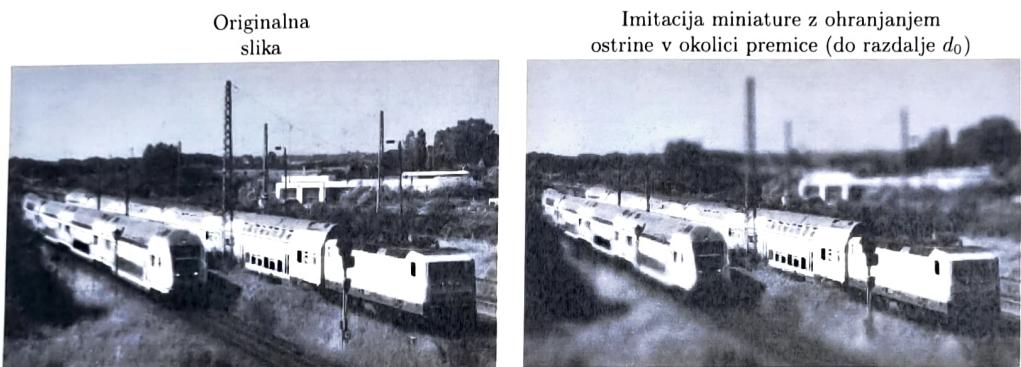


Zagovor laboratorijskih vaj - Naloga 7.1

Pripravila: Gašper Podobnik & Tomaž Vrtovec

Naloga

Fotografiranje na podlagi vrtenja in premika objektiva (*ang. tilt-shift photography*) se uporablja za odpravo perspektivnih distorzij ter za nastavljanje selektivne ostrine slike, saj omogoča prikaz objektov, ki so lahko precej različno oddaljeni od točke zajema slike, z enako ostrino. Selektivno ostrino slike lahko simuliramo tudi s pomočjo postopkov za obdelavo slik, in sicer tako, da ohranimo ostrino slike samo vzdolž izbranega področja (npr. vzdolž premice in njene okolice), z oddaljevanjem od tega področja pa ostrino slike zmanjšujemo oz. sliko vedno bolj gladimo. Rezultat je zelo poznani učinek *imitacije miniature*.



Dana je dvodimenzionalna (2D) sivinska slika `train-400x240-08bit.raw` velikosti $X \times Y = 400 \times 240$ slikovnih elementov, ki je zapisana v obliki surovih podatkov (RAW) z 8 biti na slikovni element. Slikovni elementi so izotropni, za merske enote pa povsod upoštevajte *slikovni element*.

1. Napišite funkcijo za izračun razdalje med točko in premico:

```
def distancePoint2Line(iL, iP):
    """
    Funkcija za izracun razdalje tocke od premice
    """

    return oD
```

kjer vhodni argument $iL = [k, n]$ predstavlja parametre premice (k je naklon premice, n je presečišče premice z ordinatno osjo), $iP = [x_p, y_p]$ pa koordinate točke, medtem ko izhodni argument oD predstavlja razdaljo d med točko (x_p, y_p) in premico $ax + by + c = 0$, in sicer:

$$d = \frac{|ax_p + by_p + c|}{\sqrt{a^2 + b^2}}.$$

Preizkusite funkcijo na podatkih $k = 0,22$ in $n = 100$.

2. Napišite funkcijo za izračun koeficientov jedra filtra za **uteženo** Gaussovo povprečenje:

```
def weightedGaussianFilter(iS, iWR, iStdR, iW):
    """
        Funkcija za izracun jedra fitlra za linearne utezeno Gaussovo
        povprecenje
    """

    return oK, oStd
    iWR[0], iWR[1]
```

kjer vhodni argument $iS = [M, N]$ predstavlja vektor velikosti filtra v obliki matrike velikosti $N \times M$ (naravni lihi števili), $iWR = [w_1, w_2]$ vektor mejnih vrednosti uteži (meje intervala), $iStdR = [\sigma_1, \sigma_2]$ vektor mejnih standardnih odklonov Gaussove porazdelitve v 2D (meje intervala), $iW = w$ pa podano utež. Izhodni argument oK predstavlja matriko jedra filtra, $oStd = \sigma$ pa njegov standardni odklon.

Pri vrednosti uteži w_1 naj bo standardni odklon filtra enak σ_1 , pri vrednosti uteži w_2 naj bo enak σ_2 , med skrajnima mejama intervala pa naj bo standardni odklon linearne porazdeljen glede na utež. Gaussova porazdelitev v 2D s povprečno vrednostjo nič je enaka:

$$f(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}.$$

Preizkusite funkcijo na podatkih $M = 7$, $N = 7$, $w_1 = 0$, $w_2 = 10$, $\sigma_1 = 0,1$ in $\sigma_2 = 10$.

3. Napišite funkcijo za generiranje slike z imitacijo miniature:

```
def imitateMiniature(iImage, iS, iStdR, iL, iD0):
    """
        Funkcija za izracun imitacije miniature
    """

    return oImage, oVal
```

kjer vhodni argument $iImage$ predstavlja vhodno sliko, $iS = [M, N]$ predstavlja vektor velikosti uteženega Gaussovega filtra v obliki matrike velikosti $N \times M$ (naravni lihi števili), $iStdR = [\sigma_1, \sigma_2]$ vektor mejnih standardnih odklonov Gaussove porazdelitve v 2D (meje intervala), $iL = [k, n]$ predstavlja parametre premice (k je naklon premice, n je presečišče premice z ordinatno osjo), $iD0 = d_0$ pa predstavlja mejno razdaljo od premice. Izhodni argument $oImage$ predstavlja sliko z imitacijo miniature, $oVal = [d(x, y), \sigma(x, y)]$ pa matriko razdalj od premice d in pripadajočega standardnega odklona uteženega Gaussovega filtra σ za vsako točko (x, y) slike (vsaka vrstica matrike ustreza eni točki).

Imitacijo miniature generirajte tako, da slike pri razdaljah $d \leq d_0$ od dane premice ne glađite, pri razdaljah $d > d_0$ pa pričnete sliko gladiti v vse smeri z **linearnim** povečevanjem moči glajenja, in sicer pri razdalji d_0 začnete s standardnim odklonom filtra σ_1 , standardni odklon σ_2 pa filter doseže pri največji mogični razdalji od dane premice. Problem glajenja na področju tej slike rešite z razširitvijo prostorske domene slike na podlagi ekstrapolacije sivinskih vrednosti.

Preizkusite funkcijo na dani sliki s podatki iz točke 1 in 2 ter $d_0 = 25$. Izrišite sliko z imitacijo miniature ter potek standardnega odklona uteženega Gaussovega filtra σ v odvisnosti od razdalje d od dane premice.

Zagovor laboratorijskih vaj - Naloga 2023-1

Pripravila: Gašper Podobnik & Tomaž Vrtovec

Naloga

Kolaž (beseda izhaja iz francoske besede *coller*, ki pomeni (*z*)*lepiti skupaj*) je tehnika likovnega ustvarjanja, ki se uporablja predvsem v vizualnih umetnostih, najdemo pa jo tudi v glasbi, s katero nastane umetnost iz skupka različnih oblik in tako ustvari novo celoto.

Vaša naloga bo preslikava intenzitet vhodne RGB slike, iz katere boste nato ustvarili enostaven kolaž.

Vhodna RGB slika



Dana je dvodimensionalna (2D) barvna (RGB) slika `planina-509x339-08bit.jpeg` velikosti $X \times Y \times RGB_{dim} = 509 \times 339 \times 3$ slikovnih elementov, ki je zapisana v formatu `jpeg` z 8 biti na slikovni element. Slikovni elementi so izotropni.

1. Naložite dano sliko in napišite funkcijo za standardizacijo intenzitet slike:

```
def std_channels(iImage, x_R, y_R, x_G, y_G, x_B, y_B):
    return oImage
```

kjer vhodni argument `iImage` predstavlja vhodno RGB sliko, argumenti $\{x_R, x_G, x_B\}$ in $\{y_R, y_G, y_B\}$ pa predstavljajo parametre x in y za posamezen barvni kanal RGB slike. Izhodni argument `oImage` je standardizirana slika velikosti $X \times Y \times 3$ (X in Y sta dimenzijsi vhodne slike).

Standardizacija je sivinska preslikava, ki je določena s spodnjo enačbo:

$$I_{std} = \frac{I - x}{y},$$

kjer sta x in y parametra te preslikave, I predstavlja vhodno sliko, I_{std} pa standardizirano sliko.

Funkcija naj standardizira vsak (barvni) kanal slike posebej, torej ločeno za rdeč, zelen in moder barvni kanal, pri čemer se za vsak kanal uporabi pripadajoča vrednost iz vhodnih parametrov. Izhodna slika je RGB slika, sestavljena iz vseh treh standardiziranih kanalov. *Namig:* pred standardizacijo ustreznno nastavite podatkovni tip slike.

Vhodno sliko standardizirajte, pri čemer za manjkajoče vhodne parametre uporabite sledeče vrednosti: $x_R=100$, $y_R=50$, $x_G=60$, $y_G=30$, $x_B=40$, $y_B=100$. Rezultat shranite v spremenljivko `img_std`. Dobljeno sliko tudi prikažite.

2. Napišite funkcijo, ki pripravi predlogo območij (koščkov), ki sestavljajo kolaž, za sliko velikosti $X \times Y$:

```
def image2pieces(X, Y, N):
    return oLabelImage, oPts
```

kjer vhodna argumenta X in Y predstavljata velikost slike, ki jo želimo razdeliti na območja, argument N pa število območij, ki jih želimo imeti v novi sliki. Izhodni argument oLabelImage je matrika velikost $X \times Y$, v kateri elementi z isto vrednostjo označujejo posamezno območje slike, oPts pa matrika velikosti $N \times 2$, ki nosi koordinate vseh središč območij. Število različnih vrednosti v matriki oLabelImage je torej enako številu območij N .

Funkcija naj nključno generira N parov koordinat $\text{oPts} = \{(x_j, y_j), j = 1, 2, \dots, N\}$, ki ležijo znotraj meja slike in predstavljajo središča območij. Pri tem si lahko pomagate s funkcijo `np.random.choice` (koordinate točk naj bodo cela števila). Pazite na to, da bodo vse točke različne (da ne pride do podvajanja). Vrednost posameznega elementa v izhodni matriki oLabelImage naj bo enaka indeksu najbližjega središča iz matrike oPts , torej j oz. $j - 1$ (zaradi Pythonovega načina indeksiranja).

Preizkusite funkcijo, pri čemer za vrednosti X in Y uporabite velikost vhodne slike, N pa nastavite na 255. Dobljeno matriko oLabelImage prikažite kot sivinsko sliko, pri čemer nastavite `cmap='jet'`.

3. Napišite funkcijo, ki iz vhodne slike ustvari kolaž:

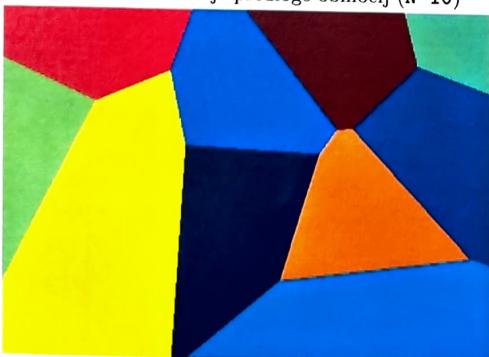
```
def img2collage(iImage, N):
    return oImage, oLabelImage
```

kjer vhodni argument iImage predstavlja vhodno sliko, N pa število območij, ki jih želimo imeti v izhodni sliki.

S funkcijo `image2pieces` najprej pridobite predlogo območij (koščkov) za vhodno sliko. Na podlagi pridobljene predlage območij nato izračunajte intenzitete izhodne slike tako, da bo intenziteta na posameznem območju izhodne slike enaka povprečni intenziteti istoležnega območja na vhodni sliki (povprečno intenzito ločeno izračunajte za vsak barvni kanal).

Preizkusite funkcijo na sliki `img_std` za dva različna parametra, in sicer za $N=255$ in $N=1000$, ter prikažite dobljeni sliki. Prikažite tudi obe predlogi območij (oLabelImage).

Primer vizualizacije predloge območij ($N=10$)



Primer izhodne RGB slike kolaža ($N=255$)



Zagovor laboratorijskih vaj - Naloga 2022-1

Pripravila: Gašper Podobnik & Tomaž Vrtovec

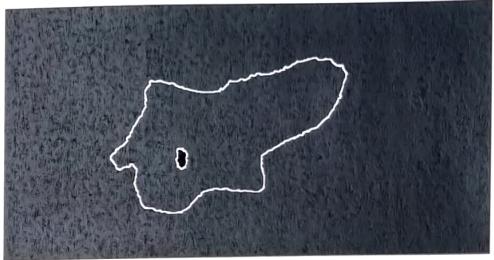
Naloga

Računanje relativnih razdalj med različnimi objekti v sliki je uporabno orodje, ki ga lahko izkoristimo v različne namene. V nadaljevanju boste iz originalne slike, ki prikazuje zemljevid Blejskega jezera in okolice, najprej izločili masko jezera, nato pa izračunali sliko razdalj med točkami, ki ležijo znotraj jezera, ter obalo jezera.

Vhodna slika



Slika meje med kopnim in jezerom



Dana je dvodimenzionalna (2D) barvna (RGB) slika `bled-lake-decimated-uint8.jpeg` velikosti $X \times Y = 693 \times 340$ slikovnih elementov, ki je zapisana v formatu jpeg z 8 biti na slikovni element. Slikovni elementi so izotropni, za merske enote pa povsod upoštevajte *slikovni element*.

1. Napišite funkcijo, ki na izhodu vrne masko območij, kjer na sliki prevladuje modra barva:

```
def get_blue_region(iImage, iThreshold):
    """
    Funkcija, ki vrne masko modrih območij na sliki
    """

    return oImage
```

kjer vhodni argument `iImage` predstavlja vhodno RGB sliko, `iThreshold` pa vrednost praga, ki se uporabi za izločanje maske modrih območij. Izhodni argument `oImage` je matrika velikosti $Y \times X$ (X in Y sta dimenzijsi vhodne slike), v kateri imajo slikovni elementi, kjer na vhodni sliki prevladuje modra barva, vrednost 255, v nasprotnem primeru pa vrednost 0.

"Modra komponenta" (RGB) slikovnega elementa ima visoko vrednost v primeru, ko le-ta prikazuje modro barvo ter tudi v primeru, ko prikazuje belo oz. (zelo) svetlo barvo. Zato je potrebno upragovanje narediti v dveh korakih in sicer tako da:

- določite masko *modrih in svetlih območij*, ki ima vrednost `True` v tistih slikovnih elementih, kjer velja $b > iThreshold$ pri čemer je b modra komponenta RGB slike, ter vrednost `False` povsod, kjer ta pogoj ne drži,
- določite masko *svetlih območij*, ki ima vrednost `True` le v tistih slikovnih elementih, kjer so vse tri barvne komponente večje od `iThreshold`, drugje pa ima vrednost `False`,

- dobljeni maski iz prve in druge točke združite na način, da dobite le masko modrih območij (brez belih oz. svetlih območij). Masko skalirajte, tako da bo vsebovala le vrednosti 0 in 255 (glej navodila v prejšnjem odstavku).

Preizkusite funkcijo na vhodni sliki. Nastavite `iThreshold = 235`. Dobljeno sliko prikažite.

- Izhodna maska iz prejšnje naloge še ni primerna za računanje razdalj od obale jezera, saj poleg maske jezera vsebuje še veliko število majhnih pik oz. svetlih območij. Da se jih znebite, opravite morfološko erozijo ter nato še morfološko dilacijo. Obe operaciji izvedite z enakim filtrom - njegovo velikost in obliko sami določite tako, da na izhodu dobite samo masko jezera (brez dotokov v jezero). Pri tem pazite, da otoček sredi jezera ostane viden. Masko shranite v spremenljivko `lake_mask` in jo prikažite.

Iz dobljene maske nato s poljubno metodo izločite robove tako, da bodo imeli slikovni elementi, ki ustrezajo robovom na sliki, vrednost 255, vsi ostali pa vrednost 0. Slika robov naj bo enaka oz. podobna sliki meje med kopnim in jezerom, ki je prikazana na začetku navodil. Sliko robov shranite v spremenljivko `lake_edge_mask` ter jo tudi prikažite.

- Napišite funkcijo, ki vrne matriko vseh koordinat, ki sestavljajo rob, ki ga prikazuje vhodna slika robov:

```
def find_edge_coordinates(iImage):
    """
    Funkcija za iskanje koordinat robnih slikovnih elementov

    return oEdges
```

kjer vhodni argument `iImage` predstavlja vhodno sliko robov, na kateri vsak slikovni element, ki ima vrednost večjo od nič, predstavlja robno točko. Izhodni argument `oEdges` naj bo matrika x in y koordinat robnih točk sledeče oblike:

$$oEdges = \begin{bmatrix} x_1 & y_1 \\ x_2 & y_2 \\ \vdots \\ x_N & y_N \end{bmatrix},$$

kjer je N število vseh robnih točk na vhodni sliki.

Preizkusite funkcijo na sliki robov `lake_edge_mask` in izpišite velikost matrike.

- Napišite funkcijo, ki vrne sliko razdalj:

```
def compute_distances(iImage, iMask = None):
    """
    Funkcija za racunanje razdalj od robov

    if iMask is None:
        # TODO: nastavite zeleno vrednost

    # TODO: funkcija za izracun slike razdalj
```

```
return oImage
```

kjer vhodni argument `iImage` predstavlja vhodno sliko robov, na kateri vsak slikovni element, ki ima vrednost večjo od nič, predstavlja robno točko, `iMask` pa masko točk, ki je enakih dimenzij kot `iImage`. Funkcija naj za vsak slikovni element (x, y) , kjer je `iMask[y, x]` > 0 izračuna razdaljo do roba, ki je definiran z vhodnim argumentom `iImage`¹. Privzeta vrednost `iMask` naj bo `None`.

Če uporabnik vhodnega argumenta `iMask` ne določi oz. ga postavi na vrednost `None`, naj se izračunajo vse razdalje na sliki (torej razdalje do vseh slikovnih elementov na sliki).

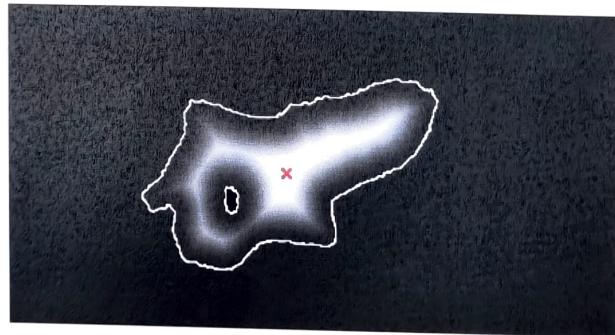
Namig: na začetku funkcije `compute_distances` iz `iImage` izločite vse koordinate robov na sliki. Pri tem si lahko pomagate s funkcijo `find_edge_coordinates`.

Preizkusite funkcijo na sliki robov `lake_edge_mask`. Vhodni argument `iMask` nastavite na `lake_mask`.

V istem prikaznem oknu prikažite:

- sliko razdalj, ki jo predhodno normalizirate na interval $[0, 255]$,
- sliko robov ter
- točko, ki je najbolj oddaljena od obale. Točko prikažite z rdečim križcem.

Izpišite tudi x in y koordinato te točke ter njeno oddaljenost od obale. Spodnja slika prikazuje pričakovani izris rezultatov.



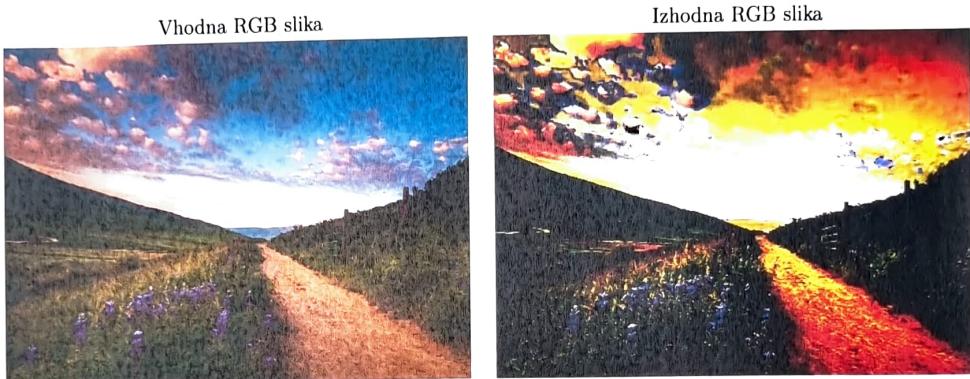
¹Opomba: uporabite razdaljo, ki je najkrajša izmed razdalj do vseh robnih točk

Zagovor laboratorijskih vaj

Pripravila: Gašper Podobnik & Tomaž Vrtovec

Naloga

RGB barvni model sliko predstavi s tremi kanali: kanalom za rdečo, zeleno in modro barvno komponento. Kot alternativen temu načinu zapisa so se uveljavili še drugi barvi modeli, npr. CMYK, HSV, YUV. Danes boste spoznali barvni model HSL, kjer prvi kanal predstavlja odtenek barve (ang. *hue*), drugi intenzivnost (ang. *saturation*), tretji pa svetlost (ang. *lightness*). Vaša naloga bo preslikava intenzitet iz modela RGB v model HSL, kjer boste opravili filtriranje. Dobljeno sliko boste nato preslikali nazaj v model RGB.



Dana je dvodimensionalna (2D) barvna (RGB) slika `planina-uint8.jpeg` velikosti $X \times Y \times RGB_{dim} = 612 \times 408 \times 3$ slikovnih elementov, ki je zapisana v formatu jpeg z 8 biti na slikovni element. Slikovni elementi so izotropni.

1. Naložite dano sliko in napišite funkcijo za standardizacijo intenzitet slike:

```
def standardize_image(iImage):
    """
    Funkcija, ki standardizira vsak kanal vhodne slike
    """

    return oImage
```

kjer vhodni argument `iImage` predstavlja vhodno RGB sliko. Izhodni argument `oImage` je standardizirana slika velikosti $X \times Y \times 3$ (X in Y sta dimenzijsi vhodne slike).

Standardizacija je sivinska preslikava, ki je določna s spodnjo enačbo:

$$I_{std} = \frac{I - \hat{v}}{\sigma},$$

kjer I predstavlja vhodno sliko, \hat{v} povprečno intenzitet v vhodni sliki, σ standardno deviacijo intenzitet v vhodni sliki, I_{std} pa standardizirano sliko (intenziteta 0 v standardizirani sliki torej ustrezava povprečni vrednosti, intenziteta 1 pa standardni deviaciji v vhodni sliki).

Funkcija naj standardizira vsak (barvni) kanal slike posebej, torej ločeno za rdeč, zelen in moder barvni kanal. Izhodna slika je RGB slika, sestavljena iz vseh treh standardiziranih kanalov.

Namig: pred standardizacijo ustrezno nastavite podatkovni tip slike. Za izračun povprečna vrednosti in standardne deviacije lahko uporabite vgrajene funkcije.

Vhodno sliko standardizirajte in rezultat shranite v spremenljivko `img_std`. Dobljeno sliko prikažite.

- Napišite funkcijo za pretvorbo slike iz barvnega modela RGB v HSL:

```
def rgb2hsl(iRGB):
    """
    Funkcija za pretvorbo RGB v HSL
    """
    return oHSL
```

kjer vhodni argument `iRGB` predstavlja vhodno RGB sliko. Izhodni argument `oHSL` pa je slika zapisana v HSL modelu. Pomagajte si z naslednjimi izračuni, pri čemer so r_i , g_i in b_i vrednosti rdeče, zelene in modre barvne komponente v poljubnem slikovnem elementu (x_i, y_i) :

$$\begin{aligned} v_i &= \max(r_i, g_i, b_i) \\ c_i &= \max(r_i, g_i, b_i) - \min(r_i, g_i, b_i) \\ l_i &= v_i - \frac{c_i}{2} \\ h_i &= \begin{cases} 60^\circ \cdot \left(\frac{g_i - b_i}{c_i}\right), & v_i = r_i \\ 60^\circ \cdot \left(2 + \frac{b_i - r_i}{c_i}\right), & v_i = g_i \\ 60^\circ \cdot \left(4 + \frac{r_i - g_i}{c_i}\right), & v_i = b_i \\ 0, & c_i = 0 \end{cases} \\ s_i &= \begin{cases} 0, & l_i = 0 \text{ ali } l_i = 1 \\ \frac{c_i}{1 - |2v_i - c_i - 1|}, & \text{drugače} \end{cases} \end{aligned}$$

Dobljene vrednosti h_i , s_i , v_i , ki sestavljajo HSL zapis (*hue*, *saturation* in *value*), zložite v matriko enake velikosti kot je vhodna RGB slika, tako da bo imel vsak slikovni element (x_i, y_i) vrednosti (h_i, s_i, v_i) .

Preizkusite funkcijo na normalizirani vhodni sliki `img_std` in dobljeno sliko shranite v spremenljivko `img_hsl` ter jo prikažite.

- Sledi korak, kjer filtrirate sliko. Iz HSL slike `img_hsl` izločite rezino, ki ustreza kanalu H , in jo shranite v spremenljivko `h_slice`. Vrednosti v `h_slice`, ki so večje ali enake od 100 in manjše do 250, delite s številom pet. Z dobljeno rezino sestavite novo HSL sliko (rezini S in V ostajata nespremenjeni), ki jo shranite v spremenljivko `img_hsl_transformed` in jo prikažite.

- Napišite funkcijo za pretvorbo slike iz barvnega modela HSL v RGB:

```
def hsl2rgb(iHSL):
    """
    Funkcija za pretvorbo HSL v RGB
    """
    return oRGB
```

kjer vhodni argument `iHSL` predstavlja vhodno HSL sliko, `oRGB` pa izhodno RGB sliko. Pomagajte si z naslednjimi izračuni, pri čemer so h_i , s_i in v_i vrednosti treh HSL komponent v poljubnem slikovnem elementu (x_i, y_i) , *mod* pa pomeni modulo (ostanek pri celoštevilskem deljenju):

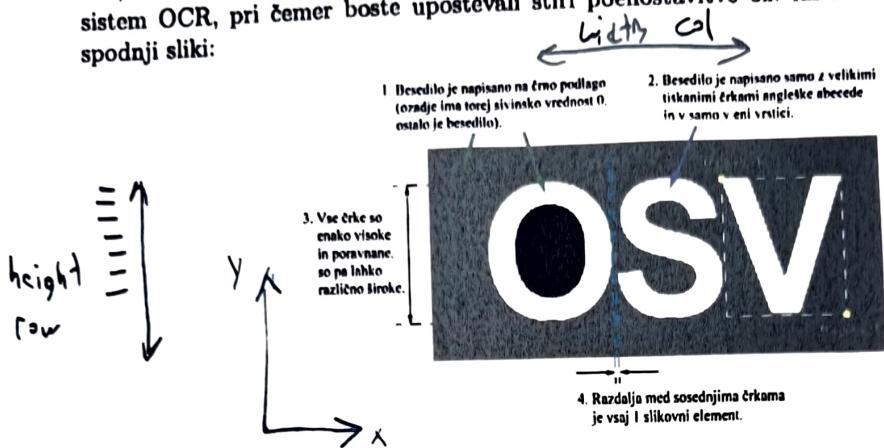
$$\begin{aligned}
 c_i &= (1 - |2l_i - 1|) \cdot s_i \\
 \hat{h}_i &= \frac{h_i}{60^\circ} \\
 x_i &= c_i \cdot (1 - |\hat{h}_i \bmod 2 - 1|) \\
 (\hat{r}_i, \hat{g}_i, \hat{b}_i) &= \begin{cases} (c_i, x_i, 0), & 0 \leq \hat{h}_i < 1 \\ (x_i, c_i, 0), & 1 \leq \hat{h}_i < 2 \\ (0, c_i, x_i), & 2 \leq \hat{h}_i < 3 \\ (0, x_i, c_i), & 3 \leq \hat{h}_i < 4 \\ (x_i, 0, c_i), & 4 \leq \hat{h}_i < 5 \\ (c_i, 0, x_i), & 5 \leq \hat{h}_i < 6 \end{cases} \\
 m_i &= l_i - \frac{c_i}{2} \\
 (r_i, g_i, b_i) &= (\hat{r}_i + m_i, \hat{g}_i + m_i, \hat{b}_i + m_i)
 \end{aligned}$$

Dobljene vrednosti r_i , g_i , b_i zložite v matriko enake velikosti kot je vhodna HSL slika, tako da bo imel vsak slikovni element (x_i, y_i) vrednosti (r_i, g_i, b_i) .

Preizkusite funkcijo na sliki `img_hsl` in `img_hsl_transformed` ter prikažite dobljeni sliki.

Naloga

Optično razpoznavanje znakov (ang. optical character recognition, OCR) je operacija obdelave slik, ki v dani sliki besedila razpozna znake oz. črke. Pri tej nalogi boste implementirali preprost sistem OCR, pri čemer boste upoštevali štiri poenostavitve oz. lastnosti, ki so prikazane na spodnji sliki:



Dana je dvodimensionalna (2D) sivinska slika `text-156x060-08bit.raw` velikosti $X \times Y = 156 \times 60$ slikovnih elementov, ki vsebuje neznano besedilo neznane dolžine. Dana je tudi tridimensionalna (3D) slika `chars-100x100x26-08bit.raw` velikosti $X \times Y \times Z = 100 \times 100 \times 26$ slikovnih elementov, v kateri vsak prečni prerez vsebuje sliko ene črke angleške abecede, in sicer v vrstnem redu od A (prvi prerez) do Z (zadnji prerez). Obe sliki sta zapisani v obliki surovih podatkov (RAW) z 8 biti na slikovni element, pri čemer so slikovni elementi izotropni.

1. Napišite funkcijo za določanje lokacije črk v dani sliki besedila:

```
def getCharLocation(iImage):
```

```
    return oLoc
```

kjer vhodni argument `iImage` predstavlja sliko besedila, medtem ko izhodni argument `oLoc` predstavlja lokacije posameznih črk v obliki matrike $L: \text{col} \times \text{row}$

kjer se nároček: vrstica stolpec vrstica stolpec, kjer se čka konča

$$L = \begin{bmatrix} r_{1,1} & c_{1,1} & r_{1,2} & c_{1,2} \\ r_{2,1} & c_{2,1} & r_{2,2} & c_{2,2} \\ \vdots & \vdots & \vdots & \vdots \\ r_{i,1} & c_{i,1} & r_{i,2} & c_{i,2} \\ \vdots & \vdots & \vdots & \vdots \\ r_{N,1} & c_{N,1} & r_{N,2} & c_{N,2} \end{bmatrix},$$

kjer i označuje i -to zaporedno črko, N pa predstavlja število vseh črk v dani sliki besedila. Vsaka vrstica zgornje matrike pripada torej eni črki, vrednosti v stolcih pa so koordinate (v slikovnih elementih) oglišč pravokotnika, ki zaobjema črko (glej primer za tretjo črko, označen z rumeno barvo na sliki zgoraj). Preizkusite funkcijo na dani sliki besedila.

2. Napišite funkcijo za določanje slike izbrane črke v dani sliki besedila:

```
def getCharImage(iImage, iLoc, iNum):  
  
    return oImage
```

kjer vhodni argument `iImage` predstavlja sliko besedila, `iLoc = L` lokacije črk v dani sliki besedila in `iNum = i` izbrano i-to zaporedno črko (vrednosti $0, 1, 2, \dots, N - 1$), medtem ko izhodni argument `oImage` predstavlja sliko izbrane i-te črke. Preizkusite funkcijo na izbrani črki za dano sliko besedila.

3. Napišite funkcijo za razpoznavanje posamezne črke na podlagi njene slike:

```
def recognizeChar(iImage, iCharImage):  
  
    return oChar
```

kjer vhodni argument `iImage` predstavlja sliko izbrane črke besedila, `iCharImage` pa 3D slika nabora črk, pri čemer vsak prečni prerez predstavlja posamezno črko angleške abecede (tj. prvi prerez črko A, zadnji prerez črko Z). Izhodni argument `oChar` predstavlja črko, ki pripada sliki izbrane črke.

Črko razpoznate na osnovi primerjanja slike dane črke z vsemi slikami iz nabora črk ter iskanja najmanjše razlike med slikama, ki jo določite kot srednjo kvadratno napako (MSE) sivinskih vrednosti:

$$MSE_j = \frac{1}{XY} \sum_{x=1}^X \sum_{y=1}^Y (f(x, y) - g_j(x, y))^2,$$

kjer je $f(x, y)$ slika dane črke, $g_j(x, y)$ je slika j-te črke iz nabora črk, MSE_j pa je pripadajoča MSE. Za primerjavo morata biti sliki $f(x, y)$ in $g_j(x, y)$ enakih dimenzij, tj. $X \times Y$, kar dosežete z ustrezno interpolacijo ene od slik. Za pretvorbo med številčnimi kodami in čkrami uporabite Pythonovo funkcijo `chr`, ki vrne črko za dano ASCII kodo (`chr(65) → A'', chr(66) → B'', ..., chr(90) → Z''`).

Namig: Slike črk v prečnih prerezih 3D slike niso "izrezane", ampak obdane z ozadjem. Črke lahko "izrežete" z uporabo funkcij `getCharLocation()` (točka 1) in `getCharImage()` (točka 2).

Preizkusite funkcijo na sliki izbrane črke.

4. Napišite funkcijo za razpoznavanje celotnega besedila iz dane slike besedila:

```
def recognizeText(iImage, iCharImage):  
  
    return oText
```

kjer vhodni argument `iImage` predstavlja sliko besedila, `iCharImage` 3D slika nabora črk, medtem ko izhodni argument `oText` predstavlja besedilo, ki ga vsebuje dana slika besedila. Preizkusite funkcijo na dano sliko besedila.