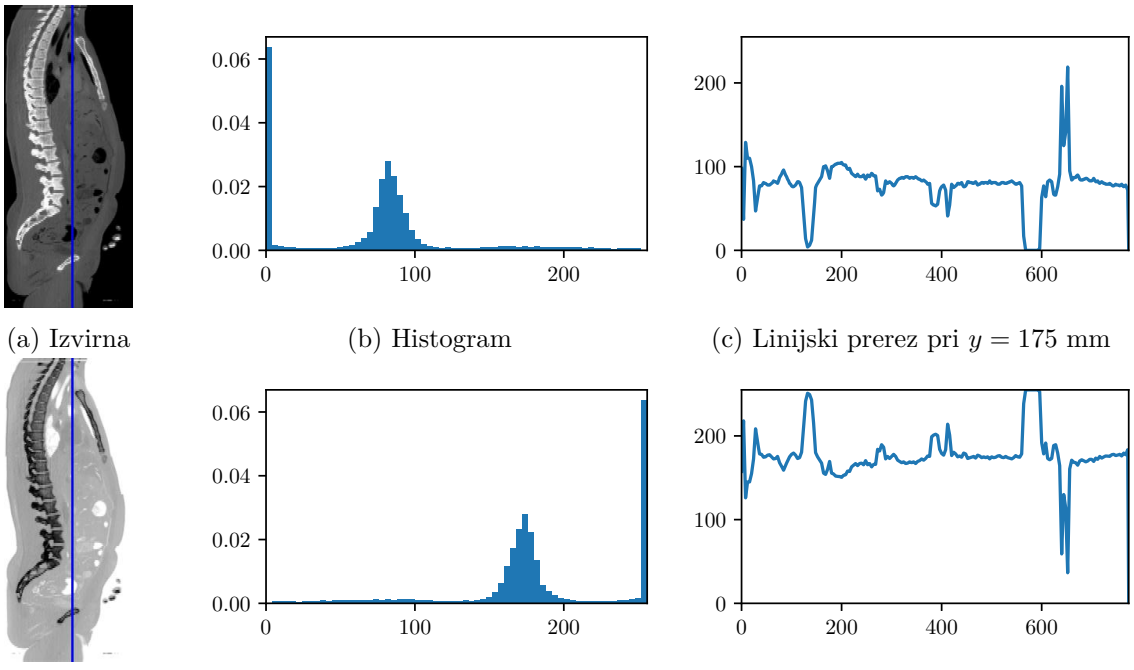
```
1 pp.figure()
2
3 IprecniMax = I.max(0)
4 funkcije.imShowEx(IprecniMax, extent=[0, Wmm, 0, Hmm],
5                   title='Maks. projekcija vzdolž z',
6                   subplot=131)
```

```
7
8  IcelniMax = I.max(1)
9  funkcije.imShowEx(IcelniMax, extent=[0, Wmm, 0, Dmm],
10                   title='Maks. projekcija vzdolž y',
11                   subplot=132)
12
13 IstranskiMax = I.max(2)
14 funkcije.imShowEx(IstranskiMax, extent=[0, Hmm, 0, Dmm],
15                   title='Maks. projekcija vzdolž x',
16                   subplot=133)
17
18 pp.show()
```

2. (a) V modulu `funkcije` ustvarimo funkcijo `imScale`.

```
1 def imScale(img, slope, intersection):
2   return img.astype(np.float)*slope + intersection
```

(b) Izvedemo invertiranje slike in prikažemo rezultat (slika 5.10).



(a) Izvirna (b) Histogram (c) Linijski prerez pri $y = 175$ mm

(d) Invertirana (e) Histogram (f) Linijski prerez pri $y = 175$ mm

Slika 5.10: Invertiranje slike.

```
1 IxScaled = funkcije.imScale(Ix, -1.0, 255.0)
2 pp.figure()
3
4 # Ix
```

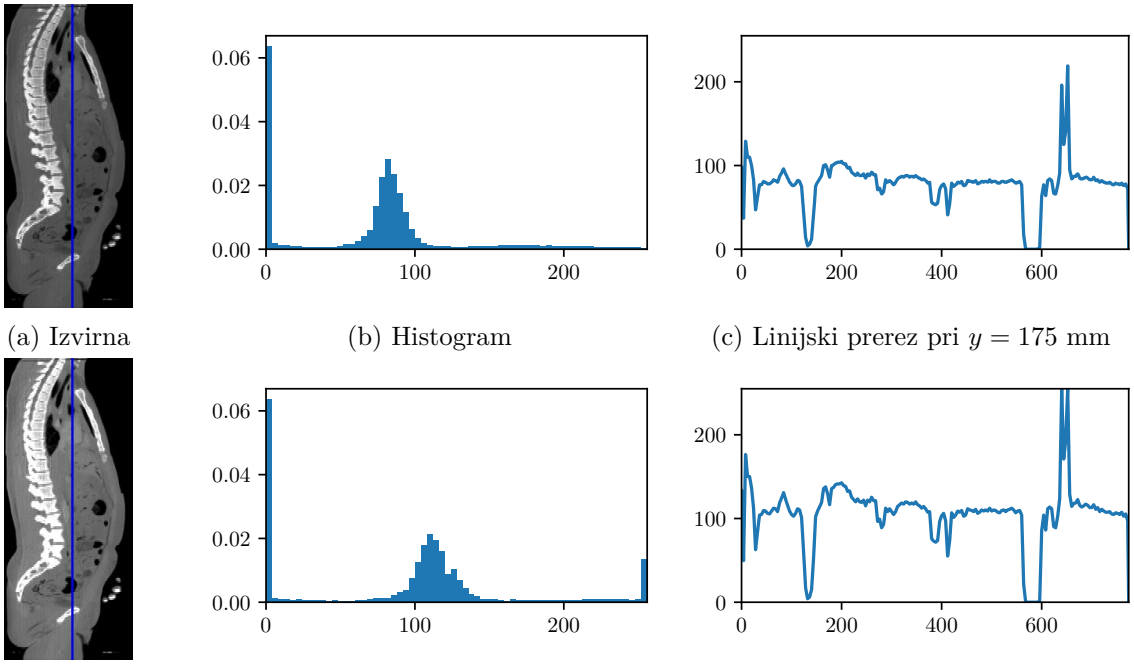
```
5  funkcije.imShowEx(Ix, extent=IxExtent,
6                    subplot=231, title='Izvirna slika')
7  pp.plot(yi, zi, '-b')
8  pp.subplot(2, 3, 2)
9  pp.hist(Ix.flatten(), nBins, [0, 256], normed=True)
10 pp.xlim([0, 256])
11 pp.subplot(2, 3, 3)
12 pp.plot(zi, interp.interp2(y, z, Ix, yi, zi))
13 pp.ylim([0, 255])
14
15 # IxScaled
16 funkcije.imShowEx(IxScaled, extent=IxExtent,
17                    subplot=234, title='imScale')
18 pp.plot(yi, zi, '-b')
19 pp.subplot(2, 3, 5)
20 pp.hist(IxScaled.flatten(), nBins, [0, 256], normed=True)
21 pp.xlim([0, 256])
22 pp.subplot(2, 3, 6)
23 pp.plot(zi, interp.interp2(y, z, IxScaled, yi, zi))
24 pp.ylim([0, 255])
25
26 pp.show()
```

3. (a) V modulu `funkcije` ustvarimo funkcijo `imWindow`.

```
1  def imWindow(img, center, width, ls=255):
2    oimg = np.zeros(img.shape)
3    ind1 = img < center - width*0.5
4    oimg[ind1] = 0
5    ind2 = img > center + width*0.5
6    oimg[ind2] = ls
7    ind3 = np.logical_not(ind1 | ind2)
8    oimg[ind3] = ls/width*(img[ind3] - (center - width*0.5))
9
10   return oimg
```

(b) Izvedemo zahtevano oknjenje slike in prikažemo rezultat (slika 5.11).

```
1  tmp = np.sort(Ix.flatten())
2  start = tmp[round(tmp.size*0.05)] + 1
3  end = tmp[round(tmp.size*0.95)] - 1
4  center = 0.5*(start + end)
5  width = end - start
6  IxWindow = funkcije.imWindow(Ix, center, width, 255)
7
8  pp.figure()
9
10 # Ix
11 funkcije.imShowEx(Ix, extent=IxExtent,
12                    subplot=231, title='Izvirna slika')
```



(a) Izvirna (b) Histogram (c) Linijski prerez pri $y = 175$ mm

(d) Oknjena (e) Histogram (f) Linijski prerez pri $y = 175$ mm

Slika 5.11: Oknjenje slike z izločitvijo 5 % najtemnejših in 5 % najsvetlejših slikovnih elementov.

```
13 pp.plot(yi, zi, '-b')
14 pp.subplot(2, 3, 2)
15 pp.hist(Ix.flatten(), nBins, [0, 256], normed=True)
16 pp.xlim([0, 256])
17 pp.subplot(2, 3, 3)
18 pp.plot(zi, interp.interp2(y, z, Ix, yi, zi))
19 pp.ylim([0, 255])
20
21 # IxWindow
22 funkcije.imShowEx(IxWindow, extent=IxExtent,
23                   subplot=234, title='imWindow')
24 pp.plot(yi, zi, '-b')
25 pp.subplot(2, 3, 5)
26 pp.hist(IxWindow.flatten(), nBins, [0, 256], normed=True)
27 pp.xlim([0, 256])
28 pp.subplot(2, 3, 6)
29 pp.plot(zi, interp.interp2(y, z, IxWindow, yi, zi))
30 pp.ylim([0, 255])
31
32 pp.show()
```

4. (a) V modulu `funkcije` ustvarimo funkcijo `imThreshold`.

```
1 def imThreshold(img, threshold, ls=255):
```

```
2     oimg = np.zeros(img.shape)
3     oimg[img < threshold] = 0
4     oimg[img >= threshold] = ls
5
6     return oimg
```

(b) Preverimo delovanje funkcije in izrišemo izvirno ter upragovljeno sliko (slika 5.12).

(a) Izvirna (b) Upragovljena s $t = 127$

Slika 5.12: Izvirna in upragovljena slika.

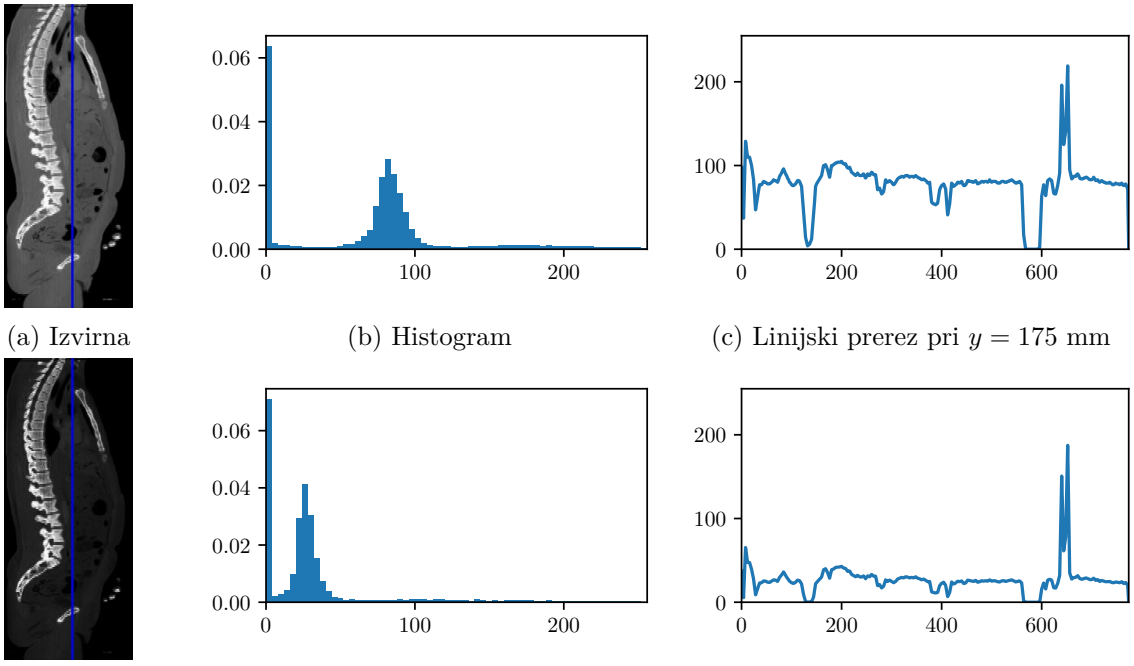
```
1  IxThreshold = funkcije.imThreshold(Ix, 127, 255)
2
3  pp.figure()
4
5  # Ix
6  funkcije.imShowEx(Ix, extent=IxExtent,
7                    subplot=121, title='Izvirna slika')
8
9  # IxThreshold
10 funkcije.imShowEx(IxThreshold, extent=IxExtent,
11                   subplot=122, title='IxThreshold')
12
13 pp.show()
```

5. (a) V modulu `funkcije` ustvarimo funkcijo `imGamma`.

```
1  def imGamma(img, gamma, lr=255):
2    return lr**(1.0 - gamma)*img.astype(np.float)**gamma
```

(b) Izvedemo nelinearno Gama preslikavo `gamma=2` slike in prikažemo rezultat (slika 5.13).

```
1  IxGamma = funkcije.imGamma(Ix, 2)
2
3  pp.figure()
4
5  # Ix
6  funkcije.imShowEx(Ix, extent=IxExtent,
```



(a) Izvirna (b) Histogram (c) Linijski prerez pri $y = 175$ mm

(d) Preslikana (e) Histogram (f) Linijski prerez pri $y = 175$ mm

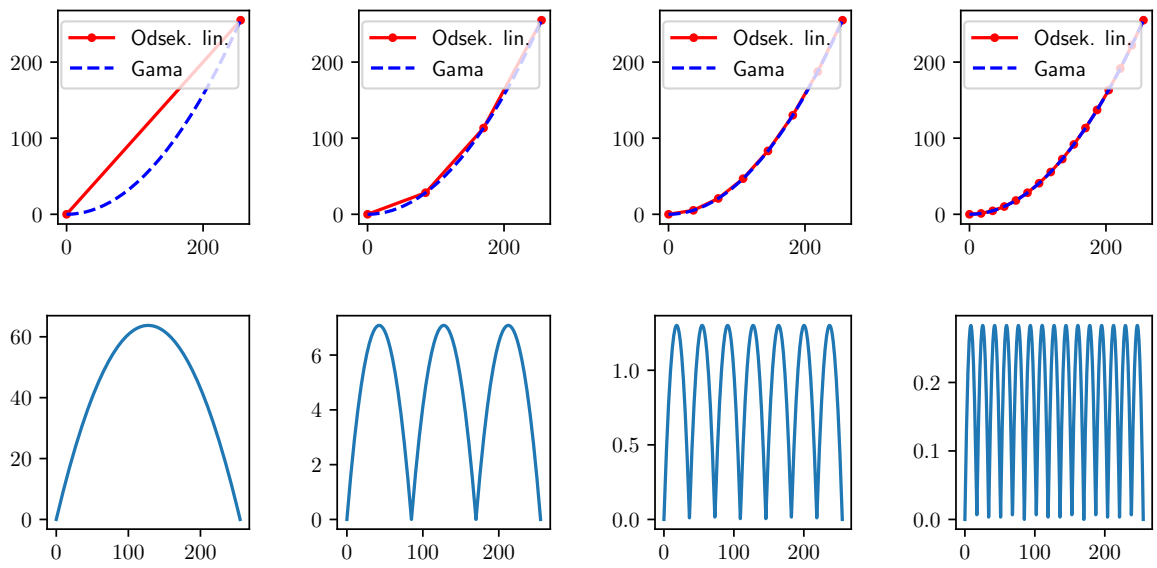
Slika 5.13: Nelinearna Gama preslikava.

```
7                       subplot=231, title='Izvirna slika')
8   pp.plot(yi, zi, '-b')
9   pp.subplot(2, 3, 2)
10  pp.hist(Ix.flatten(), nBins, [0, 256], normed=True)
11  pp.xlim([0, 256])
12  pp.subplot(2, 3, 3)
13  pp.plot(zi, interp.interp2(y, z, Ix, yi, zi))
14  pp.ylim([0, 255])
15
16  # IxWindow
17  funkcije.imShowEx(IxGamma, extent=IxExtent,
18                    subplot=234, title='IxGamma')
19  pp.plot(yi, zi, '-b')
20  pp.subplot(2, 3, 5)
21  pp.hist(IxGamma.flatten(), nBins, [0, 256], normed=True)
22  pp.xlim([0, 256])
23  pp.subplot(2, 3, 6)
24  pp.plot(zi, interp.interp2(y, z, IxGamma, yi, zi))
25  pp.ylim([0, 255])
26
27  pp.show()
```

6. (a) V modulu `funkcije` ustvarimo funkcijo `imMultiscale`.

```
1  def imMultiscale(img, r, s):
2    oimg = np.zeros(img.shape)
3    for i in range(len(r) - 1):
4      ind = (img >= r[i]) & (img <= r[i + 1])
5      oimg[ind] = (s[i + 1] - s[i])/(r[i + 1] - r[i])* \
6        (img[ind] - r[i]) + s[i]
7
8    return oimg
```

(b) Preizkusimo aproksimacijo nelinearne Gama preslikave z odsekoma linearno preslikavo. Ugotovimo, da lahko nelinearno Gama preslikavo za $\gamma = 2$ zadovoljivo aproksimiramo z odsekoma linearno preslikavo, ki jo sestavlja 10 odsekov (slika 5.14).



(a) 1 odsek (b) 3 odseki (c) 7 odsekov (d) 15 odsekov

Slika 5.14: Aproksimacija nelinearne Gama preslike pri $\gamma = 2$ z odsekoma linearno preslikavo (prva vrstica) in pripadajoče napake aproksimacije (druga vrstica).

```
1  gamma = 2
2  gammaApproxN = [2, 4, 8, 16]
3  N = len(gammaApproxN)
4  gammaApproxImg = []
5  gammaRef = funkcije.imGamma(Ix, gamma)
6  rGamma = np.linspace(0, 255, 1000)
7  sGamma = funkcije.imGamma(rGamma, gamma)
8
9  pp.figure()
10 pp.suptitle('Aproksimacija gama preslikave (gamma={}) '\
```

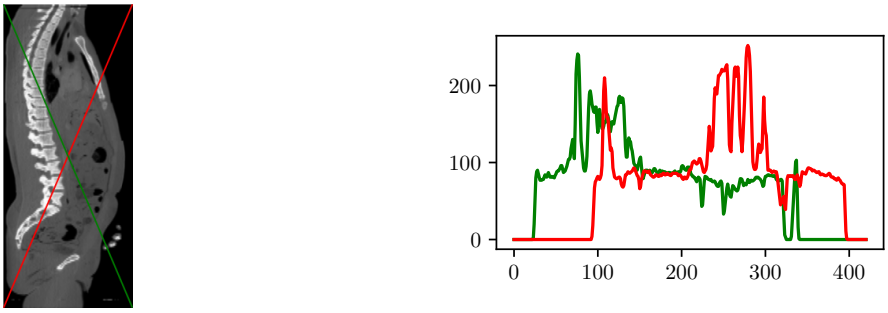
```
            'z odsekoma linearno preslikavo'.format(gamma))

for index, n in enumerate(gammaApproxN):
  r = np.linspace(0.0, 255.0, n)
  s = funkcije.imGamma(r, gamma)
  approx = funkcije.imMultiscale(Ix, r, s)
  gammaApproxImg.append(approx)

  funkcije.imShowEx(approx, extent=IxExtent,
                    subplot=(3, N, index + 1),
                    title='Št. segmentov n={}'.format(n),
                    vmin=0, vmax=255)

  pp.subplot(3, N, N + index + 1)
  pp.title('Napaka')
  pp.imshow(gammaRef - approx, cmap='gray')
  pp.colorbar()

  pp.subplot(3, N, 2*N + index + 1)
  pp.plot(r, s, '.-r', label='Odsek. lin.')
  pp.plot(rGamma, sGamma, '--b', label='Gama')
  pp.legend()

pp.show()
```

7. (a) V modulu `funkcije` ustvarimo funkcijo `imProfile2d`.

```
def imProfile2d(img, x, y,
                t1, t2, step=1, order=1):
  t1 = np.asarray(t1, dtype=np.float)
  t2 = np.asarray(t2, dtype=np.float)
  s = t2 - t1
  s = s/np.linalg.norm(s)
  xi = np.arange(t1[0], t2[0], s[0]*float(step))
  yi = np.arange(t1[1], t2[1], s[1]*float(step))
  op = interp.interp2(x, y, img, xi, yi)

  return op, xi, yi
```

Izrišemo diagonalna linijska prereza 2D slike I_x (slika 5.15).

```
p2a, p2ax, p2ay = funkcije.imProfile2d(
  Ix, y, z,
  [0.0, 0.0], [y[-1], z[-1]],
  step=2.0, order=1)
p2b, p2bx, p2by = funkcije.imProfile2d(
  Ix, y, z
  [y[-1], 0.0], [0.0, z[-1]],
  step=2.0, order=1)

```



Slika 5.15: Diagonalna linijska prereza slike I_x.

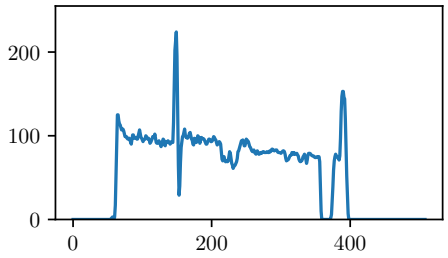
```
pp.figure()
funkcije.imShowEx(Ix, subplot=121, extent=IxExtent)
pp.plot([0.0, y[-1]], [0.0, z[-1]], '-r')
pp.plot([y[-1], 0.0], [0.0, z[-1]], '-g')
pp.subplot(1, 2, 2)
pp.plot(p2a, '-r')
pp.plot(p2b, '-g')
pp.ylim([0, 255])
pp.show()
```

(b) V modulu `funkcije` ustvarimo še funkcijo `imProfile3d`.

```
def imProfile3d(img, x, y, z,
                t1, t2, step=1, order=1):
  t1 = np.asarray(t1, dtype=np.float)
  t2 = np.asarray(t2, dtype=np.float)
  s = t2 - t1
  s = s/np.linalg.norm(s)
  xi = np.arange(t1[0], t2[0], s[0]*float(step))
  yi = np.arange(t1[1], t2[1], s[1]*float(step))
  zi = np.arange(t1[2], t2[2], s[2]*float(step))
  op = interp.interp3(x, y, z, img, xi, yi, zi)

  return op, xi, yi, zi
```

Izrišemo diagonalni prerez 3D slike (5.16).

```
p3, p3x, p3y, p3z = funkcije.imProfile3d(
  I, x, y, z,
  [0.0, 0.0, 0.0], [x[-1], y[-1], z[-1]],
  step=2.0, order=1)

pp.figure()
pp.plot(p3)
pp.ylim([0, 255])
pp.show()
```

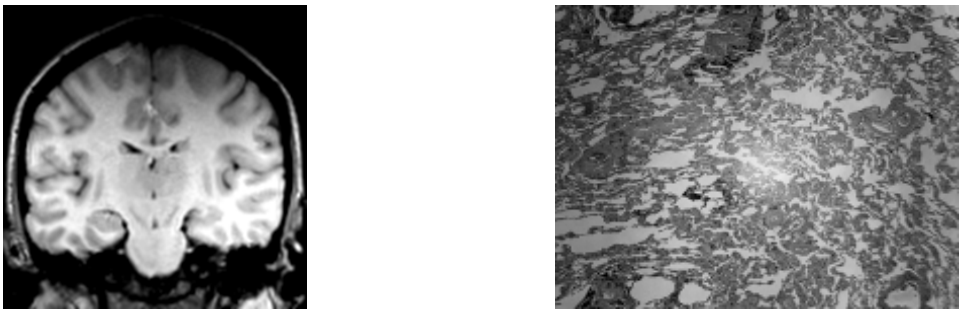


Slika 5.16: Diagonalni linijski prerez 3D slike z enim krajiščem v koordinatnem izhodišču.

Poglavje 6

Kalibracija in obnova sivinskih vrednosti

Sivinske vrednosti biomedicinskih slik, predvsem mikroskopskih in magnetno resonančnih, so zaradi nehomogenosti osvetlitve vzorca in/ali neenakomerne prostorske občutljivosti naprav za zajem slik pogosto prostorsko nehomogene. Nehomogenosti se odražajo v neenakomernem kontrastu in svetlosti istega tkiva, katerega sivinske vrednosti so odvisne od prostorske lege na zajeti sliki. Opisani neželeni pojav imenujemo prostorska nehomogenost sivinskih vrednosti, ki lahko predstavlja veliko težavo pri avtomatski obdelavi, razgradnji in kvantitativni analizi slik. Sivinske nehomogenosti zajete slike $g(x, y)$ lahko pogosto zadovoljivo opišemo z linearnim



Slika 6.1: Primera prostorske nehomogenosti sivinskih vrednosti v magnetnoresonančni (levo) in mikroskopski sliki shading_lung.png (desno).

modelom prostorske nehomogenosti [7]:

$$g(x, y) = m(x, y) \cdot f(x, y) + a(x, y), \tag{6.1}$$

kjer $m(x, y)$ predstavlja multiplikativno, $a(x, y)$ pa aditivno komponento nehomogenosti. Linearno kalibracijo nehomogenosti izvajamo tako, da ocenimo multiplikativno in aditivno komponento nehomogenosti s pomočjo dveh kalibracijskih slik, in sicer svetle slike $g_B(x, y)$ in temne slike $g_D(x, y)$. Za to potrebujemo dva homogena difuzna kalibra različne svetlosti. Pogosto se zadovoljimo le z enim kalibrom, saj lahko temno sliko zajamemo tako, da izklopimo vse

svetlobne vire. V primeru kamere temno sliko zajamemo tako, da pokrijemo objektiv ali popolnoma zapremo zaslonko. Zajeta temna slika opisuje prostorsko odvisnost temnega odziva tipala, svetla slika pa njegovo prostorsko odvisno občutljivost ter prostorsko nehomogenost optičnega sistema in vzbujanja (svetila). Komponenti nehomogenosti $a(x, y)$ in $m(x, y)$ nato določimo tako, da temni $g_D(x, y)$ in svetli $g_B(x, y)$ sliki priredimo konstantni sivinski vrednosti D in B:

$$\begin{aligned} B &= \frac{g_B(x,y) - a(x,y)}{m(x,y)}, \\ D &= \frac{g_D(x,y) - a(x,y)}{m(x,y)}. \end{aligned} \tag{6.2}$$

Iz zgornjih enačb lahko izrazimo aditivno $a(x, y)$ in multiplikativno $m(x, y)$ komponento nehomogenosti kot:

$$\begin{aligned} a(x,y) &= \frac{B \cdot g_D(x,y) - D \cdot g_B(x,y)}{B - D}, \\ m(x,y) &= \frac{g_B(x,y) - g_D(x,y)}{B - D}. \end{aligned} \tag{6.3}$$

Poljubno zajeto sivinsko sliko $g(x, y)$ nato kalibriramo kot:

$$f(x,y) = \frac{g(x,y) \cdot (B - D) - B \cdot g_D(x,y) + D \cdot g_B(x,y)}{g_B(x,y) - g_D(x,y)}. \tag{6.4}$$

6.1 Naloge in vprašanja

(a) Slika shading_muscle.png. (b) slika shading_dark.png. (c) Slika shading_bright.png.

Slika 6.2: Svetlostno nehomogena slika s pripadajočo temno g_D in svetlo g_B kalibracijsko sliko.

1. Ustvarite funkcijo `imShadingCalibrate`, ki kalibrira nehomogenost slike `img` z linearnim modelom ter vrne oceno aditivne `ofa` in multiplikativne `ofm` komponente nehomogenosti. Parameter `dark` ustreza temni kalibracijski sliki g_D, parameter `bright` svetli kalibracijski sliki g_B, parametra `d` in `b` pa konstantnima sivinskima vrednostima D in B.

```
1 def imShadingCalibrate(img, dark, bright, d=0, b=255):
2   ...
3   return omg, ofa, ofm
```

(a) Izrišite histogram nehomogene slike shading_muscle.png s pripadajočim vzdolžnim sivinski prerezom.

(b) Ocenite aditivno in multiplikativno komponento nehomogenosti testne slike s pomočjo pripadajočih kalibracijskih slik shading_bright.png ($g_B(x, y)$) in shading_dark.png ($g_D(x, y)$). Sivinsko vrednost D postavite na 0, sivinsko vrednost B pa določite tako, da bosta povprečna svetlost nehomogene in obnovljene slike približno enaki. Izrišite histogram in vzdolžni sivinski prerez obnovljene slike.

(c) Kako vpliva prostorska nehomogenost na histogram in sivinski prerez slike?

2. Pogosto je kalibracija že dovolj učinkovita, če uporabimo zgolj multiplikativni kalibracijski model $g(x, y) = m(x, y) \cdot f(x, y)$. Na ta način bistveno poenostavimo postopek svetlostne kalibracije, saj potrebujemo le sliko $g_B(x, y)$ homogenega kalibracijskega objekta:

$$\begin{aligned} m(x, y) &= \frac{g_B(x, y)}{B}, \\ f(x, y) &= g(x, y) \cdot \frac{B}{g_B(x, y)}. \end{aligned} \tag{6.5}$$

Kalibracijo je mogoče izvesti tudi z uporabo zgolj aditivnega modela $g(x, y) = f(x, y) + a(x, y)$:

$$\begin{aligned} a(x, y) &= g_D(x, y) - D, \\ f(x, y) &= g(x, y) - g_D(x, y) + D. \end{aligned} \tag{6.6}$$

(a) Nadgradite funkcijo `imShadingCalibrate` tako, da bo obnova v primeru, ko je vrednost parametra `d` enaka `None`, temeljila zgolj na multiplikativni komponenti, ko je vrednost parametra `b` enaka `None`, pa zgolj na aditivni komponenti.

(b) Obnovite homogenost slike shading_muscle.png z multiplikativnim modelom ter izrišite pripadajoči histogram in vzdolžni sivinski prerez obnovljene slike. Oceno multiplikativne komponente nehomogenosti primerjajte z oceno multiplikativne komponente iz prejšnje točke. Sivinsko vrednost B določite tako, da bosta povprečna svetlost nehomogene in obnovljene slike enaki.

(c) Ali je kalibracija z aditivnim modelom primerna za testno nehomogeno sivinsko sliko in zakaj?

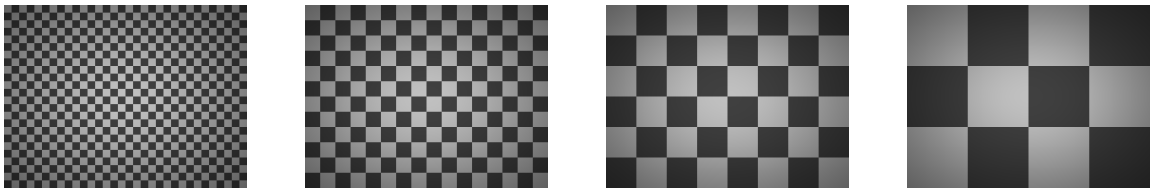
Poleg opisanih postopkov kalibracije lahko prostorsko nehomogenost sivinskih vrednosti odpravimo tudi z retrospektivni postopki, kjer informacijo o prostorski odvisnosti aditivne $a(x, y)$ in multiplikativne $m(x, y)$ komponente nehomogenosti sivinskih vrednosti ocenimo z obdelavo zajete nehomogene slike.

3. Če se na sliki nahajajo zgolj objekti, katerih frekvenčni spekter se ne prekriva s frekvenčnim spektrom svetlostnih nehomogenosti, lahko sliko zgladimo z nizkoprepustnim sitom F tako, da se objekti zlijejo z ozadjem. Dobljena slika predstavlja oceno aditivne komponente nehomogenosti:

$$a(x, y) \approx F\left(g(x, y)\right). \tag{6.7}$$

(a) Ustvarite funkcijo `imShadingFilter`, ki obnovi sivinske vrednosti nehomogene slike `img` ter vrne homogeno sliko `oimg` in oceno aditivne komponente nehomogenosti `of`. Za glajenje slike lahko uporabite funkcijo `gaussian_filter` modula `scipy.ndimage.filters`, ki filtrira sliko z Gaussovim jedrom standardne deviacije `sigma`.

```
1 def imShadingFilter(img, sigma):
2   ...
3   return oimg, of
```



(a) Slika shading_1.png (b) Slika shading_2.png (c) Slika shading_3.png (d) Slika shading_4.png

Slika 6.3: Testne slike, ustvarjene z enakim poljem svetlostne nehomogenosti.

(b) Ocenite sivinsko nehomogenost testne slike shading_muscle.png. Izrišite obnovljeno sliko, pripadajoči histogram, vzdolžni sivinski prerez ter polje nehomogenosti. Poskrbite, da bosta povprečna vrednost in standardna deviacija sivinskih vrednosti nehomogene in obnovljene slike enaki.

(c) Na podoben način obnovite še sivinske vrednosti nehomogene mikroskopske slike shading_lung.png in testnih slik shading_1.png, shading_2.png, shading_3.png ter shading_4.png. Ali lahko s filtriranjem ocenite polje nehomogenosti za vse testne slike?

4. Obnovo prostorske homogenosti z multiplikativno komponento lahko izvedemo s (homomorfnim) filtriranjem v logaritemskem prostoru, kjer multiplikativna komponenta postane aditivna:

$$\log\left(g(x,y)\right) = \log\left(f(x,y)\cdot m(x,y)\right) = \log\left(f(x,y)\right) + \log\left(m(x,y)\right). \tag{6.8}$$

Logaritem multiplikativne komponente ocenimo s filtriranjem logaritma nehomogene slike $\log\left(m(x,y)\right) = F * log\left(g(x,y)\right)$. Nato z inverzno transformacijo (eksponent) izračunamo multiplikativno komponento nehomogenosti $m(x,y) = \mathrm{e}^{F*\log(g(x,y))}$. Sledi že opisani postopek obnove homogenosti z multiplikativno komponento:

$$f(x,y) = g(x,y)\cdot \mathrm{e}^{-F*\log(g(x,y))}. \tag{6.9}$$

(a) Ustvarite funkcijo `imShadingHomomorphic`, ki obnovi sivinske vrednosti nehomogene slike `img` s homomorfnim filtriranjem ter vrne homogeno sliko `oimg` in oceno multiplikativne komponente nehomogenosti `of` ($\mathrm{e}^{F*\log(g(x,y))}$). Parameter `sigma` določa standardno deviacijo Gaussovega filtra F.

```
1 def imShadingHomomorphic(img, sigma):
2   return oimg, of
```

(b) Obnovite homogenost slike shading_muscle.png ter izrišite histogram nehomogene in obnovljene slike ter vzdolžna sivinska prereza. Določite ustrezno standardno deviacijo `sigma` Gaussovega filtra F ter prikažite oceno multiplikativne komponente nehomogenosti. Poskrbite, da bosta povprečna vrednost in standardna deviacija sivinskih vrednosti nehomogene in obnovljene slike enaki.

(c) Na podoben način obnovite še sivinske vrednosti nehomogene mikroskopske slike shading_lung.png in testnih slik shading_1.png, shading_2.png, shading_3.png ter shading_4.png. Ali lahko s homomorfnim filtriranjem ocenite polje nehomogenosti za vse testne slike?

6.2 Rešitve in odgovori na vprašanja

Pri odgovorih privzamemo, da se funkcije iz rešitev nahajajo v modulu `funkcije`, slikovno gradivo pa se nahaja v podmapi `poglavje_6`. Naprej uvozimo potrebne module, naložimo slike in definiramo pomožne spremenljivke.

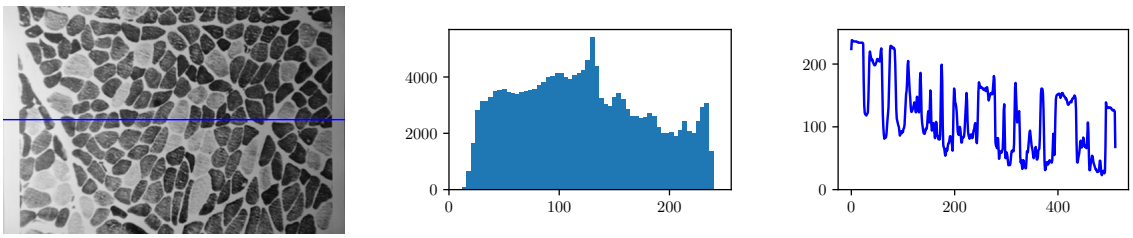
```
import numpy as np
from matplotlib import pyplot as pp
from PIL import Image as im
import funkcije

I = np.asarray(im.open('./poglavje_6/shading_muscle.png'))
HI, WI = I.shape
Id = np.asarray(im.open('./poglavje_6/shading_dark.png'))
Ib = np.asarray(im.open('./poglavje_6/shading_bright.png'))
nBins = 64

Il = np.asarray(im.open('./poglavje_6/shading_lung.png'))
HIl, WIl = Il.shape

Is1 = np.asarray(im.open('./poglavje_6/shading_1.png'))
Is2 = np.asarray(im.open('./poglavje_6/shading_2.png'))
Is3 = np.asarray(im.open('./poglavje_6/shading_3.png'))
Is4 = np.asarray(im.open('./poglavje_6/shading_4.png'))
HIs, WIs = Is1.shape
```

1. Kalibracija svetlostnih nehomogenosti. V modulu `funkcije` najprej ustvarimo funkcijo `imShadingCalibrate`.

```
def imShadingCalibrate(img, dark=None, bright=None, d=0, b=255):
  img = np.asarray(img, np.float)
  if dark is not None:
    dark = np.asarray(dark, np.float)
  if bright is not None:
    bright = np.asarray(bright, np.float)

  if bright is None:
    ofa = dark - d
    ofm = None;
    oimg= img - dark + d

  elif dark is None:
    ofa = None
    ofm = bright/b
    oimg = img*b/bright

  else:
    ofa = (d*dark  - b*bright)/float(b - d)
    ofm = (bright - dark)/float(b - d)
```

```
21      oimg = (img*(b - d) - b*dark + d*bright)/(bright - dark)
22
23  return oimg, ofa, ofm
```

(a) Izrišemo nehomogeno sliko shading_muscle.png s pripadajočim histogramom in vzdolžnim linijskim prerezom (slika 6.4).

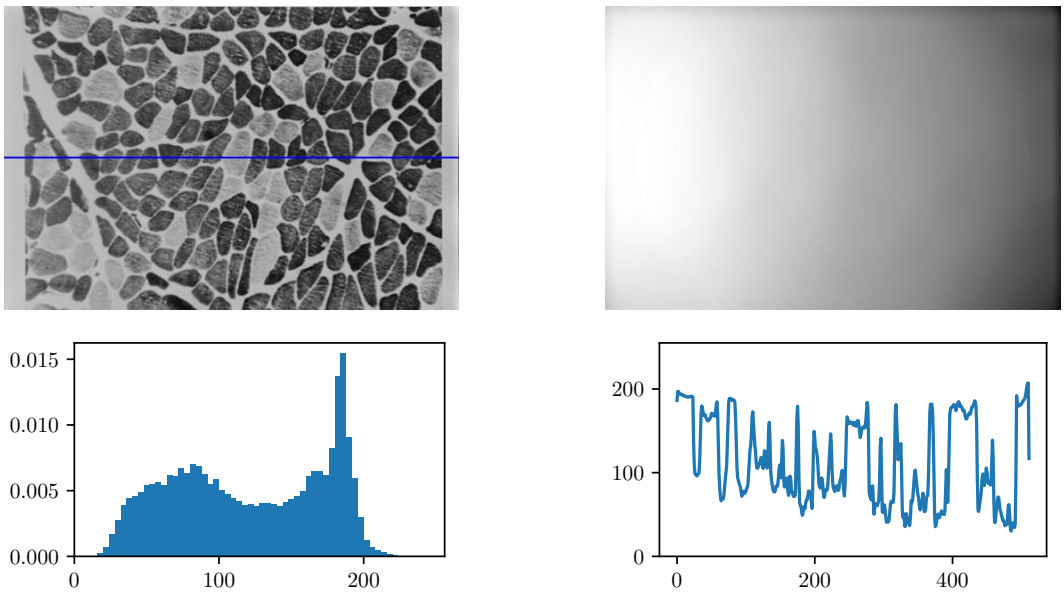


Slika 6.4: Nehomogena slika s pripadajočim histogramom in vzdolžnim linijskim prerezom.

```
1  pp.figure()
2
3  funkcije.imShowEx(I, title='Nehomogena', subplot=131,
4                    vmin=0, vmax=255)
5  pp.plot([0, WI - 1],[HI//2,HI//2], '-b')
6
7  pp.subplot(1, 3, 2)
8  pp.hist(I.flatten(), nBins, [0, 256], normed=True)
9
10 pp.subplot(1, 3, 3)
11 pp.plot(I[int(HI//2)])
12 pp.ylim([0, 255])
13
14 pp.show()
```

(b) Kalibriramo svetlostne nehomogenosti slike shading_muscle.png ter izrišemo kalibrirano sliko s pripadajočim histogramom in vzdolžnim linijskim prerezom (slika 6.5).

```
1  Ic, IcA, IcM = funkcije.imShadingCalibrate(I, Id, Ib, 0, 192)
2
3  pp.figure()
4
5  funkcije.imShowEx(Ic, title='Obnovljena', subplot=141,
6                    vmin=0, vmax=255)
7  pp.plot([0, WI - 1],[HI//2,HI//2], '-b')
8
9  funkcije.imShowEx(
10   IcM, title='Multiplikativno polje', subplot=142)
11
12 pp.subplot(1, 4, 3)
```



Slika 6.5: Svetlostno kalibrirana slika z izračunanim multiplikativnim poljem nehomogenosti, histogramom in vzdolžnim linijskim prerezom.

```
13 pp.hist(Ic.flatten(), nBins, [0, 256], normed=True)
14
15 pp.subplot(1, 4, 4)
16 pp.plot(Ic[int(HI//2)])
17 pp.ylim([0, 255])
18
19 pp.show()
```

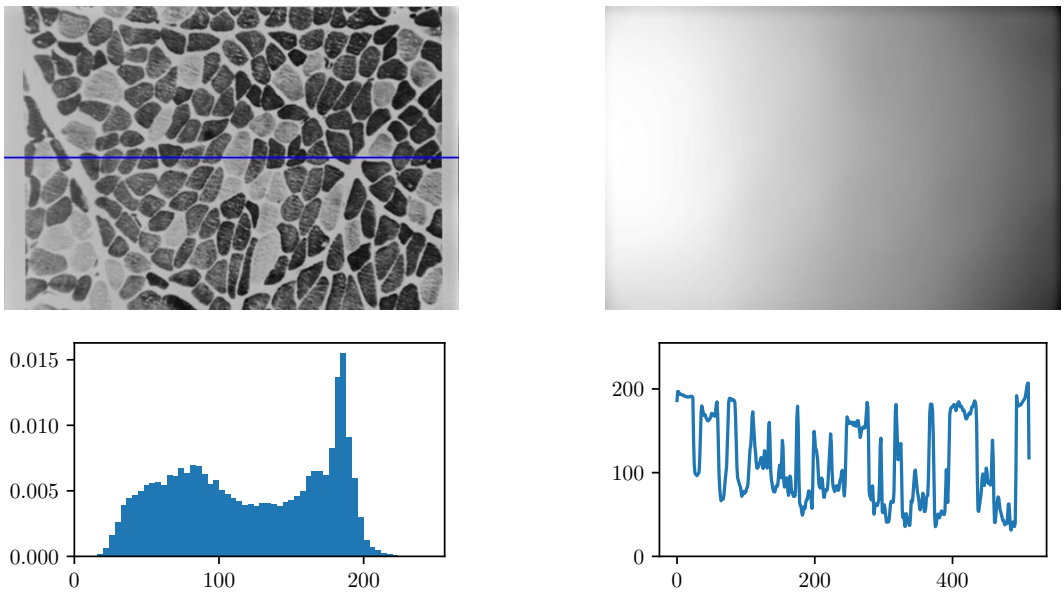
(c) Prostorska nehomogenost sivinskih vrednosti se odraža v nizkofrekvenčni komponenti linijskega prereza. V histogram vnaša razširitev porazdelitve sivinskih vrednosti, ki pripadajo določenemu tkivu.

2. Svetlostna kalibracija zgolj z multiplikativnim modelom.

(a) Glej funkcijo `imShadingCalibrate`.

(b) Kalibriramo sliko shading_muscle.png zgolj s svetlo kalibracijsko sliko (slika 6.6).

```
1 Ic1, tmp, Ic1M = funkcije.imShadingCalibrate(
2   I, bright=Ib, b=192)
3
4 pp.figure()
5 funkcije.imShowEx(Ic1, title='Obnovljena', subplot=141,
6                   vmin=0, vmax=255)
7 pp.plot([0, WI - 1],[HI//2,HI//2], '-b')
8
```



Slika 6.6: Svetlostno kalibrirana slika z izračunanim multiplikativnim poljem nehomogenosti, histogramom in vzdolžnim linijskim prerezom.

```
9  funkcije.imShowEx(
10   Ic1M, title='Multiplikativno polje', subplot=142)
11
12 pp.subplot(1, 4, 3)
13 pp.hist(Ic1.flatten(), nBins, [0, 256], normed=True)
14
15 pp.subplot(1, 4, 4)
16 pp.plot(Ic1[int(HI//2)])
17 pp.ylim([0, 255])
18
19 pp.show()
```

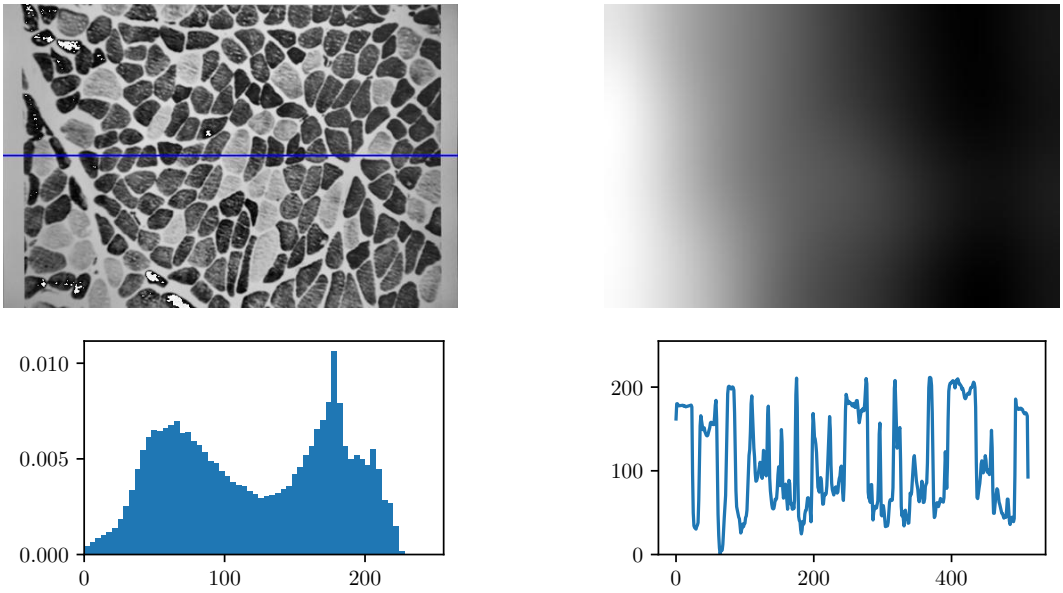
(c) Kalibracija slike z aditivnim modelom ni ustrezna, saj je svetlost temne kalibracijske slike zanemarljiva, svetlostna nehomogenost v svetli kalibracijski sliki pa izrazita.

3. Obnova svetlostne homogenosti s filtriranjem.

(a) V modulu `funkcije` ustvarimo funkcijo `imShadingFilter` ter jo preizkusimo na svetlostno nehomogeni sliki shading_muscle.png.

```
1 def imShadingFilter(img, sigma):
2   img = img.astype(np.float)
3   af = imGaussFilt2d(img, float(sigma), boundary='reflect')
4   oimg = img - af
5
6   return oimg, af
```

(b) Obnovimo svetlostno homogenost slike shading_muscle.png ter izrišemo obnovljeno sliko, oceno aditivnega polja nehomogenosti, histogram ter vzdolžni linijski prerez (slika 6.7). Uporabimo Gaussov filter $\sigma = 50$.

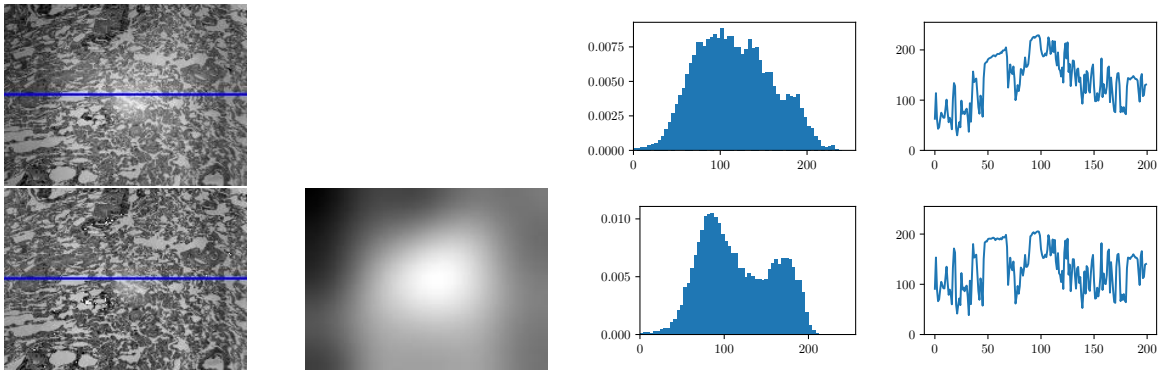


Slika 6.7: Svetlostno obnovljena slika z ocenjenim aditivnim poljem nehomogenosti, pripadajočim histogramom in vzdolžnim linijskim prerezom.

```
1  Ifc, IfA = funkcije.imShadingFilter(I, sigma=50)
2  Ifc = (Ifc - Ifc.mean())/Ifc.std()*I.std() + I.mean()
3
4  pp.figure()
5
6  funkcije.imShowEx(Ifc, title='Obnovljena',
7                    subplot=141, vmin=0, vmax=255)
8
9  funkcije.imShowEx(IfA, title='Multiplikativno polje',
10                   subplot=142)
11 pp.plot([0, WI - 1],[HI//2,HI//2], '-b')
12
13 pp.subplot(1, 4, 3)
14 pp.hist(Ifc.flatten(), nBins, [0, 256], normed=True)
15
16 pp.subplot(1, 4, 4)
17 pp.plot(Ifc[int(HI//2)])
18
19 pp.show()
```

(c) Obnovimo še svetlostno homogenost mikroskopske slike shading_lung.png (slika 6.8)

ter štirih sintetičnih slik shading_1.png, shading_2.png, shading_3.png in shading_4.png (slika 6.9).



Slika 6.8: Izvirna in svetlostno obnovljena slika z ocenjenim aditivnim poljem nehomogenosti ter pripadajoča histograma in vzdolžna linijska prereza.

```
Ilfc, IlfA = funkcije.imShadingFilter(Il, sigma=20)
Ilfc = (Ilfc - Ilfc.mean())/Ilfc.std()*Il.std() + Il.mean()

pp.figure()

funkcije.imShowEx(Ilfc, title='Obnovljena',
                  subplot=141, vmin=0, vmax=255)

funkcije.imShowEx(IlfA, title='Aditivno polje',
                  subplot=142)

pp.plot([0, WI - 1],[HI//2,HI//2], '-b')

pp.subplot(1, 4, 3)
pp.hist(Ilfc.flatten(), nBins, [0, 256], normed=True)

pp.subplot(1, 4, 4)
pp.plot(Ilfc[int(HIl//2)])


sigma = [7, 14, 28, 56]
Is = [Is1, Is2, Is3, Is4]
Ifsc = []
IfscA = []
for i in range(4):
  s = Is[i]
  sc, sca = funkcije.imShadingFilter(s, sigma=sigma[i])
  sc = (sc - sc.mean())/sc.std()*s.std() + s.mean()
  Ifsc.append(sc)
  IfscA.append(sca)
```

```
31
32    pp.figure()
33
34    funkcije.imShowEx(s, title='Izvirna', subplot=241,
35                      vmin=0, vmax=255)
36    pp.plot([0, WIs - 1],[HIs//2, HIs//2], '-b')
37
38    pp.subplot(2, 4, 3)
39    pp.hist(s.flatten(), nBins, [0, 256], normed=True)
40
41    pp.subplot(2, 4, 4)
42    pp.plot(s[int(HIs//2)])
43
44    funkcije.imShowEx(sc, title='Obnovljena',
45                      subplot=245, vmin=0, vmax=255)
46    pp.plot([0, WIs - 1],[HIs//2, HIs//2], '-b')
47
48    funkcije.imShowEx(sca, title='Aditivno polje',
49                      subplot=246)
50
51    pp.subplot(2, 4, 7)
52    pp.hist(sc.flatten(), nBins, [0, 256], normed=True)
53
54    pp.subplot(2, 4, 8)
55    pp.plot(sc[int(HIs//2)])
56
57  pp.show()
```

Z večanjem velikosti pravokotnih področij testnih slik narašča tudi napaka pri oceni polja nehomogenosti s postopkom filtriranja.

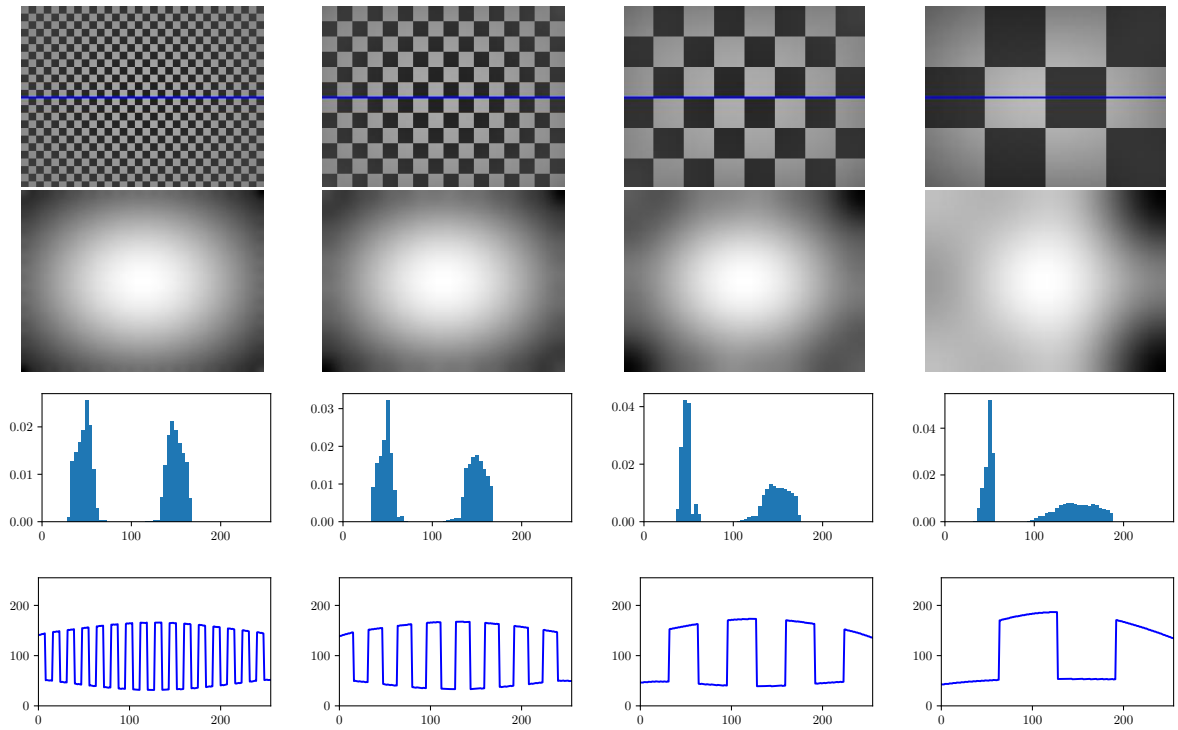
4. Obnova svetlostne homogenosti s homomorfnim filtriranjem.

   (a) V modulu `funkcije` ustvarimo funkcijo `imShadingHomomorphic` ter jo preizkusimo na svetlostno nehomogeni sliki shading_muscle.png.

```
1  def imShadingHomomorphic(img, sigma):
2    logimg = np.log(np.asarray(img, dtype=np.float) + 1.0)
3    logimgf = imGaussFilt2d(
4      logimg, float(sigma), boundary='reflect')
5    mf = np.exp(logimgf) - 1.0
6    oimg = img.astype(np.float)/mf
7
8    return oimg, mf
```

   (b) Obnovimo svetlostno homogenost slike shading_muscle.png ter izrišemo obnovljeno sliko, oceno multiplikativnega polja nehomogenosti, histogram ter vzdolžni linijski prerez (slika 6.10). Uporabimo Gaussov filter $\sigma = 50$.

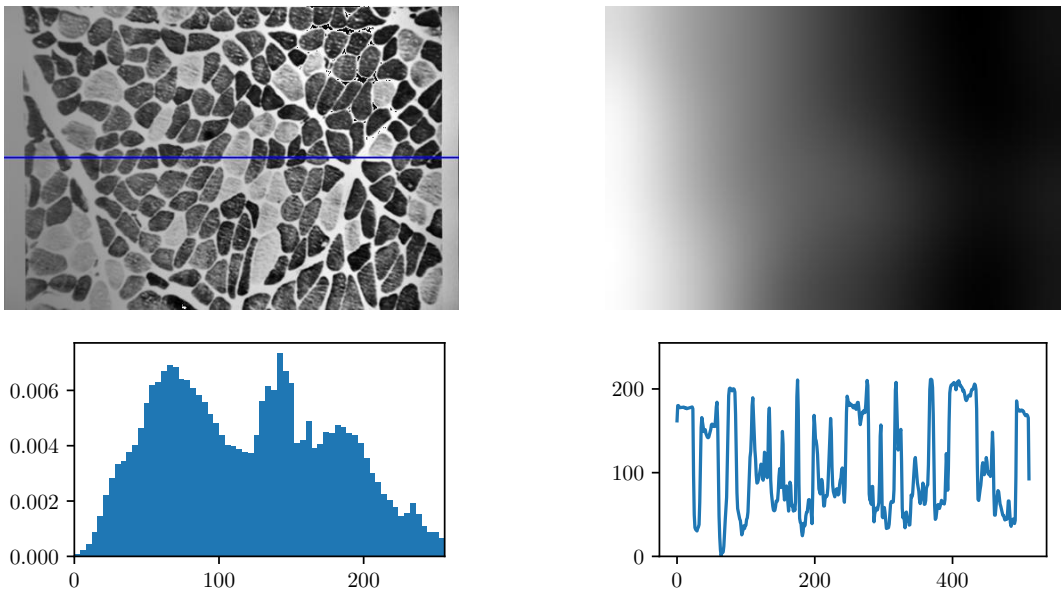
```
1  Ihfc, IhfM = funkcije.imShadingHomomorphic(I, sigma=50)
```



Slika 6.9: Obnovljene sintetične slike s pripadajočimi ocenami aditivnega polja svetlostne nehomogenosti, histogrami in vzdolžnimi linijskimi prerezi.

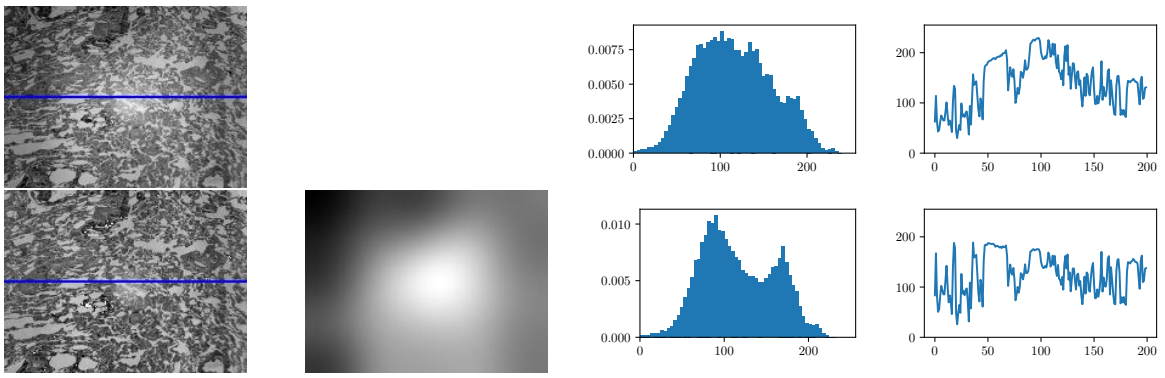
```
2  Ihfc = (Ihfc - Ihfc.mean())/Ihfc.std()*I.std() + \
3    I.mean()
4
5  pp.figure()
6
7  funkcije.imShowEx(Ihfc, title='Obnovljena',
8                    subplot=141, vmin=0, vmax=255)
9
10 funkcije.imShowEx(IhfM, title='Multiplikativno polje',
11                   subplot=142, vmin=0, vmax=255)
12 pp.plot([0, WI - 1],[HI//2, HI//2], '-b')
13
14 pp.subplot(1, 4, 3)
15 pp.hist(Ihfc.flatten(), nBins, [0, 256], normed=True)
16
17 pp.subplot(1, 4, 4)
18 pp.plot(Ihfc[int(HI//2)])
19
20 pp.show()
```

(c) Obnovimo še svetlostno homogenost mikroskopske slike shading_lung.png (slika 6.11)



Slika 6.10: Svetlostno obnovljena slika z ocenjenim multiplikativnim poljem nehomogenosti, pripadajočim histogramom in vzdolžnim linijskim prerezom.

ter štirih sintetičnih slik shading_1.png, shading_2.png, shading_3.png in shading_4.png (slika 6.12).

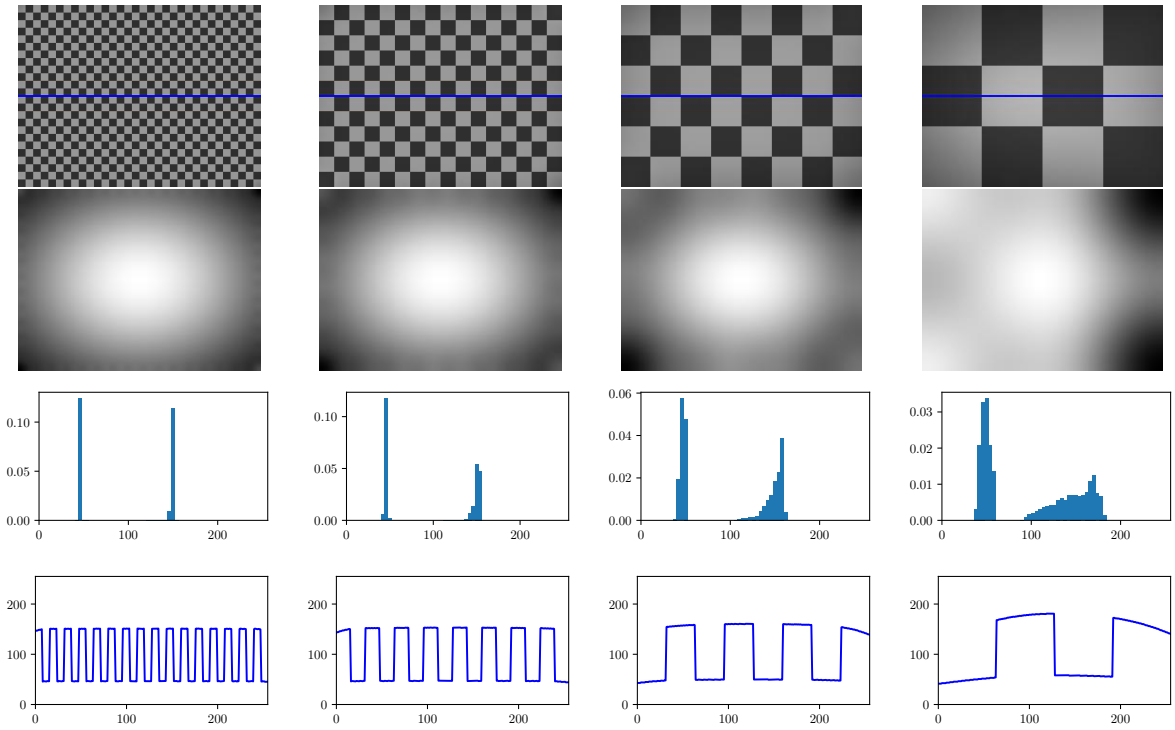


Slika 6.11: Izvirna in svetlostno obnovljena slika z ocenjenim multiplikativnim poljem nehomogenosti, pripadajoča histograma in vzdolžna linijska prereza.

```
1 Ilhfc, IlhfM = funkcije.imShadingHomomorphic(Il, sigma=20)
2 Ilhfc = (Ilhfc - Ilhfc.mean())/Ilhfc.std()*Il.std() + \
3   Il.mean()
4
5 pp.figure()
6
```

```
7  funkcije.imShowEx(Ilhfc, title='Obnovljena',
8                    subplot=141, vmin=0, vmax=255)
9
10 funkcije.imShowEx(IlhfM, title='Multiplikativno polje',
11                   subplot=142, vmin=0, vmax=255)
12 pp.plot([0, WIl - 1],[HIl//2, HIl//2], '-b')
13
14 pp.subplot(1, 4, 3)
15 pp.hist(Ilhfc.flatten(), nBins, [0, 256], normed=True)
16
17 pp.subplot(1, 4, 4)
18 pp.plot(Ilhfc[int(HIl//2)])
19
20
21 sigma = [7, 14, 28, 56]
22 Is = [Is1, Is2, Is3, Is4]
23 Ihfsc = []
24 IhfscM = []
25 for i in range(4):
26   s = Is[i]
27   sc, scm = funkcije.imShadingHomomorphic(s, sigma=sigma[i])
28   sc = (sc - sc.mean())/sc.std()*s.std() + s.mean()
29   Ihfsc.append(sc)
30   IhfscM.append(scm)
31
32   pp.figure()
33
34   funkcije.imShowEx(s, title='Izvirna',
35                     subplot=241, vmin=0, vmax=255)
36   pp.plot([0, WIs - 1],[HIs//2, HIs//2], '-b')
37
38   pp.subplot(2, 4, 3)
39   pp.hist(s.flatten(), nBins, [0, 256], normed=True)
40
41   pp.subplot(2, 4, 4)
42   pp.plot(s[int(HIs//2)])
43
44   funkcije.imShowEx(sc, title='Obnovljena', subplot=245)
45   pp.plot([0, WIs - 1],[HIs//2, HIs//2], '-b')
46
47   funkcije.imShowEx(scm, title='Multiplikativno polje',
48                     subplot=246, vmin=0, vmax=255)
49
50   pp.subplot(2, 4, 7)
51   pp.hist(sc.flatten(), nBins, [0, 256], normed=True)
52
53   pp.subplot(2, 4, 8)
54   pp.plot(sc[int(HIs//2)])
55
```

```
56 pp.show()
```



Slika 6.12: Obnovljene sintetične slike s pripadajočimi ocenami multiplikativnega polja svetlostne nehomogenosti, histogrami in vzdolžnimi linijskimi prerezi.

(d) Z večanjem velikosti pravokotnih področij testnih slik narašča tudi napaka pri oceni polja nehomogenosti s postopkom homomorfnega filtriranja.

Poglavje 7

Geometrijske preslikave slik

Z geometrijskimi preslikavami 2D slik $\mathcal{T} : R^2 \rightarrow R^2$ in 3D slik $\mathcal{T} : R^3 \rightarrow R^3$ preslikamo lokacije vseh slikovnih elementov iz (x, y) ali (x, y, z) v (x', y') oziroma (x', y', z'), pri tem pa ohranimo njihove sivinske vrednosti. Na ta način lahko izvedemo povečavo oziroma pomanjšavo (skaliranje), premik (translacijo), zasuk (rotacijo), pa tudi številne druge linearne in nelinearne geometrijske preslikave slik. Poljubno geometrijsko preslikavo 2D oziroma 3D slike zapišemo kot:

$$(x', y') = \mathcal{T}(x, y) \quad \text{ali} \quad (x', y', z') = \mathcal{T}(x, y, z). \tag{7.1}$$

Linearne preslikave $\mathcal{T}$ lahko zapišemo v matrični obliki T. V kolikor želimo tudi translacijo obravnavati kot linearno preslikavo, je potrebno uporabiti homogene koordinate $(x, y, 1)$ oziroma $(x, y, z, 1)$. Med linearnimi preslikavami je najbolj splošna afina preslikava, ki omogoča poljubno skaliranje, translacijo, rotacijo in strig. Afina preslikava je v 2D določena s 6, v 3D pa z 12 parametri:

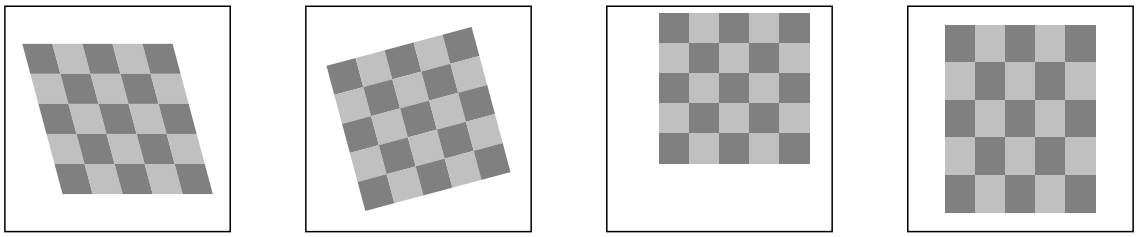
$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = T \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & t_x \\ a_{21} & a_{22} & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \text{ ali } \begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = T \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & t_x \\ a_{21} & a_{22} & a_{23} & t_y \\ a_{31} & a_{32} & a_{33} & t_y \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}, \tag{7.2}$$

kjer parametri t_x, t_y, t_z določajo translacijo slik v x, y, z smeri, parametri a_{ij} pa skaliranje, rotacijo in strig. Matriko afine preslikave T_{afina} lahko sestavimo z zaporednim matričnim množenje homogenih matrik elementarnih preslikav v želenem vrstnem redu (slika 7.1):

$$T_{afina} = T_{strig} \cdot T_{rot} \cdot T_{trans} \cdot T_{skal}. \tag{7.3}$$

V praksi se pogosto uporabljata toga preslikava, ki jo dobimo s kompozitumom rotacijske in translacijske elementarne preslikave, in podobnostna preslikava, ki jo dobimo s kompozitumom toge preslikave in skaliranja. Poleg linearnih se uporabljajo tudi nelinearne preslikave. Med njimi si bomo podrobneje ogledali projektivno preslikavo, ki je v 2D določena z 8, v 3D pa s 15 parametri. Projektivno preslikavo zapišemo z nehomogeno matriko (slika 7.3). Pri preslikavah digitalnih slik se diskretni slikovni elementi 2D referenčne slike (x_i, y_j) preslikajo na nove lokacije $\mathcal{T}(x_i, y_j) = (x'_i, y'_j)$, ki v splošnem ne sovpadajo z diskretno mrežo vzorčnih točk preslikane

(a) Strig T_{strig} (b) Rotacija T_{rot} (c) Translacija T_{trans} (d) Skaliranje T_{skal}



$$\begin{bmatrix} 1 & g_{xy} & 0 \\ g_{yx} & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \begin{bmatrix} \cos(\alpha) & -\sin(\alpha) & 0 \\ \sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \quad \begin{bmatrix} k_x & 0 & 0 \\ 0 & k_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} 1 & g_{xy} & g_{xz} & 0 \\ g_{yx} & 1 & g_{yz} & 0 \\ g_{zx} & g_{zy} & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad T_x \cdot T_y \cdot T_z \quad \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \begin{bmatrix} k_x & 0 & 0 & 0 \\ 0 & k_y & 0 & 0 \\ 0 & 0 & k_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Slika 7.1: Afina preslikava 2D in 3D slik.

$$\begin{bmatrix} \cos(\gamma) & -\sin(\gamma) & 0 & 0 \\ \sin(\gamma) & \cos(\gamma) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \begin{bmatrix} \cos(\beta) & 0 & \sin(\beta) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(\beta) & 0 & \cos(\beta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\alpha) & -\sin(\alpha) & 0 \\ 0 & \sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

(a) T_z (b) T_y (c) T_x

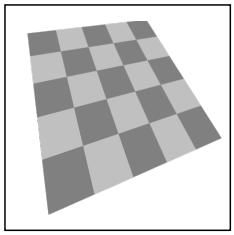
Slika 7.2: Homogene matrike, ki opisujejo 3D rotacijo okoli x, y in z koordinatnih osi.

slike (u_n, v_m). V točkah (u_m, v_n) je zato potrebno določiti sivinske vrednosti z interpolacijo. Ker pa se točke (x, y) v splošnem preslikajo na poljubno diskretno mrežo (x'_i, y'_j), lahko tudi neortogonalno, postane postopek interpolacije zelo zapleten, zato pri geometrijskem preslikovanju slik uporabljamo inverzno preslikavo $\mathcal{T}^{-1}(u_m, v_n) = (u'_m, v'_n)$, ki ohrani referenčno sliko na pravokotni diskretni mreži točk (slika 7.4).

7.1 Naloge in vprašanja

1. Ustvarite funkcijo `transformAffine2d`, ki zgradi transformacijsko matriko `T` poljubne 2D afine geometrijske preslikave, kjer je `scale` niz parametrov skaliranja k_x in k_y, `trans` niz parametrov translacije t_x in t_y, `rot` kot rotacije α v radianih in `shear` niz parametrov striga g_x in g_y. Matrično množenje najenostavneje izvedemo s funkcijo `dot` modula `numpy`.

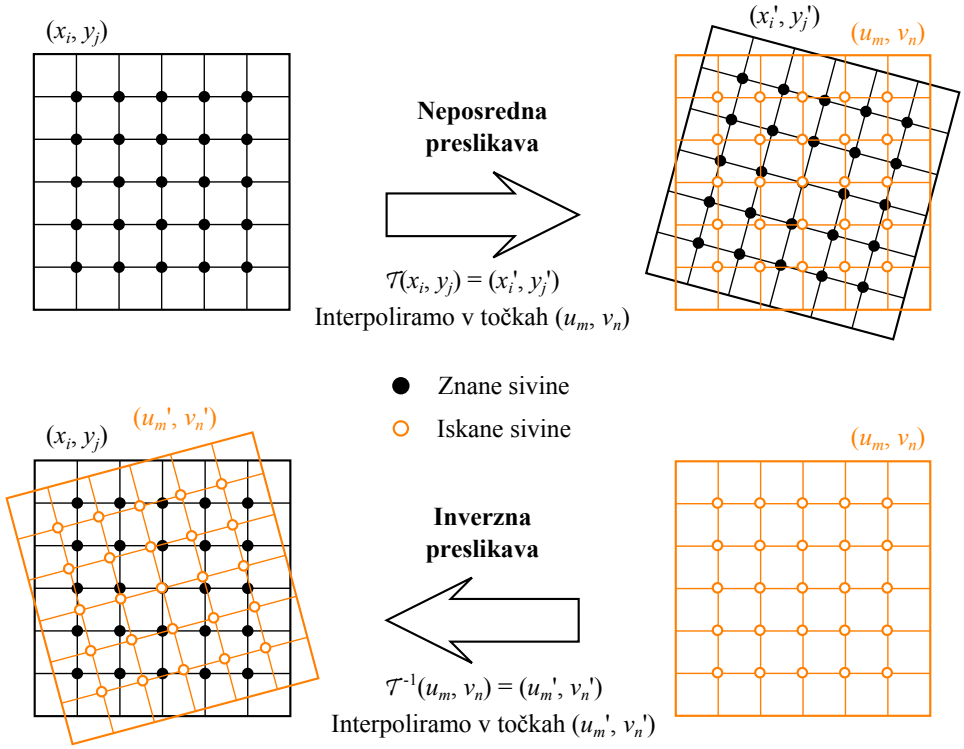
```
1 def transformAffine2d(scale=[1.0, 1.0], trans=[0.0, 0.0],
```



$$x' = \frac{a_{11}x + a_{12}y + t_x}{p_x x + p_y y + 1}$$
$$y' = \frac{a_{21}x + a_{22}y + t_y}{p_x x + p_y y + 1}$$
$$\begin{bmatrix} u \\ v \\ f \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & t_x \\ a_{21} & a_{22} & t_y \\ p_x & p_y & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$
$$x' = \frac{u}{f}$$
$$y' = \frac{v}{f}$$

$$x' = \frac{a_{11}x + a_{12}y + a_{13}z + t_x}{p_x x + p_y y + p_z z + 1}$$
$$y' = \frac{a_{21}x + a_{22}y + a_{23}z + t_y}{p_x x + p_y y + p_z z + 1}$$
$$z' = \frac{a_{31}x + a_{32}y + a_{33}z + t_z}{p_x x + p_y y + p_z z + 1}$$
$$\begin{bmatrix} u \\ v \\ w \\ f \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & t_x \\ a_{21} & a_{22} & a_{23} & t_y \\ a_{31} & a_{32} & a_{33} & t_z \\ p_x & p_y & p_z & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$
$$x' = \frac{u}{f}$$
$$y' = \frac{v}{f}$$
$$z' = \frac{w}{f}$$

Slika 7.3: Projektivna preslikava 2D in 3D slik.



Slika 7.4: Preslikave slikovnih elementov digitalnih slik.

```
                        rot=0.0, shear=[0.0, 0.0]):
  ...
  return T
```

(a) Ustvarite funkcijo `ptTransform2d`, ki s poljubno afino ali projektivno transformacijo preslika set točk `(x, y)` v `(ox, oy)`. Ko je vrednost parametra `inverse` enaka `True`, funkcija preslika točke z inverzom transformacije `T`. Inverz transformacijske matrike lahko robustno izračunamo z reševanjem linearnega sistema enačb, in sicer s funkcijo `solve` modula `numpy.linalg`.

```
def ptTransform2d(T, x, y, inverse=False):
  ...
  return ox, oy
```

(b) Ustvarite pravokotno polje referenčnih točk:

```
Y, X = meshgrid(np.arange(9), np.arange(12), indexing='ij')
```

Preizkusite delovanje funkcij `transformAffine2d` ter `ptTransform2d` tako, da set referenčnih točk rotirate za 10°, premaknete za $[10, 1]$, skalirate s faktorjem $[0.5, 0.5]$ in strižete s $[0.25, 0.5]$. Preizkusite naštete transformacije posamično in vse hkrati. S funkcijo `plot` modula `matplotlib.pylot` na isti graf izrišite referenčno (uporabite oznake `'xr'`) in transformirano (uporabite oznake `'xb'`) množico točk.

2. Ustvarite funkcijo `imTransform2d`, ki preslika vhodno sliko s poljubno afino ali projektivno preslikavo `T` in vrne transformirano sliko `oimg`. Ko je vrednost parametra `expand` enaka `'same'` naj se definicijsko območje slike ohrani, ko je `'crop'` naj definicijsko območje obsega celotno transformirano sliko, ko je `'full'` pa celotno transformirano sliko in definicijsko območje vhodne slike `img`. Parameter `center` naj določa koordinatno izhodišče preslikave. Vektorja točk `x` in `y` določata koordinatni sistem slike. Ko sta vrednosti vektorjev `x` in `y` enaki `None`, naj točka $(0.0, 0.0)$ predstavlja zgornje levo krajišče slike, velikost slikovnega elementa pa naj bo 1×1. Sivinske vrednosti transformirane slike interpolirajte s funkcijo `interp2` modula `interp`, red interpolacije pa naj bo določen s parametrom `order`.

```
def imTransform2d(img, T, x=None, y=None, center=(0.0, 0.0),
                  order=1, expand='same'):
  ...
  return oimg
```

(a) Preizkusite delovanje funkcije `imTransform2d` za različne afine preslikave. Uporabite sliko preslikave_ct.png.

(b) Izvedite rotacijo slike okoli geometričnega središča slike. Rešitev udejanite v obliki funkcije `imRotate2d`. Parameter `angle` naj določa kot rotacije v radianih, preostali parametri pa naj imajo enak pomen kot pri funkciji `imTransform2d`.

```
def imRotate2d(img, angle, order=1, expand='same'):
  ...
  return oimg
```

7.2 Rešitve in odgovori na vprašanja

Pri odgovorih privzamemo, da se funkcije iz rešitev nahajajo v modulu `funkcije`, slikovno gradivo pa se nahaja v podmapi `poglavje_7`. Naprej uvozimo potrebne module, naložimo slike in definiramo pomožne spremenljivke.

```
import numpy as np
from matplotlib import pyplot as pp
from PIL import Image as im
import funkcije

I = np.array(im.open('./poglavje_7/preslikave_ct.png'))
fi = 10/180*np.pi
```

1. V modulu `funkcije` ustvarimo funkcijo `transformAffine2d`.

```
def transformAffine2d(scale=[1.0, 1.0], trans=[0.0, 0.0],
                      rot=0.0, shear=[0.0, 0.0]):

  fi = rot

  S = np.array([[scale[0], 0, 0],
                [0, scale[1], 0],
                [0, 0, 1.0]], 'float')

  T = np.array([[1, 0, trans[0]],
                [0, 1, trans[1]],
                [0, 0, 1.0]], 'float')

  R = np.array([[np.cos(fi), -np.sin(fi), 0],
                [np.sin(fi), np.cos(fi), 0],
                [0, 0, 1.0]], 'float')

  H = np.array([[1, shear[0], 0],
                [shear[1], 1, 0],
                [0, 0, 1.0]], 'float')

  return np.dot(H, R).dot(T).dot(S)
```

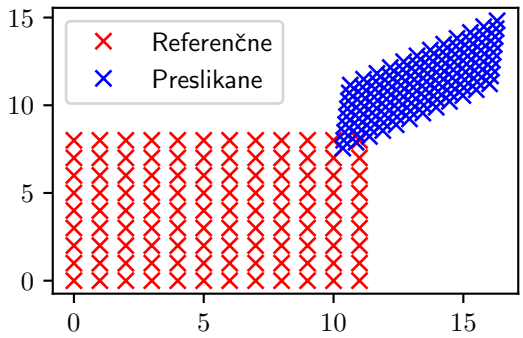
   (a) V modulu funkcije ustvarimo še funkcijo `ptTransform2d`.

```
def ptTransform2d(T, x, y, inverse=False):
  x = np.asarray(x, dtype=np.float)
  y = np.asarray(y, dtype=np.float)
  T = np.asarray(T)
  R = np.vstack(
    (x.flatten(),
     y.flatten(),
     np.ones([x.size]))
```

```
  )
  if inverse:
    Rt = np.linalg.solve(T, R)

  else:
    Rt = np.dot(T, R)

  ox, oy = Rt[0, :]/Rt[-1, :],  Rt[1, :]/Rt[-1, :]
  ox.shape, oy.shape = x.shape, y.shape

  return ox, oy
```

(b) Preverimo delovanje funkcij `transformAffine2d` in `ptTransform2d` (slika 7.5).

```
Y, X = np.meshgrid(
  np.arange(9), np.arange(12), indexing='ij')
T1 = funkcije.transformAffine2d(
  scale=[0.5, 0.5], trans=[10, 1],
  rot=fi, shear=[0.25, 0.5])
Xt1, Yt1 = funkcije.ptTransform2d(T1, X, Y)

pp.figure()
pp.plot(X.flatten(), Y.flatten(), 'rx',
  label='Referenčne')
pp.plot(Xt1.flatten(), Yt1.flatten(), 'bx',
  label='Preslikane')
pp.legend(loc='upper left')
pp.show()
```

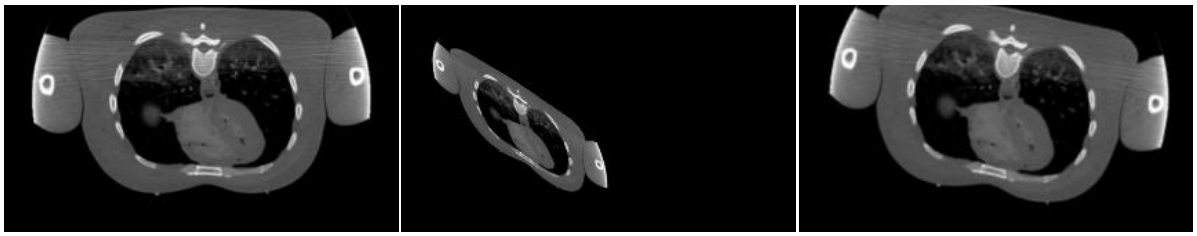


Slika 7.5: Mreža referenčnih in preslikanih točk za podano afino transformacijo.

2. V modulu `funkcije` ustvarimo funkcijo `imTransform2d`.

```
def imTransform2d(img, T, x=None, y=None, center=(0.0, 0.0),
                  order=1, expand='same'):
  H, W = img.shape
```

```
4   if x is None:
5     x = np.arange(W) - center[0]
6   if y is None:
7     y = np.arange(H) - center[1]
8
9   dx, dy = x[1] - x[0], y[1] - y[0]
10
11  if expand == 'same':
12    xmin, xmax = x.min(), x.max()
13    ymin, ymax = y.min(), y.max()
14
15  elif expand in ['full', 'crop']:
16    # oglišča referenčne slike v preslikani sliki določajo meje razširitve
17    etx, ety = ptTransform2d(t,
18      [x[0], x[0],  x[-1], x[-1]],
19      [y[0], y[-1], y[0], y[-1]])
20
21    xmin, xmax = np.floor(etx).min(), np.floor(etx).max()
22    ymin, ymax = np.ceil(ety).min(), np.ceil(ety).max()
23
24    if expand == 'full':
25      xmin = min(x.min(), xmin)
26      xmax = min(x.max(), xmax)
27      ymin = min(y.min(), ymin)
28      ymax = min(y.max(), ymax)
29
30  xt = np.arange(xmin, xmax + dx, dx)
31  yt = np.arange(ymin, ymax + dy, dy)
32  Yt, Xt = np.meshgrid(yt, xt, indexing='ij')
33  Xi, Yi = ptTransform2d(T, Xt, Yt, inverse=True)
34
35  oimg = interp.interp2(x, y, img.astype(np.float), Xi, Yi)
36
37  return oimg
```



(a) Izvirna slika (b) Primer afine transformacije. (c) Primer rotacije.

Slika 7.6: Primera afine trasformacije slike preslikave_ct.png.

(a) Preverimo delovanje funkcije `imTransform2d` (slika 7.6b).

```
It = funkcije.imTransform2d(I, T1)
```

(b) V modulu `funkcije` ustvarimo še funkcijo `imRotate2d`.

```
def imRotate2d(img, angle, order=1, expand='same'):
  H, W = img.shape
  x, y = np.arange(W), np.arange(H)
  x -= x.mean()
  y -= y.mean()
  T = transformAffine2d(rot=angle)

  return imTransform2d(img, T, x, y,
                       expand=expand, order=order)
```

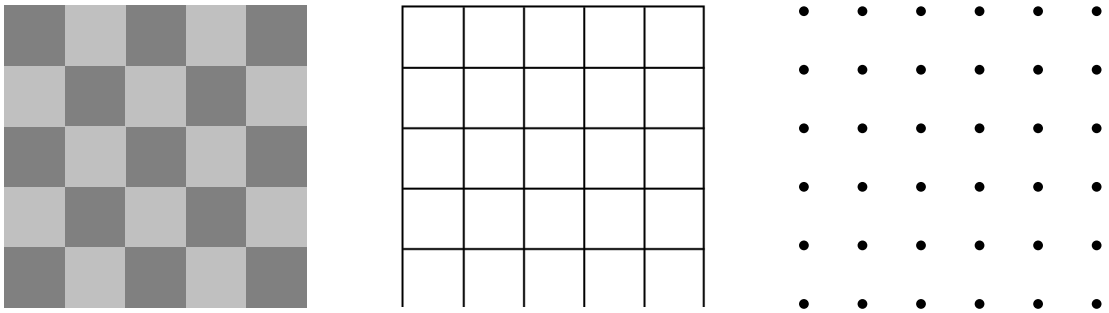
Preverimo še delovanje funkcije `inRotate2d` (slika 7.6c).

```
Ir = funkcije.imRotate2d(I, fi)

pp.figure()

funkcije.imShowEx(I, subplot=131, title='Izvirna')
funkcije.imShowEx(It, subplot=132, title='Transformirana')
funkcije.imShowEx(Ir, subplot=133, title='Rotirana')

pp.show()
```

Poglavje 8

Geometrijska kalibracija slik

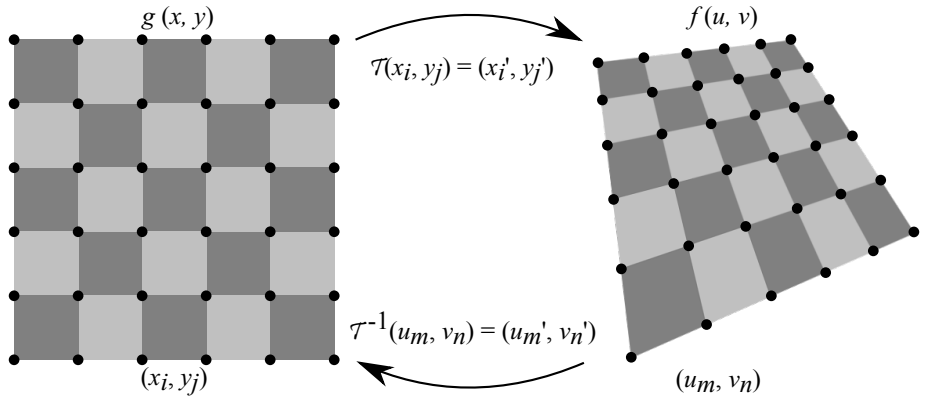
S postopkom geometrijske kalibracije slikovnega sistema zagotovimo konsistentno absolutno velikost slikovnega elementa, ki je v splošnem lahko odvisna od lege na sliki (projekcija) in dodatno popačena zaradi optičnih aberacij, med katerimi so najpogostejše radialne distorzije tipa sodček in blazinica. Po izvedeni geometrijski kalibraciji slikovnega sistema lahko enostavno izvajamo meritve dimenzij, ploščin in volumnov objektov na slikah v absolutnih enotah (npr. mm, mm^2, mm^3). Za geometrijsko kalibracijo 2D slik uporabljamo kalibre z dobro definiranimi periodičnimi geometričnimi strukturami, kot so šahovnica, mreža tankih črt ali polje točk (slika 8.1). S postopkom kalibracije geometrijsko poravnamo strukture zajete $f(u, v)$ in referenčne $g(x, y)$



(a) Šahovnica (b) Mreža črt (c) Polje točk

Slika 8.1: Primeri kalibracijskih vzorcev.

slike kalibracijskega objekta (slika 8.2). V ta namen lahko uporabimo celotno slikovno informacijo ali pa izluščimo korespondenčne pare točk referenčne in kalibracijske slike ter s pomočjo njih določimo parametre geometrijske preslikave (poravnave) tako, da se preslikane točke referenčne slike čim bolje ujemajo s korespondenčnimi točkami kalibracijske slike in obratno. Glede na število parov korespondenčnih točk in vrsto preslikave lahko točke preslikamo tako, da se popolnoma prekrivajo, tj. $\mathcal{T}(x_i, y_j) = (u_i, v_j)$, kar imenujemo interpolacijska poravnava, ali pa tako, da se točke le približno prekrijejo, tj. $\mathcal{T}(x_i, y_j) \approx (u_i, v_j)$, kar imenujemo aproksimacijska poravnava.



Slika 8.2: Geometrijska kalibracija s poravnavo struktur zajete in referenčne slike.

8.1 Toga poravnava

Toga poravnava je vedno aproksimacijska, saj že zahtevano minimalno število parov korespondenčnih točk vodi do predoločenega sistema enačb, tako da dobimo več enačb kot je neznanih parametrov. Za določitev parametrov 2D toge preslikave potrebujemo vsaj $K \geq 2$ parov korespondenčnih točk, preslikavo pa določimo z minimizacijo povprečne kvadratne evklidske razdalje E med točkami:

$$\begin{aligned} E &= \frac{1}{K}\sum_{k=0}^{K-1} \|T_{toga}(x_k, y_k) - (u_k, v_k)\|^2, \\ E &= \frac{1}{K}\sum_{k=0}^{K-1} \left((x_k \cos\alpha - y_k \sin\alpha + t_x - u_k)^2 + (x_k \sin\alpha + y_k \cos\alpha + t_y - v_k)^2 \right), \end{aligned} \tag{8.1}$$

ki jo odvajamo po parametrih t_x, t_y in α ter odvode postavimo na nič:

$$\begin{aligned} \frac{\partial E}{\partial t_x} =& \frac{2}{K}\sum_{k=0}^{K-1} \left((x_k \cos\alpha - y_k \sin\alpha + t_x - u_k) \right) = 2(\bar{x}\cos\alpha - \bar{y}\sin\alpha + t_x - \bar{u}) = 0, \\ \frac{\partial E}{\partial t_y} =& \frac{2}{K}\sum_{k=0}^{K-1} \left((x_k \sin\alpha + y_k \cos\alpha + t_y - v_k) \right) = 2(\bar{x}\sin\alpha + \bar{y}\cos\alpha + t_y - \bar{v}) = 0, \\ \frac{\partial E}{\partial \alpha} =& \frac{2}{K}\sum_{k=0}^{K-1} \left((x_k \cos\alpha - y_k \sin\alpha + t_x - u_k)\,(-x_k \sin\alpha - y_k \cos\alpha) \right) + \\ & \frac{2}{K}\sum_{k=0}^{K-1} \left((x_k \sin\alpha + y_k \cos\alpha + t_y - v_k)\,(x_k \cos\alpha - y_k \sin\alpha) \right) \\ =& 2\left(\overline{xu}\sin\alpha + \overline{yu}\cos\alpha - \overline{xv}\cos\alpha + \overline{yv}\sin\alpha\right) - \\ & 2t_x\left(\bar{x}\sin\alpha + \bar{y}\cos\alpha\right) + 2t_y\left(\bar{x}\cos\alpha - \bar{y}\sin\alpha\right) = 0. \end{aligned} \tag{8.2}$$

Rešitev sistema enačb je:

$$\begin{aligned} t_x &= \bar{u} - \bar{x}\cos(\alpha) + \bar{y}\sin(\alpha), \\ t_y &= \bar{v} - \bar{x}\sin(\alpha) - \bar{y}\cos(\alpha), \\ \alpha &= -\arctan(\frac{\overline{yu} - \overline{xv} - \bar{y}\cdot\bar{u} + \bar{x}\cdot\bar{v}}{\overline{xu} + \overline{yv} - \bar{x}\cdot\bar{u} - \bar{y}\cdot\bar{v}}), \end{aligned} \tag{8.3}$$

kjer elementi s črto označujejo povprečne vrednosti:

$$\begin{aligned} \bar{x} &= \frac{1}{K}\sum_{k=0}^{K-1} x_k, & \bar{y} &= \frac{1}{K}\sum_{k=0}^{K-1} y_k, \\ \bar{u} &= \frac{1}{K}\sum_{k=0}^{K-1} u_k, & \bar{v} &= \frac{1}{K}\sum_{k=0}^{K-1} v_k, \\ \overline{xu} &= \frac{1}{K}\sum_{k=0}^{k=K-1} x_k u_k, & \overline{yu} &= \frac{1}{K}\sum_{k=0}^{k=K-1} y_k u_k, \\ \overline{xv} &= \frac{1}{K}\sum_{k=0}^{k=K-1} x_k v_k, & \overline{yv} &= \frac{1}{K}\sum_{k=0}^{k=K-1} y_k v_k. \end{aligned} \tag{8.4}$$

8.2 Afina poravnava

Afina poravnava je v 2D enolično določena s tremi ($K = 3$) pari korespondenčnih točk (u_i, v_i) in (x_i, y_j):

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & t_x \\ a_{21} & a_{22} & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \Rightarrow T_{afina} = \begin{bmatrix} u_1 & u_2 & u_3 \\ v_1 & v_2 & v_3 \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} x_1 & x_2 & x_3 \\ y_1 & y_2 & y_3 \\ 1 & 1 & 1 \end{bmatrix}^{-1}. \tag{8.5}$$

2D afino aproksimacijsko poravnavo pa lahko uporabimo takrat, ko poznamo več kot tri pare korespondenčnih točk ($K > 3$). V tem primeru dobimo predoločen sistem enačb, zato lahko korespondenčne točke poravnamo le približno (aproksimacijsko). To storimo tako, da minimiziramo povprečno kvadratno evklidsko razdaljo E med korespondenčnimi točkami:

$$\begin{aligned} E &= \frac{1}{K}\sum_{k=0}^{K-1} \|T_{afina}(x_k, y_k) - (u_k, v_k)\|^2, \\ E &= \frac{1}{K}\sum_{k=0}^{K-1} \left((x_k a_{11} + y_k a_{12} + t_x - u_k)^2 + (x_k a_{21} + y_k a_{22} + t_y - v_k)^2\right). \end{aligned} \tag{8.6}$$

Optimalne vrednosti neznanih parametrov preslikave dobimo tako, da odvode povprečne kvadratne evklidske razdalje po vseh parametrih postavimo na nič, in s tem dobimo sledeči sistem

linearnih enačb za parametre preslikave:

$$\begin{aligned}
\frac{\partial E}{\partial a_{11}} &= \frac{2}{K}\sum_{k=0}^{K-1}\left(x_k a_{11} + y_k a_{12} + t_x - u_k\right) x_k = 2\left(\overline{xx}a_{11} + \overline{xy}a_{12} + \overline{x}t_x - \overline{xu}\right) = 0,\\
\frac{\partial E}{\partial a_{12}} &= \frac{2}{K}\sum_{k=0}^{K-1}\left(x_k a_{11} + y_k a_{12} + t_x - u_k\right) y_k = 2\left(\overline{xy}a_{11} + \overline{yy}a_{11} + \overline{y}t_x - \overline{yu}\right) = 0,\\
\frac{\partial E}{\partial t_x} &= \frac{2}{K}\sum_{k=0}^{K-1}\left(x_k a_{11} + y_k a_{12} + t_x - u_k\right) = 2\left(\overline{x}a_{11} + \overline{y}a_{11} + t_x - \overline{u}\right) = 0,\\
\frac{\partial E}{\partial a_{21}} &= \frac{2}{K}\sum_{k=0}^{K-1}\left(x_k a_{21} + y_k a_{22} + t_y - v_k\right) x_k = 2\left(\overline{xx}a_{21} + \overline{xy}a_{22} + \overline{x}t_y - \overline{xv}\right) = 0,\\
\frac{\partial E}{\partial a_{22}} &= \frac{2}{K}\sum_{k=0}^{K-1}\left(x_k a_{21} + y_k a_{22} + t_y - v_k\right) y_k = 2\left(\overline{xy}a_{21} + \overline{yy}a_{22} + \overline{y}t_y - \overline{yv}\right) = 0,\\
\frac{\partial E}{\partial t_y} &= \frac{2}{K}\sum_{k=0}^{K-1}\left(x_k a_{21} + y_k a_{22} + t_y - v_k\right) = 2\left(\overline{x}a_{21} + \overline{y}a_{22} + t_y - \overline{v}\right) = 0.
\end{aligned} \tag{8.7}$$

Elementi s črto označujejo povprečne vrednosti:

$$\begin{aligned}
\bar{x} &= \frac{1}{K}\sum_{k=0}^{K-1} x_k, & \bar{y} &= \frac{1}{K}\sum_{k=0}^{K-1} y_k,\\
\overline{xx} &= \frac{1}{K}\sum_{k=0}^{K-1} x_k^2, & \overline{yy} &= \frac{1}{K}\sum_{k=0}^{K-1} y_k^2,\\
\bar{u} &= \frac{1}{K}\sum_{k=0}^{K-1} u_k, & \bar{v} &= \frac{1}{K}\sum_{k=0}^{K-1} v_k,\\
\overline{xu} &= \frac{1}{K}\sum_{k=0}^{k=K-1} x_k u_k, & \overline{yu} &= \frac{1}{K}\sum_{k=0}^{k=K-1} y_k u_k,\\
\overline{xv} &= \frac{1}{K}\sum_{k=0}^{k=K-1} x_k v_k, & \overline{yv} &= \frac{1}{K}\sum_{k=0}^{k=K-1} y_k v_k.
\end{aligned} \tag{8.8}$$

Zgornji sistem enačb lahko preoblikujemo v pregledno matrično obliko:

$$\begin{bmatrix}
\overline{xx} & \overline{xy} & \overline{x} & 0 & 0 & 0\\
\overline{xy} & \overline{yy} & \overline{y} & 0 & 0 & 0\\
\overline{x} & \overline{y} & 1 & 0 & 0 & 0\\
0 & 0 & 0 & \overline{xx} & \overline{xy} & \overline{x}\\
0 & 0 & 0 & \overline{xy} & \overline{yy} & \overline{y}\\
0 & 0 & 0 & \overline{x} & \overline{y} & 1
\end{bmatrix}
\begin{bmatrix} a_{11}\\ a_{12}\\ t_x\\ a_{21}\\ a_{22}\\ t_y \end{bmatrix} =
\begin{bmatrix} \overline{xu}\\ \overline{yu}\\ \overline{u}\\ \overline{xv}\\ \overline{yv}\\ \overline{v} \end{bmatrix}. \tag{8.9}$$

8.3 Projektivna poravnava

Projektivno preslikavo $T_{projektivna}$ smo spoznali že v poglavju 7. Njene parametre lahko določimo s postopkom optimizacije, ki poišče takšne vrednosti parametrov preslikave, pri katerih je vrednost kvadratne evklidske razdalje E med podanimi korespondenčnimi točkami minimalna:

$$\begin{aligned} E &= \frac{1}{K}\sum_{k=0}^{K-1} \|(u_k, v_k) - T_{projektivna}(x_k, y_k)\|^2, \\ T_{projektivna} &= \arg\min(E). \end{aligned} \tag{8.10}$$

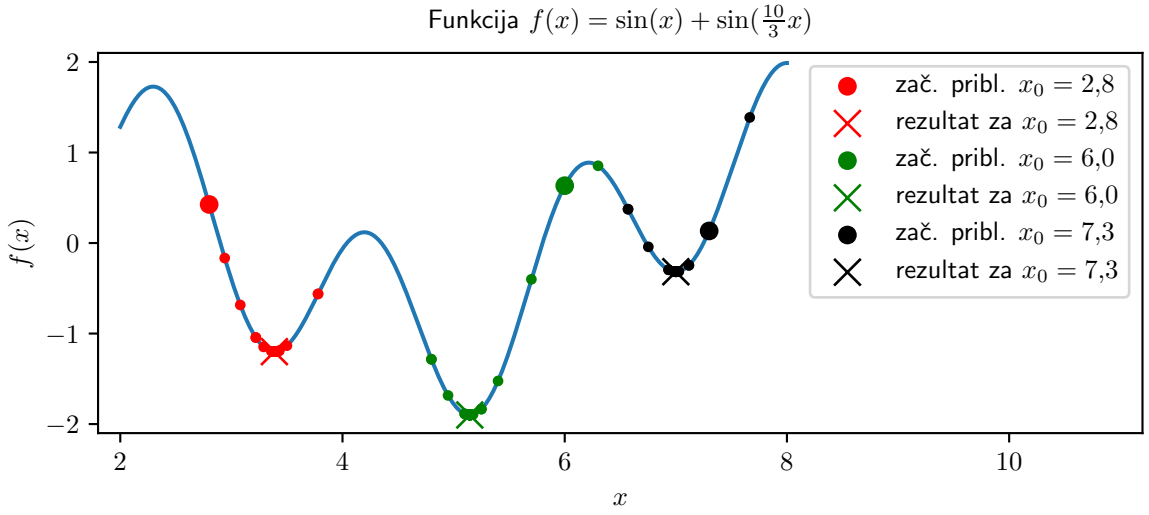
Za iskanje optimalnih vrednosti parametrov projektivne preslikave bomo uporabili optimizacijsko funkcijo `fmin` modula `scipy.optimize`, ki poišče minimum večrazsežne funkcije. V našem primeru bomo iskali minimum funkcije E (enačba 8.10). Uporabo funkcije `fmin` za iskanja minimuma enostavne dvoparametrične konveksne funkcije $f(x, y) = x^2 + y^2$, ilustrira sledeči primer.

```
1  from scipy.optimize import fmin
2
3  # funkcija, katere minimum iščemo z fmin
4  def fun(x):
5    f = x[0]**2 + x[1]**2
6    # izpišemo parametre in funkcijsko vrednost v vsaki iteraciji
7    print(f, x)
8    return f
9
10 # začetni približek parametrov funkcije
11 x0 = (1.0, 1.0)
12 # poiščemo minimum funkcije
13 x = fmin(fun, x0)
14 # izpišemo ocenjene vrednosti parametrov funkcije v njenem minimumu
15 print('Minimum fukcije x*x + y*y:')
16 print(' x = ', x[0])
17 print(' y = ', x[1])
```

Pri funkcijah, ki izkazujejo več lokalnih minimumov, je rezultat optimizacije odvisen od vrednosti začetnega približka. Tako je pri funkciji $f(x) = \sin x + \sin\left(\frac{10}{3}x\right)$ (slika 8.3), ki na intervalu $[2, 8]$ izkazuje tri lokalne minimume, rezultat optimizacije odvisen od vrednosti začetnega približka x_0. Z začetnimi približki $x_0 = 2{,}8$, $x_0 = 6{,}0$ in $x_0 = 7{,}3$ postopek optimizacije vsakokrat zaključimo v drugem/najbližjem lokalnem minimumu funkcije $f(x)$.

8.4 Odprava radialnih distorzij

V praksi se za geometrijsko kalibracijo slikovnih sistemov pogosto uporablja afini ali projektivni model geometrijske preslikave, po potrebi pa še ustrezni model za odpravo radialnih distorzij tipa sodček in blazinica (slika 8.4). Za ta namen lahko uporabimo Brawnov model radialnih

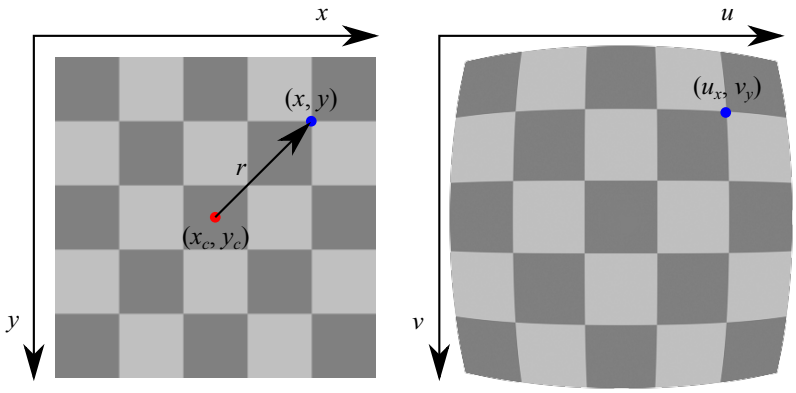


Slika 8.3: Potek in končni rezultat optimizacije funkcije $f(x) = \sin x + \sin\left(\frac{10}{3}x\right)$ v odvisnosti od vrednosti začetnega približka x_0.

distorzij, ki je določen s koordinatami središča distorzij (x_c, y_c) ter utežmi radialnih funkcij K_i:

$$\begin{aligned} u_x &= x_c + (x - x_c)(1 + K_1 r^2 + K_2 r^4 + ...), \\ v_y &= y_c + (y - y_c)(1 + K_1 r^2 + K_2 r^4 + ...), \\ r &= \sqrt{(x - x_c)^2 + (y - y_c)^2)} \\ (x_c, y_c) &- \text{ koordinate središča radialnih distorzij} \\ K_n &- \text{ uteži radialnih funkcij} \end{aligned} \tag{8.11}$$

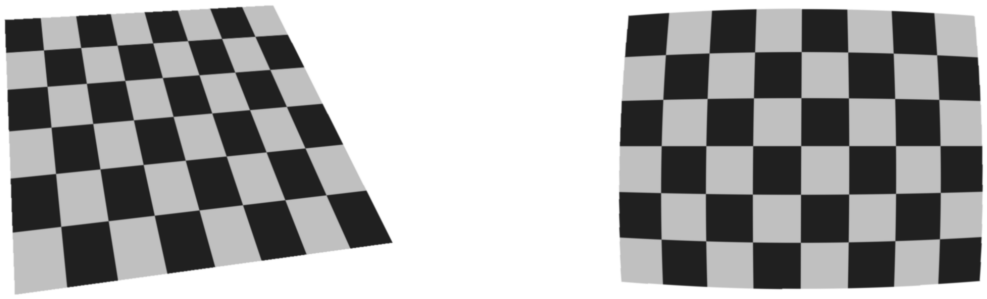
Kalibracijo izvedemo tako, da najprej opravimo transformacijo z modelom radialnih distorzij in



Slika 8.4: Radialne distorzije

šele nato uporabimo projektivno preslikavo. Ker je Brawnov model radialnih distorzij nelinearen, je za izračun parametrov modela potrebno uporabiti optimizacijski postopek, ki minimizira ustrezno kriterijsko funkcijo, npr. povprečno kvadratno evklidsko razdaljo med izbranimi korespondenčnimi točkami. V ta namen lahko uporabimo že omenjeno optimizacijsko funkcijo `fmin` modula `scipy.optimize`

8.5 Naloge in vprašanja



(a) cal_barrel.png (b) cal_projective.png

Slika 8.5: Sliki kalibracijskega objekta.

1. Ustvarite funkcijo `transformEstimate2D` za določanje parametrov geometrijskih preslikav, kjer `x` in `y` ter `u` in `v` določajo koordinate korespondenčnih točk, parameter `kind` pa tip aproksimacijske preslikave, `'rigid'` za togo, `'affine'` za afino ter `'projective'` za projektivno. Pri tem si pomagajte s funkcijo `fmin` modula `scipy.optimize`. Ko je vrednost parametra `animate=True`, v skupni graf izrišite referenčne in preslikane točke v vsaki iteraciji izračuna parametrov aproksimacijske projektivne preslikave. Parametre projektivne aproksimacijske preslikave določite tako, da neposredno minimizirate povprečno vrednost kvadratne evklidske razdalje med korespondenčnimi točkami. Začetne vrednosti parametrov projektivne aproksimacijske preslikave aproksimirajte s parametri afine aproksimacijske preslikave. Funkcija naj vrne izračunano aproksimacijsko preslikavo `T` (numpy polje velikosti 3×3) in koren povprečne kvadratne evklidske razdalje med korespondenčnimi točkami `err`.

```
1 def transformEstimate2D(x, y, u, v, kind='rigid', animate=False):
2   ...
3   return T, err
```

2. Naložite sliko cal_projective.png (slika 8.5) kalibracijskega objekta ter s pomočjo nekaj izbranih korespondenčnih točk izračunajte parametre aproksimacijske toge poravnave med kalibracijsko in referenčno sliko kalibracijskega objekta. Predpostavite, da stranice posameznih kvadratnih področij kalibracijskega objekta merijo 1 mm. Prikažite sliko kalibracijskega objekta pred in po transformaciji z izračunano aproksimacijsko preslikavo.

Kalibracijo izvedite tako, da bo na kalibrirani sliki velikost slikovnega elementa znašala 0,02 mm.

(a) Izračunajte parametre aproksimacijske afine preslikave, ki poravnajo naloženo sliko z referenčno sliko kalibracijskega objekta. Določite parametre afine interpolacijske preslikave tako, da izberete tri primerne korespondenčne točke. Prikažite sliko kalibracijskega objekta pred in po transformaciji z izračunano aproksimacijsko afino preslikavo. Določite še parametre aproksimacijske projektivne preslikave. Prikažite sliko kalibracijskega objekta pred in po transformaciji z izračunano projektivno preslikavo.

(b) Ali je afina preslikava primerna za geometrijsko kalibracijo podane slike?

3. Naložite sliko cal_barrel.png (slika 8.5) kalibracijskega objekta ter s pomočjo nekaj korespondenčnih točk izračunajte parametre afine in projektivne aproksimacijske preslikave med kalibracijsko sliko in referenčno sliko kalibracijskega objekta. Predpostavite, da stranice posameznih kvadratnih področij kalibracijskega objekta merijo 1 mm. Prikažite sliko kalibracijskega objekta pred in po transformaciji z izračunano aproksimacijsko preslikavo. Ali sta toga in afina preslikava zadostni za kalibracijo slikovnega sistema? Kalibracijo izvedite tako, da bo na kalibrirani sliki velikost slikovnega elementa 0,02 mm.

(a) Kalibracijo izvedite še z uporabo Brawnovega modela radialnih distorzij (prvi red), tako da minimizirate povprečno kvadratno evklidsko razdaljo med korespondenčnimi točkami s funkcijo `fmin` modula `scipy.optimize`. Parametri optimizacije naj obsegajo le parametre modela radialnih distorzij, parametre aproksimativne projektivne preslikave pa določite vsakokrat posebej. Začetne vrednosti parametrov radialne transformacije K_i postavite na 0, center radialnih distorzij pa v geometrično središče slike. V ta namen dopolnite funkcijo `transformEstimate2d` z dodatnim parametrom `nr`, ki določa red radialnih distorzij. Funkcija naj vrne parametre aproksimacijske projektivne preslikave `tp`, parametre radialnih distorzij `tr` (vektor $[x_c, y_c, K_1, K_2, ...]$) in koren povprečne kvadratne evklidske razdalje med korespondenčnimi točkami `err`.

```
1 def transformEstimate2d(x, y, u, v, kind='rigid', nr=1,
2                         animate=False):
3   ...
4   elif kind == 'radial':
5   ...
6     return tp, tr, err
```

Prikažite transformirano sliko kalibracijskega objekta.

(b) Postopek kalibracije nekajkrat ponovite, tako da vsakokrat ponovno označite izbrane točke na sliki kalibracijskega objekta. Na podlagi dobljenih rezultatov ocenite natančnost opisanega postopka geometrijske kalibracije.

8.6 Rešitve in odgovori na vprašanja

Pri odgovorih privzamemo, da se funkcije iz rešitev nahajajo v modulu `funkcije`, slikovno gradivo pa se nahaja v podmapi `poglavje_8`. Najprej naložimo testni sliki in ustvarimo pomožne spremenljivke.

```
\begin{python}
import os.path
import funkcije
import numpy as np
import interp
from matplotlib import pyplot as pp
from PIL import Image as im

# naložimo sliko kalibracijskega objekta
I1 = np.array(im.open('./poglavje_8/cal_projective.png'))
H1, W1 = I1.shape
xg1, yg1 = np.arange(W1), np.arange(H1)
# definiramo referenčne točke
xr1 = np.array([0, 4, 8, 8, 8, 4, 0, 0], dtype=np.float)
yr1 = np.array([0, 0, 0, 3, 6, 6, 6, 3], dtype=np.float)
# definiramo koordinatno mrežo točk kalibrirane slike
x1c = np.arange(0, 8.0 + 0.01, 0.02)
y1c = np.arange(0, 6.0 + 0.01, 0.02)
Y1c, X1c = np.meshgrid(y1c, x1c, indexing='ij')

# naložimo sliko kalibracijske tarče
I2 = np.array(im.open('./poglavje_8/cal_barrel.png'))
H2, W2 = I2.shape
xg2, yg2 = np.arange(W2), np.arange(H2)
# definiramo referenčne točke
xr2 = xr1
yr2 = yr1
# definiramo koordinatno mrežo točk kalibrirane slike
x2c, y2c = x1c, y1c
X2c, Y2c = X1c, Y1c
```

1. V modulu `funkcije` ustvarimo funkcijo `transformEstimate2d`.

```
def transformEstimate2d(x, y, u, v, kind='rigid', nr=1,
                        animate=False):
  x = np.asarray(x).astype(np.float).flatten()
  y = np.asarray(y).astype(np.float).flatten()
  u = np.asarray(u).astype(np.float).flatten()
  v = np.asarray(v).astype(np.float).flatten()
  t = tr = err = None

  if kind == 'rigid':
    xm = x.mean()
```

```
11      um = u.mean()
12      ym = y.mean()
13      vm = v.mean()
14      yum = np.mean(y*u)
15      xvm = np.mean(x*v)
16      xum = np.mean(x*u)
17      yvm = np.mean(y*v)
18      a = -np.arctan(
19        (yum - xvm - ym*um + xm*vm)/
20        (xum +  yvm - xm*um - ym*vm))
21      tx = um - xm*np.cos(a) + ym*np.sin(a)
22      ty = vm - xm*np.sin(a) - ym*np.cos(a)
23      t = np.dot(
24        transformAffine2d(trans=[tx, ty]),
25        transformAffine2d(rot=a))
26      tx, ty = ptTransform2d(t, x, y)
27      err = (((u - tx)**2 + (v - ty)**2).mean())**0.5
28      return t, err
29
30    elif kind == 'affine':
31      xxm = np.mean(x*x)
32      xym = np.mean(x*y)
33      yym = np.mean(y*y)
34      xm = x.mean()
35      ym = y.mean()
36      uxm = np.mean(x*u)
37      uym = np.mean(u*y)
38      vxm = np.mean(v*x)
39      vym= np.mean(y*v)
40      um = u.mean()
41      vm = v.mean()
42      t = np.zeros([3,3])
43      t[-1, -1] = 1.0
44      tvec = np.linalg.solve(
45        np.array(
46          [[xxm, xym, xm, 0, 0, 0],
47          [xym, yym, ym, 0, 0, 0],
48          [xm, ym, 1, 0, 0, 0],
49          [0, 0, 0, xxm, xym, xm],
50          [0, 0, 0, xym, yym, ym],
51          [0, 0, 0, xm, ym, 1]]
52        ),
53        np.array([uxm, uym, um, vxm, vym, vm]))
54      t[0] = tvec[:3]
55      t[1] = tvec[3:]
56      tx, ty = ptTransform2d(t, x, y)
57      err = (((u - tx)**2 + (v - ty)**2).mean())**0.5
58      return t, err
59
```

```
60    elif kind == 'projective':
61      if animate:
62        pp.figure()
63      t0, err = transformEstimate2d(x, y, u, v, 'affine')
64      t0 = t0.flatten()[:-1]
65      topt = opt.fmin(
66        lambda t: kProjective2d(t, x, y, u, v, animate), t0)
67      t = np.ones([9], 'float')
68      t[:8] = topt
69      t.shape = [3, 3]
70      err = kProjective2d(topt, x, y, u, v, False)
71      return t, err
72
73    elif kind == 'radial':
74      if animate:
75        pp.figure()
76      Tproj = np.zeros([3, 3])
77      xc = x.mean()
78      yc = y.mean()
79      tr0 = np.zeros([nr + 2])
80      tr0[0] = xc
81      tr0[1] = yc
82      tropt = opt.fmin(
83        lambda tr:
84          kProjectiveRadial2d(tr, x, y, u, v, Tproj, animate),
85        tr0)
86      tr = tropt
87      err = kProjectiveRadial2d(tropt, x, y, u, v, Tproj, False)
88      return Tproj, tr, err
89
90    return t, tr, err
```

V modulu `funkcije` ustvarimo še kriterijsko funkcijo `kProjective2d`, ki jo potrebujemo za izračun vrednosti parametrov projektivne aproksimacijske preslikave.

```
1  def kProjective2d(t, x, y, u, v, animate):
2    T = np.zeros([9])
3    T[:8] = t
4    T[-1] = 1.0
5    T.shape = (3,3)
6    ue, ve = ptTransform2d(T, x, y)
7    if animate:
8      pp.cla()
9      pp.plot(u, v, 'xr')
10     pp.plot(ue, ve, 'xb')
11     pp.draw()
12     pp.pause(0.01)
13
14   return (((u - ue)**2 + (v - ve)**2).mean())**0.5
```

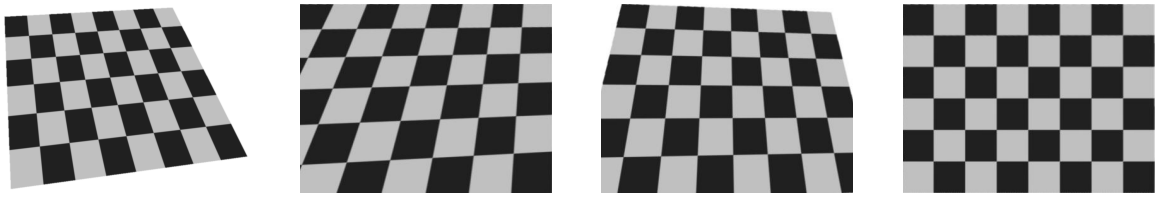
Sledi geometrijska kalibracija slike cal_projective.png.

```
# ročno označevanje referenčnih točk
pp.figure()
if os.path.isfile('./poglavje_8/rezultati/ref_pts_1.npy'):
  t = np.load('./poglavje_8/rezultati/ref_pts_1.npy')
else:
  pp.imshow(I1, cmap='gray')
  pp.title(
    'Označi oglišča in razpolovišča stranic. '
    'Prični levo zgoraj, nadaljuj v smeri urinega kazalca..')
  t = np.asarray(pp.ginput(8, timeout=600))
  np.save('./poglavje_8/rezultati/ref_pts_1.npy', t)
pp.close()
u1, v1 = t[:,0], t[:,1]

# kalibracija z interpolacijsko afino preslikavo
# uporabimo levo zgornje, desno zgornje in desno spodnje krajišče
inds = [0, 2, 4]
T1Ai, err1Ai = funkcije.transformEstimate2d(
  xr1[inds], yr1[inds], u1[inds], v1[inds], 'affine')
x1Ai, y1Ai = funkcije.ptTransform2d(T1Ai, u1, v1, inverse=True)
tmpx, tmpy = funkcije.ptTransform2d(T1Ai, X1c, Y1c)
I1Ai = interp.interp2(tmpx, tmpy, xg1, yg1, I1, cval=255)

# kalibracija z afino aproksimacijsko preslikavo
T1A, err1A = funkcije.transformEstimate2d(
  xr1, yr1, u1, v1, 'affine')
x1A, y1A = funkcije.ptTransform2d(T1A, u1, v1, inverse=True)
tmpx, tmpy = funkcije.ptTransform2d(T1A, X1c, Y1c)
I1A = interp.interp2(tmpx, tmpy, xg1, yg1, I1, cval=255)

# kalibracija s projektivno preslikavo
T1P, err1P = funkcije.transformEstimate2d(
  xr1, yr1, u1, v1, 'projective', animate=False)
x1P, y1P = funkcije.ptTransform2d(T1P, u1, v1, inverse=True)
tmpx, tmpy = funkcije.ptTransform2d(T1P, X1c, Y1c)
I1P = interp.interp2(tmpx, tmpy, xg1, yg1, I1, cval=255)

print('Kalibracija slike cal_projective.png:')
print('  Napaka za interp. afini model:', err1Ai)
print('  Napaka za aproks. afini model:', err1A)
print('  Napaka za proj. model:', err1P)

pp.figure()

pp.subplot(2, 2, 1)
pp.plot(xr1, yr1, 'xr', label='Izvirna')
pp.plot(x1A, y1A, 'xb', label='Afina')
pp.plot(x1P, y1P, 'xg', label='Projektivna')
```

```
pp.legend(loc='upper right')

pp.subplot(2, 2, 2)
pp.imshow(I1, cmap='gray')
pp.title('Izvirna')

pp.subplot(2, 2, 3)
pp.imshow(I1A, cmap='gray')
pp.title('Afina')

pp.subplot(2, 2, 4)
pp.imshow(I1P, cmap='gray')
pp.title('Projektivna')

pp.show()
```

2. Izvedemo zahtevano kalibracijo slike cal_projective.png

   (a) Parametre interpolacijske afine preslikave določimo na podlagi treh korespondenčnih točk, ki se nahajajo v levem zgornjem, desnem zgornjem in desnem spodnjem krajišču kalibracijskega objekta.



(a) Zajeta slika (b) Afina interp. (c) Afina aproks. (d) Projektivna

Slika 8.6: Geometrijska kalibracija slike cal_projective.png z afino in projektivno preslikavo.

   (b) Za kalibracijo zajete slike je potrebno uporabiti projektivno preslikavo. Koren povprečne kvadratne evklidske razdalje med korespondenčnimi točkami v prostoru zajete slike znaša približno 50 slikovnih elementov za aproksimacijsko afino in 0,5 slikovnega elementa za projektivno preslikavo.

3. V modulu `funkcije` ustvarimo kriterijsko funkcijo `kProjectiveRadial2d`, ki jo potrebujemo za izračun parametrov radialnega modela distorzij.

```
def kProjectiveRadial2d(t, x, y, u, v, Tproj, animate):
  # najprej izvedemo radialno preslikavo
  xr, yr = ptTransformRadial2d(t, x, y)

  # in se aproksimacijska projektivna preslikava
  t, err = transformEstimate2d(
    xr, yr, u, v, 'projective', False)
  [ue, ve] = ptTransform2d(t, xr, yr)
```

```
9    Tproj[:, :] = t
10
11   if animate:
12     pp.cla()
13     pp.plot(u, v, 'xr')
14     pp.plot(ue, ve, 'xb')
15     pp.draw()
16     pp.pause(0.01)
17
18   return (((u - ue)**2 + (v - ve)**2).mean())**0.5
```

V modulu `funkcije` ustvarimo še funkcijo `ptTransformRadial2d`, ki točke preslika z radialno preslikavo.

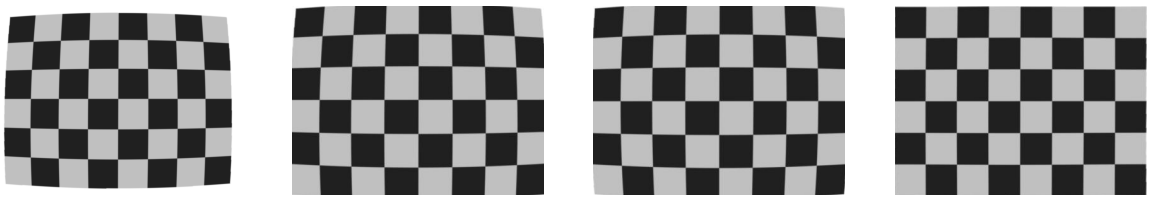
```
1  def ptTransformRadial2d(tr, x, y):
2    x = np.asarray(x)
3    y = np.asarray(y)
4
5    tr = np.asarray(tr, 'float')
6
7    xc = tr[0]
8    yc = tr[1]
9    K = tr[2:]
10
11   rr = (x - xc)**2 + (y - yc)**2
12   sr = 1
13   for i in range(len(K)):
14     sr += K[i]*(rr**(i + 1.0))
15
16   xt = xc + (x - xc)*sr
17   yt = yc + (y - yc)*sr
18
19   return xt, yt
```

(a) Z afino in projektivno preslikavo ne moremo izvesti zadovoljive geometrijske kalibracije slike cal_barrel.png (slika 8.7). Koren povprečne kvadratne evklidske razdalje med korespondenčnimi točkami v prostoru zajete slike znaša približno 10 slikovnih elementov za afino, 7 slikovnih elementov za projektivno in 0,5 slikovnega elementa za kombinacijo radialne in projektivne preslikave. Na koncu geometrijsko kalibriramo še sliko cal_barrel.png.

```
1  pp.figure()
2  if os.path.isfile('./poglavje_8/rezultati/ref_pts_2.npy'):
3    t = np.load('./poglavje_8/rezultati/ref_pts_2.npy')
4  else:
5    pp.imshow(I2, cmap='gray')
6    pp.title(
7      'Izberi oglišča in razpolovišča stranic. '
8      'Prični levo zgoraj.')
```

```
9   t = np.asarray(pp.ginput(8, timeout=600))
10  np.save('./poglavje_8/rezultati/ref_pts_2.npy', t)
11 pp.close()
12 u2, v2 = t[:,0], t[:,1]
13
14 # kalibracija z aproksimacijsko afino preslikavo
15 T2A, err2A = funkcije.transformEstimate2d(
16   xr2, yr2, u2, v2, 'affine')
17 tmpx, tmpy = funkcije.ptTransform2d(T2A, X2c, Y2c)
18 I2A = interp.interp2(tmpx, tmpy, xg2, yg2, I2, cval=255)
19
20 # kalibracija s projektivno preslikavo
21 T2P, err2P = funkcije.transformEstimate2d(
22   xr2, yr2, u2, v2, 'projective')
23 tmpx, tmpy = funkcije.ptTransform2d(T2P, X2c, Y2c)
24 I2P = interp.interp2(tmpx, tmpy, xg2, yg2, I2, cval=255)
25
26 # a - kalibracija z radialnim Brownovim modelom
27 T2RP, T2RB, err2RP = funkcije.transformEstimate2d(
28   xr2, yr2, u2, v2, 'radial', animate=True)
29 tmpx, tmpy = funkcije.ptTransformRadial2d(T2RB, X2c, Y2c)
30 tmpx, tmpy = funkcije.ptTransform2d(T2RP, tmpx, tmpy)
31 I2RP = interp.interp2(tmpx, tmpy, xg2, yg2, I2, cval=255)
32
33 print('Kalibracija slike cal_barrel.png:')
34 print('  Napaka za afini model preslikave:', err2A)
35 print('  Napaka za proj. model preslikave:', err2P)
36 print('  Napaka za proj. in rad. model preslikave:', err2RP)
37
38 # napaka pri kalibraciji z afino, projektivno in
39 # radialno/projektivno preslikavo
40 print(err2A, err2P, err2RP)
41
42 pp.figure()
43
44 pp.subplot(2, 2, 1)
45 pp.imshow(I2, cmap='gray')
46 pp.title('Zajeta')
47
48 pp.subplot(2, 2, 2)
49 pp.imshow(I2A, cmap='gray')
50 pp.title('Afina preslikava')
51
52 pp.subplot(2, 2, 3)
53 pp.imshow(I2P, cmap='gray')
54 pp.title('Projektivna preslikava')
55
56 pp.subplot(2, 2, 4)
57 pp.imshow(I2RP, cmap='gray')
```

```
58 pp.title('Radialna preslikava')
59
60 tmpx, tmpy = funkcije.ptTransformRadial2d(T2RB, xr2, yr2)
61 x2RP, y2RP = funkcije.ptTransform2d(T2RP, tmpx, tmpy)
62
63 pp.figure()
64
65 pp.plot(u2, v2, 'xr', label='Zajeta')
66 pp.plot(x2RP, y2RP, 'xb', label='Radialna + Projektivna')
67 pp.legend(loc='upper right')
68
69 pp.show()
```



(a) Zajeta slika (b) Afina (c) Projektivna (d) Radialna/projektivna

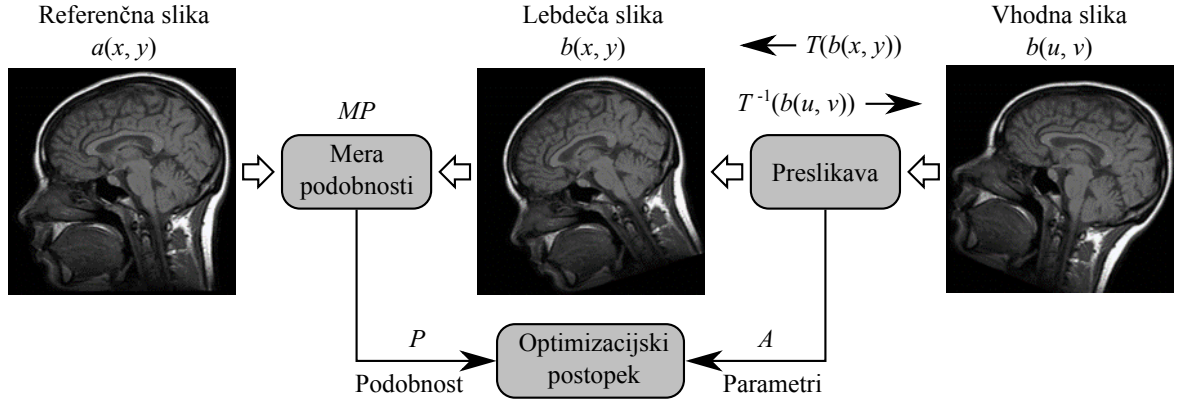
Slika 8.7: Geometrijska kalibracija slike cal_barrel.png z afino, projektivno ter s kombinacijo radialne in projektivne preslikave.

(b) Z nekajkratno ponovitvijo postopka kalibracije pridemo do ocene natančnosti, ki znaša približno 0,3 slikovnega elementa zajete slike.

Poglavje 9

Geometrijska poravnava slik s postopkom optimizacije

Geometrijska poravnava dveh ali več slik je proces iskanja optimalnih geometrijskih preslikav T, ki nam slike preslikajo tako, da se iste strukture oziroma objekti slik po poravnavi nahajajo na enakih položajih. Pred postopkom poravnave je potrebno izbrati model geometrijske preslikave, mero podobnosti ter optimizacijski postopek [8]. Tekom postopka poravnave nam izbrana optimizacijska metoda iterativno spreminja parametre geometrijske preslikave, in sicer tako, da teži k optimalni vrednosti mere podobnosti. Glavna prednost geometrijske poravnave slik s postopkom optimizacije je v splošnosti, saj ne zahteva predhodne izbire kontrolnih točk in jo zato lahko relativno preprosto avtomatiziramo. Naj bo $a(x, y)$ referenčna slika, $b(u, v)$ pa poljubna slika, ki jo želimo poravnati z referenčno sliko a. Postopek poravnave slik z optimizacijo mere podobnosti prikazuje slika 9.1. Mera podobnosti je poljubna skalarna funkcija, določena nad



Slika 9.1: Poravnava slik z optimizacijo mere podobnosti.

vsemi istoležnimi slikovnimi elementi referenčne slike $a(x, y)$ in lebdeče slike $b(x, y)$, ki ima optimalno vrednost pri optimalni poravnavi slik. Mero podobnosti je potrebno smiselno izbrati, tako da je čim manj občutljiva na motilna slikovna neskladja in čim bolj občutljiva na dejanska geometrijska neskladja med slikama. Ob predpostavki, da sta velikosti referenčne in lebdeče

slike enaki (M, N), lahko zapišemo sledeče primere mer podobnosti:

1. Srednja kvadratna napaka MSE (ang. Mean Square Error):

$$MSE(a,b) = \frac{1}{M \cdot N} \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} (a(x_i, y_j) - b(x_i, y_j))^2. \tag{9.1}$$

2. Srednja absolutna napaka MAE (ang. Mean Absolute Error):

$$MAE(a,b) = \frac{1}{M \cdot N} \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} |a(x_i, y_j) - b(x_i, y_j)|. \tag{9.2}$$

3. Korelacijski koeficient CC (ang. Correlation Coefficient):

$$\begin{aligned} CC(a,b) &= \frac{\sum_{i=0}^{M-1} \sum_{j=0}^{N-1} (a(x_i, y_j) - \bar{a}) \cdot (b(x_i, y_j) - \bar{b})}{\sqrt{\sum_{i=0}^{M-1} \sum_{j=0}^{N-1} (a(x_i, y_j) - \bar{a})^2 \cdot \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} (b(x_i, y_j) - \bar{b})^2}}, \\ \bar{a} &= \frac{1}{M \cdot N} \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} a(x_i, y_j), \\ \bar{b} &= \frac{1}{M \cdot N} \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} b(x_i, y_j). \end{aligned} \tag{9.3}$$

4. Medsebojna informacija MI (ang. Mutual Information):

$$MI(a,b) = H(a) + H(b) - H(a,b), \tag{9.4}$$

kjer je $H(a)$ entropija referenčne slike $a(x, y)$, $H(b)$ entropija lebdeče slike $b(x, y)$, $H(a, b)$ pa njuna skupna entropija:

$$\begin{aligned} H(a) &= -\sum_{l_a=0}^{L} p_a(l_a) \cdot \log\left(p_a(l_a)\right), \\ H(b) &= -\sum_{l_b=0}^{L} p_b(l_b) \cdot \log\left(p_b(l_b)\right), \\ H(a,b) &= -\sum_{l_a=0}^{L} \sum_{l_b=0}^{L} p_{ab}(l_a, l_b) \cdot \log\left(p_{ab}(l_a, l_b)\right). \end{aligned} \tag{9.5}$$

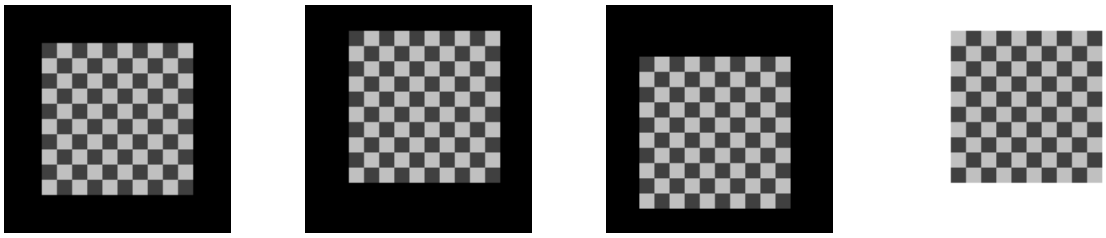
Verjetnostni porazdelitvi $p_a(l_a)$ in $p_b(l_b)$ ter skupno porazdelitev $p_{ab}(l_a, l_b)$ ocenimo z normalizacijo pripadajočih histogramov $h_a(l_a)$, $h_b(l_b)$ ter $h_{ab}(l_a, l_b)$:

$$\begin{aligned} p_a(l_a) &= \frac{h_a(l_a)}{M \cdot N}, \\ p_b(l_b) &= \frac{h_b(l_b)}{M \cdot N}, \\ p_{ab}(l_a, l_b) &= \frac{h_{ab}(l_a, l_b)}{M \cdot N}. \end{aligned} \tag{9.6}$$

Spremenljivki l_a in l_b označujeta diskretne sivinske vrednosti referenčne slike $a(x, y)$ in lebdeče slike $b(x, y)$, in sicer $l_a, l_b \in \{0, 1, 2, ..., L\}$.

9.1 Naloge in vprašanja

1. Potek mere podobnosti v bližini optimuma bistveno vpliva na številne lastnosti postopkov poravnave slik z optimizacijo. Za podano referenčno sliko reg_reference.png (slika 9.2) določite vrednosti mere podobnosti MSE, MAE, CC ter MI v parametričnem prostoru preslikave, ki ga napenjata vektorja premikov t_x in t_y na intervalu $[-15, 15]$ s korakom 1 slikovni element. Polje premikov ustvarite z ustrezno uporabo funkcije `meshgrid` modula `numpy`, za transformacijo (premik) slik uporabite funkcijo `imTransform2D`, vrednost parametra `expand` pa postavite na `'same'`. Pri izračunu medsebojne informacije verjetnost p_{ab} ocenite iz histograma, ki ima število razredov bistveno manjše od 256×256.



Slika 9.2: Od leve proti desni: referenčna slika reg_reference.png in tri vhodne slike reg_input1.png, reg_input2.png in reg_input3.png.

   (a) Ustvarite funkcijo `imSM`, ki izračuna vrednost izbrane mere podobnosti `sm` (`'MSE'`, `'MAE'`, `'CC'` ali `'MI'`) med slikama `imageA` in `imageB`. Število razredov histogramov pri izračunu medsebojne informacije naj določa parameter `nb`, razpon sivinskih vrednosti pa parameter `span`.

```
1 def imSM(imageA, imageB, sm='CC', nb=16, span=(0, 255)):
2   ...
3   return f
```

   (b) Izrišite izračunane vrednosti mer podobnosti kot slike v parametričnem prostoru preslikave.

   (c) Kakšna je optimalna vrednost posameznih mer podobnosti ter kako se mere podobnosti obnašajo v bližini optimuma (t_x=0 in t_y=0)?

   (d) Na podlagi postopka izčrpnega iskanja (ang. exhaustive search) na prej definiranem območju parametričnega prostora preslikave določite parametre premika, ki poravnajo vhodne slike reg_input1.png, reg_input2.png ter reg_input3.png z referenčno sliko reg_reference.png (slika 9.2). Postopek ponovite za vse mere podobnosti.

   (e) V čem se medsebojna informacija bistveno razlikuje od ostalih naštetih mer podobnosti?

2. Dani sta dvorazsežni magnetno resonančni sliki glave reg_MR1.png in reg_MR2.png (slika 9.3). Določite parametre toge preslikave (t_x, t_y, α), ki poravna dani sliki na podlagi optimizacije z izbrano mero podobnosti. Za optimizacijo uporabite funkcijo `fmin` modula `scipy.optimize`, za začetni približek parametrov preslikave pa uporabite ocenjene vrednosti premika in rotacije. Med poravnavo prikazujte razliko med referenčno in lebdečo sliko tako, da razliki slik prištejete 127 in ustrezno odrežete vrednosti izven 8-bitnega dinamičnega območja $[0, 255]$.

Slika 9.3: Magnetno resonančni sliki glave reg_MR1.png in reg_MR2.png.

   (a) Funkcija `fmin` modula `scipy.optimize` išče minimum večrazsežne funkcije, zato po potrebi preoblikujte naštete mere podobnosti tako, da izkazujejo najmanjšo vrednost, ko sta referenčna in lebdeča slika poravnani. V ta namen ustvarite funkcijo `imRigidRegister`, ki togo poravna lebdečo sliko `imageB` na referenčno sliko `imageA` in pri tem uporabi izbrano mero podobnosti `sm` ter začetni približek `x0` ($[t_x, t_y, \alpha]$). Funkcija naj vrne parametre `x` optimalne preslikave `T`, vrednost kriterijske funkcije v optimumu `f` ter število iteracij optimizacije `n`.

```
1 def imRigidRegister(imageA, imageB, sm, x0=[0.0, 0.0, 0.0]):
2   ...
3   return x, T, f, n
```

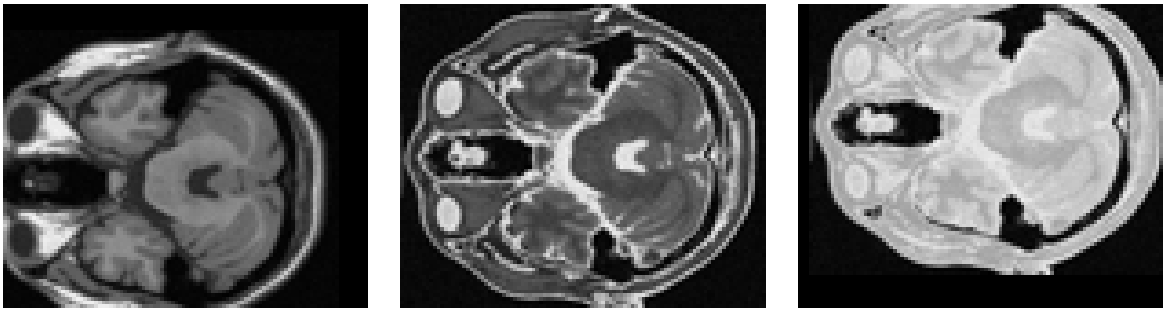
   (b) Kolikšno je število potrebnih iteracij optimizacijskega postopka za izbrano mero podobnosti? Ali se postopek poravnave vedno konča tako, da sta referenčna in lebdeča slika "poravnani"?

3. Dane so dvorazsežne $T1$, $T2$ in SD utežene magnetno resonančne slike glave (slika 9.4).

   (a) Izrišite poteke izbranih mere podobnosti za sliki SD in $T1$ ter sliki SD in $T2$ kot slike v parametričnem prostoru, ki ga napenjata vektorja premikov $t_x = [-15, 15]$ in $t_y = [-30, 0]$ s korakom 1 slikovni element. Pri tem naj bosta sliki $T1$ in $T2$ lebdeči.

   (b) Določite optimalno vrednost izbrane mere podobnosti ter pripadajoče parametre premika t_x^{opt} in t_y^{opt}.

   (c) Določite še optimalno vrednost izbrane mere podobnosti ter pripadajoče parametre premika t_x^{opt} in t_y^{opt} z uporabo optimizacijskega postopka, ki ste ga udejanili pod prejšnjo točko.



(a) $T1$ (b) $T2$ (c) SD

Slika 9.4: $T1$ (reg_T1.png), $T2$ (reg_T2.png) in SD (reg_SD.png) utežene magnetno resonančne slike glave.

9.2 Rešitve in odgovori na vprašanja

Pri odgovorih privzamemo, da se funkcije iz rešitev nahajajo v modulu `funkcije`, slikovno gradivo pa se nahaja v podmapi `poglavje_9`. Naprej uvozimo potrebne module, naložimo slike in definiramo pomožne spremenljivke.

```
import numpy as np
from matplotlib import pyplot as pp
from PIL import Image as im
import funkcije

I1ref = np.asarray(im.open('./poglavje_9/reg_reference.png'))
I1input1 = np.asarray(im.open('./poglavje_9/reg_input1.png'))
I1input2 = np.asarray(im.open('./poglavje_9/reg_input2.png'))
I1input3 = np.asarray(im.open('./poglavje_9/reg_input3.png'))

I2MR1 = np.asarray(im.open('./poglavje_9/reg_MR1.png'))
I2MR2 = np.asarray(im.open('./poglavje_9/reg_MR2.png'))

I3T1 = np.asarray(im.open('./poglavje_9/reg_T1.png'))
I3T2 = np.asarray(im.open('./poglavje_9/reg_T2.png'))
I3SD = np.asarray(im.open('./poglavje_9/reg_SD.png'))

tx = ty = np.arange(-15.0, 15.0 + 1.0)
Ty, Tx = np.meshgrid(ty, tx, indexing='ij')
Txf, Tyf = Tx.flatten(), Ty.flatten()
```

Ustvarimo pomožne funkcije za izčrpno iskanje optimalnih vrednosti mer podobnosti za dano polje premikov.

```
# izračuna vrednosti vseh mer podobnosti za polje premikov tx, ty
def rigidSM(imgA, imgB, tx, ty):
  Ia, Ib = imgA.astype('float'), imgB.astype('float')
  txshape = tx.shape
  tx = tx.flatten().astype('float')
  ty = ty.flatten().astype('float')

  MSE = np.zeros_like(tx)
  MAE = np.zeros_like(tx)
  CC = np.zeros_like(tx)
  MI = np.zeros_like(tx)

  for i in range(tx.size):
    txi, tyi = tx[i], ty[i]
    T = np.array([[1, 0, txi],[0, 1, tyi],[0, 0, 1]], dtype='float')
    MSE[i] = funkcije.imSM(Ia,
      funkcije.imTransform2d(Ib, T, expand='same'), 'MSE')
    MAE[i] = funkcije.imSM(Ia,
      funkcije.imTransform2d(Ib, T, expand='same'), 'MAE')
    CC[i] = funkcije.imSM(Ia,
```

```
21        funkcije.imTransform2d(Ib, T, expand='same'), 'CC')
22      MI[i] = funkcije.imSM(Ia,
23        funkcije.imTransform2d(Ib, T, expand='same'), 'MI')
24
25    MSE.shape = MAE.shape = CC.shape = MI.shape = txshape
26    return MSE, MAE, CC, MI
27
28  # izpiše premik pri optimalni vrednosti mer podobnosti
29  def printRigidOpt(MSE, MAE, CC, MI, tx, ty):
30    if MSE is not None:
31      ind = np.argmin(MSE)
32      print('Minimum MSE={} pri tx={}, ty={}.'.format(
33        MSE.flatten()[ind], tx.flatten()[ind], ty.flatten()[ind]))
34    if MAE is not None:
35      ind = np.argmin(MAE)
36      print('Minimum MAE={} pri tx={}, ty={}.'.format(
37        MAE.flatten()[ind], tx.flatten()[ind], ty.flatten()[ind]))
38    if CC is not None:
39      ind = np.argmax(CC)
40      print('Maximum CC={} pri tx={}, ty={}.'.format(
41        CC.flatten()[ind], tx.flatten()[ind], ty.flatten()[ind]))
42    if MI is not None:
43      ind = np.argmax(MI)
44      print('Maximum MI={} pri tx={}, ty={}.'.format(
45        MI.flatten()[ind], tx.flatten()[ind], ty.flatten()[ind]))
46
47  # izriše kriterijsko funkcijo kot sliko
48  def showSMs(MSE, MAE, CC, MI, tx, ty, title=''):
49    # extent = [left, right, bottom, top]
50    extent = [tx.min(), tx.max(), ty.max(), ty.min()]
51    pp.figure()
52    pp.suptitle(title)
53    pp.subplot(1, 4, 1)
54    pp.imshow(MSE, extent=extent),
55    pp.title('MSE'), pp.colorbar()
56    pp.subplot(1, 4, 2)
57    pp.imshow(MAE, extent=extent)
58    pp.title('MAE'), pp.colorbar()
59    pp.subplot(1, 4, 3)
60    pp.imshow(CC, extent=extent)
61    pp.title('CC'), pp.colorbar()
62    pp.subplot(1, 4, 4)
63    pp.imshow(MI, extent=extent)
64    pp.title('MI'), pp.colorbar()
```

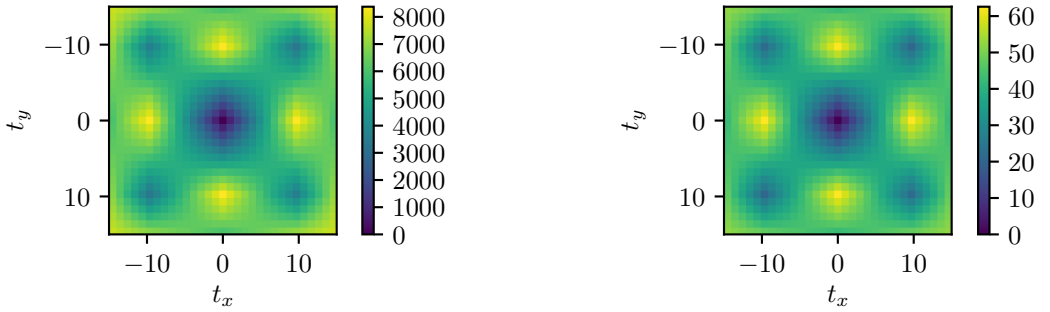
1. (a) V modulu `funkcije` ustvarimo funkcijo `imSM`.

```
1  def imSM(imageA, imageB, sm, nb=16, span=[0, 255]):
2  imageA = np.asarray(imageA, dtype=np.float)
```

```
3   imageB = np.asarray(imageB, dtype=np.float)
4   N = imageA.size
5   nb, span = int(nb), np.asarray(span, dtype=np.float)
6   sm = str(sm).lower()
7   if sm == 'mse':
8     of = ((imageA - imageB)**2).mean()
9
10  elif sm == 'mae':
11    of = (np.abs(imageA - imageB)).mean()
12
13  elif sm == 'cc':
14    imageAm = imageA.mean()
15    imageBm = imageB.mean()
16    of = ((imageA - imageAm)*(imageB - imageBm)).sum()/ \
17      np.sqrt(
18        ((imageA-imageAm)**2).sum()*
19        ((imageB - imageBm)**2).sum())
20
21  elif sm == 'mi':
22    d = span[1] - span[0]
23    imageAd = np.round((imageA - span[0])/(d/nb))
24    imageBd = np.round((imageB - span[0])/(d/nb))
25    pa = np.zeros([nb])
26    pb = np.zeros([nb])
27    pab = np.zeros([nb, nb])
28    for i in range(nb):
29      pa[i] = np.count_nonzero(imageAd == i)
30      pb[i] = np.count_nonzero(imageBd == i)
31
32    pa *= 1.0/N
33    pb *= 1.0/N
34
35    for i in range(nb):
36      for j in range(nb):
37        pab[i, j] = np.count_nonzero(
38          (imageAd == i) & (imageBd == j))
39
40    pab *= 1.0/N
41
42    pa[pa == 0] = 1.0
43    pb[pb == 0] = 1.0
44    pab[pab == 0] = 1.0
45    Ha, Hb = -(pa*np.log(pa)).sum(), -(pb*np.log(pb)).sum()
46    Hab = -(pab*np.log(pab)).sum()
47    of = Ha + Hb - Hab
48  else:
49    raise ValueError('Izbrana mera podobnosti ne obstaja.')
50
51  return of
```

(b) Izrišemo vse štiri mere podobnosti v parametričnem prostoru toge preslikave (slika 9.5).

```
MSEref, MAEref, CCref, MIref = rigidSM(I1ref, I1ref, Tx, Ty)
showSMs(MSEref, MAEref, CCref, MIref, tx, ty,
        'Mere podobnosti med premaknjeno in '
        'izvirno sliko reg_reference.png')
```



(a) MSE (b) MAE

(c) CC (d) MI

Slika 9.5: Mere podobnosti v parametričnem prostoru toge preslikave, ki je definirana s poljem premikov $t_x = t_y = [-15, 15]$

.

(c) Meri podobnosti MAE in MSE izkazujeta minimum v okolici optimalne vrednosti, meri podobnosti CC in MI pa izkazujeta maksimum v okolici optimalne vrednosti.

```
ind = np.argmin(MSEref)
print('Mera podaobnosti kot funkcija togega premika '
      'za sliko reg_reference:\n')
printRigidOpt(MSEref, MAEref, CCref, MIref, Tx, Ty)
```

(d) Parametri premikov, ki poravnajo vhodne slike z referenčno, so zbrani v tabeli 9.1. Pri treji vhodni sliki reg_input3.png je poravnava uspešna le za mero podobnosti MI.

```
MSEi1, MAEi1, CCi1, MIi1 = rigidSM(I1ref, I1input1, Tx, Ty)
print('\nPoravnava slike reg_reference in reg_input1:')
printRigidOpt(MSEi1, MAEi1, CCi1, MIi1, Tx, Ty)

MSEi2, MAEi2, CCi2, MIi2 = rigidSM(I1ref, I1input2, Tx, Ty)
print('\nPoravnava slike reg_reference in reg_input2:')
printRigidOpt(MSEi2, MAEi2, CCi2, MIi2, Tx, Ty)

MSEi3, MAEi3, CCi3, MIi3 = rigidSM(I1ref, I1input3, Tx, Ty)
print('\nPoravnava slike reg_reference in reg_input3:')
printRigidOpt(MSEi3, MAEi3, CCi3, MIi3, Tx, Ty)
```

Tabela 9.1: Parametri premikov pri optimalni vrednosti mer podobnosti, izraženi v slikovnih elementih.

Vhodna slika	t_x	t_y
reg_input1.png	-4	8
reg_input2.png	3	-9
reg_input3.png	-4	8

(e) Medsebojna informacija je statistična mera podobnosti, katere vrednost izhaja iz verjetnostnih porazdelitev sivinskih vrednosti referenčne in lebdeče slike.

2. (a) V modulu `funkcije` ustvarimo funkcijo `imRigidRegister`.

```
def imRigidRegister(imageA, imageB, sm, x0=[0.0, 0.0, 0.0],
                    animate=False):
  # pripravimo vhodne podatke
  imageA = np.asarray(imageA, dtype=np.float)
  imageB = np.asarray(imageB, dtype=np.float)
  x0 = np.asarray(x0, dtype=np.float)
  sm = str(sm).lower()

  # ponastavimo vrednost števca klicev kriterijske funkcije
  Nreg = np.array(0)

  # pripravimo okno za izrisovanje vmesnih rezultatov
  if animate:
    pp.figure()

  # začnemo poravnavo z optimizaciojo mere podobnosti
  ox = opt.fmin(
    lambda x: kFunRigidReg(
      x, imageA, imageB, sm, animate, Nreg), x0)
  # izračunamo vrednost kriterijske funkcije ob zaključku poravnave
  of = kFunRigidReg(ox, imageA, imageB, sm, False)
  # ustvarimo transformacijsko matriko toge preslikave
  ot = transformAffine2d(trans=ox[:2], rot=ox[2])
```

```
24
25    return ox, ot, of, int(Nreg)
```

Ustvarimo še kriterijsko funkcijo, ki jo minimiziramo v optimizacijskem postopku.

```
1  def kFunRigidReg(x, img1, img2, sm, animate, Nreg=0):
2    Nreg += 1
3    # ustvarim matriko toge preslikave in z njo preslikamo sliko
4    T = transformAffine2d(trans=x[:2], rot=x[2])
5    img2t = imTransform2d(img2, T, expand='same')
6    # izračunamo vrednost mere podobnosti med lebdečo in referenčno sliko
7    f = imSM(img1, img2t, sm)
8    if sm == 'cc':
9      f = 1.0 - f
10
11   elif sm == 'mi':
12     f = -f
13
14   # po potrebi prikažemo stanje poravnave
15   if animate and (Nreg % 10) == 0:
16     pp.imshow(128.0 + img2t - img1, cmap='gray',
17                       vmin=0, vmax=255)
18     pp.title('Iteracija {}, vrednost '
19                'kriterijske funkcije {}'.format(Nreg, f))
20     pp.draw()
21     pp.pause(0.01)
22
23   return f
```

(b) Postopek poravnave se vedno ne konča tako, da sta sliki poravnani. Rezultat postopka poravnave je odvisen od izbrane mere podobnosti in začetnega približka preslikave (tabela 9.2).

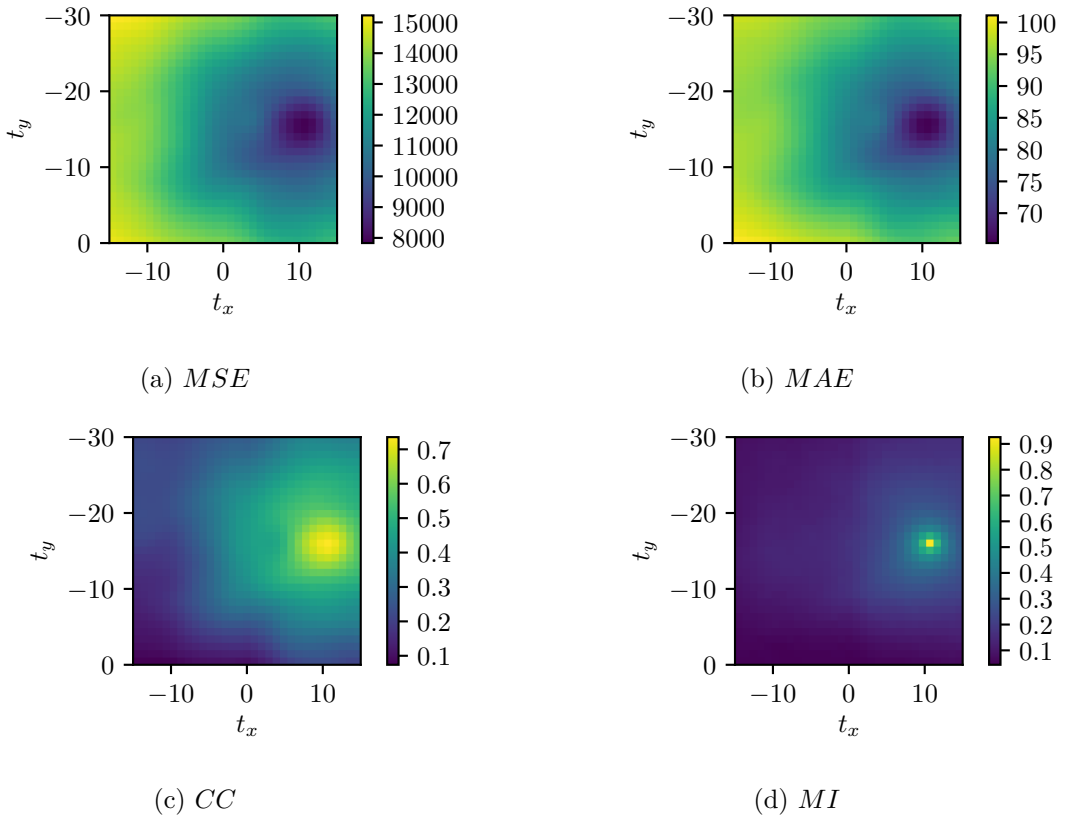
Tabela 9.2: Parametri poravnave za štiri mere podobnosti. Vrednosti parametrov premikov t_x in t_y so podane v slikovnih elementih, kot rotacije α je podan v radianih, končna vrednost mere podobnosti f_k pa v slikovnih elementih. N določa število izračunov mere podobnosti.

Mera podobnosti	t_x	t_y	α	f_k	N
MSE	15.00	-6.00	0.087	33.63	203
MAE	15.00	-6.00	0.087	2.86	178
CC	15.00	-6.00	0.087	0.0064	245
MI	15.00	-6.00	0.087	-1.64	227

```
1  X2CC = []
2  T2CC = []
3  f2CC = []
4  N2CC = []
5  for sm in ['MSE', 'MAE', 'CC', 'MI']:
```

```
6   X2CCi, T2CCi, f2CCi, N2CCi = funkcije.imRigidRegister(
7     I2MR1, I2MR2, sm, [5, 5, 0.1], animate=True)
8   print(
9     '\nParametri toge poravnave slik reg_MR1 in reg_MR2 s '
10    'SM=''{}'': tx={}, ty={}, fi={} rad, f={}, N={}'.format(
11    sm, X2CCi[0], X2CCi[1], X2CCi[2], f2CCi, N2CCi))
12  X2CC.append(X2CCi)
13  T2CC.append(T2CCi)
14  f2CC.append(f2CCi)
15  N2CC.append(N2CCi)
```

3. (a) Vrednosti štirih mer podobnosti v parametričnem prostoru preslikave, ki ga napenjata vektorja premikov $t_x = [-15, 15]$ in $t_y = [-30, 0]$ slikovnih elementov, prikazujeta sliki 9.6 in 9.7.



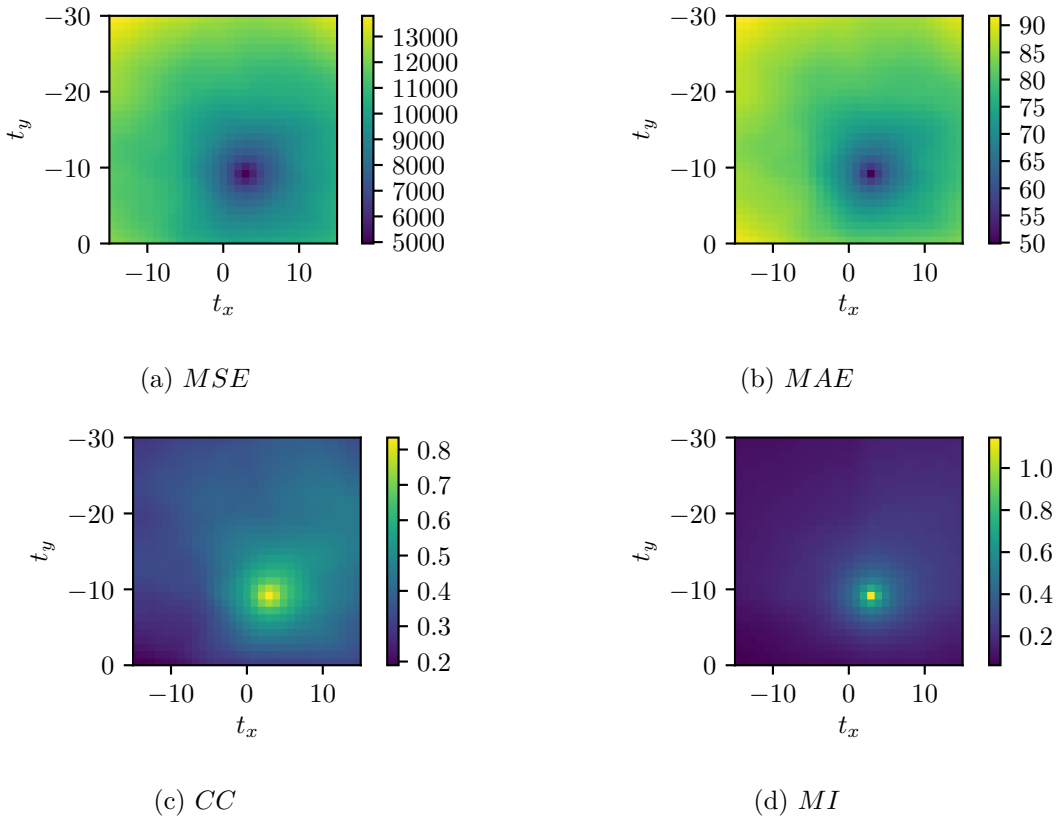
(a) *MSE* (b) *MAE*

(c) *CC* (d) *MI*

Slika 9.6: Mere podobnosti v parametričnem prostoru toge preslikave za sliki reg_SD.png in reg_T1.png.

.

```
1  tx3 = np.arange(-15.0, 15.0 + 1.0)
2  ty3 = np.arange(-30.0, 0 + 1.0)
```



(a) MSE (b) MAE

(c) CC (d) MI

Slika 9.7: Mere podobnosti v parametričnem prostoru toge preslikave za sliki reg_SD.png in reg_T2.png.

.

```
3  Ty3, Tx3 = np.meshgrid(ty3, tx3, indexing='ij')
4
5  MSET1, MAET1, CCT1, MIT1 = rigidSM(I3SD, I3T1, Tx3, Ty3)
6  print('\nMere podobnosti med slikama reg_SD in reg_T1:')
7  printRigidOpt(MSET1, MAET1, CCT1, MIT1, Tx3, Ty3)
8  showSMs(MSET1, MAET1, CCT1, MIT1, tx3, ty3,
9          'Poravnava reg_SD in reg_T1')
10
11 MSET2, MAET2, CCT2, MIT2 = rigidSM(I3SD, I3T2, Tx3, Ty3)
12 print('\nMere podobnosti med slikama reg_SD in reg_T2:')
13 printRigidOpt(MSET2, MAET2, CCT2, MIT2, Tx3, Ty3)
14 showSMs(MSET2, MAET2, CCT2, MIT2, tx3, ty3,
15         'Poravnava reg_SD in reg_T2')
```

(b) Določimo še vrednosti parametrov premika, pri katerih dobimo optimalno vrednost mere podobnosti (tabeli 9.3 in 9.4).

(c) Izberemo medsebojno informacijo MI kot mero podobnosti (tabela 9.5).

Tabela 9.3: Parametri premika t_x in t_y, izraženi v slikovnih elementih, pri optimalni vrednosti mer podobnosti za sliko reg_T1.png.

Mera podobnosti	t_x	t_y	SM
MSE	11	-16	7832.9
MAE	11	-16	65.30
CC	11	-16	0.736
MI	11	-16	0.922

Tabela 9.4: Parametri premika t_x in t_y, izraženi v slikovnih elementih, pri optimalni vrednosti mer podobnosti za sliko reg_T2.png.

Mera podobnosti	t_x	t_y	SM
MSE	3	-9	4946.1
MAE	3	-9	49.84
CC	3	-9	0.834
MI	3	-9	1.112

Tabela 9.5: Rezultati poravnave slik z mero podobnosti MI. Vrednosti parametrov premikov t_x in t_y so podane v slikovnih elementih, kot rotacije α je podan v radianih, končna vrednost mere podobnosti f_k pa v slikovnih elementih. N določa število izračunov mere podobnosti.

Vhodna slika	t_x	t_y	α	f_k	N
reg_T1.png	10.99	15.96	0.000	-0.933	85
reg_T2.png	2.99	-9.00	0.000	-1.133	84

```
ind = np.argmax(MIT1)
x0 = [Tx3.flatten()[ind], Ty3.flatten()[ind], 0]
X3CCT1, T3CCT1, f3CCT1, N3CCT1 = funkcije.imRigidRegister(
  I3SD, I3T1, 'MI', x0=x0, animate=True)
print(
  'Parametri toge preslikave, ki poravna sliki '
  'reg_SD in reg_T1: '
  'tx={}, ty={}, fi={} rad, f={}, N={}'.format(
  X3CCT1[0], X3CCT1[1], X3CCT1[2], f3CCT1, N3CCT1))

ind = np.argmax(MIT2)
x0 = [Tx3.flatten()[ind], Ty3.flatten()[ind], 0]
X3CCT2, T3CCT2, f3CCT2, N3CCT2 = funkcije.imRigidRegister(
   I3SD, I3T2, 'MI', x0=x0, animate=True)
print(
  'Parametri toge preslikave, ki poravna sliki '
  'reg_SD in reg_T2: '
  'tx={}, ty={}, fi={} rad, f={}, N={}'.format(
  X3CCT2[0], X3CCT2[1], X3CCT2[2], f3CCT2, N3CCT2))

```

```
21 pp.show()
```

Poglavje 10

Projekcije 2D slik

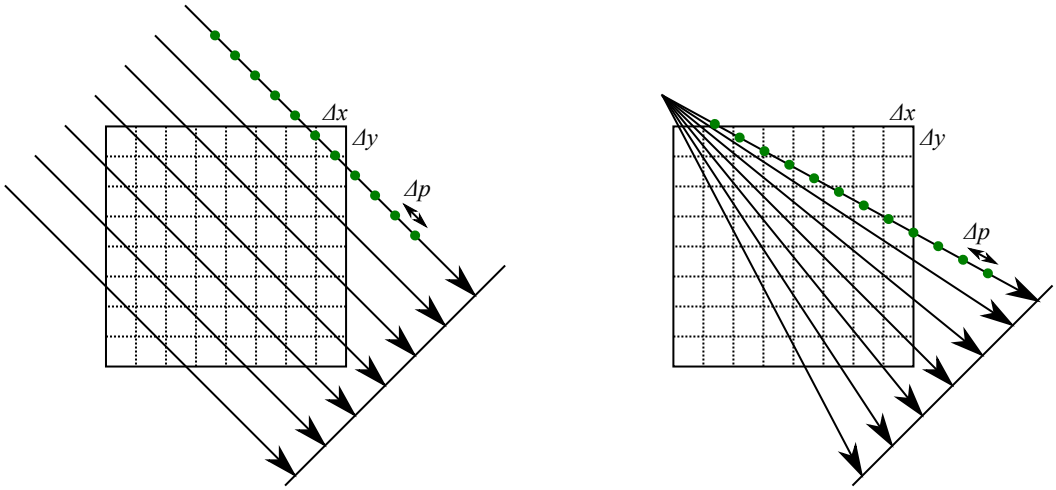
V okviru tega poglavja si bomo podrobneje ogledali dva osnovna tipa projekcij, in sicer pravokotno ter stožčasto projekcijo (slika 10.1). Stožčasta projekcija nam dobro opiše dogajanje tekom zajema sodobnih CT in rentgenskih slik. Podrobneje bomo spoznali tudi sinogram, ki ga sestavljajo projekcije enega prečnega prereza CT slike, zajete pod različnimi koti. Posamezne projekcije sinograma predstavljajo odziv linijskega tipala, projekcijske daljice pa pot žarkov od izvora do posameznega slikovnega elementa linijskega tipala. Za rotacijo točk okoli koordinatnega izhodišča lahko uporabimo rotacijsko matriko T_r, ki smo jo podrobneje spoznali v poglavju 7:

$$T_r = \begin{bmatrix} \cos\varphi & -\sin\varphi \\ \sin\varphi & \cos\varphi \end{bmatrix}. \tag{10.1}$$

Pri izračunu projekcije 2D slik vzdolž poljubnih projekcijskih daljic izberemo korak vzorčenja Δp običajno tako, da ni manjši od najmanjše razsežnosti slikovnega elementa $\Delta p \approx \min(\Delta x, \Delta y)$. Pri vzorčenju sivinskih vrednosti vzdolž projekcijskih daljic v splošnem ne naletimo na vzorčne vrednosti digitalne slike, razen ko računamo pravokotne projekcije vzdolž osnovnih koordinatnih osi slike. Zato je v splošnem potrebno sivinsko vrednost v vzorčnih točkah izračunati s pomočjo interpolacije. V okviru tega poglavja se bomo omejili na bilinearno interpolacijo, ki jo lahko izvedemo s funkcijo `interp2` modula `interp`.

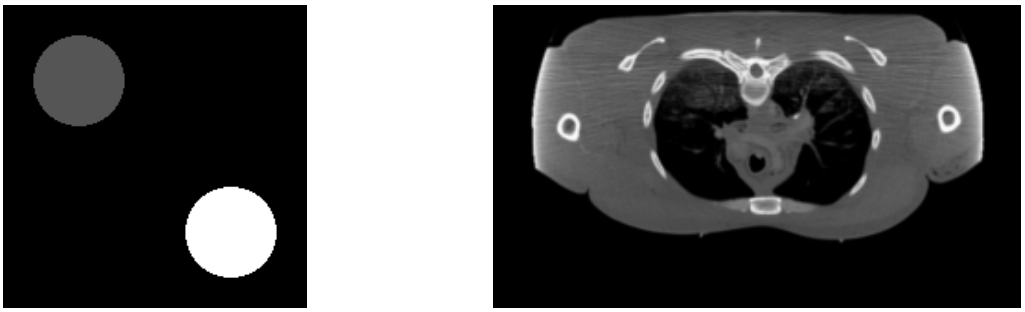
10.1 Naloge in vprašanja

1. Ustvarite funkcijo `imParallelBeamProject2d`, ki izračuna 1D projekcijo (odziv linijskega tipala), na katerega žarki padajo pravokotno, in sicer za kote rotacije `fi` med linijskim tipalom in y osjo. Predpostavite, da se vhodna slika `img` nahaja na mreži točk, ki jo napenjata vektorja `x` in `y`. Izvor vzporednih žarkov in linijsko tipalo naj bosta na razdalji `d`, število slikovnih elementov linijskega tipala naj bo `n`, velikost posameznega kvadratnega slikovnega elementa tipala pa `pixelSize`. Izračunane projekcije naj bodo shranjene v vrsticah matrike `P`. Korak vzorčenja v smeri projekcijskih daljic naj bo `step`. Parameter `kind` naj določa tip projekcije, in sicer maksimalne vrednosti `'max'` , vsote `'sum'` ali srednje vrednosti `'mean'`. Podrobnejše informacije o geometriji prikazuje slika 10.3.



(a) Pravokotna projekcija (b) Stožčasta projekcija

Slika 10.1: Ilustracija pravokotne in stožčaste projekcije 2D slike na linijsko tipalo.



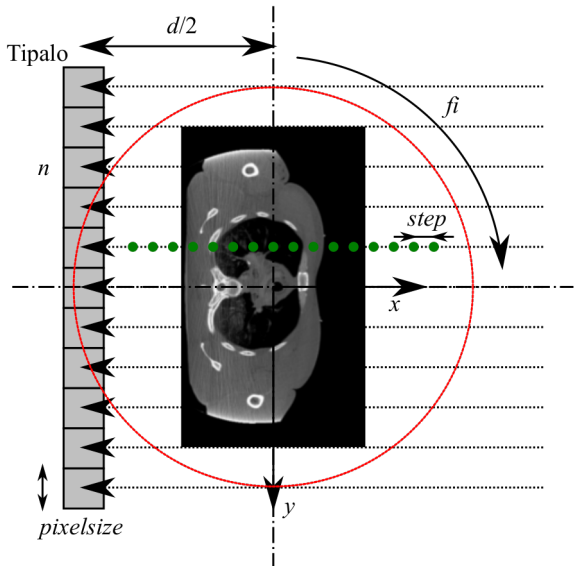
(a) Testna slika (b) Prerez CT slike

Slika 10.2: Testna slika proj2d_testna.png (200×200 slikovnih elementov velikosti 1×1 mm) ter prečni prerez CT slike trupa proj2d_ct.png (287×165 slikovnih elementov velikosti 2×2 mm).

```
1 def imParallelBeamProject2d(fi, step, img, x, y,
2                              d, n, pixelSize, kind='sum'):
3   ...
4   return P
```

(a) Naložite testno sliko proj2d_test.png (slika 10.2), izračunajte 2D projekcije (`kind='sum'`) za kote med linijskim tipalom in y osjo iz intervala $[0, 359]^\circ$ s korakom 1° ter izrišite projekcije `P` kot sivinsko sliko. Vrednosti parametrov funkcije `imParallelBeamProject2d` naj bodo sledeče: `d=800` mm, `n=512`, `pixelSize=1` mm, `step=2` mm, geometrično središče slike pa postavite v koordinatno izhodišče. Velikost slikovnih elementov testne slike znaša 1×1 mm.

(b) Na enak način izračunajte še projekcije prečnega prereza CT slike trupa proj2d-



Slika 10.3: Pravokotna projekcija 2D prereza CT slike na linijsko tipalo.

_ct.png (slika 10.2) ter izračunane projekcije prikažite kot sliko. Velikost slikovnih elementov prečnega prereza CT slike znaša 2×2 mm.

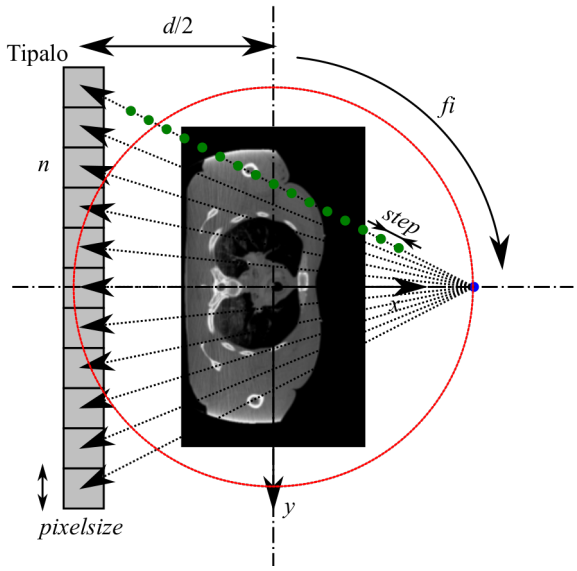
2. Ustvarite še funkcijo `imFanBeamProject2d` za izračun stožčastih projekcij, ki jih dobimo, ko na linijsko tipalo vpadajo žarki iz točkastega izvora. Pomen parametrov funkcije naj bo enak kot pod prejšnjo točko. Podrobnejše informacije o geometriji postavitve prikazuje slika 10.4.

```
def imFanBeamProject2d(fi, step, img, x, y,
    d, n, pixelSize, kind='sum'):
  ...
  return P
```

   (a) Naložite testno sliko proj2d_test.png, izračunajte 2D projekcije (`kind='sum'`) za kote med linijskim tipalom in y osjo iz intervala $[0, 359]^\circ$ s korakom 1° ter izrišite projekcije `P` kot sivinsko sliko. Vrednosti parametrov naj bodo sledeče: `d=800` mm, `n=512`, `pixelSize=1` mm, `step=2` mm, geometrično središče slike pa postavite v koordinatno izhodišče. Velikost slikovnega elementa testne slike znaša 1×1 mm.

   (b) Preizkusite, kako vrednost parametra `step` vpliva na sinogram in kako na čas, ki ga za računanje porabi funkcija `imFanBeamProject2d`.

   (c) Izračunajte še projekcije prečnega prereza CT slike trupa proj2d_ct.png ter projekcije `P` prikažite kot sivinsko sliko. Velikost slikovnih elementov prečnega prereza CT slike znaša 2×2 mm. Izračunane projekcije predstavljajo surove podatke, ki jih zajeme CT naprava, in so osnova za rekonstrukcijo enega prečnega prereza 3D CT slike.



Slika 10.4: Stožčasta projekcija 2D prereza CT slike na linijsko tipalo.

(d) Programsko kodo prilagodite tako, da bodo izračuni projekcij hitri (nekaj deset ms za posamezno projekcijo), saj bomo izračunane projekcije (sinograme) potrebovali v poglavju 12, kjer jih bomo uporabili za rekonstrukcijo prereza 3D CT slike.

10.2 Rešitve in odgovori na vprašanja

Pri odgovorih privzamemo, da se funkcije iz rešitev nahajajo v modulu `funkcije`, slikovno gradivo pa se nahaja v podmapi `poglavje_10`. Naprej uvozimo potrebne module, naložimo slike in definiramo pomožne spremenljivke.

```
import funkcije
import numpy as np
from matplotlib import pyplot as pp
from PIL import Image as im
import time

It = np.array(im.open('./poglavje_10/proj2d_test.png'))
Ht, Wt = It.shape
xt = np.arange(-(Wt - 1)*0.5, (Wt - 1)*0.5 + 0.5, 1)
yt = np.arange(-(Ht - 1)*0.5, (Ht - 1)*0.5 + 0.5, 1)

Ict = np.array(im.open('./poglavje_10/proj2d_ct.png'))
Hct, Wct = Ict.shape
dy, dx = 2, 2
xct = np.arange(-dx*(Wct - 1)*0.5, dx*(Wct - 1)*0.5 + 0.5*dx, dx)
yct = np.arange(-dy*(Hct - 1)*0.5, dy*(Hct - 1)*0.5 + 0.5*dy, dy)

fi = np.deg2rad(np.arange(360))
```

1. V modulu `funkcije` ustvarimo funkcijo `imParallelBeamProject2d`.

```
def imParallelBeamProject2d(fi, step, image, x, y,
                            d, n, pixelSize, kind='sum'):
  # pripravimo vhodne podatke
  kind = str(kind).lower()
  if kind == 'sum':
    fun = np.sum
  elif kind == 'max':
    fun = np.max
  elif kind == 'mean':
    fun = np.mean
  else:
    raise ValueError(
      'Vrednost parametra "kind" je lahko "sum" ali "max".')
  image = np.asarray(image, dtype=np.float)
  x = np.asarray(x, dtype=np.float)
  y = np.asarray(y, dtype=np.float)
  if isinstance(fi, float) or isinstance(fi, int):
    fi = [fi]
  fi = np.asarray(fi, dtype=np.float)
  pixelSize = float(pixelSize)
  step = float(step)
  d = float(d)
```

```
23
24    # položaj vira
25    ys = pixelSize*np.arange(n, dtype=np.float)
26    ys -= ys.mean()
27    xs = np.tile(d*0.5, ys.shape)
28    # položaji tipal
29    # yd = ys
30    # xd = -d*0.5
31
32    # vzorčne točke vzdolž projekcijskih daljic
33    t = np.arange(0, d + step, step)
34
35    # ustvari projekcijske žarke (vsak v svoji vrstici)
36    Xsamples = np.zeros([n, t.size])
37    Ysamples = np.zeros([n, t.size])
38    for i in range(n):
39      Xsamples[i,:] = xs[i] + t*(-1.0)
40      Ysamples[i,:] = ys[i] + t*(0.0)
41    # vzorčne točke shranimo v dvovrstično matriko - x v prvo, y v drugo vrstico
42    pts = np.vstack([Xsamples.flatten(), Ysamples.flatten()])
43
44    # ustvarimo podatkovno polje projekcij/sinograma
45    P = np.zeros([fi.size, n])
46    for i in range(fi.size):
47      # rotiramo vzorčne točke za dani kot rotacije
48      tmp = np.dot(np.array([
49        [np.cos(fi[i]), -np.sin(fi[i])],
50        [np.sin(fi[i]), np.cos(fi[i])]]), pts)
51      # interpolacija sivinskih vrednosti v vzorčnih točkah
52      proji = interp.interp2(tmp[0], tmp[1], x, y, image)
53      # preoblikuj podatke tako, da so žarki v vrsticah
54      proji.shape = Xsamples.shape
55      # izvedemo zahtevano operacijo vzdolž projekcijskih daljic
56      P[i] = fun(proji, 1)
57
58    return P
```

(a) Preizkusimo delovanje funkcije `imParallelBeamProject2d` na sliki proj2d_test.png. Izračunani sinogram prikazuje slika 10.5.

```
1  start = time.perf_counter()
2  ItPS = funkcije.imParallelBeamProject2d(
3    fi, 2, It, xt, yt, 800, 512, 1, 'sum')
4  print('Izračun {} pravokotnih projekcij testne slike '
5        'v {} s'.format(fi.size, time.perf_counter() - start))
6
7  pp.figure()
8  pp.suptitle('Testna slika s pripadajočim sinogramom - '
9              'pravokotna projekcija.')
```

```
10
11 pp.subplot(1, 2, 1)
12 pp.imshow(It, cmap='gray')
13
14 pp.subplot(1, 2, 2)
15 pp.imshow(ItPS, cmap='gray')
16
17 pp.show()
```

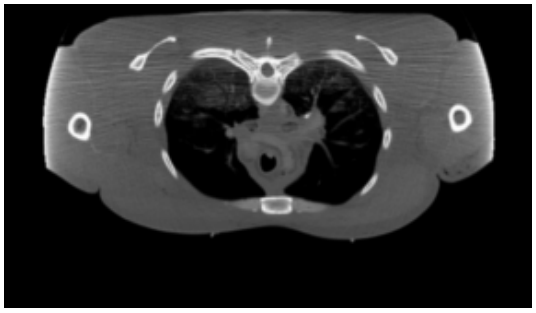
Slika 10.5: Testna slika proj2d_test.png s pripadajočim sinogramom, izračunanim s pravokotno projekcijo.

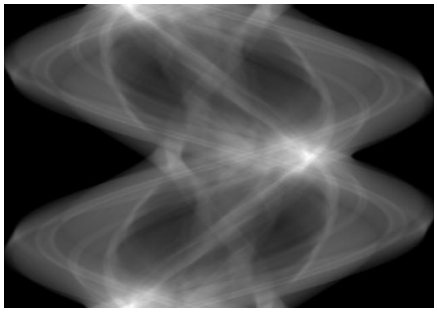
(b) Preizkusimo delovanje funkcije `imParallelBeamProject2d` še na prečnem prerezu CT slike proj2d_ct.png. Izračunani sinogram prikazuje slika 10.6.

```
1  start = time.perf_counter()
2  IctPS = funkcije.imParallelBeamProject2d(
3    fi, 2, Ict, xct, yct, 800, 512, 1, 'sum')
4  print('Izračun {} pravokotnih projekcij CT rezine '
5        'v {} s'.format(fi.size, time.perf_counter() - start))
6
7  pp.figure()
8  pp.suptitle('Rezina CT slike s pripadajočim sinogramom - '
9              'pravokotna projekcija.')
10
11 pp.subplot(1, 2, 1)
12 pp.imshow(Ict, cmap='gray')
13 pp.subplot(1, 2, 2)
14 pp.imshow(IctPS, cmap='gray')
15
16 pp.show()
```

2. V modulu `funkcije` ustvarimo funkcijo `imFanBeamProject2d`.

```
1 def imFanBeamProject2d(fi, step, image, x, y,
2     d, n, pixelSize, kind='sum'):
3
4   kind = str(kind).lower()
```





Slika 10.6: Prečni prerez CT slike proj2d_ct.png s pripadajočim sinogramom, izračunanim s pravokotno projekcijo.

```
5   if kind == 'sum':
6     fun = np.sum
7   elif kind == 'max':
8     fun = np.max
9   elif kind == 'mean':
10    fun = np.mean
11  else:
12    raise ValueError(
13      'Vrednost parametra "kind" je lahko "sum" ali "max".')
14
15  # pripravi sliko, koordinatno mrežo točk in kote
16  image = np.asarray(image, dtype=np.float)
17  x = np.asarray(x, dtype=np.float)
18  y = np.asarray(y, dtype=np.float)
19  if isinstance(fi, float) or isinstance(fi, int):
20    fi = [fi]
21  fi = np.asarray(fi, dtype=np.float)
22  pixelSize = float(pixelSize)
23  step = float(step)
24  d = float(d)
25
26  # položaj vira
27  ys = 0.0
28  xs = d*0.5
29  # položaji tipal
30  yd = pixelSize*np.arange(n, dtype=np.float)
31  yd -= yd.mean()
32  xd = -d*0.5
33
34  # dolžina najdaljšega žarka
35  dmax = np.sqrt(yd[0]**2 + d**2)
36  # točke na projekcijski premici
37  t = np.arange(0, dmax + step, step)
38
39  # ustvari projekcijske žarke (vsak v svoji vrstici)
```

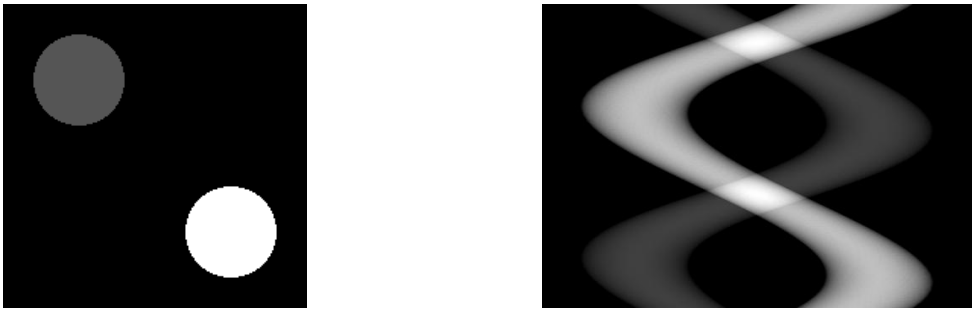
```
  Xsamples = np.zeros([n, t.size])
  Ysamples = np.zeros([n, t.size])
  for i in range(n):
    s = np.array([xd - xs, yd[i] - ys])
    s /= np.linalg.norm(s)
    Xsamples[i,:] = xs + t*s[0]
    Ysamples[i,:] = ys + t*s[1]

  # podatkovno polje sinograma
  P = np.zeros([fi.size, n])
  # interpolacijske točke žarkov
  pts = np.vstack([Xsamples.flatten(), Ysamples.flatten()])
  for i in range(fi.size):
    # rotiraj interpolacijske točke
    tmp = np.dot(np.array([
      [np.cos(fi[i]), -np.sin(fi[i])],
      [np.sin(fi[i]), np.cos(fi[i])]]), pts)
    # interpoliraj
    proji = interp.interp2(tmp[0], tmp[1], x, y, image)
    # preoblikuj podatke, tako da so žarki v vrsticah
    proji.shape = Xsamples.shape
    P[i] = fun(proji, 1)

  return P
```

(a) Preizkusimo delovanje funkcije `imFanBeamProject2d` na testni sliki proj2d_test.png. Izračunani sinogram prikazuje slika 10.7.

```
start = time.perf_counter()
ItSS = funkcije.imFanBeamProject2d(
  fi, 2, It, xt, yt, 800, 512, 1, 'sum')
print('Izračun {} stožčastih projekcij testne slike '
      'v {} s'.format(fi.size, time.perf_counter() - start))

pp.figure()
pp.suptitle('Testna slika s pripadajočim sinogramom - '
            'stožčasta projekcija.')

pp.subplot(1, 2, 1)
pp.imshow(It, cmap='gray')

pp.subplot(1, 2, 2)
pp.imshow(ItSS, cmap='gray')

pp.show()
```

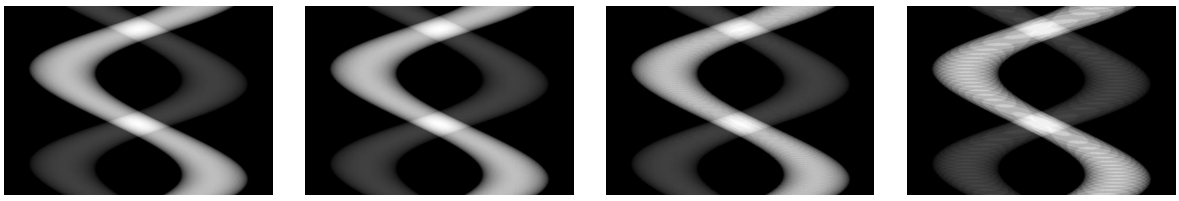
(b) Ocenimo še vpliv vzorčnega koraka na izračun sinogramov (slika 10.8 ter tabela 10.1).

```
step = [1.0, 2.0, 4.0, 8.0]
TSS = []
```



Slika 10.7: Testna slika proj2d_test.png s pripadajočim sinogramom, izračunanim s stožčasto projekcijo.

```
ISSt = []

pp.figure()
pp.suptitle('Sinogram testne slike '
            'v odvisnosti od koraka vzorčenja.')
for stepi in step:
  start = time.perf_counter()
  s = funkcije.imFanBeamProject2d(
    fi, stepi, It, xt, yt, 800, 512, 1, 'sum')
  TSS.append(time.perf_counter() - start)
  ISSt.append(s)
  pp.subplot(1, 4, len(TSS))
  pp.imshow(s, cmap='gray')
  pp.title('Korak vzorčenja {:.1f} mm v {:.1f} s'.format(
           stepi, TSS[-1]))

pp.show()
```



(a) 1 mm (b) 2 mm (c) 4 mm (d) 8 mm

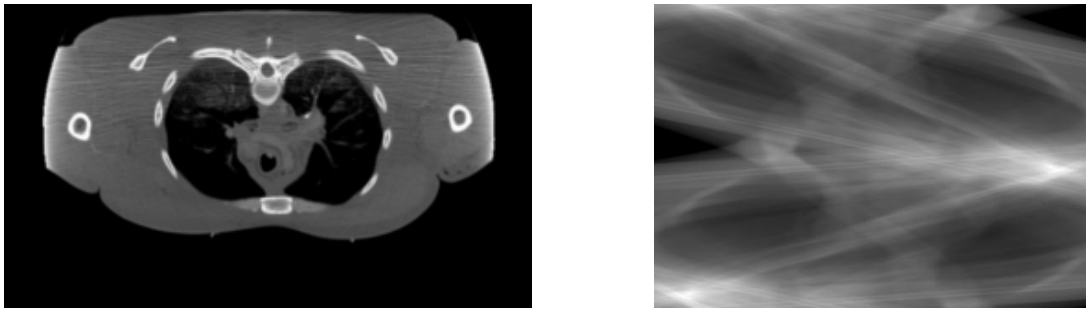
Slika 10.8: Vpliv koraka vzorčenja na izračun sinogramov.

Tabela 10.1: Vpliv koraka vzorčenja na čas izračuna sinogramov.

Korak (mm)	1	2	4	8
Čas (s)	55.1	27.2	13.4	6.7

(c) Preizkusimo delovanje funkcije `imFanBeamProject2d` še na prečnem prerezu CT slike proj2d_ct.png. Izračunani sinogram prikazuje slika 10.9

```
start = time.perf_counter()
IctSS = funkcije.imFanBeamProject2d(
  fi, 2, Ict, xct, yct, 800, 512, 1, 'sum')
print('Izračun {} stožčastih projekcij CT rezine v '
      '{} s'.format(fi.size, time.perf_counter() - start))

pp.figure()
pp.suptitle('Rezina CT slike s pripadajočim sinogramom - '
        'stožčasta projekcija.')

pp.subplot(1, 2, 1)
pp.imshow(Ict, cmap='gray')

pp.subplot(1, 2, 2)
pp.imshow(IctSS, cmap='gray')

pp.show()
```



Slika 10.9: Prečni prerez CT slike proj2d_ct.png s pripadajočim sinogramom, izračunanim s stožčasto projekcijo.

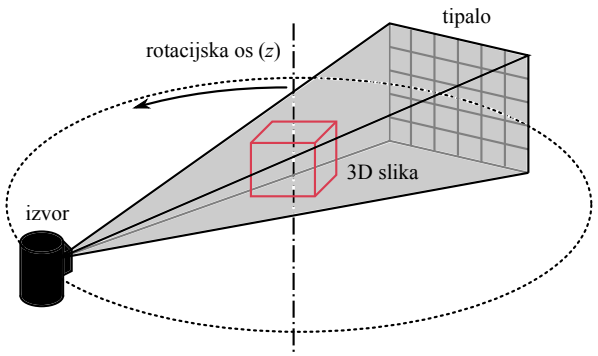
(d) Glej implementacijo funkcij `imParallelBeamProject2d` in `imFanBeamProject2d` v modulu `funkcije`.

Poglavje 11

Projekcije 3D slik

V poglavju 10 smo si podrobneje ogledali pravokotne in stožčaste projekcije 2D slik. V tem poglavju bomo pristop posplošili na 2D pravokotne in stožčaste projekcije 3D slik (slika 11.1). Pri obravnavi sledimo že opisanim postopkom projekcije 2D slik. Korak vzorčenja Δp v smeri projekcijskih daljic izberemo tako, da ustreza najmanjši velikosti slikovnega elementa $\Delta p \approx \min(\Delta x, \Delta y, \Delta z)$. Pri vzorčenju sivinskih vrednosti vzdolž projekcijskih daljic v splošnem ne naletimo na vzorčne vrednosti digitalne slike, zato je potrebno sivinske vrednosti v vzorčnih točkah izračunati z interpolacijo. V okviru tega poglavja se bomo omejili na trilinearno interpolacijo, ki je udejanjena v funkciji `interp3` modula `interp`. Za rotacijo točk okoli z koordinatne osi, ki sovpada z osjo CT naprave, bomo uporabili 3D rotacijsko matriko T_z:

$$T_z = \begin{bmatrix} \cos\gamma & -\sin\gamma & 0 \\ \sin\gamma & \cos\gamma & 0 \\ 0 & 0 & 1 \end{bmatrix}. \tag{11.1}$$

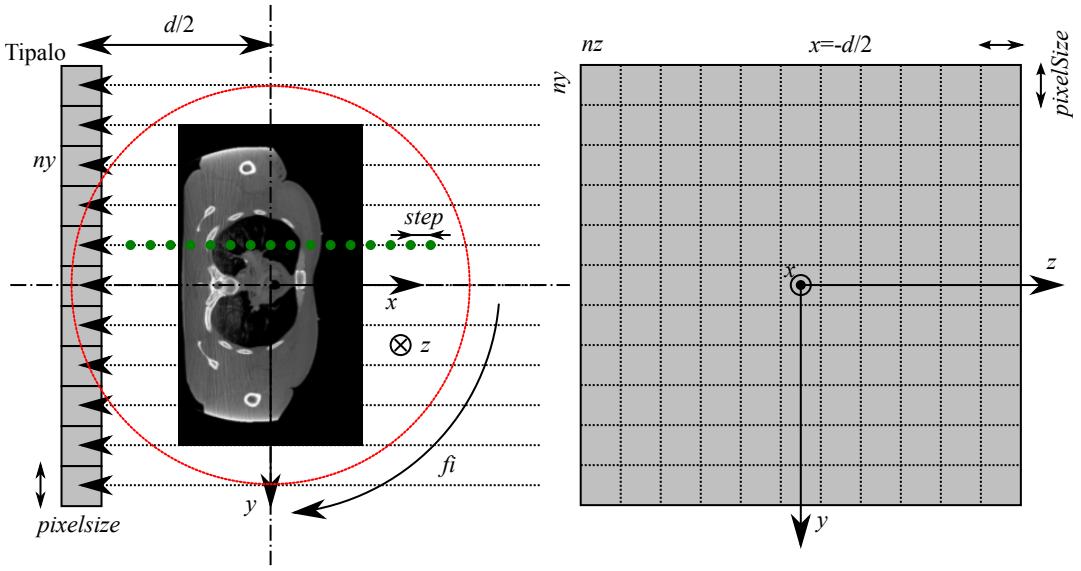


Slika 11.1: Ilustracija stožčaste projekcije 3D slike na 2D tipalo.

11.1 Naloge in vprašanja

1. Ustvarite funkcijo `imParallelBeamProject3d`, ki izračuna pravokotne 2D projekcije za kote rotacije `fi` med senzorskim poljem in y osjo. Predpostavite, da se vhodna 3D slika `img` nahaja na mreži točk, ki jo napenjajo vektorji `x`, `y` in `z`. Izvor vzporednih žarkov in tipalo naj bosta oddaljena za `d`, število slikovnih elementov tipala, ki se nahaja v yz ravnini, naj bo `ny`×`nz`, velikost kvadratnih slikovnih elementov tipala pa `pixelSize`. Os tipala naj sovpada z x koordinatno osjo. Izračunane projekcije naj bodo shranjene v 3D polju `P`, in sicer tako, da tretja razsežnost polja predstavlja kote rotacije `fi`. Korak vzorčenja v smeri projekcijskih daljic naj določa parameter `step`. Parameter `kind` naj določa tip projekcije, in sicer maksimalne vrednosti `kind='max'` ali vsote sivinskih vrednosti `kind='sum'`. Podrobnejše informacije o geometriji prikazuje slika 11.2.

```
def imParallelBeamProject3d(
    fi, step, image, x, y, z,
    d, ny, nz, pixelSize, kind='max'):
  ...
  return P
```



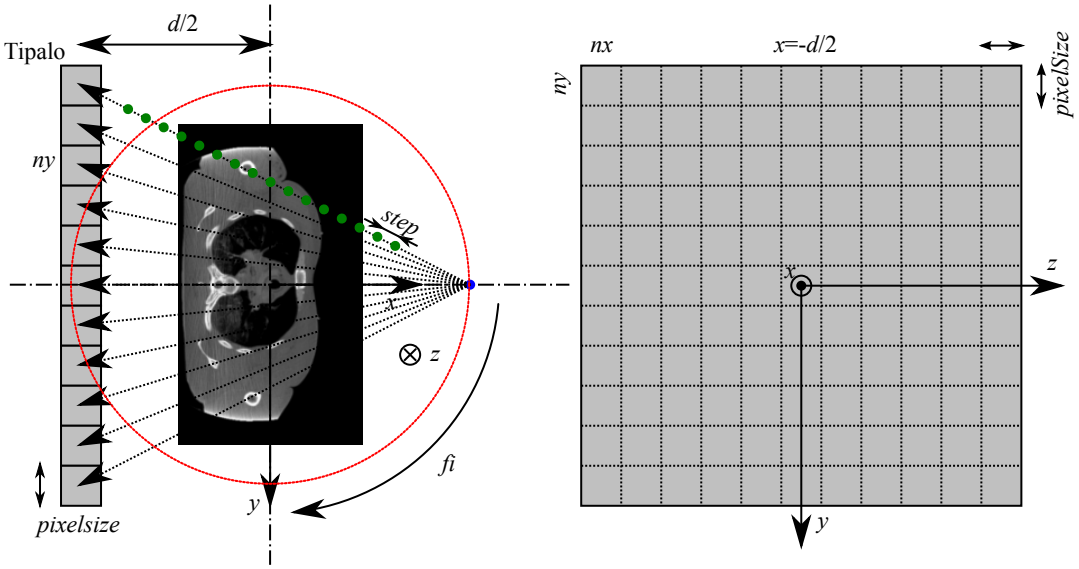
Slika 11.2: Pravokotna projekcija 3D CT slike na 2D tipalo, prikazana v yz ravnini.

(a) S funkcijo `imLoadRaw3d` naložite 3D CT sliko ct_287x165x194_uint8.raw velikosti 287 × 165 × 194 slikovnih elementov tipa `np.uint8`, ki so shranjeni v vrstnem redu `order='xyz'`, ter izračunajte 2D projekcijo maksimalne vrednosti za kot `fi`=90° med ravnino tipala in y koordinatno osjo. Izrišite izračunano projekcijo. Vrednosti parametrov naj bodo sledeče: `d=800` mm, `ny=128`, `nz=128`, `pixelSize=10` mm, `step=2` mm, geometrično središče 3D CT slike pa postavite v koordinatno izhodišče. Velikost slikovnega elementa slike je 2 × 2 × 4 mm.

(b) Na enak način izračunajte še zaporedje projekcij 3D CT slike za kote `fi` iz intervala $[0, 355]°$ s korakom $5°$ ter projekcije prikažite kot neskončno animacijo, kjer v isto grafično okno zaporedoma izrisujete posamezne projekcije. Ko izrišete zadnjo projekcijo, animacijo ponovite.

2. Ustvarite še funkcijo `imFanBeamProject3D`, ki izračuna stožčaste 2D projekcije. Pomen parametrov funkcije je enak kot pod prejšnjo točko. Podrobnejše informacije o geometriji projekcije prikazuje slika 11.3.

```
def imFanBeamProject3D(
    fi, step,
    iImg, iGridX, iGridY, iGridZ,
    image, x, y, z,
    d, ny, nz, pixelSize, kind='max'):
  ...
  return P
```



Slika 11.3: Stožčasta projekcija 3D CT slike na 2D tipalo, prikazana v yz ravnini.

(a) Naložite 3D CT sliko ct_287x165x194_uint8.raw, izračunajte 2D projekcij maksimalne vrednosti za kot `fi`=90° med ravnino senzorja in y koordinatno osjo ter izrišite projekcijo `P`. Vrednosti parametrov naj bodo sledeče: `d=800` mm, `ny=128`, `nz=128`, `pixelSize=20` mm, `step=2` mm, geometrično središče 3D CT slike pa postavite v koordinatno izhodišče.

(b) Preizkusite, kako vrednost parametra `step` vpliva na izračunano projekcijo in kako na čas, ki ga za izračun projekcije porabi funkcija `imFanBeamProject3D`.

(c) Izračunajte še zaporedje projekcij 3D CT slike za kote `fi` iz intervala $[0, 355]°$ s korakom $5°$ ter projekcije prikažite kot neskončno animacijo, kjer v isto grafično okno

zaporedoma izrisujete posamezne projekcije. Ko izrišete zadnjo projekcijo, animacijo ponovite.

11.2 Rešitve in odgovori na vprašanja

Pri odgovorih privzamemo, da se funkcije iz rešitev nahajajo v modulu `funkcije`, slikovno gradivo pa se nahaja v podmapi `poglavje_11`. Naprej uvozimo potrebne module, naložimo slike in definiramo pomožne spremenljivke.

```
import funkcije
import numpy as np
from matplotlib import pyplot as pp
from PIL import Image as im
import time

# naložimo 3D CT sliko
Ict = funkcije.imLoadRaw3d(
  './poglavje_11/ct_287x165x194_uint8.raw',
  287, 165, 194)
D, H, W = Ict.shape # velikost slike
dz, dy, dx = 4.0, 2.0, 2.0
# določimo vektorje, ki napenjajo koordinate slikovnih elementov
x = np.arange(-dx*(W - 1)*0.5, dx*(W - 1)*0.5 + 0.5*dx, dx)
y = np.arange(-dy*(H - 1)*0.5, dy*(H - 1)*0.5 + 0.5*dy, dy)
z = np.arange(-dz*(D - 1)*0.5, dz*(D - 1)*0.5 + 0.5*dz, dz)

# koti za katere bomo izračunali projekcije
fi = np.deg2rad(np.arange(0.0, 360.0, 5))
```

1. V modulu `funkcije` ustvarimo funkcijo `imParallelBeamProject3D`.

```
def imParallelBeamProject3D(
    fi, step, image, x, y, z,
    d, ny, nz, pixelSize, kind='sum'):

 kind = str(kind).lower()
  if kind == 'sum':
    fun = np.sum
  elif kind == 'max':
    fun = np.max
  else:
    raise ValueError('Vrednost parametera kind je lahko '
                     '"max" ali "sum".')

  # pripravi sliko, koordinatno mrežo točk in kote
  im = np.asarray(image, dtype='float')
  x = np.asarray(x, dtype='float')
  y = np.asarray(y, dtype='float')
  z = np.asarray(z, dtype='float')
  if isinstance(fi, float): fi = [fi]
  fi = np.asarray(fi).astype('float')

```

```
22    # vektorja, ki napenjata pozicije tipal
23    yd = pixelSize*(np.arange(ny))
24    yd -= yd.mean()
25    zd = pixelSize*np.arange(nz)
26    zd -= zd.mean()
27
28    # pozicije tipal
29    Zd, Yd = np.meshgrid(zd, yd, indexing='ij')
30    # Xd = -d*0.5
31
32    # pozicije izvorov
33    Xs = d*0.5
34    Ys, Zs = Yd, Zd
35
36    # korak vzorčenja
37    dmax = d
38    t = np.arange(0, dmax, step)
39    n = t.size
40
41    # ustvarimo daljice vzdolž katerih vzorčimo 3D sliko
42    Xsamples = np.zeros([nz, ny, n])
43    Ysamples = np.zeros_like(Xsamples)
44    Zsamples = np.zeros_like(Xsamples)
45    # smerni vektor vzorčnih daljic s = [-1, 0, 0]
46    for i in range(nz):
47      for j in range(ny):
48        Xsamples[i,j,:] = Xs + (-1)*t
49        Ysamples[i,j,:] = Ys[i,j] + (0)*t
50        Zsamples[i,j,:] = Zs[i,j] + (0)*t
51
52    # izračunamo sivine vzdolž točk projekcijskih daljic (tretja razsežnost)
53    P = np.zeros([nz, ny, fi.size])
54    tmppts = np.vstack([Xsamples.flatten(),
55      Ysamples.flatten(), Zsamples.flatten()])
56    for i in range(fi.size):
57      # rotiraj interpolacijske točke
58      tmp = np.dot(
59        np.array([[np.cos(fi[i]), -np.sin(fi[i]), 0.0],
60                  [np.sin(fi[i]), np.cos(fi[i]),  0.0],
61                  [0,       0.0,      1.0]]), tmppts)
62      # intepolacija
63      proji = interp.interp3(tmp[0], tmp[1], tmp[2], x, y, z, im)
64      proji.shape = Xsamples.shape
65      # integriraj (ali max) vzdolž daljice
66      P[:,:,i] = fun(proji, 2)
67
68    return P
```

(a) Izračunamo projekcijo za kot `fi`=90° med ravnino tipala in y koordinatno osjo (slika

Slika 11.4: Pravokotna projekcija pri kotu 90°.

11.4).

```
IctP90 = funkcije.imParallelBeamProject3D(
  np.deg2rad(90), 2.0,
  Ict, x, y, z, 800.0, 128, 128, 10.0, 'max')

pp.figure()
pp.imshow(np.squeeze(IctP90), cmap='gray')
pp.title('Pravokotna projekcija 90')
pp.show()
```

(b) Izračunamo projekcije za kote `fi` iz intervala $[0, 355]°$ s korakom 5° (slika 11.5).

```
start = time.perf_counter()
IctP = funkcije.imParallelBeamProject3D(
  fi, 2.0, Ict, x, y, z, 800.0, 128, 128, 10.0, 'max')
dt = time.perf_counter() - start
print('{} pravokotnih projekcij izračunanih v '
      '{:.1f} s.'.format(fi.size, dt))

pp.figure()
pp.ion()

frame, title = None, None
for i in range(IctP.shape[2]):
  if frame is None:
    frame = pp.imshow(
      IctP[:,:,i], vmin=0, vmax=255, cmap='gray')
  else:
    frame.set_array(IctP[:,:,i])

  if title is None:
    title = pp.title('Fi: {:.0f}'.format(np.rad2deg(fi[i])))
  else:
    title.set_text('Fi: {:.0f}'.format(np.rad2deg(fi[i])))

  pp.draw()
```

(a) 0° (b) 90° (c) 180° (d) 270°

Slika 11.5: Primer pravokotnih projekcij za kote rotacije 0°, 90°, 180° in 270°.

```
25      pp.pause(.1)
26
27  pp.ioff()
```

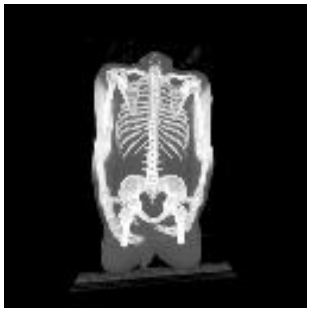
2. V modulu `funkcije` ustvarimo funkcijo `imFanBeamProject3D`.

```
1  def imFanBeamProject3D(
2      fi, step, image, x, y, z,
3      d, ny, nz, pixelSize, kind='max'):
4
5    kind = str(kind).lower()
6    if kind == 'sum':
7      fun = np.sum
8    elif kind == 'max':
9      fun = np.max
10   else:
11     raise ValueError('Vrednost parametera kind je lahko '
12                      '"max" ali "sum".')
13
14   # pripravi sliko, koordinatno mrežo točk in kote
15   im = np.asarray(image, dtype='float')
16   x = np.asarray(x, dtype='float')
17   y = np.asarray(y, dtype='float')
18   z = np.asarray(z, dtype='float')
19   if isinstance(fi, float):
20     fi = [fi]
21   fi = np.asarray(fi).astype('float')
22
23   # vektorja, ki napenjata pozicije tipal
24   yd = pixelSize*(np.arange(ny))
25   yd -= yd.mean()
26   zd = pixelSize*np.arange(nz)
27   zd -= zd.mean()
28
29   # pozicije tipal
30   Zd, Yd = np.meshgrid(zd, yd, indexing='ij')
31   Xd = -d*0.5
```

```
32
33    # pozicije izvorov
34    Xs = d*0.5
35    Ys, Zs = 0.0, 0.0
36
37    # korak vzorčenja
38    dmax = d
39    t = np.arange(0, dmax, step)
40    n = t.size
41
42    # ustvarimo daljice vzdolž katerih vzorčimo 3D sliko
43    Xsamples = np.zeros([nz, ny, n])
44    Ysamples = np.zeros_like(Xsamples)
45    Zsamples = np.zeros_like(Xsamples)
46    # smerni vektor vzorčnih daljic s = [-1, 0, 0]
47    for i in range(nz):
48      for j in range(ny):
49        # smerni vektor žarka
50        s = np.array([Xd - Xs, Yd[i,j] - Ys, Zd[i,j] - Zs],
51          dtype='float')
52        s /= np.linalg.norm(s)
53        Xsamples[i,j,:] = Xs + s[0]*t
54        Ysamples[i,j,:] = Ys + s[1]*t
55        Zsamples[i,j,:] = Zs + s[2]*t
56
57    # izračunamo sivine vzdolž točk projekcijskih daljic (tretja razsežnost)
58    P = np.zeros([nz, ny, fi.size])
59    tmppts = np.vstack([Xsamples.flatten(),
60      Ysamples.flatten(), Zsamples.flatten()])
61    for i in range(fi.size):     # rotiraj interpolacijske točke
62      tmp = np.dot(
63        np.array([[np.cos(fi[i]), -np.sin(fi[i]), 0.0],
64                  [np.sin(fi[i]), np.cos(fi[i]),  0.0],
65                  [0,       0.0,       1.0]]), tmppts)
66      # intepolacija
67      proji = interp.interp3(tmp[0], tmp[1], tmp[2], x, y, z, im)
68      proji.shape = Xsamples.shape
69      # integriraj (ali max) vzdolž daljice
70      P[:,:,i] = fun(proji, 2)
71
72    return P
```

(a) Izračunamo projekcijo za kot `fi`=90° med ravnino tipala in y koordinatno osjo (slika 11.6).

```
1  IctS90 = funkcije.imFanBeamProject3D(
2    np.deg2rad(90), 2.0,
3    Ict, x, y, z, 800.0, 128, 128, 20.0, 'max')
4
```

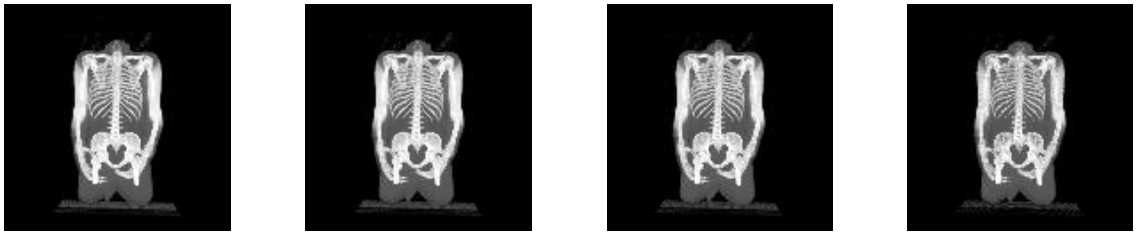


Slika 11.6: Stožčasta projekcija pri kotu 90°.

```
pp.figure()
pp.imshow(np.squeeze(IctS90), cmap='gray')
pp.title('Pravokotna projekcija 90')
pp.show()
```

(b) Ovrednotimo vpliv velikosti vzorčnega koraka `step` na izračun projekcij (slika 11.7) in čas, ki ga porabi funkcija imFanBeamProject3d za izračun projekcije (tabela 11.1 in slika 11.8). Izručune projekcij lahko izdatno pospešimo z uporabo grafičnih procesnih enot [9].

```
steps = [1, 2, 4, 8]
IctST = []
TSo = []
TSfb = []
pp.figure()
pp.suptitle('Vpliv vzorčnega koraka na izračun projekcij.')

for i in range(len(steps)):
  start = time.perf_counter()
  funkcije.imParallelBeamProject3D(
    np.deg2rad(90), steps[i],
    Ict, x, y, z, 800.0, 128, 128, 10.0, 'max')
  TSo.append(time.perf_counter() - start)
  print('Pravokotna projekcija s korakom vzorčenja {} mm '
        'izračunana v {:.0f} ms'.format(
          steps[i], TSo[-1]*1000))

  start = time.perf_counter()
  tmp = funkcije.imFanBeamProject3D(
    np.deg2rad(90), steps[i],
    Ict, x, y, z, 800.0, 128, 128, 20.0, 'max')
  IctST.append(np.squeeze(tmp))
  TSfb.append(time.perf_counter() - start)
  print('Stožčasta projekcija s korakom vzorčenja {} mm '
        'izračunana v {:.0f} ms'.format(
          steps[i], TSfb[-1]*1000))
```



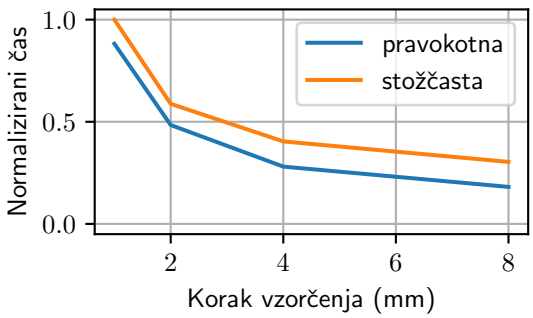
(a) 1 mm (b) 2 mm (c) 4 mm (d) 8 mm

Slika 11.7: Vpliv koraka vzorčenja na stožčaste projekcije pri kotu rotacije 90°.

```
27    pp.subplot(1, 4, i + 1)
28    pp.imshow(IctST[-1], cmap='gray')
29    pp.title('{} mm'.format(steps[i]))
30
31  pp.show()
```

Tabela 11.1: Vpliv koraka vzorčenja na čas potreben za izračun projekcije.

	Korak vzorčenja (mm)			
Projekcija	1	2	4	8
Pravokotna	1.67 s	0.89 s	0.56 s	0.36 s
Stožčasta	1.95 s	1.15 s	0.76 s	0.57 s



Slika 11.8: Vpliv koraka vzorčenja na časovno učinkovitost izračuna projekcij.

(c) Izračunamo projekcije za kote `fi` iz intervala $[0, 355]°$ s korakom 5° (slika 11.9).

```
1  start = time.perf_counter()
2  IctS = funkcije.imFanBeamProject3D(
3    fi, 2.0, Ict, x, y, z, 800.0, 128, 128, 20.0, 'max')
4  dt = time.perf_counter() - start
5  print('{} stožčastih projekcij izračunanih '
6        'v {:.1f} s.'.format(fi.size, dt))
```

(a) 0° (b) 90° (c) 180° (d) 270°

Slika 11.9: Primeri stožčastih projekcij za kote rotacije 0°, 90°, 180° in 270°.

```
7
8  pp.figure()
9  pp.ion()
10
11 frame, title = None, None
12 for i in range(IctS.shape[2]):
13   if frame is None:
14     frame = pp.imshow(
15       IctS[:,:,i], vmin=0, vmax=255, cmap='gray')
16   else:
17     frame.set_array(IctS[:,:,i])
18
19   if title is None:
20     title = pp.title('Fi: {:.0f}'.format(np.rad2deg(fi[i])))
21   else:
22     title.set_text('Fi: {:.0f}'.format(np.rad2deg(fi[i])))
23
24   pp.draw()
25   pp.pause(.1)
26
27 pp.ioff()
```

Poglavje 12

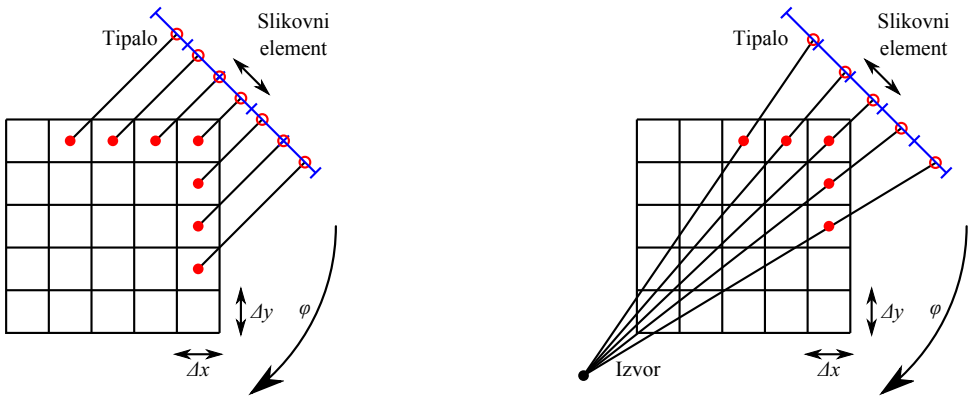
Rekonstrukcija slik s povratno projekcijo

V poglavjih 10 in 11 smo si podrobneje ogledali pravokotne in stožčaste projekcije 2D in 3D slik. V tem poglavju bomo obravnavali inverzni problem, in sicer rekonstrukcijo slike objekta na osnovi množice projekcij. Podrobnejšo obravnavo tematike lahko najdete v [10]. Eden od načinov rekonstrukcije slik je uporaba postopka povratne projekcije (angl. backprojection). Povratno projekcijo izvedemo tako, da vzdolž daljic od tipala do izvora vsakemu slikovnemu elementu rekonstruirane slike prištejemo pripadajočo sivinsko vrednost projekcije. Postopek ponovimo za vse projekcije, ki so določene s kotom rotacije φ. Eno izmed možnih izvedb opisanega postopka prikazuje slika 12.1. Pri takšni izvedbi postopka povratne projekcije središče vsakega izmed slikovnih elementov rekonstruirane slike pravokotno ali stožčasto projiciramo na linijsko tipalo. Na ta način dobljene točke (projekcije) v splošnem ne sovpadajo s koordinatami središč posameznih slikovnih elementov linijskega tipala, zato je potrebno sivinske vrednosti v projiciranih točkah določiti z 1D interpolacijo. Interpolirane sivinske vrednosti nato prištejemo pripadajočim slikovnim elementom rekonstruirane slike in postopek ponovimo za vse projekcije. Za določanje povratne projekcije pri poljubnem projekcijskem kotu je potrebno vzorčne točke rekonstruirane slike ustrezno rotirati okoli koordinatnega izhodišča. V ta namen lahko uporabimo rotacijsko matriko T_r:

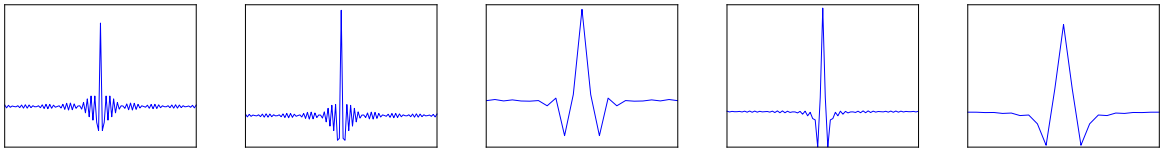
$$T_r = \begin{bmatrix} \cos\varphi & -\sin\varphi \\ \sin\varphi & \cos\varphi \end{bmatrix}. \tag{12.1}$$

S povratno projekcijo dobimo rekonstruirano sliko, ki je zamegljena (slika 12.3) in neprimerna za uporabo. Nastalo težavo rešimo tako, da uporabimo postopek s filtrirano povratno projekcijo (angl. filtered backprojection), pri kateremu signal projekcije filtriramo z visoko-prepustnim sitom in šele nato izvedemo povratno projekcijo. Tipična jedra tovrstnih 1D visoko-prepustnih filtrov so prikazana na sliki 12.2. Celotno rekonstrukcijo slike lahko povzamemo v treh točkah:

1. izvedemo konvolucijo med jedrom izbranega visoko-prepustnega filtra in danimi projekcijami,
2. določimo vzorčno mrežo točk rekonstruirane slike in



Slika 12.1: Pravokotna in stožčasta povratna projekcija.

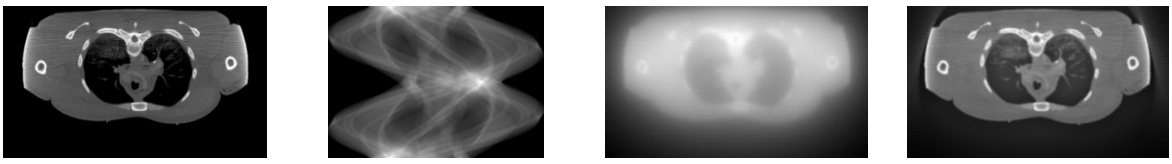


(a) Klančinski (b) Shepp-Loganov (c) Kosinusni (d) Hammingov (e) Hannov

Slika 12.2: Primeri visoko-prepustnih sit za filtriranje projekcij.

3. z opisanim postopkom izvedemo povratne projekcije vseh filtriranih projekcij.

Primer slike s pripadajočim sinogramom in rekonstrukcijo na osnovi tega sinograma z in brez filtriranja, je prikazan na sliki 12.3.



(a) CT prerez (b) Sinogram (c) PP (d) Filtrirana PP

Slika 12.3: Prerez CT slike s pripadajočim sinogramom in povratnima projekcijama (PP).

12.1 Naloge in vprašanja

1. Ustvarite funkcijo `imParallelBeamBackproject2d`, ki izračuna vsoto pravokotnih povratnih projekcij, in sicer za kote rotacije `fi` med linijskim tipalom in y osjo. Izvor vzporednih žarkov in tipalo naj bosta oddaljena za `d`, število slikovnih elementov linijskega tipala, ki se nahaja pri $y = d/2$, naj bo `n`, velikost kvadratnih slikovnih elementov tipala pa `pixelSize`. V vrsticah podatkovnega polja `sinogram` naj se nahajajo 1D projekcije za vse

kote `fi`. Predpostavite, da se izhodna 2D slika `img` nahaja na mreži točk, ki jo napenjata vektorja `x` in `y`.

```
def imParallelBeamBackproject2d(fi, sinogram, x, y,
                                d, n, pixelSize):
  ...
  return img
```

(a) Naložite sintetično sliko bproj_test.png in s funkcijo `imParallelBeamProject2d` izračunajte 2D pravokotne projekcije oziroma sinogram za kote `fi` iz intervala $[0, 359]^\circ$ s korakom 1°. Velikost slikovnega elementa testne slike znaša 1×1 mm. Sinogram testne slike uporabite za rekonstrukcijo s povratno projekcijo (`d`=1000 mm, `n`=512, `pixelSize`=1 mm).

(b) Na enak način izračunajte še 2D pravokotne projekcije prečnega prereza CT slike trupa bproj_ct.png. Velikost slikovnega elementa prereza CT slike znaša 2×2 mm. Sinogram CT prereza uporabite za rekonstrukcijo s povratno projekcijo (`d`=1000 mm, `n`=512, `pixelSize`=2 mm).

2. Ustvarite še funkcijo `imFanBeamBackproject2d`, ki izračuna vsoto stožčastih povratnih projekcij. Pomen parametrov naj bo enak enak kot pri funkciji `imParallelBeamBackproject2d` pod prejšnjo točko.

```
def imFanBeamBackproject2d(fi, sinogram, x, y,
                           d, n, pixelSize):
  ...
  return img
```

(a) Naložite 2D sintetično sliko bproj_test.png in s funkcijo `imFanBeamProject2D` izračunajte 2D stožčaste projekcije povprečne vrednosti oz. sinogram za kote `fi` iz intervala $[0, 359]^\circ$ s korakom 1°. Izračunani sinogram testne slike uporabite za rekonstrukcijo s povratno projekcijo s funkcijo `imFanBeamBackproject2d` (`d`=1000 mm, `n`=512, `pixelSize`=1 mm).

(b) Na enak način izračunajte še 2D stožčaste projekcije prečnega prereza CT slike trupa bproj_ct.png. Izračunani sinogram CT prereza uporabite za rekonstrukcijo s povratno projekcijo s funkcijo `imParallelBeamBackproject2d` (`d`=1000 mm, `n`=512, `pixelSize`=2 mm).

3. Uporabite priloženo funkcijo `create` modula `hpfilter`, da ustvarite izbrano jedro 1D visoko-prepustnega sita. Z vrednostjo parametra `kind` določite tip filtra, ki je lahko `'ramp'`, `'shepp-logan'`, `'cosine'`, `'hamming'` ali `'hann'`. S pomočjo funkcije `convolve` modula `scipy` izračunajte konvolucijo med jedrom 1D filtra in posameznimi projekcijami v sinogramih, ki ste jih ustvarili pri prejšnjih točkah.

(a) Filtrirane sinograme prikažite in jih primerjajte z izvirnimi.

(b) Ponovno izračunajte povratni projekciji testne slike in CT prereza, a tokrat uporabite filtrirane sinograme. Kako in zakaj filtrirani sinogrami vplivajo na kvaliteto rekonstruiranih slik?

4. Preučite vpliv števila in izbire 1D projekcij na kvaliteto rekonstruiranih slik.

   Projekcije vzorčite po kotu φ s korakom $1°$, $2°$, $4°$ in $8°$. Prikažite povratne projekcije prereza CT slike in obrazložite vpliv koraka vzorčenja na kvaliteto rekonstruiranih slik.

12.2 Rešitve in odgovori na vprašanja

Pri odgovorih privzamemo, da se funkcije iz rešitev nahajajo v modulu `funkcije`, slikovno gradivo pa se nahaja v podmapi `poglavje_12`. Naprej uvozimo potrebne module, naložimo slike in definiramo pomožne spremenljivke.

```
import funkcije
import numpy as np
from matplotlib import pyplot as pp
from PIL import Image as im
import time
import hpfilter
import os.path

# naloži testno slike in pripravi koordinatno mrežo
It = np.asarray(im.open('./poglavje_12/bproj_test.png'))
xt = np.arange(It.shape[1], dtype=np.float)
yt = np.arange(It.shape[0], dtype=np.float)
xt -= xt.mean()
yt -= yt.mean()

# naloži rezino CT slike in pripravi koordinatno mrežo
Ict = np.asarray(im.open('./poglavje_12/bproj_ct.png'))
xct = 2.0*np.arange(Ict.shape[1], dtype=np.float)
yct = 2.0*np.arange(Ict.shape[0], dtype=np.float)
xct -= xct.mean()
yct -= yct.mean()

fi = np.deg2rad(np.arange(0, 360))

# geometrija pravokotne projekcije
dP, nP, psP = 1000, 512, 1
# geometrija Stožčaste projekcije
dS, nS, psS = 1000, 512, 2

# ponovno računanje sinogramov
recompute = True
```

1. V modulu `funkcije` ustvarimo funkcijo `imParallelBeamBackproject2D`.

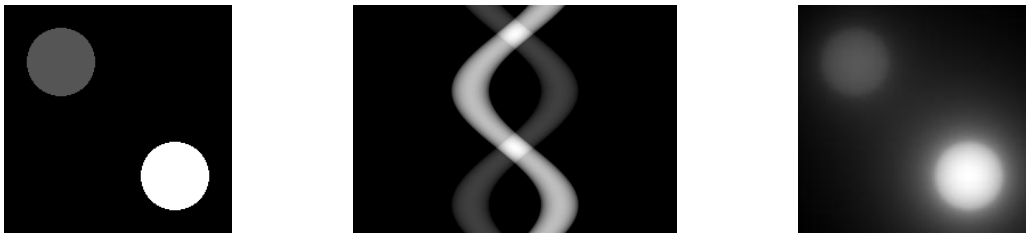
```
def imParallelBeamBackproject2D(fi, sinogram, x, y,
                                d, pixelSize):
  # priprava podatkov
  if isinstance(fi, float) or isinstance(fi, int):
    fi = [fi]
  fi = np.asarray(fi, dtype=np.float)
  sinogram = np.asarray(sinogram, dtype=np.float)
  x = np.asarray(x, dtype=np.float)
  y = np.asarray(y, dtype=np.float)
```

```
  d = float(d)
  pixelSize = float(pixelSize)

  # število slikovnih elementov linijskega tipala
  n = sinogram.shape[1]

  # koordinate slikovnih elementov rekonstruirane slike
  [Yg, Xg] = np.meshgrid(y, x, indexing='ij')
  # koordinate razporedimo v dvovrstično matriko - x v prvo, y v drugo vrstico
  pts = np.vstack([Xg.flatten(), Yg.flatten()])

  # akumulator povratne projekcije
  oBP = np.zeros(Yg.shape)
  for i in range(fi.size):
    # rotacija slikovnih elementovb v obratni smeri
    # kot bi rotirali tipalo-izvor
    Tr = np.array([[np.cos(-fi[i]), -np.sin(-fi[i])],
                   [np.sin(-fi[i]),  np.cos(-fi[i])]])
    ptsr = np.dot(Tr, pts)
    # y koordinate projiciranih slikovnih elementov na tipalu izražene z
    # naslovom tipala
    yt = ptsr[1,:]/pixelSize + n*0.5
    # poiščemo projekcije z veljavnimi naslovi
    validinds = (yt >= 0) & (yt <= n - 1)
    yt = yt[validinds]

    # z linearno interpolacijo določimo vrednosti v projiciranih točkah
    ind = np.floor(yt)
    delta = yt - ind
    intind = ind.astype('int')
    vt = sinogram[i, intind]*(1.0 - delta) +  \
      sinogram[i, np.minimum(intind + 1, n - 1)]*delta
    validinds.shape = Xg.shape

    # prištejemo izračunane vrednosti akumulatorju povratne projekcije
    oBP[validinds] += vt

  return oBP
```

(a) Izračunamo sinogram in pripadajočo povratno projekcijo testne slike z uporabo pravokotne projekcije (slika 12.4).

```
# izračunaj sinogram, če je to potrebno
if os.path.isfile('./poglavje_12/rezultati/ItSP.npy') and \
    not recompute:
  ItSP = np.load('./poglavje_12/rezultati/ItSP.npy')
else:
  ItSP = funkcije.imParallelBeamProject2d(fi, 1, It,
    xt, yt, dP, nP, psP, 'mean')
```

```
8   np.save('./poglavje_12/rezultati/ItSP.npy', ItSP)
9
10  start = time.perf_counter()
11  inds = np.arange(360)
12  ItBpP = funkcije.imParallelBeamBackproject2D(
13    fi[inds], ItSP[inds, :], xt, yt, dP, psP)
14  T = time.perf_counter() - start
15
16  pp.figure()
17
18  pp.subplot(1, 3, 1)
19  pp.suptitle('Testna slika - pravokotna projekcija')
20  pp.imshow(It, cmap='gray')
21  pp.title('Izvirna')
22
23  pp.subplot(1, 3, 2)
24  pp.imshow(ItSP, cmap='gray')
25  pp.title('Sinogram')
26
27  pp.subplot(1, 3, 3)
28  pp.imshow(ItBpP, cmap='gray')
29  pp.title('Povratna projekcija')
30
31  pp.show()
```



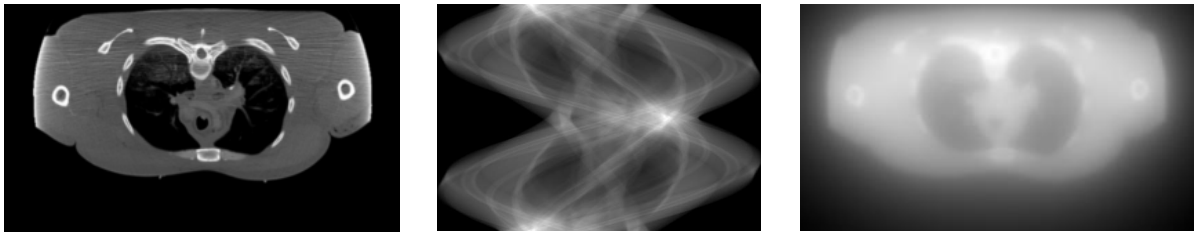
(a) Izvirna (b) Sinogram (c) Povratna projekcija

Slika 12.4: Testna slika s pripadajočim sinogramom in pravokotno povratno projekcijo.

(b) Izračunamo sinogram in pripadajočo povratno projekcijo prečnega prereza CT slike z uporabo pravokotne projekcije (slika 12.5).

```
1  # izračunaj sinogram, če je to potrebno
2  if os.path.isfile('./poglavje_12/rezultati/IctSP.npy') and \
3      not recompute:
4    IctSP = np.load('./poglavje_12/rezultati/IctSP.npy')
5  else:
6    IctSP = funkcije.imParallelBeamProject2d(
7      fi, 1, Ict, xct, yct, dP, nP, psP, 'mean')
8    np.save('./poglavje_12/rezultati/IctSP.npy', IctSP)
9
```

```
start = time.perf_counter()
inds = np.arange(360)
IctBpP = funkcije.imParallelBeamBackproject2D(fi[inds],
  IctSP[inds, :], xct, yct, dP, psP)
T = time.perf_counter() - start

pp.figure()

pp.subplot(1, 3, 1)
pp.suptitle('Rezina CT slike - pravokotna projekcija')
pp.imshow(Ict, cmap='gray')
pp.title('Izvirna')

pp.subplot(1, 3, 2)
pp.imshow(IctSP, cmap='gray')
pp.title('Sinogram')

pp.subplot(1, 3, 3)
pp.imshow(IctBpP, cmap='gray')
pp.title('Povratna projekcija')

pp.show()
```



(a) Izvirna (b) Sinogram (c) Povratna projekcija

Slika 12.5: Prerez CT slike s pripadajočim sinogramom in pravokotno povratno projekcijo.

2. V modulu `funkcije` ustvarimo funkcijo `imFanBeamBackproject2D`.

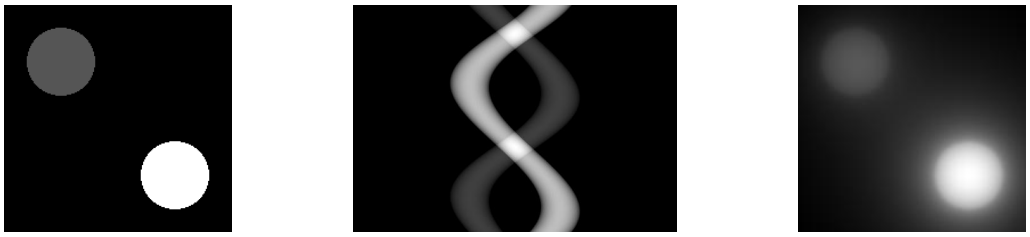
```
def imFanBeamBackproject2D(fi, sinogram, x, y,
                           d, pixelSize):
  # priprava podatkov
  if isinstance(fi, float) or isinstance(fi, int):
    fi = [fi]
  fi = np.asarray(fi, dtype=np.float)
  sinogram = np.asarray(sinogram, dtype=np.float)
  x = np.asarray(x, dtype=np.float)
  y = np.asarray(y, dtype=np.float)
  d = float(d)
  pixelSize = float(pixelSize)

```

```
13    # število slikovnih elementov linijskega tipala
14    n = sinogram.shape[1]
15
16    # koordinate slikovnih elementov rekonstruirane slike
17    [Xg, Yg] = np.meshgrid(x, y)
18    # koordinate razporedimo v dvovrstično matriko - x v prvo, y v drugo vrstico
19    pts = np.vstack([Xg.flatten(), Yg.flatten()])
20
21    # akumulator povratne projekcije enake velikosti kot sta Xg in Yg
22    oBP = np.zeros(Yg.shape)
23    for i in range(fi.size):
24      # rotacija slikovnih elementovb v obratni smeri
25      # kot bi rotirali tipalo-izvor
26      Tr = np.array([[np.cos(-fi[i]), -np.sin(-fi[i])],
27                     [np.sin(-fi[i]),  np.cos(-fi[i])]])
28      ptsr = np.dot(Tr, pts)
29      # y koordinate projiciranih slikovnih elementov na tipalu izražene z
30      # naslovom tipala
31      yt = 2.0*ptsr[1,:]/(1.0 - 2.0*ptsr[0,:]/d)/ \
32          pixelSize + n*0.5
33      # poiščemo projekcije z veljavnimi naslovi
34      validinds = (yt >= 0) & (yt <= n - 1)
35      yt = yt[validinds]
36
37      # z linearno interpolacijo določimo vrednosti v projiciranih točkah
38      ind = np.floor(yt)
39      delta = yt - ind
40      intind = ind.astype('int')
41      vt = sinogram[i, intind]*(1.0 - delta) + \
42          sinogram[i, np.minimum(intind + 1, n - 1)]*delta
43      validinds.shape = Xg.shape
44
45      # prištejemo izračunane vrednosti akumulatorju povratne projekcije
46      oBP[validinds] += vt
47
48    return oBP
```

(a) Izračunamo sinogram in pripadajočo povratno projekcijo testne slike z uporabo stožčaste projekcije (slika 12.6).

```
1  if os.path.isfile('./poglavje_12/rezultati/ItSS.npy') and \
2      not recompute:
3    ItSS = np.load('./poglavje_12/rezultati/ItSS.npy')
4  else:
5    ItSS = funkcije.imFanBeamProject2d(fi, 1, It,
6      xt, yt, dS, nS, psS, 'mean')
7    np.save('./poglavje_12/rezultati/ItSS.npy', ItSS)
8
9  start = time.perf_counter()
```

```
inds = np.arange(360)
ItBpS = funkcije.imFanBeamBackproject2D(
  fi[inds], ItSS[inds, :], xt, yt, dS, psS)
T = time.perf_counter() - start

pp.figure()

pp.subplot(1, 3, 1)
pp.suptitle('Testna slika - stožčasta projekcija')
pp.imshow(It, cmap='gray')
pp.title('Izvirna')

pp.subplot(1, 3, 2)
pp.imshow(ItSS, cmap='gray')
pp.title('Sinogram')

pp.subplot(1, 3, 3)
pp.imshow(ItBpS, cmap='gray');
pp.title('Povratna projekcija')

pp.show()
```



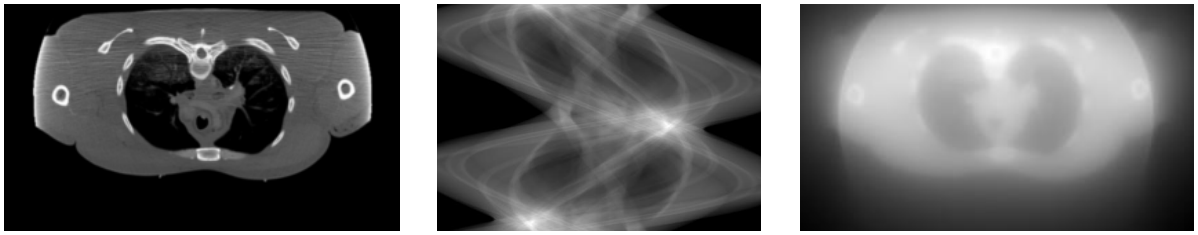
(a) Izvirna (b) Sinogram (c) Povratna projekcija

Slika 12.6: Testna slika s pripadajočim sinogramom in stožčasto povratno projekcijo.

(b) Izračunamo sinogram in pripadajočo povratno projekcijo prečnega prereza CT slike z uporabo stožčaste projekcije (slika 12.7).

```
# izračunaj sinogram, če je to potrebno
if os.path.isfile('./poglavje_12/rezultati/IctSS.npy') and \
    not recompute:
  IctSS = np.load('./poglavje_12/rezultati/IctSS.npy')
else:
  IctSS = funkcije.imFanBeamProject2d(fi, 2, Ict,
    xct, yct, dS, nS, psS, 'mean')
  np.save('./poglavje_12/rezultati/IctSS.npy', IctSS)

start = time.perf_counter()
inds = np.arange(360)
IctBpS = funkcije.imFanBeamBackproject2D(fi[inds],
```

```
13    IctSS[inds, :], xct, yct, dS, psS)
14  T = time.perf_counter() - start
15
16  pp.figure()
17
18  pp.subplot(1, 3, 1)
19  pp.suptitle('Rezina CT slike - stožčasta projekcija')
20  pp.imshow(Ict, cmap='gray')
21  pp.title('Izvirna')
22
23  pp.subplot(1, 3, 2)
24  pp.imshow(IctSS, cmap='gray')
25  pp.title('Sinogram')
26
27  pp.subplot(1, 3, 3)
28  pp.imshow(IctBpS, cmap='gray')
29  pp.title('Povratna projekcija')
30
31  pp.show()
```



(a) Izvirna (b) Sinogram (c) Povratna projekcija

Slika 12.7: Prerez CT slike s pripadajočim sinogramom in stožčasto povratno projekcijo.

3. V modulu `funkcije` ustvarimo funkcijo za filtriranje sinograma `filterSinogram`.

```
1  def filterSinogram(sinogram, filt='hamming'):
2    if isinstance(filt, str):
3      filt = hpfilter.create(filt)
4    oSinogram = np.zeros(sinogram.shape, dtype=np.float)
5    for i in range(sinogram.shape[0]):
6      oSinogram[i] = convolve(sinogram[i], filt, 'same')
7
8    return oSinogram
```

Filtriramo sinograme in izvedemo povratno projekcijo za testno sliko in prečni prerez CT slike.

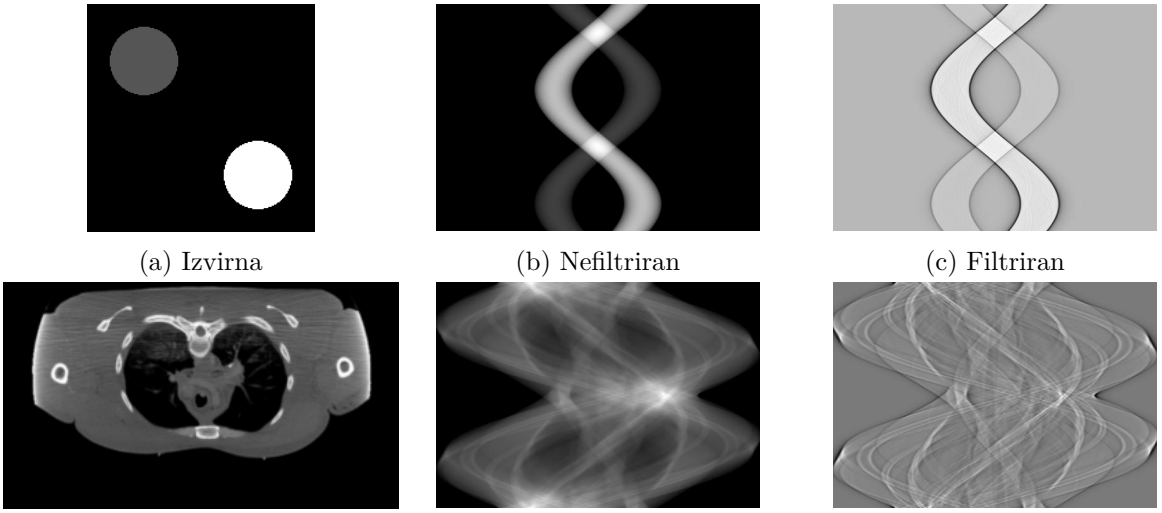
```
1  filt = hpfilter.create('hamming')
2
3  # filtriramo sinograme
```

```
4  ItSPF = funkcije.filterSinogram(ItSP, filt)
5  ItSSF = funkcije.filterSinogram(ItSS, filt)
6
7  IctSPF = funkcije.filterSinogram(IctSP, filt)
8  IctSSF = funkcije.filterSinogram(IctSS, filt)
9
10 inds = np.arange(360)
11 # rekonstrukcija s filtriranimi sinogrami
12 ItFBpP = funkcije.imParallelBeamBackproject2D(
13   fi[inds], ItSPF[inds, :], xt, yt, dP, psP)
14 ItFBpS = funkcije.imFanBeamBackproject2D(
15   fi[inds], ItSSF[inds, :], xt, yt, dS, psS)
16
17 IctFBpP = funkcije.imParallelBeamBackproject2D(
18   fi[inds], IctSPF[inds, :], xct, yct, dP, psP)
19 IctFBpS = funkcije.imFanBeamBackproject2D(
20   fi[inds], IctSSF[inds, :], xct, yct, dS, psS)
21
22 # testna slika
23 pp.figure()
24 pp.suptitle(
25   'Primerjava povratnih projekcij na podlagi '
26   'filtriranih in nefiltriranih sinogramov - '
27   'pravokotna projekcija')
28
29 pp.subplot(1, 3, 1)
30 pp.imshow(It, cmap='gray')
31 pp.title('Izvirna')
32
33 pp.subplot(1, 3, 2)
34 pp.imshow(ItBpP, cmap='gray')
35 pp.title('Nefiltrirana VP')
36
37 pp.subplot(1, 3,3)
38 pp.imshow(ItFBpP, cmap='gray')
39 pp.title('Filtrirana VP')
40
41 pp.figure()
42 pp.suptitle(
43   'Primerjava povratnih projekcij na podlagi '
44   'filtriranih in nefiltriranih sinogramov - '
45   'stožčasta projekcija')
46
47 pp.subplot(1, 3, 1)
48 pp.imshow(It, cmap='gray')
49 pp.title('Izvirna')
50
51 pp.subplot(1, 3, 2)
52 pp.imshow(ItBpS, cmap='gray')
```

```
53 pp.title('Nefiltrirana VP')
54
55 pp.subplot(1, 3, 3)
56 pp.imshow(ItFBpS, cmap='gray')
57 pp.title('Filtrirana VP')
58
59 # CT rezina
60 pp.figure()
61 pp.suptitle(
62   'Primerjava povratnih projekcij na podlagi '
63   'filtriranih in nefiltriranih sinogramov - '
64   'pravokotna projekcija')
65
66 pp.subplot(1, 3, 1)
67 pp.imshow(Ict, cmap='gray')
68 pp.title('Izvirna')
69
70 pp.subplot(1, 3, 2)
71 pp.imshow(IctBpP, cmap='gray')
72 pp.title('Nefiltrirana VP')
73
74 pp.subplot(1, 3, 3)
75 pp.imshow(IctFBpP, cmap='gray')
76 pp.title('Filtrirana VP')
77
78 pp.figure()
79 pp.suptitle(
80   'Primerjava povratnih projekcij na podlagi '
81   'filtriranih in nefiltriranih sinogramov - '
82   'stožčasta projekcija')
83
84 pp.subplot(1, 3, 1)
85 pp.imshow(Ict, cmap='gray')
86 pp.title('Izvirna')
87
88 pp.subplot(1, 3, 2)
89 pp.imshow(IctBpS, cmap='gray')
90 pp.title('Nefiltrirana VP')
91
92 pp.subplot(1, 3, 3)
93 pp.imshow(IctFBpS, cmap='gray')
94 pp.title('Filtrirana VP')
95
96 pp.show()
```

(a) Primerjava med nefiltriranimi in filtriranimi sinogrami, dobljenimi s pravokotno (slika 12.8) in stožčasto projekcijo (slika 12.9).

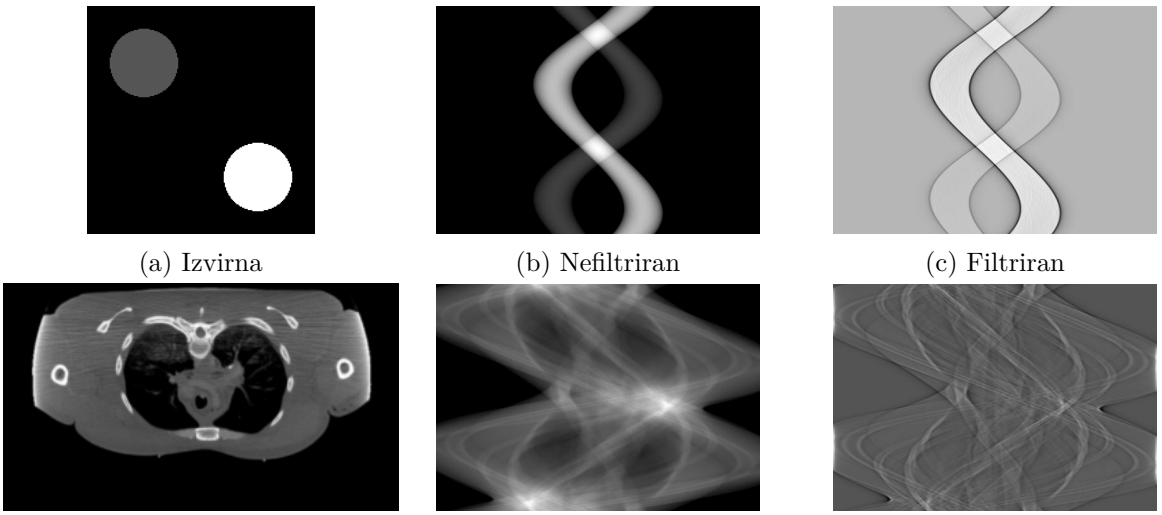
(b) Slika 12.10 prikazuje primerjavo povratnih projekcij na podlagi nefiltriranih in filtri-



(a) Izvirna (b) Nefiltriran (c) Filtriran

(d) Izvirna (e) Nefiltriran (f) Filtriran

Slika 12.8: Primerjava med nefiltriranimi in filtriranimi sinogrami, dobljenimi s pravokotno projekcijo.



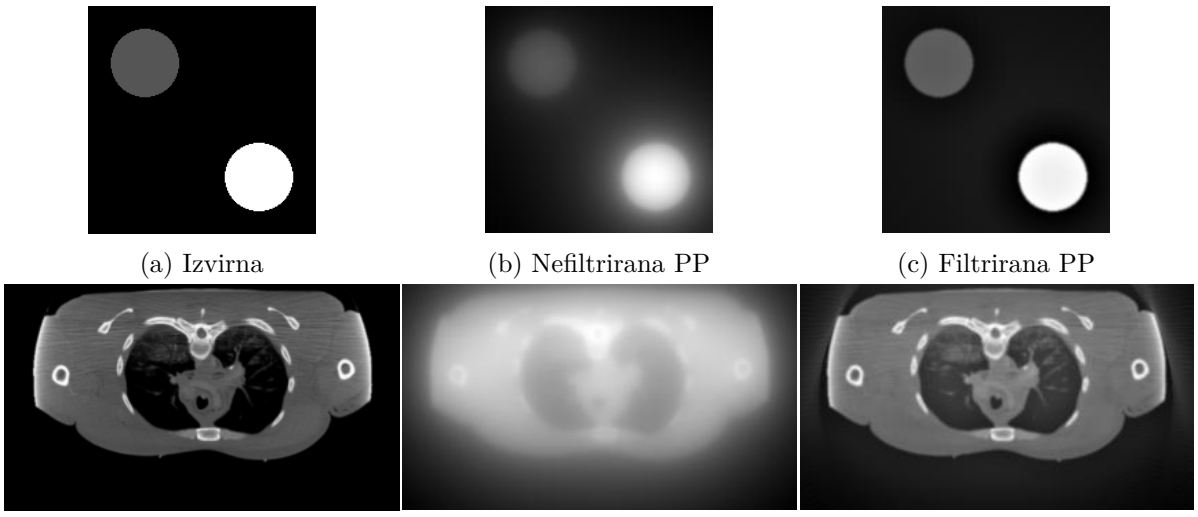
(a) Izvirna (b) Nefiltriran (c) Filtriran

(d) Izvirna (e) Nefiltriran (f) Filtriran

Slika 12.9: Primerjava med nefiltriranimi in filtriranimi sinogrami, dobljenimi s stožčasto projekcijo.

ranih sinogramov, dobljenih s pravokotno projekcijo, slika 12.11 pa še primerjavo za stožčasto projekcijo.

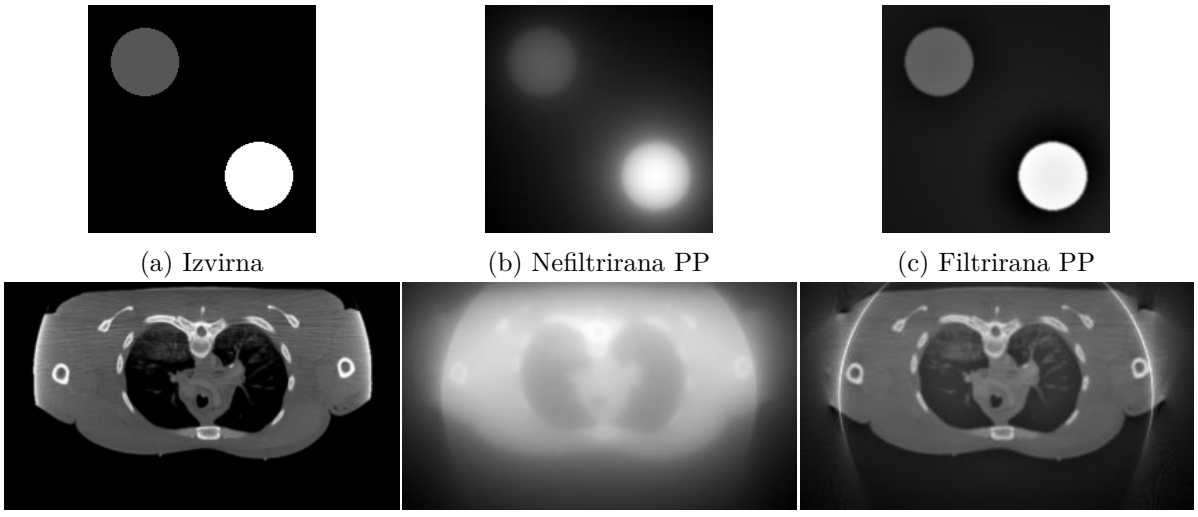
4. Ovrednotimo še vpliv števila projekcij na kakovost povratne projekcije (slika 12.12) in časovno zahtevnost izračunov (tabela 12.1 in slika 12.13).



(a) Izvirna (b) Nefiltrirana PP (c) Filtrirana PP

(d) Izvirna (e) Nefiltrirana PP (f) Filtrirana PP

Slika 12.10: Primerjava povratnih projekcij (PP) za nefiltrirane in filtrirane sinograme, dobljene s pravokotno projekcijo.



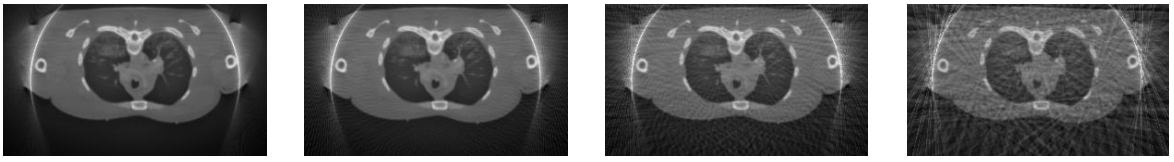
(a) Izvirna (b) Nefiltrirana PP (c) Filtrirana PP

(d) Izvirna (e) Nefiltrirana PP (f) Filtrirana PP

Slika 12.11: Primerjava povratnih projekcij (PP) za nefiltrirane in filtrirane sinograme, dobljene s stožčasto projekcijo.

```
r = [1, 2, 4, 8]
IctFbpFi = []
IctPbpFi = []

pp.figure()
```

```
6  pp.suptitle(
7    'Vpliv števila projekcij na '
8    'kakovost povratne projekcije.')
9
10 pp.subplot(1, len(r) + 1, 1)
11 pp.imshow(Ict, cmap='gray')
12 pp.title('Izvirna')
13
14 TSo = []
15 TSfb = []
16 for i in range(len(r)):
17   inds = np.arange(0, 360, r[i])
18
19   tstart = time.perf_counter()
20   tmp = funkcije.imParallelBeamBackproject2D(
21     fi[inds], IctSP[inds, :], xct, yct, dP, psP)
22   IctPbpFi.append(tmp)
23   TSo.append(time.perf_counter() - tstart)
24   print('Pravokotna povratna projekcija s korakom vzorčenja {} '
25         'izračunana v {:.0f} ms'.format(r[i], TSo[-1]*1000))
26
27   tstart = time.perf_counter()
28   tmp = funkcije.imFanBeamBackproject2D(
29     fi[inds], IctSSF[inds, :], xct, yct, dS, psS)
30   IctFbpFi.append(tmp)
31   TSfb.append(time.perf_counter() - tstart)
32   print('Stožčasta povratna projekcija s korakom vzorčenja {} '
33         'izračunana v {:.0f} ms'.format(r[i], TSfb[-1]*1000))
34
35   pp.subplot(1, len(r) + 1, i + 2)
36   pp.imshow(IctFbpFi[-1], cmap='gray')
37   pp.title('Korak {} stopinj'.format(r[i]))
38
39 pp.show()
```

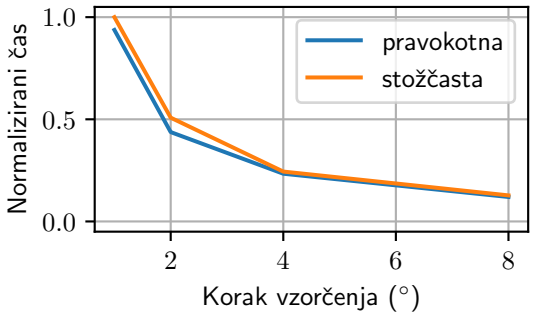


(a) $\Delta\varphi = 1°$ (b) $\Delta\varphi = 2°$ (c) $\Delta\varphi = 4°$ (d) $\Delta\varphi = 8°$

Slika 12.12: Vpliv števila projekcij na kakovost povratne projekcije kot funkcija koraka vzorčenja $\Delta\varphi$.

Tabela 12.1: Vpliv števila projekcij na čas potreben za izračun povratne projekcije.

	Korak vzorčenja $\Delta\varphi$ (°)			
Projekcija	1	2	4	8
Pravokotna	0.47 s	0.24 s	0.13 s	0.07 s
Stožčasta	0.56 s	0.30 s	0.14 s	0.07 s



Slika 12.13: Vpliv koraka vzorčenja na časovno učinkovitost izračuna stožčastih in pravokotnih povratnih projekcij.

Dodatek A

Modul interp

Modul `interp` z računsko učinkovito izvedbo interpolacijo, ki temelji na funkciji `map_coordinates` knjižnice `scipy.ndimage`.

```
# -*- coding: utf-8 -*-
from scipy.ndimage import map_coordinates
import numpy as np

def interp1(xi, x, f, order=1, **kwargs):
  '''
  Interpolacija 1D signalov, definiranih na enakomerno razporejeni
  mreži točk.

  Parametri
  ---------
  xi: numpy.ndarray
    Točke v katerih je potrebno izračunati (interpolirati)
    funkcijske vrednosti.
  x: numpy.ndarray
    Enakomerno razporejene vzorčne točke v katerih je funkcijska
    vrednost f znana.
  f: numpy.ndarray
    Funkcijske vrednosti v vzorčnih točkah x. Velikost podatkovnega
    polja f mora ustrezati (x.size,).
  order: int
    Red interpolacije. 0 - najbližji sosed, 1 - linearna ...
  kwargs: dict
    Preostali poimensko podani parametri se posredujejo funkciji
    map_coordinates.

  Vrne
  ----
  fxi: numpy.ndarray
    Funkcijske vrednosti v točkah f[xi].

  Primer
```

```
33    ------
34    >>> import numpy as np
35    >>> from matplotlib import pyplot as pp
36    >>>
37    >>> x = np.linspace(0, np.pi, 5)
38    >>> x_ref = np.linspace(x[0], x[-1], 1000)
39    >>> f = np.cos(x)
40    >>>
41    >>> xi = np.linspace(0, np.pi, 50)
42    >>> fi_lin = interp1(xi, x, f)
43    >>> fi_quad = interp1(xi, x, f, 2)
44    >>>
45    >>> pp.figure()
46    >>> pp.plot(x_ref, np.cos(x_ref), '-k', label='cos(x)')
47    >>> pp.plot(x, f, 'or', label='vzorčne točke', markersize=6)
48    >>> pp.plot(xi, fi_lin, 'xg', label='linearna', markersize=6)
49    >>> pp.plot(xi, fi_quad, 'xb', label='kvadratična', markersize=6)
50    >>> pp.legend()
51    '''
52    if x is not None:
53      x = np.asarray(x).flatten()
54      if x.size != f.size:
55        raise IndexError(
56          'Dolžini vektorjev "x" in "f" morata biti enaki!')
57      indx = (xi - x[0])*((x.size - 1)/(x[-1] - x[0]))
58    else:
59      indx = xi
60
61    return map_coordinates(np.asarray(f), np.asarray([indx]),
62                           order=order, **kwargs)
63
64  def interp2(xi, yi, x, y, f, order=1, **kwargs):
65    '''
66    Interpolacija 2D funkcij, definiranih na enakomerno razporejeni
67    mreži točk.
68    Predpostavimo, da prva razsežnost f pripada y koordinatni osi,
69    druga razsežnost pa x koordinatni osi, in sicer kot f[y, x].
70
71    Parametri
72    ---------
73    xi: numpy.ndarray
74      X komponente točk v katerih je potrebno izračunati (interpolirati)
75      funkcijske vrednosti.
76    yi: podatkovno polje
77      Y komponente točk v katerih je potrebno izračunati (interpolirati)
78      funkcijske vrednosti.
79    x: numpy.ndarray
80      Enakomerno razporejene vzorčne točke vzdolž x koordinatne osi.
81    y: numpy.ndarray
```

```
82      Enakomerno razporejene vzorčne točke vzdolž y koordinatne osi.
83    f: numpy.ndarray
84      Funkcijske vrednosti na 2D mreži vzorčnih točkah, ki jo napenjata
85      vektorja x in y. Velikost podatkovnega polja f
86      mora ustrezati (y.size, x.size).
87    order: int
88      Red interpolacije. 0 - najbližji sosed, 1 - linearna ...
89    kwargs: dict
90      Preostali poimensko podani parametri se posredujejo funkciji
91      map_coordinates.
92
93    Vrne
94    ----
95    fxyi: numpy.ndarray
96      Funkcijske vrednosti v točkah f[yi, xi].
97
98    Primer
99    ------
100   >>> import numpy as np
101   >>> from matplotlib import pyplot as pp
102   >>> from mpl_toolkits.mplot3d import Axes3D
103   >>>
104   >>> x = np.linspace(-1, 1, 10)
105   >>> y = np.linspace(-1, 1, 10)
106   >>> Y, X = np.meshgrid(y, x, indexing='ij')
107   >>> f = 1.0/(X**2 + Y**2 + 1)
108   >>>
109   >>> xi = np.linspace(0, 1, 30)
110   >>> yi = np.linspace(0, 1, 30)
111   >>> Yi, Xi = np.meshgrid(yi, xi, indexing='ij')
112   >>> fi = interp2(Xi, Yi, x, y, f)
113   >>>
114   >>> fig = pp.figure()
115   >>> ax = fig.add_subplot(111, projection='3d')
116   >>> ax.plot_wireframe(X, Y, f, color='r', label='vzorčne')
117   >>> ax.plot_wireframe(Xi, Yi, fi, color='g', label='interpolirane')
118   >>> ax.legend()
119   '''
120
121   f = np.asarray(f)
122   xi, yi = np.asarray(xi), np.asarray(yi)
123
124   if x is not None:
125     x = np.asarray(x)
126     if x.size != f.shape[1]:
127       raise IndexError(
128         'Dolžina vektorja "x" se mora ujemati '
129         's številom stolpcev "f"!')
130     indx = (xi - x[0])*((x.size - 1)/(x[-1] - x[0]))
```

```
131     else:
132       indx = xi
133 
134     if y is not None:
135       y = np.asarray(y)
136       if y.size != f.shape[0]:
137         raise IndexError(
138           'Dolžina vektorja "y" se mora ujemati '
139           's številom vrstic "f"!')
140       indy = (yi - y[0])*((y.size - 1)/(y[-1] - y[0]))
141     else:
142       indy = yi
143 
144     return map_coordinates(f, np.asarray([indy, indx]),
145                            order=order, **kwargs)
146 
147 def interp3(xi, yi, zi, x, y, z, f, order=1, **kwargs):
148     '''
149     Interpolacija 3D funkcij, definiranih na enakomerno razporejeni
150     mreži točk.
151     Predpostavimo, da prva razsežnost f pripada z koordinatni osi,
152     druga razsežnost y koordinatni osi, tretja razsežnost pa
153     x koordinatni osi, in sicer kot f[z, y, x].
154 
155     Parametri
156     -----------
157     xi: numpy.ndarray
158       X komponente točk v katerih je potrebno izračunati (interpolirati)
159       funkcijske vrednosti.
160     yi: numpy.ndarray
161       Y komponente točk v katerih je potrebno izračunati (interpolirati)
162       funkcijske vrednosti.
163     zi: numpy.ndarray
164       Z komponente točk v katerih je potrebno izračunati (interpolirati)
165       funkcijske vrednosti.
166     x: numpy.ndarray
167       Enakomerno razporejene vzorčne točke vzdolž x osi.
168     y: numpy.ndarray
169       Enakomerno razporejene vzorčne točke vzdolž y osi.
170     z: numpy.ndarray
171       Enakomerno razporejene vzorčne točke vzdolž z osi.
172     f: numpy.ndarray
173       Funkcijske vrednosti na 3D mreži vzorčnih točkah, ki jo napenjajo
174       vektorji x, y in z. Velikost podatkovnega polja f
175       mora ustrezati (z.size, y.size, x.size).
176     order: int
177       Red interpolacije. 0 - najbližji sosed, 1 - linearna ...
178     kwargs: dict
179       Preostali poimensko podani parametri se posredujejo
```

```
180     funkciji map_coordinates.
181
182   Vrne
183   ----
184   fxyzi: numpy.ndarray
185     Funkcijske vrednosti v točkah f[zi, yi, xi].
186
187   Primer
188   ------
189   >>> import numpy as np
190   >>>>
191   >>> x = np.linspace(-1, 1, 10)
192   >>> y = np.linspace(-1, 1, 10)
193   >>> z = np.linspace(-1, 1, 10)
194   >>> Z, Y, X = np.meshgrid(z, y, x, indexing='ij')
195   >>> f = 1.0/(X**2 + Y**2 + Z**2 + 1)
196   >>>
197   >>> xi = np.linspace(0, 1, 30)
198   >>> yi = np.linspace(0, 1, 30)
199   >>> zi = np.linspace(0, 1, 30)
200   >>> Zi, Yi, Xi = np.meshgrid(zi, yi, xi, indexing='ij')
201   >>>
202   >>> fi = interp3(Xi, Yi, Zi, x, y, z, f)
203   '''
204   f = np.asarray(f)
205   xi, yi = np.asarray(xi), np.asarray(yi)
206
207   if x is not None:
208     x = np.asarray(x)
209     if x.size != f.shape[2]:
210       raise IndexError(
211         'Dolžina vektorja "x" se mora ujemati '
212         's številom stolpcev "f"!')
213     indx = (xi - x[0])*((x.size - 1)/(x[-1] - x[0]))
214   else:
215     indx = xi
216
217   if y is not None:
218     y = np.asarray(y)
219     if y.size != f.shape[1]:
220       raise IndexError(
221         'Dolžina vektorja "y" se mora ujemati '
222         'številom vrstic f!')
223     indy = (yi - y[0])*((y.size - 1)/(y[-1] - y[0]))
224   else:
225     indy = yi
226
227   if z is not None:
228     z = np.asarray(z)
```

```
229     if z.size != f.shape[0]:
230       raise IndexError(
231         'Dolžina vektorja "z" se mora ujemati '
232         's številom prerezov "f"!')
233     indz = (zi - z[0])*((z.size - 1)/(z[-1] - z[0]))
234   else:
235     indy = zi
236
237   return map_coordinates(f, np.asarray([indz, indy, indx]),
238                          order=order, **kwargs)
239
240 def interpn(ti, t, f, order=1, **kwargs):
241   '''
242   Interpolacija poljubno razsežnih funkcij, definiranih na enakomerno
243   razporejeni mreži točk.
244
245   Parametri
246   ---------
247   ti: numpy.ndarray
248     Seznam komponent točk v katerih je potrebno izračunati
249     (interpolirati) funkcijske vrednosti.
250   t: numpy.ndarray
251     Seznam vektorjev enakomerno razporejenih vzorčnni točk vzdolž vseh
252     koordinatnih osi (razsežnosti).
253   f: numpy.ndarray
254     Funkcijske vrednosti na ND mreži vzoržnih točkah, ki jo napenjajo
255     vektorji v t. Velikost podatkovnega polja f mora
256     ustrezati (t[0].size, t[1].size, ..., t[-1].size).
257   order: int
258     Red interpolacije. 0 - najbližji sosed, 1 - linearna ...
259   kwargs: dict
260     Preostali poimensko podani parametri se posredujejo funkciji
261     map_coordinates.
262
263   Vrne
264   ----
265   fti: numpy.ndarray
266       Funkcijske vrednosti v točkah f[t[0], t[1], ..., t[-1]].
267
268   Primer
269   ------
270   >>> import numpy as np
271   >>> from matplotlib import pyplot as pp
272   >>> from mpl_toolkits.mplot3d import Axes3D
273   >>>
274   >>> x = np.linspace(-1, 1, 15)
275   >>> y = np.linspace(-1, 1, 10)
276   >>> Y, X = np.meshgrid(y, x, indexing='ij')
277   >>> f = 1.0/(X**2 + Y**2 + 1)
```

```
278     >>>
279     >>> xi = np.linspace(0, 1, 30)
280     >>> yi = np.linspace(0, 1, 30)
281     >>> Yi, Xi = np.meshgrid(yi, xi, indexing='ij')
282     >>> fi = interpn([Yi, Xi], [y, x], f)
283     >>>
284     >>> fig = pp.figure()
285     >>> ax = fig.add_subplot(111, projection='3d')
286     >>> ax.plot_wireframe(X, Y, f, color='r', label='vzorčne')
287     >>> ax.plot_wireframe(Xi, Yi, fi, color='g', label='interpolirane')
288     >>> ax.legend()
289     '''
290
291     f = np.asarray(f)
292
293     if t is not None:
294       N = len(t) # space dimensionality
295       if N != len(ti):
296         raise IndexError(
297           'Število elementov "ti" in "t" se mora biti enako!')
298
299         tind = []
300         for i in range(N):
301           if t[i] is not None:
302             tmp = t[i].flatten()
303             tind.append((ti[i] - tmp[0])*
304                         ((tmp.size - 1)/(tmp[-1] - tmp[0])))
305           else:
306             tind.append(ti[i])
307     else:
308       tind = ti
309
310     return map_coordinates(f, np.asarray(tind), order=order, **kwargs)
```

Dodatek B

Modul hpfilter

Modul `hpfilter` s funkcijo `create`, ki ustvari konvolucijsko jedro visokoprepustnega filtra.

```
# -*- coding: utf-8 -*-
import numpy as np

def create(kind='hamming', fstop=0.1):
  '''
  Funkcija ustvari konvolucijsko jedro visokoprepustnega filtra.

  Parametri
  ---------
  kind: str
    Tip visokoprepustnega filtra, in sicer "ramp", "shepp-logan",
    "cosine", "hamming" ali "hann".
  fstop: float
    Delež frekvenčnega področja pod Nyquistovo mejo, ki ga
    odreže filter (od 0 do 1).

  Vrne
  ----
  kernel: vektor
    Konvolucijsko jedro visokoprepustnega filtra.
  '''
  len = 64
  d = 1 - fstop
  order = max(64, 2**np.ceil(np.log2(2*len)))

  filt = 2.0*np.arange(0, order*0.5)/order
  w = 2.0*np.pi*np.arange(0, filt.size)/order

  if kind == 'ramp':
    pass
  elif kind == 'shepp-logan':
    filt[1:] *= np.sin(w[1:]/(2.0*d))/(w[1:]/(2.0*d))
  elif kind == 'cosine':
```

```
34      filt[1:] *= np.cos(w[1:]/(2.0*d))
35    elif kind == 'hamming'   :
36     filt[1:] *= 0.54 + 0.46*np.cos(w[1:]/d)
37    elif kind == 'hann':
38     filt[1:] *= (1.0 + np.cos(w[1:]/d))/2.0
39    elif kind == 'none':
40     filt.fill(1.0)
41    else:
42     raise ValueError(
43       'Vrednost parametra "kind" je lahko '
44       '"ramp", "shepp-logan", "hamming" ali "hann"!')
45
46    filt[w > np.pi*d] = 0
47    filt = np.hstack([filt, filt[-2:0:-1]])
48
49    filter = (np.fft.ifft(filt))
50    filterCutoff = 0.95
51    filterAbs = np.abs(filter)/np.abs(filter).sum()
52    filterSum = filterAbs[0]
53    for i in range(int(np.floor(filterAbs.size*0.5))):
54      filterSum += filterAbs[i + 1] + filterAbs[-1-i]
55      if filterSum >= filterCutoff:
56          break
57
58    kernel = np.hstack([filter[-1 - i:], filter[:i + 1 + 1]])
59
60    return np.real(kernel)
```

Literatura

[1] M. Bürmen, *Uvod v programski jezik Python*. Ljubljana: Založba FE, 1st ed., 2016.

[2] P. J. Deitel and H. M. Deitel, *Intro to Python for Computer Science and Data Science: Learning to Program with AI, Big Data and The Cloud*. United Kingdom: Pearson Education, 2019.

[3] J. D. Murray and W. VanRyper, *Encyclopedia of Graphics File Formats*. Boston, USA: O'Reilly Media, 2nd ed., 1996.

[4] G. D. Boreman, *Modulation Transfer Function in Optical and Electro-Optical Systems*. 1000 20th Street, Bellingham, WA 98227-0010 USA: SPIE, July 2001.

[5] D. Malacara, *Color Vision and Colorimetry: Theory and Applications*, vol. PM204. Bellingham, USA: SPIE Press, 2 ed., 2011.

[6] G. Wyszecki and W. S. Stiles, *Color Science: Concepts and Methods, Quantitative Data and Formulae*. New York: John Wiley & Sons, Inc., 2nd ed., 2000. Library Catalog: www.wiley.com.

[7] B. Likar, *Biomedicinska slikovna informatika in diagnostika*. Založba FE in FRI, 1st ed., 2008.

[8] J. V. Hajnal, D. L. G. Hill, and D. L. G. Hill, *Medical Image Registration*. CRC Press, June 2001.

[9] O. M. Dorgham, S. D. Laycock, and M. H. Fisher, "GPU Accelerated Generation of Digitally Reconstructed Radiographs for 2-D/3-D Image Registration," *Ieee Transactions on Biomedical Engineering*, vol. 59, pp. 2594–2603, Sept. 2012. Place: Piscataway Publisher: Ieee-Inst Electrical Electronics Engineers Inc WOS:000307895000024.

[10] J. Hsieh, *Computed Tomography: Principles, Design, Artifacts, and Recent Advances*. SPIE Press, 2nd ed., 2015.

[11] J. L. Prince and J. Links, *Medical Imaging Signals and Systems*. Prentice Hall, 2nd ed., 2014.

[12] P. Suetens, *Fundamentals of Medical Imaging*. Cambridge University Press, 2nd ed., 2009.

[13] "BrainWeb: Simulated Brain Database." `https://brainweb.bic.mni.mcgill.ca/brainweb/`.