Vaja 1: Od kamere do preproste detekcije predmetov

Vsako od vaj zagovarjate pri asistentu. Za uspešen zagovor morate rešiti vse naloge, razen tistih, ki so označene z ★. Če katera od nalog zahteva skiciranje ali ročno računanje, morate to pokazati na zagovoru. Programerske naloge morate rešiti samostojno in izvorno kodo ter rezultate predstaviti na zagovoru. Če kode, ki rešuje vaše naloge, ne boste poznali v detajle, lahko asistent vaš zagovor zavrne. V primeru zavrnitve zagovora, ga lahko ponovno opravljate z zamudo, a bo zato vaše končno število točk kaznovano s faktorjem **0.75**.

Za rešen obvezen del nalog lahko dobite največ **75** točk, dodatne naloge pa vam lahko prinesejo do **25** dodatnih točk. Za pomanjkljive rešitve ali slabo teoretično znanje vam asistent lahko odšteje točke.

Pogoste napake in nasveti
- Zamenjava x in y osi
- Napačen podatkovni tip: float, uint8
- Napačno definicijsko območje podatkov: [0,255], [0,1]
- Rešitev razvijajte na enostavnih primerih (lahko so sintetični)

---

1. Obdelava slik

Namen te naloge je spoznavanje z osnovno funkcionalnostjo `OpenCV` in z zapisom slik v obliki matrik.

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

Nalaganje in prikaz slike:

```
import cv2
import matplotlib.pyplot as plt
import numpy as np

ime_slike = 'umbrellas.jpg'
im = cv2.imread(ime_slike)
im = cv2.cvtColor(im, cv2.COLOR_BGR2RGB)

print(f'{im.shape=}')
print(f'{im.dtype=}')

plt.clf()
plt.imshow(im)
plt.show()
```

Slika, ki ste jo naložili, ima tri barvne kanale (RGB: Red-Green-Blue), in je shranjena v 3D matriki velikosti visina $\times$ sirina $\times$ kanali $= h \times w \times 3$. Ugotovite velikost slike preko atributa `.shape` ter tip matrike z atributom `.dtype`.

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

Pri prikazu sivinske slike se bo ta izrisala s privzeto barvno tabelo. Barvna tabela pomeni način, kako vam MatPlotLib prikaže odtenek sivine. Na primer, temne odtenke lahko prikaže modro, svetle pa z rdečo. Poskusite spremeniti barvno tabelo v sliki s spreminjanjem parametra `cmap` funkcije `imshow`. Poskusite z vrednostmi `jet`, `bone`, `gray`, npr. `plt.imshow(im, cmap='gray')`.

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

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

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

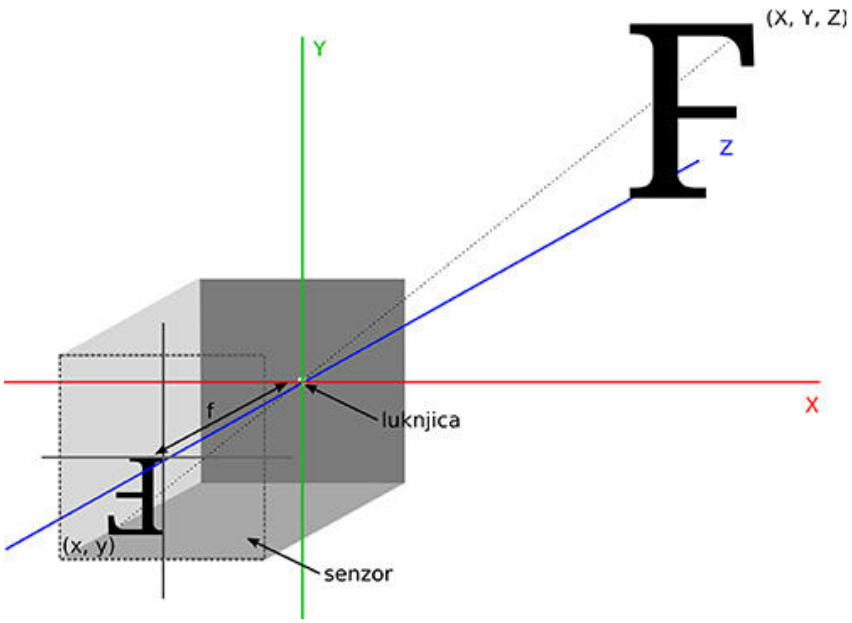
2. Kamera

Namen te naloge je spoznati model kamere z luknjico (angl. *pinhole camera*) ter osnove digitalnega zajema slike. Kot ste izvedeli na predavanjih, model kamere z luknjico opisuje par preprostih geometrijskih razmerij:

$$x = -f\frac{X}{Z}$$

$$y = -f\frac{Y}{Z},$$

ki točko v prostoru s koordinatami (X, Y, Z) preko goriščne razdalje f poveže z njeno projekcijo na sliki (x, y). Grafično je povezava prikazana na Sliki 1.



Rešite naslednje naloge:

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

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

c) Zakaj se kamere z luknjico uporabljajo bolj kot teoretičen model in ne tudi v praksi? Naštejte prednosti in slabosti kamer z lečami.

Sliko, ki je projicirana na zadnjo stran kamere, zajamemo v digitalno obliko z (največkrat) matričnim senzorjem. Parametra senzorja kamere sta njegova velikost in gostota. Velikost navadno podajamo v milimetrih, gostoto pa v točkah na palec (angl. dots per inch oz. DPI). Standardna konverzija med enotama je $1 \text{ inch} = 2.54 \text{ cm}$.

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

★ e) (10 točk) Za naslednjo nalogo boste potrebovali spletno kamero ter knjižnico `OpenCV`. Spletna kamera sicer ni čista kamera z luknjico, vsebuje lečo, zato pri zajemu slike prihaja do določene stopnje popačenja. Kljub temu z uporabo kamere preizkusite zakonitosti, ki jih opisuje enačba kamere z luknjico v praksi. Kamero postavite na statično mesto s pogledom na mizo. Pred kamero na izmerjeno razdaljo od nje postavite objekt. S programom za zajem slik iz kamere pridobite več (vsaj šest) zaporednih slik objekta pri čemer objekt premikajte na različne razdalje in zabeležite oddaljenost od kamere. Nato posamezne slike odprite s python skripto in zabeležite

višino objekta v številu slikovnih elementov (pomagajte si s knjižnico `PyPlot` (funkcija `plt.ginput()`), lahko pa to naredite tudi v programu za urejanje slik). Na podlagi višine v slikovnih elementih in oddaljenosti od kamere lahko določite kakšna bo velikost v slikovnih elementih pri drugi razdalji od kamere. Preverite oceno še z dejansko meritvijo in ocenite napake.

3. Histogrami

V nadaljevanju si bomo ogledali, kako gradimo in prikazujemo histograme. Histogrami so uporaben način opisa slike, s katerim lahko veliko povemo o porazdelitvi slikovnih elementov v sliki, s tem pa do neke mere tudi nekaj o vsebini slike. Zaradi njihove preprostosti in uporabnosti so zelo razširjeni, pogosto se uporabljajo npr. v fotografiji.

Podana je 3-bitna sivinska slika:

5	3	2	7	1
7	1	0	0	0
4	5	7	1	1
1	3	2	1	1
5	3	1	6	3

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

b) Na podlagi izračunanega histograma določite kumulativni histogram slike.

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

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

e) Implementirajte funkcijo `histogram_stretch()`, ki vhodno sliko popravi tako, da njene vrednosti raztegne preko celotnega spektra sivinskih nivojev. Za okvirni potek algoritma se zgledujte po prosojnicah s predavanj. Ker funkcija izvede enako operacijo na vseh slikovnih elementih slike, implementirajte celotno funkcijo brez zank z uporabo matričnih operacij.

Namig: Za vhodno sliko lahko določite najvišjo $v_{\max}$ in najnižjo $v_{\min}$ sivinsko vrednost z `np.max(I)` in `np.min(I)`.

Pomembno: Pri nalogi ni dovoljena uporaba integriranih funkcij, izračun novih

vrednosti morate implementirati sami.

Za preizkus funkcije preberite z diska datoteko `phone.jpg` (ki je že sivinska slika) in zanjo izrišite histogram z 256 celicami. S histograma lahko opazite, da najnižja in najvišja sivinska vrednost v sliki nista 0 in 255. Nato na sliki opravite razteg histograma ter prikažite rezultat v obliki slike in njenega histograma (histogrami naj vsebujejo 256 celic).

f) Upragovanje slike je zelo uporabno pri implementaciji preproste detekcije objektov, vendar je določitev praga pogosto problematična. Preizkusite OpenCV funkcijo `cv2.threshold()`, ki zna določiti prag samodejno z uporabo različnih metod, vključno z *Otsujevo metodo* (angl. Otsu's method)[1], ki temelji na analizi histograma slike.

g) Kakšen histogram je idealen za Otsujevo metodo, ali, povedano drugače, na kakšnih slikah ta metoda deluje dobro?

4. Barvni prostori

Barvo lahko zapišemo v različnih barvnih prostorih. Vsak barvni prostor ima svoje značilnosti. V okviru naloge si boste pogledali, kako lahko s pretvorbo med RGB in HSV barvnima prostoroma izvedemo razčlenitev slike na podlagi barve.

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

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

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

d) Različni barvni prostori so koristni tudi za upragovanje. V RGB barvnem prostoru je recimo težko na preprost način določiti področja, ki pripadajo določenemu barvnemu odtenku. Napišite skripto, ki sliko `trucks.jpg` upraguje po modrem kanalu za vrednost praga 150. Prikažite originalno sliko ter binarno sliko eno ob drugi (uporabite `plt.subplot()`.

[1] Otsu's method

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

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

```
import numpy as np
  plt.figure()
  plt.imshow(np.meshgrid(np.linspace(0, 1, 255), np.ones((10, 1)))[0],
cmap=plt.cm.hsv)
  plt.show()
```

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

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

5. Regije in morfološke operacije

V tej nalogi si boste ogledali, kako iz binarne slike izluščiti posamezne regije, kako te regije zapisati na različne načine, ter, kako si lahko z uporabo morfoloških operacij pomagamo pri detekciji regij v šumnih slikah.

Za razčlenitev regij na sliki se zaradi njegove časovne predvidljivosti najpogosteje uporablja algoritem zaporednega označevanja regij, oziroma povezanih komponent, ki ste ga obravnavali na predavanjih. Za začetek ponovimo delovanje algoritma z ročnim reševanjem preprostega primera:

a) Podana je binarna slika, za katero določite rezultat algoritma barvanja regij po prvem in drugem sprehodu po sliki ter povezave med oznakami za okolico $\mathcal{N}_4$. Na sliki so z 1 označena področja objektov v sliki, celice z vrednostjo 0 pa so ozadje.

0	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	1	1
0	1	0	1	0	0	1	1	1
1	1	1	1	0	1	0	0	0
1	0	0	1	1	0	0	0	0
0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	1	1	1
0	1	0	0	0	1	1	1	1
1	1	0	0	0	0	0	0	1

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

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

Binarne slike, pridobljene iz realnih podatkov, pogosto niso tako lepe kot slike, s katerimi smo delali do sedaj. V realnih slikah je veliko šuma, ki se v binarni sliki odraža kot drobne regije ali luknjice v regijah, ki motijo nadaljnje procesiranje. V takih primerih so lahko uporabimo morfološke operacije, s pomočjo katerih lahko tak šum odpravimo. V okviru te naloge bomo spoznali morfološki operaciji skrči (`erode`) in razširi (`dilate`) v kontekstu odstranjevanja šuma iz binarne slike.

Najprej na kratko ponovite delovanje obeh operacij z naslednjo nalogo:

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

$$K = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

0	0	0	0	0	0	0	0	1
0	0	0	1	0	0	0	1	0
0	1	0	1	0	0	1	1	0
1	1	1	1	0	1	0	0	0
1	0	0	1	1	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	1	1	0
0	1	0	0	0	1	1	1	0

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

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

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

★ h) (5 točk) Pridobljeno znanje o uporabi morfoloških operacij preizkusite še na bolj realnem primeru. Preberite sliko `bird.jpg`, spremenite jo v sivinsko ter določite prag, da dobite čim boljšo masko objekta. Ker popolne maske ne morete dobiti samo z globalnim pragom, jo izboljšajte z uporabo morfoloških operacij. Število točk, ki jih boste dobili za nalogo je odvisno od kakovosti rezultata.

6. Delo z živimi slikami

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

funkcijo bomo definirali tipko *q*.

```
import numpy as np
  import cv2

  cap = cv2.VideoCapture(0)  # 0 = privzeta spletna kamera
  # Namesto številke lahko podamo tudi ime video datoteke

  # Zanka za pridobitev naslednje slike
  while(True):
      # Poskusimo pridobiti trenutno sliko s spletne kamere
      ret, frame = cap.read()

      # Če to ni mogoče (kamera izključena, itd.), končamo z izvajanjem
funkcije
      if not ret:
          break

      # Prikaži trenutno sliko
      cv2.imshow('frame', frame)

      # Ob pritisku tipke 'q' prekini izvajanje funkcije
      if cv2.waitKey(1) & 0xFF == ord('q'):
          break

  # Zapri kamero in okno
  cap.release()
  cv2.destroyAllWindows()
```

V zgornjem primeru za prikaz uporabimo funkcijo `cv2.imshow()`, ki ji podamo ime okna in sliko. Več slik naenkrat lahko izrišemo z zaporednimi klici funkcije z različnimi imeni oken.

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

★ c) (10 točk) Izberite si predmet poljubne barve in z uporabo znanja iz prejšnjih nalog upragujte sliko s takšnimi vrednostmi, da boste iz nje izluščili izbrani predmet. Z uporabo OpenCV funkcije `cv2.findContours()` izrišite konturo okoli izluščenih predmetov. Število točk, ki jih boste dobili za to nalogo bo odvisno od robustnosti delovanja vaše metode.