

UNIVERZITET U TUZLI
FAKULTET ELEKTROTEHNIKE



SEMINARSKI RAD

OBARADA I ANALIZA MEDICINSKIH SLIKA

Tema:
Prepoznavanje kvadrata na slici

Ismar Osmanović / Lejla Zahirović

Sadržaj

1	Ideja projekta	1
2	Realizacija projekta	2
2.1	Određivanje pragova poje	2
2.2	Maska	3
2.3	Morfološke operacije	4
2.4	bwboundary	7
2.5	Funkcija brojanje	8
2.6	Poziv funkcije i ispis	9
3	Primjeri	11
4	Zaključak:	12
5	Izvori:	13

1 Ideja projekta

Segmentacija slike predstavlja jedan od osnovnih problema u obradi slike. Zadatak našeg projekta je kreirati algoritam koji će segmentirati, a zatim i prebrojati crvene i zelene kvadrate na slici. Započinjemo sa slikom studenata u amfiteatru koji drže crvene i zelene kartone.



Slika 1: Početna slika

2 Realizacija projekta

```
1 img = imread('img/studenti.jpg');
```

Imread funkcija učitava sliku iz file-a koji je specificiran pomoću njegovog imena, utvrđujući format na osnovu sadržaja istog. img je varijabla u kojoj se čuva učitana slika. Nakon što je slika učitana, sadržaj slike (u formi matrice piksela) biće sačuvan u varijabli. Ova funkcija je dio biblioteke koju koristi MatLab.

2.1 Određivanje pragova poje

```
1 hsvImg = rgb2HSV(img);
```

Koristeći funkciju `rgb2HSV` slika se konvertuje iz RGB color prostora u HSV (Hue, Saturation, Value) color prostor. Dok RGB prostor koristi tri komponente (crvenu, zelenu i plavu) za reprezentaciju boja, HSV prostor koristi tri komponente:

- Hue (nijansa),
- Saturation (intenzitet boje),
- Value (osvjetljenost boje).

Dakle, img je ulazna slika u RGB formatu, koja je pritom učitana pomoću `imread` funkcije, a `hsvImg` je promjenljiva u kojoj će biti sačuvana slika u HSV formatu.

Zašto je primjenjena ova konverzija, iz jednog color prostora u drugi, na ovom primjeru?

U opštem slučaju se u ovaj format koristi zbog svojih prednosti u obradi same slike. Naime, ovaj format pruža lakšu manipulaciju bojama. Naprimjer, H komponenta je nezavisna od svjetlosti i zasićenosti, pa je koristeći navedeni format lakše izolavati određene boje na slici, što će kasnije biti primjenjeno, nego isključivo koristeći RGB format. U implementaciji koda je upravo bio zadatak pronaći piksele crvene i zelene boje na datim slikama. Međutim, u prvobitnom prostoru, taj zadatak bi bio znatno otežan jer bi bilo potrebno razmatrati sve tri komponente, a u ovom slučaju smo posmatrali samo H komponentu za filtriranje crvenih tonova, kao i zelenih. Preostale dvije mogu biti korištene za kontrolu jačine boje i osvjetljenosti.

```
1 lowerRed1 = [0, 0.3, 0.3]; % Donja granica za crvenu
2 upperRed1 = [7/360, 1, 1]; % Gornja granica za crvenu
3 lowerRed2 = [353/360, 0.3, 0.3]; % Donja granica za crvenu
4 upperRed2 = [1, 1, 1]; % Gornja granica za crvenu
```

Ove linije koda definišu pragove za filtriranje crvene boje u HSV color prostoru. Konkretno, koristi se Hue (H), Saturation (S) i Value (V) komponente za postavljanje opsega boje crvene (a kasnije i zelene), tako da se mogu prepoznati pikseli koji odgovaraju toj boji u slici.

Zašto se koristi ovaj pristup? Ovaj pristup koristi dva opsega vrednosti za Hue (H), jer crvena boja u HSV prostoru nije predstavljena sa samo jednim opsegom. U HSV prostoru, crvena boja se prostire kroz početni i krajnji dio opsega Hue vrednosti:

- Prvi opseg (`lowerRed1` i `upperRed1`) pokriva crvenu boju koja je bliža početku opsega (oko 0, što je početak crvene).
- Drugi opseg (`lowerRed2` i `upperRed2`) pokriva crvenu boju koja je bliža kraju opsega (oko 1, što je kraj crvene boje).

Ovaj pristup omogućava preciznije prepoznavanje crvene boje u cijeloj paleti. Sličan pristup je korišten i za zelenu boju ali samo sa jednim opsegom pošto se zelena boja ne nalzi na krajevima HSV prostora.

2.2 Maska

Generalno, maska se koristi za segmentaciju ili izdvavanje određenih dijelova date slike po moću nekih kriterijuma. Koristeći ovaj koncept željeni dijelovi slike se izdvoje, dok ostali bivaju zanemareni.

Maska je zapravo binarna slika, što dalje znači da se sastoji isključivo od 1 i 0 te ona označava koji dijelovi slike trebaju biti podvrgnuti željenoj analzi, obradi ili promjeni u opštem slučaju. Naime, vrijednost 1 označava da je odgovarajući piksel od interesa, dok se onaj sa 0 zanemaruje. Ovaj koncept je implementiran na datom kodu za pronalaženje samo crvenih i/ili zelenih piksela na slikama.

Dalje se na njima mogu primjeniti filteri ili operacije. Dakle, maska predstavlja jedan od ključnih alata u obradi i analizi slike te se koristi upravo za: izolaciju određenih dijelova slike, filtriranje, segmentaciju objekta, obrada na temelju boje ili intenziteta, praćenje objekta, kao i pobošljavanje preformansi (u raddu sa velikim slikama ili skupovima podataka).

```

1 % Kreiranje maske
2 mask1 = (hsvImg(:,:,1) >= lowerRed1(1) & hsvImg(:,:,1) <= upperRed1(1) &
3 ...
4 ...
5 ...
6 ...
7 ...
8 ...
9 ...
10 ...
11 ...
12 ...
13 ...
14 redMask = mask1 | mask2;
15 greenMask = mask3;
```

`hsvImg(:,:,1) >= lowerRed1(1) & hsvImg(:,:,1) <= upperRed1(1)` provjerava da li vrijednost Hue svakog piksela leži između minimalne (`lowerRed1(1)`) i maksimalne (`upperRed1(1)`) granice za Hue. Ovaj uslov osigurava da boja piksela bude u opsegu za crvenu boju.

`hsvImg(:,:,2) >= lowerRed1(2) & hsvImg(:,:,2) <= upperRed1(2)` provjerava Saturation, tj. zasićenost boje, da li je unutar određenog opsega.

`hsvImg(:,:,3) >= lowerRed1(3) & hsvImg(:,:,3) <= upperRed1(3)` provjerava Value, tj. svjetlost boje, da li je unutar željenog opsega.

Ovaj uslov znači da će maska `mask1` biti true (1) za piksele koji imaju crvenu boju u određenom opsegu, prema vrijednostima iz `lowerRed1` i `upperRed1`.

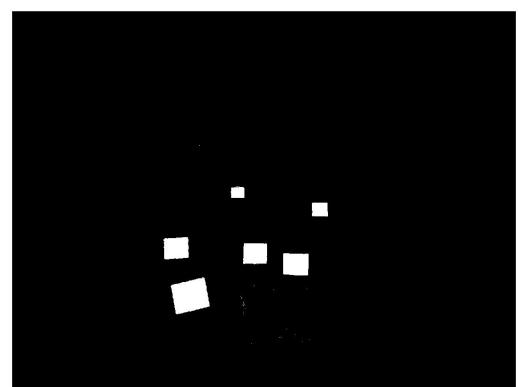
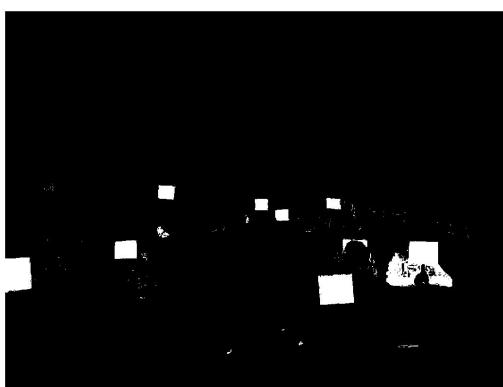
Mask2 ima isti oblik, ali koristi druge granice (`lowerRed2` i `upperRed2`), što omogućava da se prepoznaaju i druge nijanse crvene boje (na primjer, svjetlija crvena).

```
redMask = mask1 | mask2;
```

Ova linija koristi logički operator `|` (ili) kako bi spojila dvije maske. Znači da će `redMask` biti true (1) na mjestima gdje je barem jedan od uslova zadovoljen, tj. ili piksela u maski 1 ili piksela u maski 2.

Analogan pristup se koristi i za `greenMask`.

Nakon ovih linija maske imaju sljedeći izgled:



Slika 2: a) redMask b) greenMask

Vidimo da su sa početne slike izdvojena svi regioni gdje se pojavljuju naši kvadrati.

2.3 Morfološke operacije

```
1 redMask =imerode(redMask, strel('disk', 3));
2 redMask =imerode(redMask, strel('disk', 3));
3 redMask =imopen(redMask, strel('disk', 10));
4 redMask =imdilate(redMask, strel('disk', 10));
```

U obradi slike, matematička morfologija je korištena kao sredstvo za identificiranje i ekstrakciju značajnih deskriptora slike na osnovu osobina oblika unutar slike.

Ključno područje primjene morfologije predstavlja segmentacija slike, zajedno sa automatskim brojenjem i inspekcijom. Morfološke operacije su također osnovni alati te se kao takvi primarno koriste u binarnim slikama, ali se mogu primjeniti i na sive slike. Ove operacije direktno koriste matematičke metode za analizu i promjenu strukture objekta.

Ključni operatori su *dilatacija* i *erozija*.

Moguće je sve složenije procedure svesti na dvije pomenute. Osvrćući se na sam pojam binarne slike, on je obrađen u prethodnom tekstu. Međutim, važno je pomenuti da se binarna slika sastoji od bilo koje grupe povezanih piksela. Tu dolazi do razlike između dvije vrste povezanosti i to

4-povezanost i 8-povezanost. 4-povezanost se temelji na povezanosti piksela koji su u neporednoj horizontalnoj ili vertikalnoj blizini. To znači da su povezani samo oni pikseli koji se nalaze uzduž stranica kvadrata, a ne dijagonalno. Pikseli (i, j) su povezani ako postoji neki piksel u njegovom neposrednom okruženju, koji je u istoj boji (npr. bijeli piksel u binarnoj slici), a taj piksel se nalazi u jednoj od sledećih pozicija:

- $(i-1, j)$ — gornji susjed
- $(i+1, j)$ — donji susjed
- $(i, j-1)$ — lijevi susjed
- $(i, j+1)$ — desni susjed

Ovaj tip povezanosti primjenjuje se kada je potrebno razmatrati samo vertikalne i horizontalne susjede za povezivanje piksela.

Za 4-povezanost, N4, koristi se city-block ili Manhattan distanca. 8-povezanost se temelji na širem okruženju, gdje su povezani pikseli koji se nalaze u neposrednom susjedstvu, uključujući vertikalne, horizontalne, i dijagonalne susjede. Dakle, osim vertikalnih i horizontalnih piksela, uključeni su i dijagonalni susjedi.

Pikseli (i, j) su povezani ako postoji neki piksel u njegovom neposrednom okruženju, koji je u istoj boji, a taj piksel se nalazi u jednoj od sledećih pozicija:

- $(i-1, j)$ — gornji susjed
- $(i+1, j)$ — donji susjed
- $(i, j-1)$ — lijevi susjed
- $(i, j+1)$ — desni susjed
- $(i-1, j-1)$ — gornji lijevi dijagonalni susjed
- $(i-1, j+1)$ — gornji desni dijagonalni susjed
- $(i+1, j-1)$ — donji lijevi dijagonalni susjed
- $(i+1, j+1)$ — donji desni dijagonalni susjed

Primenjuje se kada je potrebno razmatrati sve susjede, uključujući dijagonalne, za povezivanje piksela. Za 8-povezanost, N8, koristi se Chebysheva ili šahovska distanca. Dakle, korišćenje 8-povezanosti može biti korisno kada želimo da prepoznamo objekte koji su "nešto razdvojeni", ali su dijagonalno povezani, dok 4-povezanost može biti korisna kada želimo da analiziramo striktne vertikalne ili horizontalne veze između objekata.

Sljedeći bitan pojam, koji služi objašnjavanju gornjih operacija, jeste strukturni element.

Strukturni elementi (SE) su osnovni alati u morfološkoj obradi slike. Oni su matrice sa fiksnim oblikom koje se koriste za primjenu različitih morfoloških operacija na binarnim ili grayscale slikama. Strukturni element definiše oblik i veličinu regiona u okolini svakog piksela slike, koji se koristi za manipulaciju tim pikselima tokom morfoloških operacija. SE mogu imati različite oblike kao što su kružni, kvadratni, pravougaoni, dok njihova veličina zavisi direktno od njihove aplikacije u samom kodu.

Navedeni pojmovi su ključni pri pristupu eroziji i dilataciji.

Najprije, **erozija** se koristi za smanjenje objekata na slici. Ova operacija omogućava uklanjanje manjih struktura i detalja u slici, kao i smanjenje veličine objekata u binarnoj slici. Erozija je vrlo korisna u segmentaciji, uklanjanju šuma i prepoznavanju oblika. Erozija funkcioniše tako što, za svaki piksel u slici, gleda sve njegove susjedne piksele. Ako svi susjedni pikseli koji čine oblik strukturnog elementa imaju vrijednost 1 (odnosno pripadaju objektu), onda centralni piksel ostaje 1. Ako neki od susjednih piksela nije 1, centralni piksel se postavlja na 0, što znači da se objekat smanjuje.

S druge strane, **dilatacija** se koristi za proširivanje objekata na slici. Ova operacija omogućava povećanje veličine objekata u binarnoj slici i povezivanje objekata koji su blizu jedan drugog. Dilatacija je suprotnost eroziji, jer proširuje piksele objekta umjesto da ih smanjuje. Dilatacija koristi strukturni element (SE) koji se "širi" oko svakog piksela objekta u slici. Za svaki piksel u slici, dilatacija postavlja vrijednost 1 u centralni piksel (ako je bar jedan piksel iz strukturnog elementa 1). Ako je bar jedan od susjednih piksela 1, centralni piksel postaje 1. Dakle, objekat se "proširuje" tako da susjedni pikseli postaju dio objekta.

Dilatacija i erozija se često koriste zajedno. Na primjer, primjena dilatacije može učiniti objekat većim, dok erozija može ukloniti nepotrebne dijelove objekta. Ove operacije zajedno čine osnovu mnogih "otvoreno" i "zatvoreno" morfoloških operacija.

Otvaranje je definisano kao morfološka operacija erozije praćena sa dilatacijom sa istim strukturnim elementom. Erozija uklanja neke piksele objekta sa ivica regionalnog objekta, dok dilatacija dodaje piksele objekta. Tako veličina objekta ostaje ista, ali su njegove konture zaglađenije.

Zatvaranje je morfološka operacija koja se sastoji od dilatacije praćene erozijom sa istim strukturnim elementom. Zatvaranje zaglađuje konture objekata, ali i popunjava uske prekide ili praznine i eliminiše male šupljine, mijenjajući te male regije pozadine u regije objekta.

Gore navedeni kod koristi nekoliko morfoloških operacija za obradu slike u kojoj je detektovana crvena boja, pri čemu je isti pristup urađen i za zelenu boju. Morfološke operacije koriste strukturne elemente za manipulaciju oblikom objekata u slici. U ovom slučaju, strukturni element je disk, što znači da se koristi diskasti oblik za primjenu operacija.

```
redMask =imerode(redMask, strel('disk', 3));
```

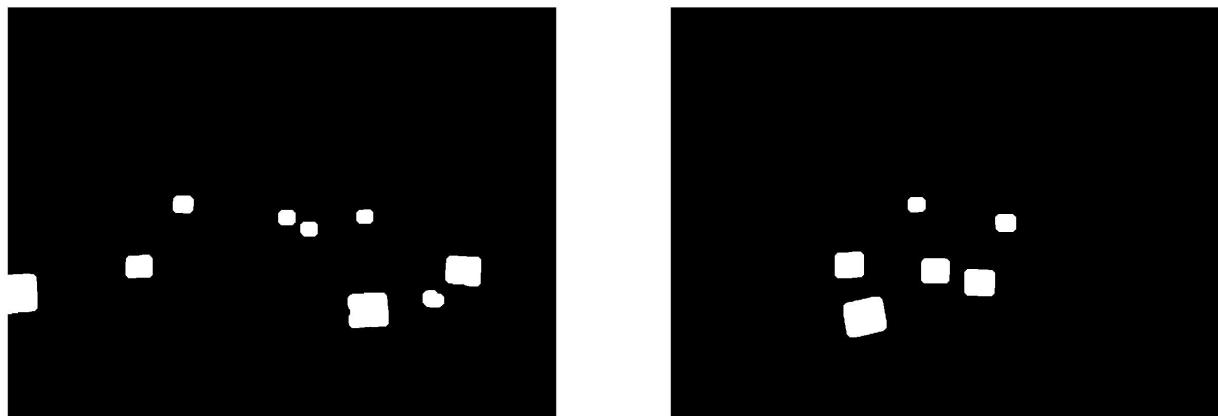
Ovdje se primjenjuje erozija na redMask koristeći **diskasti strukturni element** sa poluprečnikom od 3 piksela. Ovaj diskasti strukturni element je matrica koja sadrži vrijednosti 1 u obliku diska (poluprečnik 3). Erozija će ukloniti piksele sa ivica objekta. Zbog ove operacije, objekti u redMask postaju manji, a sitni objekti mogu biti uklonjeni. Ova operacija se radi dva puta uzastopno.

```
redMask = imopen(redMask, strel('disk', 10));
```

U ovoj liniji je primjenjena operacija otvaranja. Prvo se smanjuju objekti pomoću erozije, a zatim se proširuju pomoću dilatacije. Ovaj postupak služi za uklanjanje sitnih objekata i šupljina u većim objektima, dok istovremeno zadržava osnovnu strukturu objekata. Zbog erozije, manji objekti nestaju, dok dilatacija omogućava da preostali objekti ponovo postanu "veći" i sačuvaju osnovni oblik.

```
redMask = imdilate(redMask, strel('disk', 10));
```

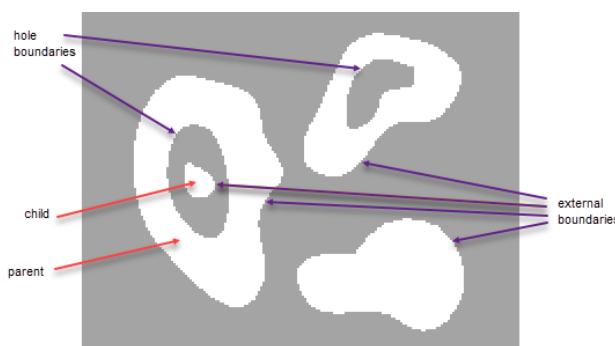
Ovom linijom se proširuju preostali objekti u slici koristeći diskasti strukturni element. Ovo može povezati objekte koji su se prethodno razdvojili i može proširiti ivice objekata, čineći ih "većim". Ovaj niz operacija omogućava precizno filtriranje i obradu binarnih slika, sa ciljem da se izdvoje važni objekti (u ovom slučaju crvena boja) i da se poboljša njihova vidljivost i konture na slici.



Slika 3: redMask i greenMask nakon morfoloških operacija

2.4 bwboundary

bwboundary predstavlja alat iz IPT-a korištenju za pronađak granica povezanih regiona na binarnoj slici. Ova komanda ima mogućnost traženja vanjskih i unutrašnjih granica kao i granica *child* i *parent* objekata na slici.



Slika 4: Tipovi granica objekata na slici

```
1 Br = bwboundaries(redMask, 'noholes');  
2 Bg = bwboundaries(greenMask, 'noholes');
```

Ova funkcija prima binarnu sliku na kojoj je potrebno pronaći granice (to je u našem slučaju masku) i kao rezultat vraća matricu određenih granica. Pošto je u našem slučaju potrebno pronaći samo vanjske granice koristimo parametar "noholes" koji traži samo zajedničke vanjske granice *parent* i *child* objekata. Funkcija **bwboundaries** implementira Moore-susjedni algoritam. Funkciju je moguće pozvati da koristi 4 ili 8 povezanost susjednih pixela s time da je default argument 4-povezanost.

Funkcija je pozvana posebno za crvenu i posebno za zelenu masku.

Sada kada imamo matricu naših granica moguće je sekvencijalno proći kroz nju i odrediti koje granice odgovaraju obliku traženog kvadrata.

2.5 Funkcija brojanje

Pošto imamo dvije matrice granica iz kojih je potrebno izvući kvadrate odlučili smo se sljedeći algoritam smjestiti u našu funkciju pod nazivom `Funkcija_brojanje` koja prima matricu granica, boju kvadrata i posebnu funkciju. Za ovaj pristup odlučili smo se iz sljedećeg razloga. Pošto radimo nad slikom studenata u amfiteatru, veličina kvadrata na slici koji drži student u prvom redu će biti manja od kvadrata koji drži student u posljednjem redu amfiteatra. Iz ovog razloga odredili smo funkciju prema kojoj se na osnovu površine kvadrata može odrediti njegova relativna visina na slici. To znači da kada znamo površinu granice možemo izračunati njenu predviđenu visinu na slici i uporediti je sa stvarnom visinom na kojoj se objekat nalazi. Ako postoji značajno odstupanje to znači da naša granica ne odgovara kvadratu.

Funkcija koja odgovara multimedijalnoj sali našeg fakulteta:

```
1 f = @(x) 0.03*x+600 + 20;
```

Ovu funkciju smo kreirali na osnovu slika napravljenih u multimedijalnoj sali. Za svaki amfiteatar bilo bi potrebno odrediti funkciju koja ga opisuje (Zavisi od nagiba, visine klupa i sl.). Definicija funkcije:

```
1 function output = Funkcija_brojanje( B, color, funkcija)
2 squareCount = 0;
3     for k = 1:length(B)
4         boundary = B{k};
5
6         % Racunanje povrsine i obima
7         area = polyarea(boundary(:,2), boundary(:,1));
8         perimeter = sum(sqrt(sum(diff(boundary).^2, 2)));
```

Kao što smo rekli, u funkciji prolazimo sekvencijalno kroz svaku granicu iz B matrice. Za svaku granicu računamo površinu i njen obim.

```
1
2     % Racunanje velicine okvira
3     minX = min(boundary(:,2));
4     maxX = max(boundary(:,2));
5     minY = min(boundary(:,1));
6     maxY = max(boundary(:,1));
7
8     width = maxX - minX;
9     height = maxY - minY;
10    %Racunanje odnosa viine i sirine
```

Zatim računamo visinu. Iz ovih podataka ćemo odrediti odnos visine i širine. Kvadrati koje koristimo imaju odnos 4/5.

U sljedećim ugnježdenim if izrazima određujemo naše kvadrate.

```
1     if perimeter > 0
2
3         if aspectRatio >= 0.5 && aspectRatio <= 2
4
5             if mean(boundary(:,1)) < funkcija(area)
6                 squareCount = squareCount + 1;
7                 centroidX = mean(boundary(:,2));
8                 centroidY = mean(boundary(:,1));
9
10                label = sprintf('%d', squareCount);
11                text(centroidX, centroidY, label, 'Color', color, '
12                    FontSize', 20, 'FontWeight', 'bold', 'HorizontalAlignment', 'center');
13                continue;
14            end
15        end
```

U prvom se pitamo da li je obim veći od 0. Zatim se pitamo da li je odnos visine i širine u granicama. Na ovaj način smo isključili sve visoke i uske kao i široke i niske oblike. Zadnji if provjerava da li visina odgovara izračunatoj visini.

Ukoliko je sve od navedenog ispravno, inkrementiramo brojač i postavljamo `label` sa brojem na mjesto našeg kvadrata.

Na samom kraju funkcije vraćamo broj kvadrata.

```
1 output = squareCount;
```

2.6 Poziv funkcije i ispis

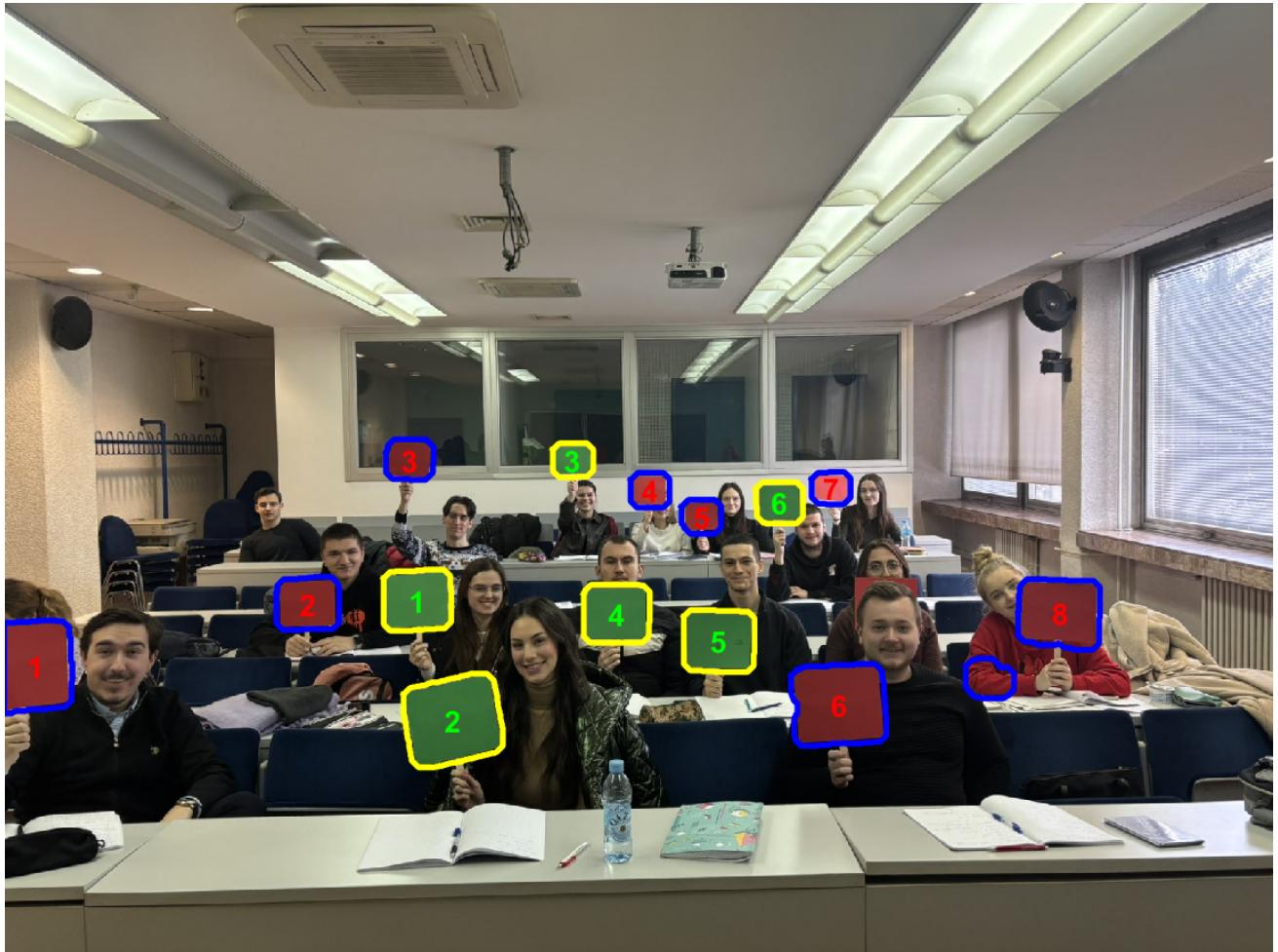
Iz glavnog programa vršimo poziv funkcije:

```
1 RsquareCount = Funkcija_brojanje(Br, 'red', f);
2 GsquareCount = Funkcija_brojanje(Bg, 'green', f);
```

Nakon ovih linija broj crvenih i zelenih kvadrata sačuvan je odgovarajuće varijable. Preostaje samo ispisati broj na ekran i opisati linije u boji oko svih granica.

```
1 fprintf('\nNumber of red squares: %d\n', RsquareCount);
2 fprintf('\nNumber of red squares: %d\n', GsquareCount);
3 xlabel(sprintf('Broj crvenih: %d Broj zelenih: %d', RsquareCount,
4 GsquareCount), 'FontSize', 40);
5 hold on;
6 for i = 1:length(Br)
7     boundary = Br{i};
8     plot(boundary(:,2), boundary(:,1), 'Color', 'blue', 'LineWidth', 2);
9 end
10
11 for i = 1:length(Bg)
12     boundary = Bg{i};
13     plot(boundary(:,2), boundary(:,1), 'Color', 'yellow', 'LineWidth', 2);
14 end
```

Rezultat:



Slika 5: 8 crvenih 6 zelenih

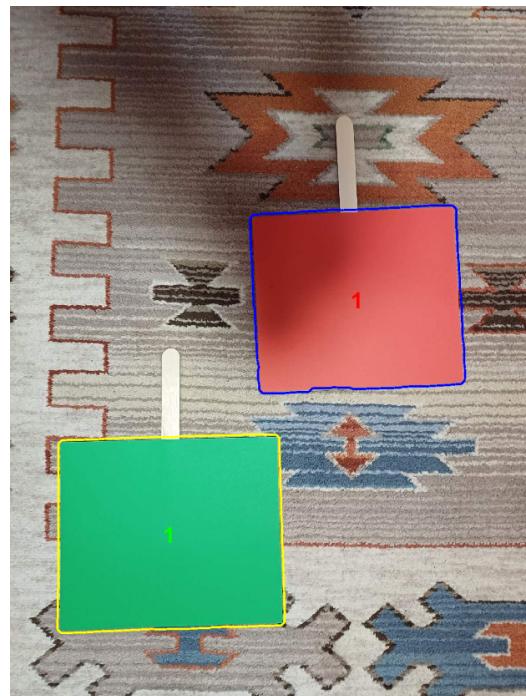
Kvadrati su tačno prebrojani, a crveni džemper ispod crvenog kvadrata broj 8. je izbačen po kriterijumu visine.

3 Primjeri



Slika 6: 4 crvenih 11 zelenih

U ovom primjeru uspješno smo eliminisali boundary crvene jakne ispod zelenog kvadrata br. 5, ali ne i crveni džemper na crvenom kvadratu br. 4.



Slika 7: 1 crvenih 1 zelenih

4 Zaključak:

U okviru ovog seminarskog rada uspješno smo implementirali program za brojanje crvenih i zelenih kvadrata na slici korištenjem MATLAB-a. Program je razvijen primjenom tehnika obrade slike, uključujući segmentaciju boja i analizu objekata, što je omogućilo precizno prepoznavanje i brojanje kvadrata različitih boja.

Rezultati pokazuju da program postiže visoku tačnost u identifikaciji i klasifikaciji kvadrata.

Iako su rezultati zadovoljavajući, u budućem radu postoji prostor za dodatna poboljšanja.

5 Izvori:

1. <https://www.mathworks.com/help/matlab>
2. <https://www.mathworks.com/help/matlab/ref/boundary>