Uvod v računalniško obdelavo slik

February 23, 2021

Table of Contents

Note Če še nimate nameščenega *Python*-a in *OpenCV*-ja sledite navodilom na povezavi: :ref:`navodila_python`.

Na tej vaji se boste spoznali z osnovnimi koncepti računalniške obdelave slik, programskim jezikom *Python* (predpostavljamo, da ste osnovno znanje že osvojili pri predmetu Programiranje in numerične metode v ekosistemu Pythona) in predvsem s programsko knjižnico *OpenCV*.

1 Uvod

OpenCV (*Open Source Computer Vision Library*) je knjižnica programskih funkcij, ki je v osnovi namenjena procesiranju slik v realnem času. Gre za cross-platform knjižnjico in je prosta za uporabo. Prav v tem leži ena njenih največjih prednosti, saj jo lahko uporabljamo v različnih programskih jezikih (C++, Python, Java, MATLAB) in v različnih operacijskih sistemih (Windows, Linux, macOS, Android, iOS).

Na današnji vaji bomo spoznali osnovno fukcionalnost, trike in pasti.

Note

Kot velikokrat v življenju, tudi v programiraju velja, da do končnega cilja ne vodi le ena (pravilna) pot, ampak je le-teh (skorajda *neskončno*) mnogo. Različne poti se lahko bolj ali manj primerne, boli ali manj upoštavajo različne konvencije ipd... Pri predmetu LMS se bomo poskušali čim bolj držati pravil, ki ste jih spoznali pri predmetu Programiranje in numerične metode v ekosistemu Pythona); včasih najbrž neuspešno. Koda, ki jo bomo pisali bo daleč od optimalne Python kode, ampak bo zelo *skriptna* in čim bolj podobna pseudo kodi.

1.1 Vaja 1: branje in pisanje slik z diska

Napišimo za začetek enostaven *Python* skripto, ki bo z diska prebrala sliko in jo prikazala.

Če pogedamo sedaj vrstico po vrstico.

V prvi vrstici vljučimo *OpenCV*.

V vrstici *3* preberemo sliko z diska s klicom `cv2.imread()`. `cv.` pomeni, da bomo sedaj klicali funkcijo iz knjičnice, `imread` pa je ime funkcije. Vidite lahko, da funkcija sprejme samo en argument. Ta je tipa string in opisuje pot na disku do slike.

Note Ta navodila so pisana v *OS Ubuntu/Linux*. Zato so vse poti napisane v Linux "stilu". Če uporabaljte *OS Windows*, je zadeva nekoliko bolj zapletena. Za opisovanje poti imate nekako 3 možnosti. Recimo, da je vaša slika na lokaciji `"C:\SomeFolder\SomeOtherFolder\my_image.jpg"`.

1. pot podate z dvojno levo poševnico (*backslash*-om), torej: `"C:'\\SomeFolder\\SomeOtherFolder\\my_image.jpg"`
2. pot podate s poševnico (*slash*-om), torej: `"C:/SomeFolder/SomeOtherFolder/my_image.jpg"`
3. pot podate kot *dobeseden string* (*literal string*), torej: `r"C:\SomeFolder\SomeOtherFolder\my_image.jpg"`

Težava je v tem, da če pot kopirate, bodo v imenu leve poševnice, ti pa so rezervisani za *posebne znake* (*special characters*, npr. `"\n"` pomeni novo vrstico, `"\t"` pomeni *tab*, `"\s"` presledek itd...).

Python ne loči med `"` in `'`, torej lahko uporabljete kateregakoli, morata pa biti **v paru enaka znaka**.

Warning Ne glede na operacijski sistem pa **morate podati celotno pot; torej od diska do končnice!** Izjema so t.i. *relativne poti*.

V četrti vrstici sliko prikažemo s klicom funkcije `cv2.imshow()`. Funkcija ima 2 parametra: prvi je ime okna (glej *sliko 1*), z drugim pa podamo sliko/matriko, ki jo želimo izrisati.

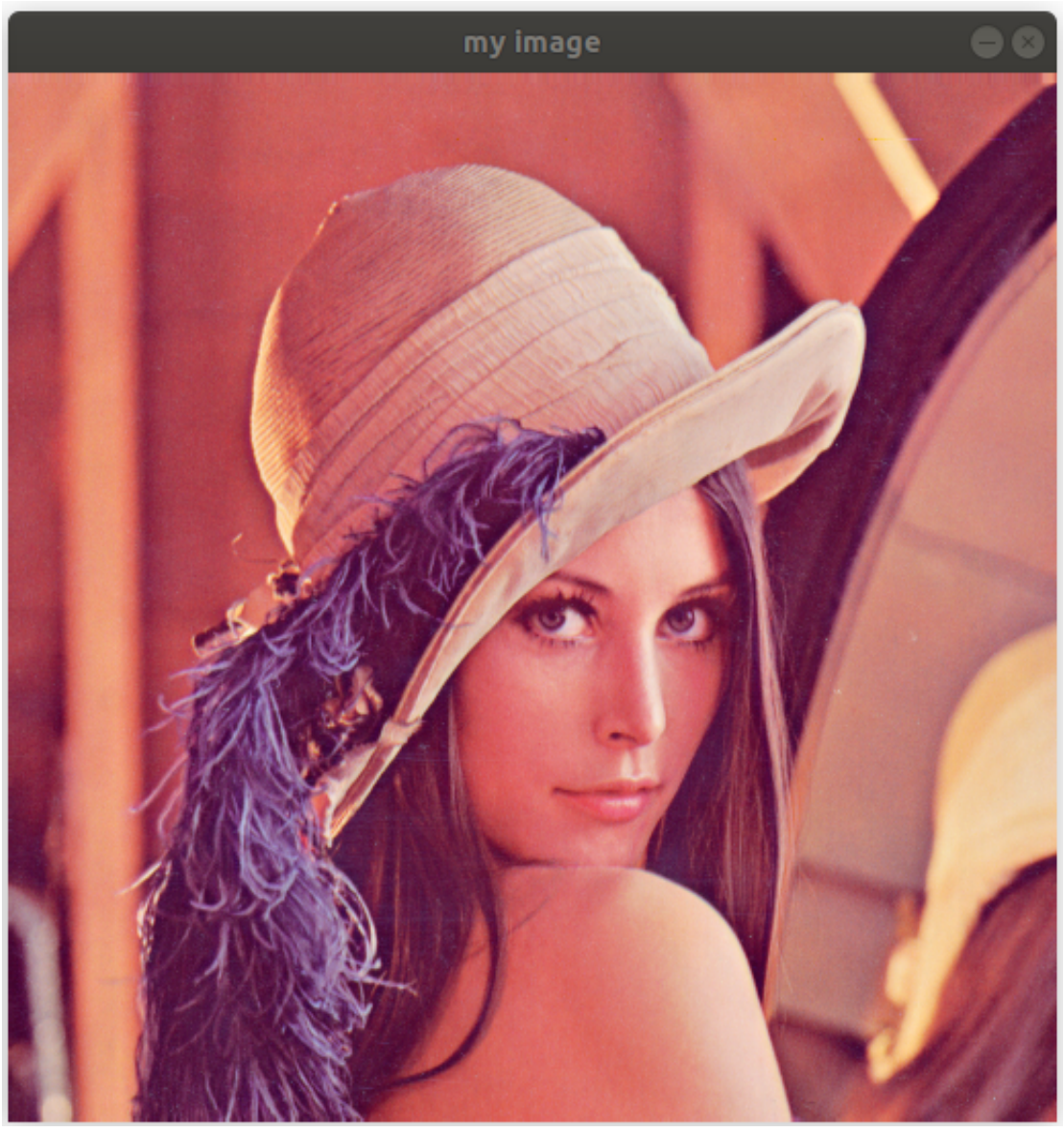

Figure 1.1. Slika 1. Prikazana slika.

Note Slike so v računalništvu matrike. Če ima slika *640x480* (*širina* x *višina*) pikslov, imamo torej matriko *480x640* (pazite, pri matrikah **najprej podamo število vrstic, potem število stolpcev!**) kjer vrednost vsakega elementa popoisuje intenziteto posameznega piksla (v primeru sivinske - *grayscale* slike). Če imamo opravka z barvno sliko (npr. *RGB* barvni model) imamo 3 matrike, po

eno za posamezen kanal (barvo).

V peti vrstici s fukcijo `cv2.waitKey()` čakamo na pritisk katerikoli tipke. Če pogledamo v dokumentacijo fukcije, je tam zapisamo `retval = cv.waitKey([, delay] )`. Branje in razumevanje dokumentacije je ena najpomembnejših vrlin, ki jo kot programerji moramo osvojiti. Poglejmo, kaj lahko iz tega razberemo. Znotaj `()` vidimo `[, delay]`. To pomeni, da fukcija sprejme 1 argument, ki pa je opcijski (to povesta `[]`): to pomeni, da ga lahko podamo, ni pa nujno. Če nadalnje preberemo dokumentacijo piše: *"The function waitKey waits for a key event infinitely (when delay≤0) or for delay milliseconds, when it is positive."*. Torej, če argumenta `delay` ne podamo, bo funkcija čakala na pritist tipke "v neskončnost", če pa jo, bo počakala samo število milisekund, kot smo ga podali s parametrom `delay`. Vidimo lahko tudi, da fukcija vrne `retval`. V dokumentacij piše: *"It returns the code of the pressed key or -1 if no key was pressed before the specified time had elapsed."*. Ugotovimo lahko, katero tipko smo pritisni (vrne **ASCII** kodo, najdete jo lahko v stolpcu **Dec** v tabeli, ali s *Python* klicom `ord('<vaša črka>')`).

Po 5. vrstici bi se torej izvajanje programa moralo ustaviti in prikazati sliko. Ko pritisnemo tipko ze izvajanje skripte nadaljuje z zadnjo vrstico, kjer vsa odprta okna zapremo.

Note Če ne kličemo funkcije `cv2.destroyAllWindows()` običajno okno ostane odprto in neodzivno (ne moremo ga zapreti), kar je lahko moteče.

Kot lahko vidimo, je prikazana slika barvna, torej jo sestavljajo 3 kanali, rdeč, zelen in model. Dopolnimo skripto tako, da sliko razdelimo na posamezne kanale, jih prikažemo, premešamo in sliko spet shranimo.

Dodali smo vrstice *5-10*, *12* in *15*.

Note Čeprav običajo slišite "RGB", *OpenCV* uporablja zaporedje **blue, green, red**.

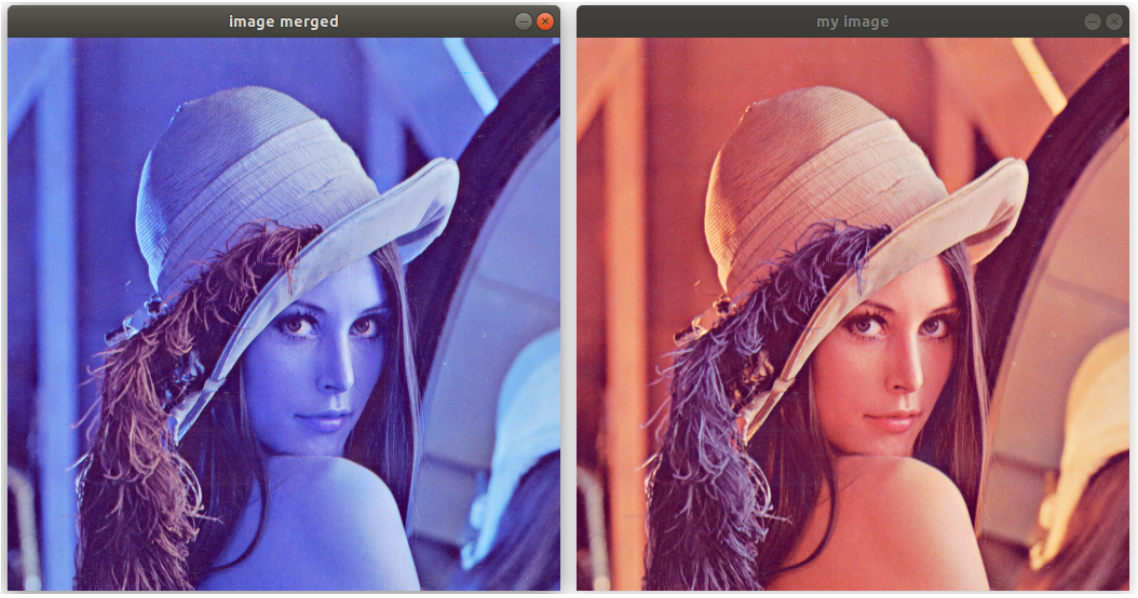

Figure 1.2. Slika 2. Originalna *Lenna* in *Lenna* s premešanimi kanali.

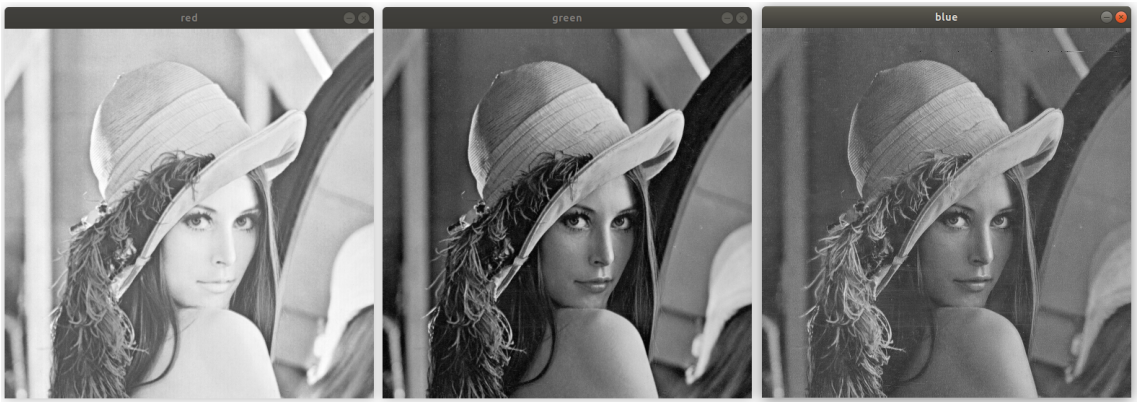

Figure 1.3. Slika 3. Posamezni kanali.

1.2 Vaja 2: zajemanje in prikazovanje slike s kamere

V tem delu vaje bomo pogledali, kako s pomočjo *OpenCV*-ja zajamemo sliko s kamere, ki je priklopljena na računalnik.

Skripta je sedaj nekoliko daljša.

Najpomembnejša razlika v primerjavi s prejšnjo skripto je vrstica *3*. `VideoCapture` je razred za zajem videa iz različnih video datotek, sekvenc slik in *hardware*-a - kamer. Če podrobneje pogledamo dokumentacij omenjenega razreda lahko vidimo, da imamo na voljo 5 razlčnih konstruktorjev :

1. `<VideoCapture object> = cv.VideoCapture( )` ... *default* konstruktor, za enkrat za nas ni zanimiv
2. `<VideoCapture object> = cv.VideoCapture( filename )` ... "odpremo" video datototeko
3. `<VideoCapture object> = cv.VideoCapture( filename, apiPreference )` ... "odpremo" video datototeko in določimo s katero *video* knjižnico
4. `<VideoCapture object> = cv.VideoCapture( index )` ... "odpremo" kamero
5. `<VideoCapture object> = cv.VideoCapture( index, apiPreference )` ... "odpremo" kamero in določimo s katero *video* knjižnico

Warning Kadarkoli "odpremo" nek kos *hardware*-a, ga moramo potem tudi zapreti (glejte vrstico *20*). Če tega ne storimo, v ob naslednjem poskusu odpiranja to ne bo mogoče, saj je npr. kamera že odprta. Tudi v dokumentaciji piše: *In C API, when you finished working with video, release CvCapture structure with cvReleaseCapture()...*

Note Dokumentacija *OpenCV*-ja je primarno napisana za *C++*; za *Python* je precej skopa in moramo velikokrat prebrati kaj piše za *C++* in to "preversti" v *Python*.

V vrstici *3* lahko vidite, da smo uporabiti opcijo *4*. `index` je tipa `int` in določa, katero kamero želimo odpreti.

Note *Python* tako kot vsak resen programski jezik začenja šteti z **0** in ne z **1** (*I'm looking at you,*

MATLAB!); torej, *index prvega elementa* je *0*!

Klic nam vrne objekt tipa *VideoCapture*, ki ga "ujamemo" v spremenljivo `cam`. V vrstivi 6 začnemo neskončno **while** zanko.

V vrstici *7* kličemo metodo razreda *VideoCapture* `read()`. V dokumentaciji lahko preberemo, da metoda *Grabs, decodes and returns the next video frame.*. Sintaksa je zapisana kot `retval, image = cv.VideoCapture.read( [, image] )`. Spet lahko vidimo, da sprejme en opcijski parameter `image`, ki pa ga lahko "ujamemo" tudi na izhodnji strani. Glede `retval` pa piše *false if no frames has been grabbed*. Tako v vrsticah *8-9* preverim, ali je branje uspelo in če ni *neskončno* zanko zaključimo.

V vrstici *10* preverimo, kakšno je treutno stanje spremenljivke `flip` in eventuelno izvedemo vrstico *11*. **Vaša naloga je**, da na spletu poiščete dokumentacijo *OpenCV* funkcije `cv2.flip()` in **ugotovite**, kaj funkcija naredi in kaj sta oba parametra.

V vrsticah *14-17* imamo 2 klica povezana s tipko, ki smo jo (eventuelno) pritisnili v vrstici *13*. *27* je koda tipke *ESC* in v tem primeru zanko zaključimo, če pa pritisnemo tipko *f* spremenimo *True/False* stanje spremenljivke `flip`.

1.3 Vaja 3: globina slike (*image depth*) in prikazovanje slike

Če uporabljate *Jupyter Notebook*, spremenljivke ostanejo v spominu tudi, ko se izvajanje skripte konča. V naslednjo celico napišite klic:

```
print(type(frame[0, 0, 0]))
output: <class 'numpy.uint8'>
```

Slika v spremenljivki `frame` je v resnici trodimenzionalna matrika, vredsnoti znotraj oglatih oklepajev pa koordinata piksla, ki ga želimo izpisati: *prva* vrednost določa vrstico, *druga* stolpec in *tretja* kanal, če imamo RGB sliko. Če je slika sivinska moramo podati samo 2 vrednosti. Vidimo lahko, da je element tipa `uint8`. To pomeni, da je *unsigned integer* (celo število brez predznaka) popisan z osmimi biti. Najmanjša vrednost, ki jo torej lahko opišemo je torej '0' (binarno *0000 0000*, hex *00*), največja pa *255* (binarno *1111 1111*, hex *ff*). **Vprašanje:** kaj se zgodi, če *uint8* spemelnjivki z vrednostjo *255* prištejemo *1*?

Napišimo kratko skripto, ki bo izrisala 3 slike: črno, sivo in belo.

Kot lahko vidite, smo sedaj uvozili še eno knjižnico, numpy (vrstica *1*). Gre za "*The fundamental package for scientific computing with Python*". Ker je njeno ime dolgo, ga okrajšamo, tako da jo lahko kličemo kot `np.`. V vrstici *4* določimo dmenzije slik. V vrsticah *6-8*, naredimo najprej 3 matrike enic, ki jih potem pomnožimo z *0* za črno sliko, *126* za sivo in *255* za belo. Ko slike izrišemo, vidimo nekaj takega:

Figure 1.4. Slika 4. Bela, siva in črna generirana slika.

V vrsticah *6-8* smo nastavili, da je posamezen element matrik tipa `uint8`. To imenujemo **image depth** (v prostem prevodu *"globina" slike*). Spremenimo sedaj *globino* naših slik v `float32`, torej število s plavajočo vejico in 32 biti, in skripto ponovno poženimo.

Figure 1.5. Slika 5. Enake 3 slike kot na *sliki 4*, le da so sedaj v formatu *float32*.

Opazimo lahko, da sta sedaj tako slika *gray* kot slika *white* beli. Zakaj?

Note Zapomnite si: če z *OpenCV* funkcijo `imshow()` izrisujemo slike *globine* **integer**, bo **0 ... črna**, **255 ... bela**, če pa izrisujemo slike *globine* **float**, bo **0.0 ... črna**, **1.0 ... bela**!

1.4 Vaja 4: filtriranje slik

Pri procesiranju slik se pogosto poslužujemo različnih filtriranj. Filtriranje običajno temelji na principu konvolucije. V *OpenCV*-ju imamo na voljo različne predefinirane filtre, lahko pa ga ustvarimo sami. Karakteristike filtra določa njegovo *jedro* (*angl. kernel*). Poglejmo si primer enega najbolj uporablajnih filtrov - *Gaussov filter*.

```
kernel:
 [[0.17820325]
 [0.21052228]
 [0.22254895]
 [0.21052228]
 [0.17820325]]

difference:
 0.0
```

V vrstici *7* uporabimo predefiniran *Gaussov* filter v funkciji `GaussianBlur`. Medtem v vrstici 9 generiramo eno-dimenzionalen Gaussov filter z enakimi parametri, potem pa ga uporabimo v `sepFilter2D`. Tej funkciji podamo posebej tilter v *X* in *Y* smeri. V *output* oknu lahko vidimo izpisa; vrednosti *kernela* in pa razliko med obema slikama, ki nam potrdi, da smo v obeh primerih sliko enako sfiltrirali.

Naredimo sedaj lasten *Moving Average* filter.

```
[[0.00826446 0.00826446 0.00826446 0.00826446 0.00826446 0.00826446
0.00826446 0.00826446 0.00826446 0.00826446 0.00826446]
 [0.00826446 0.00826446 0.00826446 0.00826446 0.00826446 0.00826446
0.00826446 0.00826446 0.00826446 0.00826446 0.00826446]
 [0.00826446 0.00826446 0.00826446 0.00826446 0.00826446 0.00826446
0.00826446 0.00826446 0.00826446 0.00826446 0.00826446]
 [0.00826446 0.00826446 0.00826446 0.00826446 0.00826446 0.00826446
0.00826446 0.00826446 0.00826446 0.00826446 0.00826446]
 [0.00826446 0.00826446 0.00826446 0.00826446 0.00826446 0.00826446
0.00826446 0.00826446 0.00826446 0.00826446 0.00826446]
 [0.00826446 0.00826446 0.00826446 0.00826446 0.00826446 0.00826446
0.00826446 0.00826446 0.00826446 0.00826446 0.00826446]
 [0.00826446 0.00826446 0.00826446 0.00826446 0.00826446 0.00826446
0.00826446 0.00826446 0.00826446 0.00826446 0.00826446]
 [0.00826446 0.00826446 0.00826446 0.00826446 0.00826446 0.00826446
0.00826446 0.00826446 0.00826446 0.00826446 0.00826446]
 [0.00826446 0.00826446 0.00826446 0.00826446 0.00826446 0.00826446
0.00826446 0.00826446 0.00826446 0.00826446 0.00826446]
 [0.00826446 0.00826446 0.00826446 0.00826446 0.00826446 0.00826446
0.00826446 0.00826446 0.00826446 0.00826446 0.00826446]
 [0.00826446 0.00826446 0.00826446 0.00826446 0.00826446 0.00826446
0.00826446 0.00826446 0.00826446 0.00826446 0.00826446]]
```

V vrsticah *7* in *8* generiamo *11x11* matriko enic in jo v normaliziramo (vsota vseh uteži mora biti *1.0*). Na *sliki 6* sta prikazani izvorna in filtrirana slika.

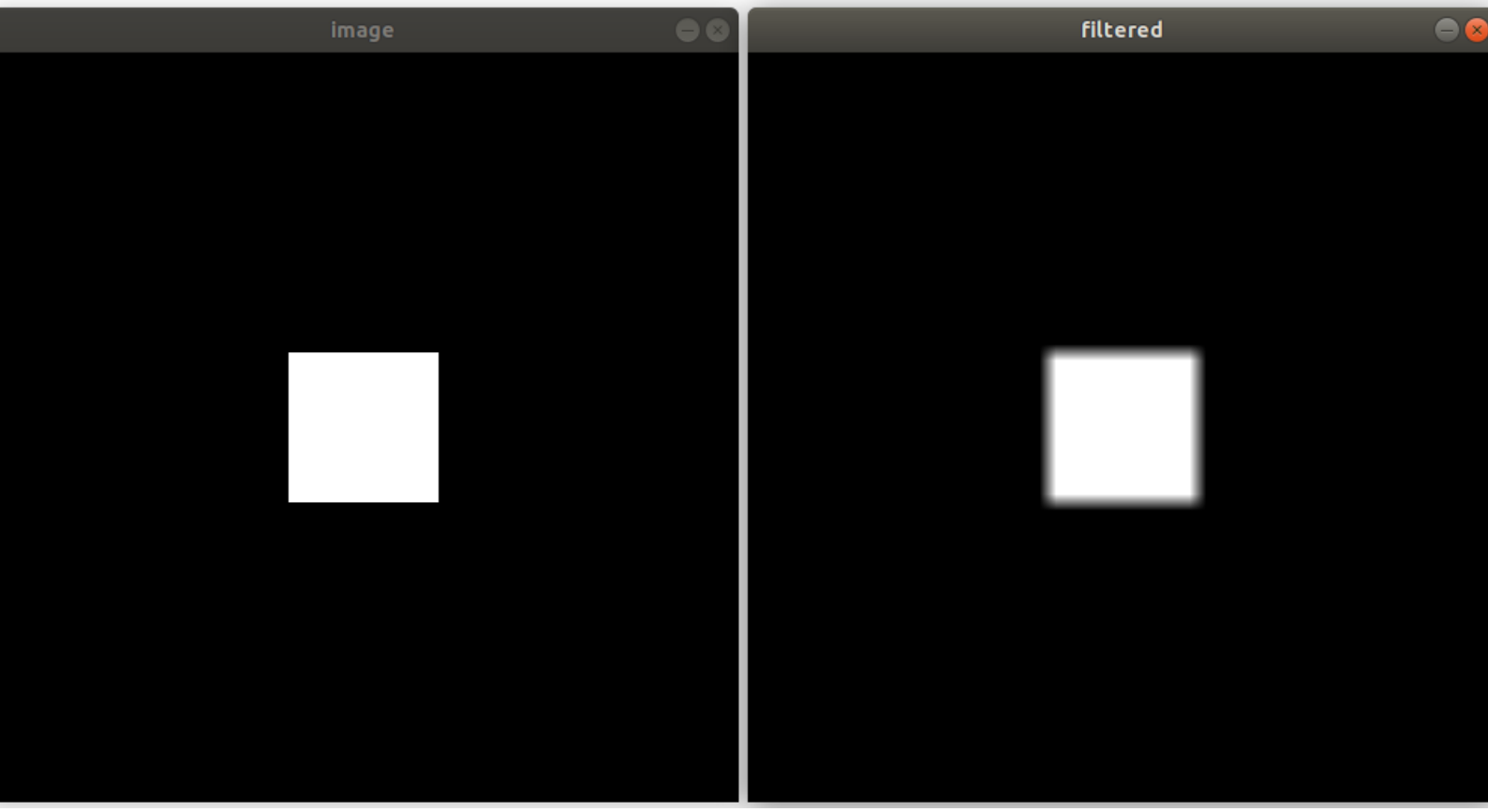

Figure 1.6. Slika 6. Izvorna in filtrirana slika.

2 Median filter

Median filter je filter, kjer aktivnemu pikslu pripišemo vrednost mediane znotraj konvolucijskega okna. Gre za filter, s katerim lahko odstranimo npr. termičen šum.

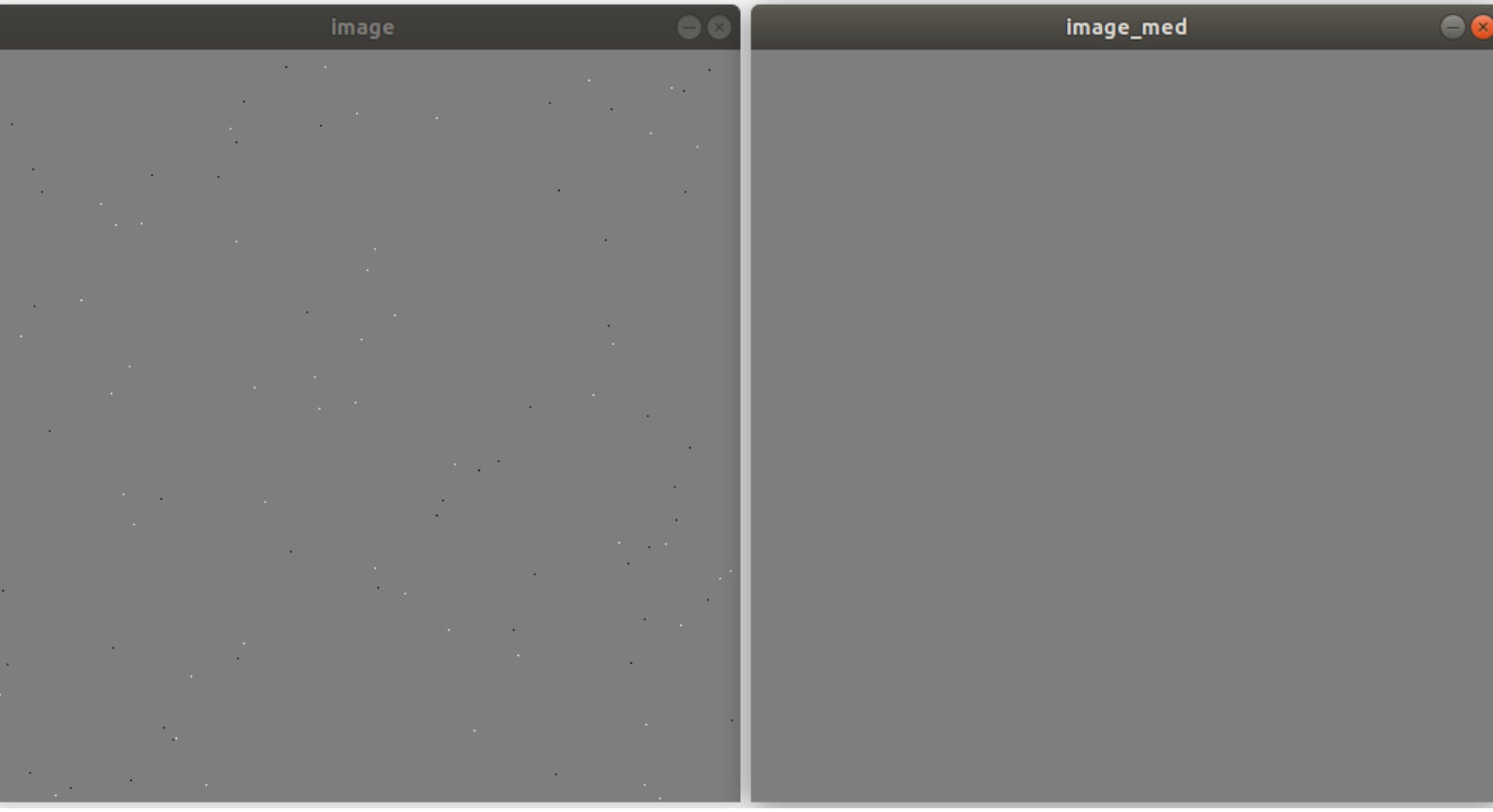

Figure 2.1. Slika 7. Izvorna in filtrirana slika z *median* filtrom.