Vaja 1: Od kamere do preproste detekcije predmetov

Robotika in računalniško zaznavanje

2017/2018

Najprej si ustvarite delovno mapo, v katero boste pisali rešitve vseh nalog za predmet. V tej mapi si boste za vsako vajo posebej ustvarili podmapo.

Za vajo ustvarite mapo `vaja1`. Sem si prenesite tudi razpakirano vsebino datoteke `vaja1.zip` s spletne strani predmeta. Rešitve nalog boste pisali v *Matlab/Octave* skripte in jih shranili v mapo `vaja1`. Da uspešno opravite vajo, jo morate predstavili asistentu na vajah. Pri nekaterih nalogah so vprašanja, ki zahtevajo skiciranje, ročno računanje in razmislek. Odgovore na ta vprašanja si zabeležite v pisni obliki in jih prinesite na zagovor. Deli nalog, ki so označeni s simbolom ★ niso obvezni. Brez njih lahko za celotno vajo dobite največ 75 točk (zgornja meja je 100 točk kar pomeni oceno 10). Vsaka dodatna naloga ima zraven napisano tudi število točk. V nekaterih vajah je dodatnih nalog več in vam ni potrebno opraviti vseh.

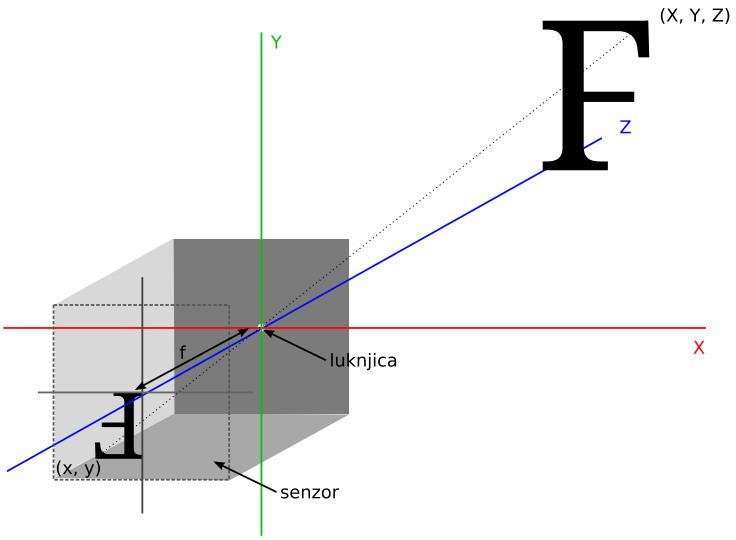
Če okolja *Matlab/Octave* še niste uporabljali, si pred nadaljevanjem poglejte še skripto z osnovami jezika *Matlab/Octave*, ki vam razloži osnovno sintakso jezika. Skritpta je dostopna na spletni strani predmeta.

Naloga 1: Kamera

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

$$x = -f\frac{X}{Z} \text{ in } y = -f\frac{Y}{Z}, \tag{1}$$

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



Rešite naslednje naloge:

(a) ***Vprašanje:*** Kockasta škatla z velikostjo stranice $10cm$ 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) ***Vprašanje:*** 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 *Matlab/Octave* skripte, ki jo shranite v datoteko `vaja1_naloga1b.m` , 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 `plot`.

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

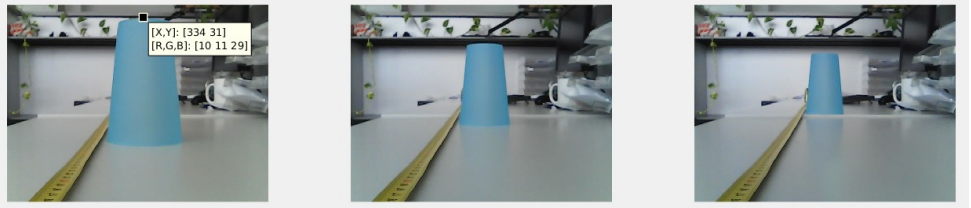
Sliko, ki je projecirana na zadnjo stran kamere, zajamemo v digitalno obliko z (največkrat) matričnim senzorjem. Več o tem ste povedali že na predavanih, tu pa ponovimo samo, da sta najpomembnejša parametra senzorja njegova velikost in gostota. Ker se gostota senzorja največkrat podaja v točkah na inčo (*dots per inch, DPI*), si je na tem mestu pametno zapomniti, da je $1inch = 2.54cm$.

(d) ***Vprašanje:*** S kamero z goriščno razdaljo $f = 60mm$ 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 program za zajem slik (izberite ga sami glede na vaš operacijski sistem). 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 naložite v *Matlab/Octave* in zabeležite višino objekta v številu slikovnih elementov (v Matlabu si lahko pomagate z orodjem *Data select*, ki je na voljo kot gumb v orodni vrstici okna za prikaz slike, 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.



Naloga 2: Procesiranje slik

Namen te naloge je spoznavanje z osnovno funkcionalnostjo *Matlab/Octave* in z zapisom slik v matrikah. Ukazi, ki jih boste preizkusili pri tej vaji so: `imread`, `imshow`, `imagesc`, `colormap` in `imrotate`. Sledite naslednjim točkam in kodo pišite v datoteko `vaja1_naloga2.m` :

(a) Preberite sliko iz datoteke `umbrellas.jpg` in jo prikažite v oknu s pomočjo ukazov:

```
A = imread('umbrellas.jpg'); % Image A is in 8-bit format (uint8)
figure(1); clf; imagesc(A); % open figure window, clear, draw
figure(2); clf; imshow(A);
```

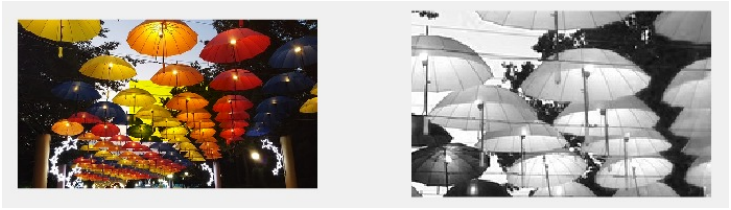
(b) Na prvi pogled ni bistvene razlike med načinom prikaza slik pri uporabi funkcij `imagesc` ali `imshow` – razliko si bomo pogledali v kratkem. Slika, ki ste jo naložili ima tri kanale, RGB, in je spravljena v 3D matriko velikosti $visina \times sirina \times kanali = h \times w \times 3$. Izpišite si na zaslon velikost slike tako, da napišete `[h, w, d] = size(A)`. Spremenite barvno sliko v sivinsko tako, da povprečite kanale in sliko zopet prikažite z dvemi funkcijami za prikaz. 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 `double`.

```
Ad = double(A); % division operation not defined for uint8
[h,w,d] = size(A) % output the size of the image (notice the absence of semicolon at the end of ↩
    the line)
A_gray = Ad(:,:,1) + Ad(:,:,2) + Ad(:,:,3)) / 3.0;
figure; imshow(uint8(A_gray)); % change image to 8-bit before displaying it
```

(c) V zgornjem primeru se je slika izrisala s privzeto barvno tabelo. Barvna tabela pomeni način, kako vam Matlab prikaže odtenek sivine. Na primer, temne odtenke lahko prikaže modro, svetle pa rdeče. Poskusite spremeniti barvno mapo v sliki z ukazom `colormap` (za primer uporabe glej Matlabovo pomoč). Poskusite s tabelami `jet`, `bone`, `gray`.

(d) Izrežite pravokotno regijo s slike in jo izrišite kot novo sliko. Regijo v prvi sliki bomo označili tako, da tretji kanal (modri) postavimo na nič.

```
A1 = A;
A1(130:260,240:450,3) = 0 ;
figure;
subplot(1,2,1);
imshow(A1);
A2 = A(130:260,240:450, 1);
subplot(1,2,2);
imshow(A2);
```



(e) Izrišite sivinsko sliko, kjer del sivinske verzije slike slike negirate.

```
A3 = A_gray;
A3(130:260,240:450) = 255 - A3(130:260,240:450) ;
figure;
imshow(A3);
```

(f) 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. Za izhodišče uporabite spodnjo kodo.

```
A5 = A_gray;
A5 = A5 > 150 ; % all elements with value > 150 are changed to 1, others are changed to 0
figure(1);
imagesc(A5);
```

(g) ★ (5 točk) Zadnjo nalogo predelajte tako, da bo upragovana slika prišla iz kamere. Predelajte primer iz prejšnje naloge, da se slike iz kamere v zanki najprej preberejo, pretvorijo v sivinske slike in upragujejo, rezultat pa se prikaže na zaslonu.

Naloga 3: Histogrami

V nadaljevanju si bomo ogledali, kako gradimo in prikazujemo histograme. Histogrami so zelo uporaben način opisa slike, s katerim lahko veliko povemo o porazdelitvi slikovnih elementov v sliki, s tem pa do neke mere tudi kaj o vsebini slike. Zaradi njihove preprostosti in uporabnosti so zelo razširjeni, zato posvetite pozornost procesu grajenja, ki ga bomo opisali v tej nalogi.

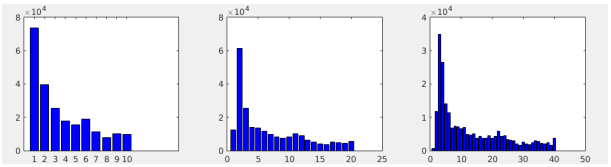
(a) ***Vprašanje:*** Podana je 3-bitna sivinska slika:

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

- Določite histogram za podano sliko (v obliki tabele in z grafično predstavitvijo).
- Na podlagi izračunanega histograma določite kumulativni histogram slike.
- Kakšen bi bil histogram, če bi bila slika 4-bitna?

(b) Računanje histograma je implementirano v *Matlab/Octave* v okviru funkcije `hist`. V skripti preberite sliko iz datoteke `umbrellas.jpg` in jo spremenite v sivinsko. Ker funkcija `hist` ne deluje na slikah ampak na zaporedju vrednosti, moramo sliko najprej preoblikovati iz matrike velikosti $(N \times M)$ v 1-D vektor velikosti $NM \times 1$, ta vektor pa se nato uporabi za izračun histograma. Prav tako je pomembno, da so vhodne vrednosti tipa *double*. Izrišite histograme za različno število celic.

```
I = double(rgb2gray(imread('umbrellas.jpg')));
P = I(:); % A handy way to turn 2D matrix into a vector of numbers
figure(1); clf;
bins = 10 ;
H = hist(P, bins);
subplot(1,3,1); bar(H, 'b');
bins = 20 ;
H = hist(P, bins);
subplot(1,3,2); bar(H, 'b');
bins = 40 ;
H = hist(P, bins);
subplot(1,3,3); bar(H, 'b');
```



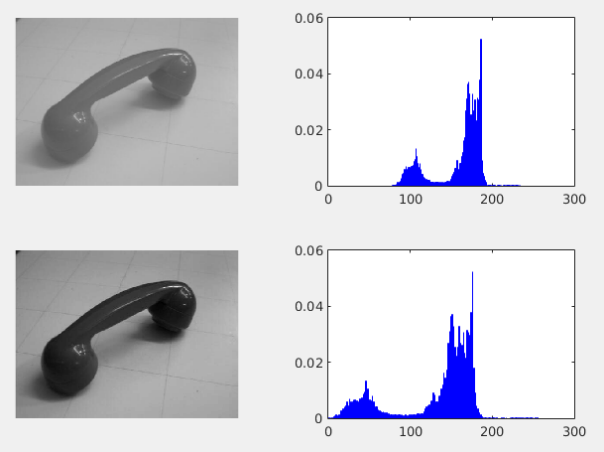
Opozorilo: Če funkciji `hist` ne podate izhodnega parametra, bo le-ta privzeto odprla novo okno in izrsala rezultat. Vendar to ne pomeni, da lahko za izris histogramov v nadaljevanju uporabljate funkcijo `hist`, raje uporabite funkcijo `bar`, ki skrbi samo za izris. Tak dvo-fazni način namreč nudi več fleksibilnosti.

(c) V okviru naslednje točke boste implementirali operacijo *razteg histograma*. Z uporabo pripravljenega skeleta implementirajte funkcijo `histstretch`, ki vhodno sliko popravi tako, da bo njene vrednosti raztegnjne preko celotnega spektra sivinskih nivojev. Za okvirni potek algoritma se zgledujte po prosojnicah z vaj in predavanjih. Ker funkcija opravlja enako operacijo na vseh slikovnih elementih slike, zaradi hitrosti implementirajte celotno funkcijo brez zank z uporabo matričnih operacij.

```
S = histstretch( I )
 % 1. determine the minimum and maximum value of I
 % 2. the minimum and maximum of the output image S are known
 % 3. use the stretch formula to compute new value for each pixel
```

Namigi: Za vhodno sliko lahko določite najvišjo (`v_max`) in najnižjo (`v_min`) sivinsko vrednost z `max( I(:) )` in `min( I(:) )`. Pomembno: pri nalogi je prepovedana uporaba funkcije `imadjust` ali podobnih 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 sivinska vrednost v sliki ni 0, in najvišja ni 255. Nato na sliki opravite raztek histograma ter prikažite rezultat v obliki slike in njenega histograma (za prikaz uporabite funkcijo `imhow`, histogrami naj vsebujejo 256 celic).

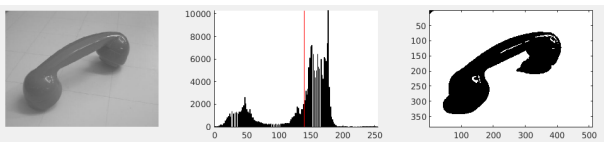


Vprašanje: Zakaj operacijo imenujemo *razteg histograma*, če histograma v funkciji sploh ne uporabimo neposredno? Katera funkcija, ki jo uporabljamo za prikaz slik že sama dela nekaj podobnega?

★ (5 točk) Če implementirate funkcijo `histstretch` brez eksplicitnih zank, dobite dodatne točke.

(d) Upragovanje slike je zelo uporabno pri implementaciji preproste detekcije objektov, vendar je določitev praga pogosto problematična. Preizkusite funkcijo `im2bw`, ki prag določi samodejno z uporabo Otsu-jeve metode, ki temelji na analizi histograma slike.

Vprašanje: Kakšen histogram je idealen za Otsu-jevo metodo, ali, povedan drugače, na kakšnih slikah ta metoda deluje dobro?



Naloga 4: Barvni prostori

Kot ste slišali že na predavanjih, lahko barvo 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 na preprost način dosežemo zanimive rezultate. Najprej pa dve računski nalogi, v okviru katerih boste osvežili poznavanje algoritmov pretvorbe, ki ste jih spoznali na predavanjih (so napisani v prosojnicah):

Vprašanje: 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.
Vprašanje: 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.

Nadaljne naloge rešujete v okolju *Matlab/Octave*:

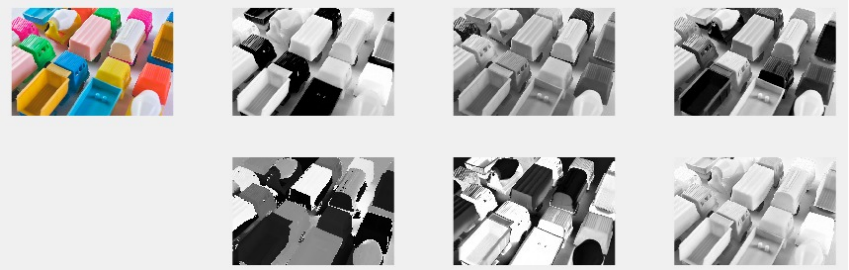
(a) Preberite sliko iz datoteke `trucks.jpg` ter jo prikažite na zaslonu kot RGB sliko ter vsako komponento posebej kot sivinsko sliko.

Spremenite sliko iz RGB barvnega prostora v HSV barvni prostor s funkcijo `rgb2hsv` ter prikažite na zaslonu vsako komponento posebej kot sivinsko sliko. Pri postopku pazite 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).

Pri pisanju skripte si lahko pomagate s sledečim skeletom:

```
I1 = imread('trucks.jpg');

figure(1);
hold on;
colormap gray;
subplot(2, 4, 1);
imshow(I1);

% Draw RGB channels as separate subplots

I2 = rgb2hsv(I1);

% Draw HSV channels as separate subplots

hold off;
```

Vprašanje: Kako si razlagate vrednosti posameznih komponent HSV prostora glede na vrednosti komponent RGB prostora?



(b) Različni barvni prostori so koristni tudi, ko pride do upragovanja. V RGB barvnem prostoru je recimo težko na preprost način določiti področja, ki pripadajo določenemu barvnemu odtenku. Napišite skripto, ki sliko iz datoteke `trucks.jpg` upraguje najprej po modrem kanalu za vrednost praga 200. Prikažite originalno sliko ter binarno sliko eno ob drugi.

Vprašanje: Je rezultat operacije kvalitetna segmentacija modrih regij? Razložite razlog.

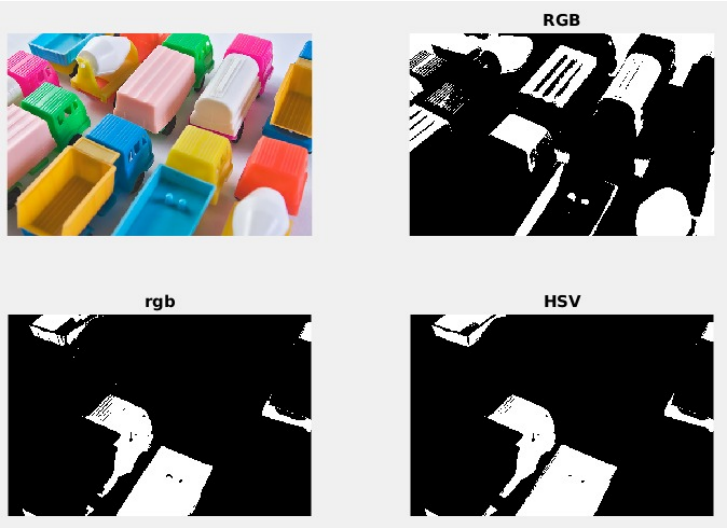
(c) Za osnovne komponene (rdeča, zelena, modra) se upragovanje poenostavi, če sliko preslikamo v normalizirani RGB prostor, kjer je vrednost vsake barve deljena z vsoto vrednosti vseh treh komponent (takemu prostoru rečemo tudi *normalizirani RGB* ali *rgb*). Dodajte kodo, ki modro komponento slike po celicah deli z vsoto soležnih treh barvnih komponent (uporabite funkcijo `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.

(d) Na tak način lahko določimo regije rdeče, zelene in modre barve. Če pa bi želeli izluščiti poljuben odtenek, se je najbolj intuitivno poslužiti preslikave v HSV barvni prostor. 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. V okolju *Matlab/Octave* se to 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:

```
I = ones(20, 255, 3);
I(:, :, 1) = ones(20, 1) * linspace(0, 1, 255);
image([0, 1], [0, 1], hsv2rgb(I));
```

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

(e) ★ (5 točk) Kako bi lahko upragovanje na podlagi HSV barvnega prostora naredili bolj robustno? Namig: Zakaj je pri katerih območjih težko natančno določiti odtenek? Lahko z uporabo dodatne informacije taka področja izločimo? Preverite svojo rešitev z implementacijo.



(f) Uporabite narejeno upragovanje v HSV prostoru za določitev regij rumene barve, nato dobljeno binarno sliko očistite še z uporabo morfologije. Očiščeno binarno sliko nato razdelite na regije ter za vsako regijo izpišite momente $m_{0,0}$, $m_{1,0}$ in $m_{0,1}$.

(g) ★ (5 točk) Implementirajte funkcijo `immask`, ki ji lahko 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 soležnega elementa v maski 0. Funkcijo implementirajte brez eksplicitnih zank. Namig: delo s posameznimi kanali je lažje, kot delo s celotno barvno sliko. Za združevanje kanalov lahko uporabite funkcijo `cat`. Funkcijo preizkusite enem izmed primerov iz prejšnjih nalog.

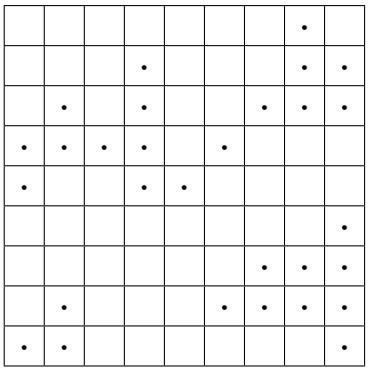
Naloga 5: Regije in morfološke operacije

V okviru te vaje boste preučili tri različne tematike. V tretji nalogi pa boste na primeru pogledali še, kako lahko s pretvorbami med barvnimi prostori dosežemo zanimive rezultate ter kako lahko na podlagi upragovanja v barvnih prostorih izluščimo področja v sliki.

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.

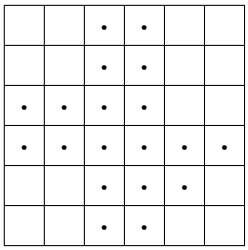
Za razčlenitev regij na sliki se zaradi njegove časovne predvidljivosti najpogosteje uporablja algoritem zaporednega označevanja regij, oziroma povezanih komponent, ki ste ga obravnavali na predavanjih. Za začetek ponovimo delovanje algoritma z ročnim reševanjem preprostega primera:

(a) ***Vprašanje:*** Podana je črno-bela slika za katero določite rezultat algoritma barvanja regij po prvem in drugem spehodu po sliki (za lažje reševanje sta podani dve prazni sliki, v kateri lahko zapišete rešitve) ter povezave med oznakami za okolico N_4. Na sliki so z • označena področja objektov v sliki, prazne celice pa so ozadje.



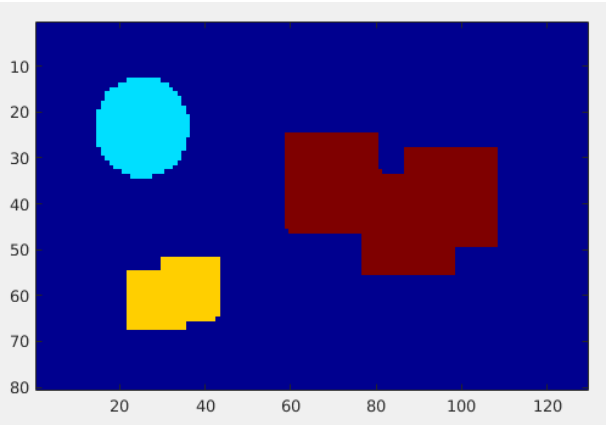
Lastnosti posameznih regij lahko opišemo na več načinov, na predavanjih ste obravnavali verižne kode ter momente. Najprej preverite vaše znanje o opisih z naslednjo nalogo.

(b) ***Vprašanje:*** Za podano regijo določite absolutno, relativno verižno kodo za okolico N_8 ter momente $(0, 2)$, $(1, 2)$ in $(0, 1)$.



Nadaljne naloge spet rešujete v okolju *Matlab/Octave*:

(c) Preberite sliko iz datoteke `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 ustreznega praga (recimo 127). Uporabite funkcijo `bwlabel`, ki je vgrajena v okolje *Matlab/Octave* in implementira algoritem označevanja regij. Prikažite rezultat kot črnobelo sliko (uporabite funkcijo `imagesc`). Iz rezultata funkcije `bwlabel` izluščite število regij ter v enem oknu skupaj prikažite maske, ki označujejo posamezne regije.



(d) Implementirajte funkcijo `moment`, ki za dano binarno sliko, ki opisuje eno regijo, vrne p, q moment regije. Definicijo momenta regije najdete na prosojnicah s predavanj. Uporabite naslednji nastavek:

```
function [moment] = moment(R, p, q)
 ...
```

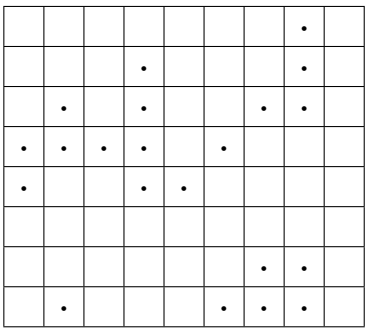
S pomočjo vaše implementacije funkcije `moment` implementirajte še funkcijo `central_moment`, ki vrne p, q centralni moment regije.

(e) V praksi obstaja še veliko načinov opisa lastnosti regij, veliko jih je implementiranih v okviru vgrajene funkcije `regionprops`. Funkcija dejansko interno požene algoritem povezanih komponent, zato ji ni potrebno podajati vsake regije posebej, ampak lahko podamo kar celotno binarno sliko, rezultat pa bo polje izračunanih lastnosti za vse regije. Uporabite funkcijo, da za posamezne regije določite centroid ter okvir okoli regije (`BoundingBox`), nato pa za vsako regijo te lastnosti prikažite nad sliko maske (za izris uporabite funkcijo `plot`).

Na žalost binarne slike, pridobljene iz realnih podatkov, pogosto niso tako lepe kot slike, s katerimi smo delali do sedaj. V slikah je veliko šuma, ki velikokrat rezultat šuma v izvorni sliki, v binarni sliki pa nastopa kot drobne regije ali luknjice v regijah, ki motijo nadaljnje procesiranje. V takih primerih so zelo priročni 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.

(a) ***Vprašanje:*** Najprej na kratko ponovite delovanje obeh operacij z naslednji izpitno nalogo: Podana je črno-bela slika za katero določite rezultat algoritma filtriranja s filtroma razširi ter skrči za jedro K.

$$K = \begin{array}{|c|c|c|}\hline \cdot & \cdot & \cdot \\ \hline \cdot & \cdot & \cdot \\ \hline & \cdot & \\ \hline \end{array}$$

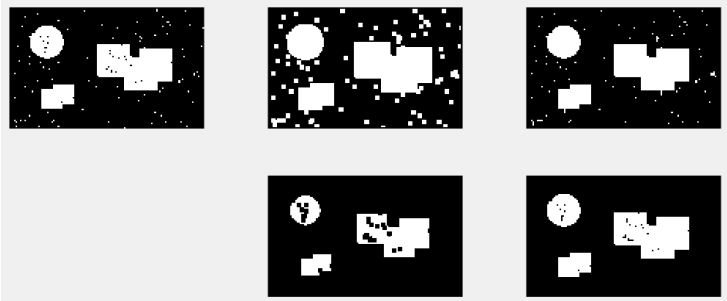


(b) Naložite sliko iz datoteke `regions_noise.png`, jo spremenite v črnobelo in nato v binarno z uporabo ustreznega praga. Uporabite funkcijo `bwlabel` za določitev regij in regije preštejte. Kaj opazite? Kako velike so posamezne regije?

(c) Na sliki preizkusite vgrajeni funkciji `imdilate` in `imerode`.

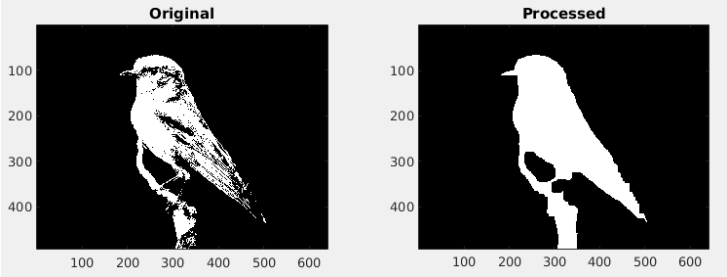
```
I = imread('regions_noise.png') > 127;

se = ones(3, 3); % initialize a simple strucural element

A1 = imdilate(I, se);
```

```
A2 = imerode(A1, se);

B1 = imerode(I, se);
B2 = imdilate(B1, se);

figure(1);
subplot(2, 3, 1);
imshow(I);
subplot(2, 3, 2);
imshow(A1);
subplot(2, 3, 3);
imshow(A2);

subplot(2, 3, 5);
imshow(B1);
subplot(2, 3, 6);
imshow(B2);
```



(d) S pomočjo funkcij `imerode` in `imdilate` implementirajte še operaciji odpiranja (*opening*) in zapiranja (*closing*) ter rezultat primerjajte z vgrajenima funkcijama `imopen` in `imclose`. Prikažite rezultat obeh operacij na upragovani sliki iz datoteke `regions_noise.png`. Glede na rezultate premislite, kako bi odpravi celoten šum, ki je prisoten v izvorni binarni sliki? Rešitev implementirajte in preizkusite.

(e) ★ (5 točk) Pridobljeno znanje o uporabite morfoloških operacij preizkusite še na bolj realnem primeru. Preberite sliko iz datoteke `bird.jpg`, jo spremenite v sivinsko ter določite prag, da dobite čim boljšo masko objekta (lahko uporabite tudi funkcijo `im2bw`). Ker popolne maske ne morete dobiti samo z globalnim pragom, jo izboljšajte z uporabo morfoloških operacij.



(f) ★ (15 točk) Napišite skripto ki iz datoteke naloži sliko `candy.jpg` in uporabnika interaktivno prosi za vnos točke na njej z uporabo funkcije `ginput`. Če točka leži na enem izmed bonbonov naj skripta prešteje bonbone iste barve. Segmentirajte ospredje od ozadja z uporabo praga ter procesirajte masko z morfološkimi operacijami, da odstranite šum, določite posamezne povezane komponente in za njih

določite povprečno barvo kot povprečje barv vseh slikovnih elementov komponente. Ko uporabnik izbere bonbon naj skripta najde vse regije s podobno barvo (za primerjavo lahko uporabite Evklidsko razdaljo in prag). Rezultat predstavite tako, da izpišete število bonbonov s funkcijo `text` ter na sliki označite njihova središča s funkcijo `plot`. Končno število točk bo določeno na podlagi robustnosti vaše rešitve.