

Primena fazi logike u obradi slika

Jelena Mrdak
Tijana Jevtić

17. januar 2019

Sažetak

U ovom radu je predstavljena primena fazi logike u obradi slika. Njen značaj za ovu oblast je prikazan kroz detaljan opis nekoliko algoritama. Takođe, ispitivano je da li metode za obradu slika zasnovane na fazi logici daju bolje rezultate od tradicionalnih metoda i da li su one efikasnije.

Kodovi su pisani u jeziku *C++* pomoću biblioteke *OpenCV*. Napominjemo da su svi kodovi, kako oni navedeni u radu, tako i oni koji nisu, ali koji su korišćeni za poređenje, pisani ručno od strane autora, te da svi rezultati i zaključci zavise od njihove implementacije. Takođe, prikazana vremena izvršavanja u velikoj meri zavise od mašine na kojoj se kodovi izvršavaju, te mogu varirati, ali su sva vremena dobijena testiranjem na istom računaru u sličnim uslovima.

Sadržaj

1	Uvod	3
2	FCM	4
2.1	Binarizacija slike	6
2.2	FCM i k-means	8
3	Detekcija ivica	10
3.1	FED	10
3.2	FED i CED	13
4	Poboljšavanje kvaliteta slike korišćenjem fazi logike	14
4.1	Tehnika za poboljšavanje kvaliteta crno-belih slika sa lošim kontrastom [3]	15
4.2	Tehnika za poboljšavanje kvaliteta slika u boji sa lošim kontrastom [4] . .	19
4.3	Poređenje rezultata metoda za poboljšavanje kvaliteta slika u boji, crno-belih slika i metoda podešavanja histograma	23
5	Zaključak	25

1 Uvod

U matematici smo do sad navikli da je nešto tačno ili netačno, da nešto pripada ili ne pripada skupu. Međutim, nekad je teško povući granicu i odrediti kad je nešto 0, a kad 1. Što smo bliži granici, to je nesigurnost veća. Na primer, ako želimo da klasifikujemo ljude na niske i visoke i postavimo granicu na 170cm , dobijamo da je osoba visoka 170cm visoka, dok je osoba visoka 169cm niska, što i nema mnogo smisla.

Fazi logiku je 1965. godine uveo Lotfi Zadeh¹ i ona nam u velikoj meri može pomoći za rešavanje pomenutih problema. Još pre Zadeha, Pirs² je ozbiljnije proučavao neodređenost (eng. vagueness). On nije verovao u jasnu granicu između tačnog i netačnog, već u to da je život kontinuum. Takođe, i drugi poznati naučnici i filozofi su se dalje bavili ovom idejom. Rasel³ je govorio da je sav jezik neodređen. Neodređenost je pitanje stepena (eng. Vagueness is a matter of degree). [6]

Fazi skupovi se razlikuju od klasičnih skupova kod kojih je granica jasna (element ili pripada ili ne pripada skupu). Zadeh je uopštio klasične skupove tako što je proširio skup valuacije $\{0, 1\}$ na interval realnih brojeva $[0, 1]$. Stepen pripadnosti nekog elementa fazi skupu opisuje koliko taj element odgovara pojmu koji je reprezentovan fazi skupom. Konkretno, element x pripada skupu A sa stepenom $\mu_A(x)$, gde je $\mu_A : A \rightarrow [0, 1]$ karakteristična funkcija skupa A .

U ovom radu smo koristili upravo ove ideje kako bismo odredili da li je piksel crn ili beo, da li je piksel ivica ili nije, da li je dovoljno osvetljen ili ne.

Obrada slika korišćenjem fazi logike nije jedinstvena teorija, već kolekcija različitih fazi metoda za obradu slika, koji mogu da razumeju, opišu i obrade sliku, njene delove ili karakteristike, posmatrajući je kao fazi skup. Reprezentacija i obrada slike zavise od izabranog metoda i problema koji je potrebno rešiti. Najrazličitije fazi tehnike omogućavaju odgovarajući okvir za napredak jer su zasnovane na znanju (eng. knowledge-based). [7] Mogu da obrade podatke sa greškom u slučaju da je ista nastala usled neodređenosti i dvosmislenosti (eng. ambiguity) (ali ne zbog nasumičnosti (eng. randomness)). [4]

Fazi tehnike za obradu slika, uostalom kao i sve ostale fazi tehnike, sastoje se iz tri faze: fazifikacija, odgovarajuća funkcija za promenu vrednosti funkcije pripadnosti i defazifikacija. Prva i poslednja faza su neophodne zbog toga što ne postoji odgovarajući fazi hardver. [7]

Slika X , dimenzija $M \times N$, može biti predstavljena kao fazi skup [8]

$$X = \sum_{i=1}^M \sum_{j=1}^N x_{ij}, \mu_{ij},$$

¹Lotfi A. Zadeh (1921 – 2017) matematičar, kompjuterski naučnik, inženjer elektrotehnike.

²Charles Sanders Peirce (1839 – 1914) američki filozof, logičar i matematičar.

³Bertrand Russell (1872 — 1970) britanski filozof i matematičar.

gde $\mu_{ij} \in [0, 1]$ predstavlja vrednost funkcije pripadnosti.

2 FCM

Fuzzy C-means (FCM) je jedan od najpopularnijih algoritama za fazi klasterovanje. U ovom poglavlju ćemo ga najpre detaljno opisati, a zatim ćemo ga iskoristiti za binarizaciju slike.

Cilj ovog algoritma je da skup $X = \{x_1, x_2, \dots, x_n\}$ particioniše na k delova (klastera) po nekom kriterijumu. Preciznije, kriterijum je minimizacija sledeće funkcije:

$$F(\bar{w}, \bar{c}) = \sum_{i=1}^n \sum_{j=1}^k w_{ij}^m \|x_i - c_j\|^2,$$

gde $w_{ij} \in [0, 1]$ predstavlja pripadnost tačke x_i j -tom klasteru i $\sum_{j=1}^k w_{ij} = 1$, dok je c_j centroid j -tog klastera. Realni parametar $m > 1$ predstavlja faktor fazifikacije i on se zadaje unapred. U nastavku ćemo preciznije odrediti ove koeficijente. Sada ćemo samo ukratko opisati korake algoritma.

FCM je veoma sličan algoritmu k-means i sastoji se iz sledećih koraka:

- Ako je slika u boji, konvertovati je u sivu.
- Izabrati broj klastera k .
- Svakoj tački x_i dodeliti koeficijente $w_{ij} \in [0, 1], j = 1, 2, \dots, k$.
- Ponavljati sve dok ne dođe do konvergencije:
 - Izračunati centroide za svaki klaster.
 - Ažurirati koeficijente.
- Tačku x_i dodeliti klasteru kom najviše pripada, tj. r -tom klasteru, gde je $w_{ir} = \max_j w_{ij}$.

Teorema 2.1. *Potrebni uslovi za minimizator (\bar{w}^*, \bar{c}^*) funkcije $F(\bar{w}, \bar{c})$ su:*

$$c_j = \frac{\sum_{i=1}^n w_{ij}^m \cdot x_i}{\sum_{i=1}^n w_{ij}^m} \quad (1)$$

$$w_{ij} = \frac{1}{\sum_{u=1}^k \left(\frac{\|x_i - c_u\|}{\|x_i - c_j\|} \right)^{\frac{2}{m-1}}} \quad (2)$$

Dokaz. Pronaći ćemo potencijalne tačke lokalnih uslovnih ekstremuma. Koristićemo Lagranževe množioce. Posmatraćemo pomoćnu funkciju:

$$J(\bar{w}, \bar{c}, \bar{\lambda}) = \sum_{i=1}^n \sum_{j=1}^k w_{ij}^m \|x_i - c_j\|^2 - \sum_{i=1}^n \lambda_i \left(\sum_{j=1}^k w_{ij} - 1 \right).$$

Tačke koje tražimo moraju da zadovoljavaju uslov $\nabla J = \mathbf{0}$. Dakle,

$$\frac{\partial J}{\partial c_j} = 0, 1 \leq j \leq k \quad (3)$$

$$\frac{\partial J}{\partial w_{ij}} = 0, 1 \leq i \leq n, 1 \leq j \leq k \quad (4)$$

$$\frac{\partial J}{\partial \lambda_i} = 0, 1 \leq i \leq n \quad (5)$$

Rešavanjem (3) dobijamo (1). Iz (4) imamo

$$mw_{ij}^{m-1} \|x_i - c_j\|^2 - \lambda_i = 0,$$

odnosno

$$w_{ij} = \left(\frac{\lambda_i}{m \|x_i - c_j\|^2} \right)^{\frac{1}{m-1}}. \quad (6)$$

Iz (5) dobijamo:

$$\begin{aligned} 1 &= \sum_{u=1}^k w_{iu} \\ &= \sum_{u=1}^k \left(\frac{\lambda_i}{m \|x_i - c_u\|^2} \right)^{\frac{1}{m-1}} \\ &= \sum_{u=1}^k \left(\frac{m \|x_i - c_u\|^2}{\lambda_i} \right)^{\frac{1}{1-m}} \\ &= \sum_{u=1}^k \frac{(m \|x_i - c_u\|^2)^{\frac{1}{1-m}}}{\lambda_i^{\frac{1}{1-m}}} \\ &= \frac{1}{\lambda_i^{\frac{1}{1-m}}} \sum_{u=1}^k (m \|x_i - c_u\|^2)^{\frac{1}{1-m}}, \end{aligned}$$

pa zaključujemo da je

$$\lambda_i = \left(\sum_{u=1}^k (m \|x_i - c_u\|^2)^{\frac{1}{1-m}} \right)^{1-m}.$$

Konačno, zamenjujući poslednju jednakost u (6), dobijamo (2). \square

FCM algoritam za određivanje minimizatora funkcije F je iteracija kroz potrebne uslove.

2.1 Binarizacija slike

Ispod je prikazan kod za binarizaciju slike koji koristi FCM algoritam. Napominjemo da se zbog čitljivosti koda u ovom delu nismo odlučili za efikasnu implementaciju. O tome će biti više reči u narednoj sekciji.

```
#include <iostream>
#include <opencv2/highgui/highgui.hpp>

int main(int argc, const char *argv[])
{
    if (argc != 2) {
        std::cerr << "Usage: ./binarization path_to_img" << std::endl;
        return 1;
    }

    // read image
    cv::Mat img = cv::imread(argv[1], cv::IMREAD_GRAYSCALE);
    cv::Mat img_binary = cv::Mat(img.rows, img.cols, CV_8UC1,
        cv::Scalar(255));

    // weights
    std::vector<std::vector<float>> w1(img.rows,
        std::vector<float>(img.cols, 0));
    std::vector<std::vector<float>> w2(img.rows,
        std::vector<float>(img.cols, 0));
    // fuzzification factor
    double m = 2;
    // centroids
    std::pair<float, float> c;

    // init weights
    for (int i = 0; i < img.rows; i++) {
        for (int j = 0; j < img.cols; j++) {
            w1[i][j] = img.at<unsigned char>(i,j)/255.0;
            w2[i][j] = 1 - w1[i][j];
        }
    }

    // stopping criteria
    float eps = 0;

    do {
        // calculate centroids
        std::pair<float, float> c1_fraction{0,0};
        std::pair<float, float> c2_fraction{0,0};
```

```

for (int i = 0; i < img.rows; i++) {
    for (int j = 0; j < img.cols; j++) {
        c1_fraction.first += std::pow(w1[i][j], m)*img.at<unsigned
        ↵ char>(i,j);
        c1_fraction.second += std::pow(w1[i][j], m);
        c2_fraction.first += std::pow(w2[i][j], m)*img.at<unsigned
        ↵ char>(i,j);
        c2_fraction.second += std::pow(w2[i][j], m);
    }
}

auto old_c = c;
c = {c1_fraction.first/c1_fraction.second,
    ↵ c2_fraction.first/c2_fraction.second};
eps = (old_c.first-c.first)*(old_c.first-c.first) +
    ↵ (old_c.second-c.second)*(old_c.second-c.second);

// update weights
for (int i = 0; i < img.rows; i++) {
    for (int j = 0; j < img.cols; j++) {
        float d1 = std::abs(img.at<unsigned char>(i,j)-c.first);
        float d2 = std::abs(img.at<unsigned char>(i,j)-c.second);
        w1[i][j] = 1/(std::pow(d1/d1, 2/(m-1)) + std::pow(d1/d2,
        ↵ 2/(m-1)));
        w2[i][j] = 1/(std::pow(d2/d1, 2/(m-1)) + std::pow(d2/d2,
        ↵ 2/(m-1)));
    }
}

} while(eps > 1);

// cluster pixels based on weights
for (int i = 0; i < img_binary.rows; i++) {
    for (int j = 0; j < img_binary.cols; j++) {
        img_binary.at<unsigned char>(i,j) = (w1[i][j] > w2[i][j]) ? 255 :
        ↵ 0;
    }
}

// show and save binary image
namedWindow("Display window", cv::WINDOW_AUTOSIZE);
imshow("Display window", img_binary);
cv::waitKey(0);
imwrite("fcm.png", img_binary);

```

```
    return 0;  
}
```

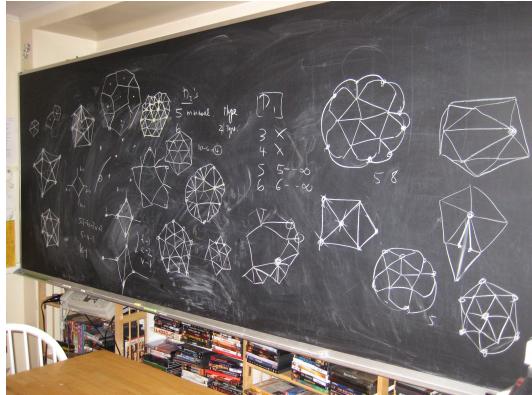
Rezultat izvršavanja algoritma je prikazan ispod.



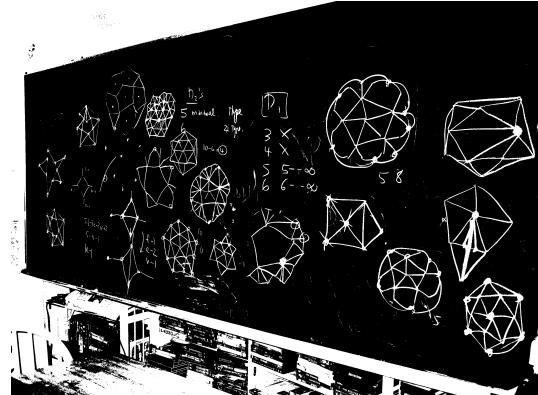
Slika 1: Original



Slika 2: Rezultat



Slika 3: Original



Slika 4: Rezultat

2.2 FCM i k-means

U ovom odeljku ćemo uporediti rezultate algoritama FCM i k-means, kao i vremena njihovih izvršavanja.

Napomena 2.1. Koristićemo efikasniju implementaciju FCM algoritma od one date u sekciji 2.1.

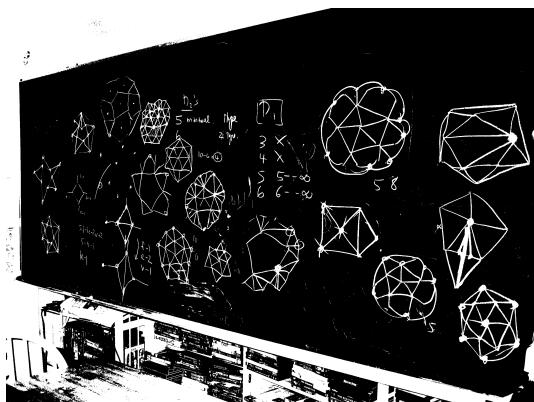
Na sledećim slikama su prikazani rezultati.



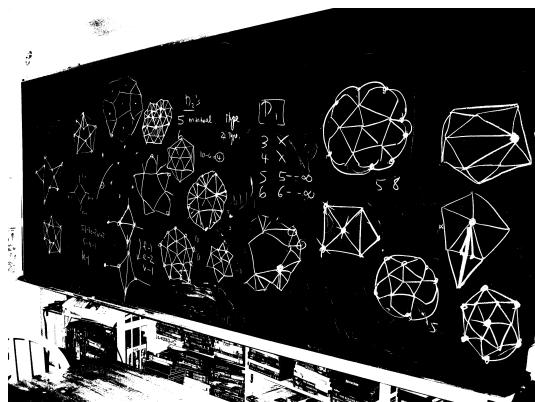
Slika 5: FCM rezultat
Broj iteracija: 7



Slika 6: k-means rezultat
Broj iteracija: 6



Slika 7: FCM rezultat
Broj iteracija: 7



Slika 8: k-means rezultat
Broj iteracija: 7

Možemo primetiti da su slike 5 i 6 identične, dok se slike 7 i 8 neznatno razlikuju. Međutim, vremena izvršavanja se primetno razlikuju. U Tabeli 1 su prikazana vremena (u sekundama) potrebna da algoritmi obrade Sliku 1 500, 1000 i 1500 puta. Slično, u Tabeli 2 su prikazana vremena potrebna da se obradi Slika 3 50, 100 i 150 puta.

broj izvršavanja	FCM	k-means
500	4	21
1000	8	42
1500	12	63

Tabela 1: Original Slika 1

broj izvršavanja	FCM	k-means
50	8	44
100	17	89
150	25	133

Tabela 2: Original Slika 3

Prikazaćemo još dva testa urađena na dve nove slike.

broj izvršavanja	FCM	k-means
100	15	119
150	22	179
200	31	238

Tabela 3:

broj izvršavanja	FCM	k-means
1000	3	36
1500	5	53
2000	6	71

Tabela 4:

Na osnovu podataka iz tabela, zaključujemo:

test	k-means/FCM
1	5
2	5
3	8
4	12

Tabela 5: Koliko puta je FCM brži od k-means

Treba napomenuti da su ovi rezultati okvirni, jer u velikoj meri zavise od implementacije samih algoritama. Naime, za centoride u k-means algoritmu je korišćen celobrojni tip (int), dok je centroide u FCM algoritmu korišćen realni tip (float). U oba slučaja su se algoritmi zaustavljali kad je promena u centroidima bila manja od jedan. Dakle, već tu može doći do razlike u broju iteracija. Međutim, i dalje očekujemo da će FCM biti brži od k-means.

Takođe, ističemo da FCM koristi više memorije nego k-means.

3 Detekcija ivica

Detekcija ivica ima veliki značaj u obradi slika. Koristi se u raznim algoritmima kao što su segmentacija slike, detekcija i izdvajanje karakteristika, pa čak nekad i u kompresiji slika.

Ivice možemo definisati kao mesta na slici gde se intenzitet naglo menja, tj. gde je razlika vrednosti susednih piksela velika. Postoje mnogi algoritmi koji se bave ovim problemom, međutim, nijedan nije savršen. Primerom ćemo najbolje ilustrovati šta mislimo kad to kažemo. Naime, Sobelov algoritam je dobar kada treba detektovati oblike, ali ne radi dobro u realnom vremenu gde je brzina ključna (direktan prenos nekog događaja). Za takve situacije je prikladniji Canny algoritam.

U nastavku ćemo videti još jedan pristup ovom problemu. Koristićemo fazi logiku i fazi skupove.

3.1 FED

Kao što smo već napomenuli, koristićemo fazi logiku i fazi skupove da bismo detektovali ivice na slici. Preciznije, napravićemo fazi skup koji sadrži uređene parove (piksel,

vrednost karakteristične funkcije). Taj skup će predstavljati ivice, dok će nam vrednosti karakteristične funkcije govoriti u kojoj meri piksel pripada tom skupu.

Postavlja se pitanje šta izabratи za karakterističnu funkciju. Podsetimo se, ivica je mesto где se intenzitet naglo menja. Shodno tome, treba uzeti u obzir razliku intenziteta piksela koji trenutno posmatramo i njegovih suseda. Kada je ta razlika velika, vrednost naše funkcije treba da teži jedinici, a kada je razlika mala, treba da teži nuli. Jedna takva funkcija je:

$$\mu_{edge}(p) = 1 - \frac{1}{1 + \frac{\sum_{n \in N(p)} \|p-n\|}{L-1}}, \quad (7)$$

gde je p piksel, $N(p)$ skup piksela iz njegove okoline, L broj sivih nijansi (za 8-bitnu sliku, to je 256) i $\|\cdot\|$ norma koju ćemo definisati kao absolutnu vrednost razlike intenziteta piksela.

Pre nego što damo kod, ukratko ćemo opisati korake algoritma:

- **Preprocesiranje** - ako je slika u boji, konvertovati je u sivu sliku.
- **Preprocesiranje** - primeniti Gausov filter na sliku kako bismo je malo zamutili.
- **Fazifikacija** - računanje karakteristične funkcije μ_{edge} za svaki piksel sa slike. Ta-kodje, čuvanje najveće vrednosti funkcije (promenljiva MAX).
- **Normiranje vrednosti** - vrednosti karakteristične funkcije podeliti sa MAX:

$$\mu_{edge}(p) = \frac{\mu_{edge}(p)}{MAX}$$

- **Defazifikacija** - na osnovu vrednosti $\mu_{edge}(p)$ i nekog unapred datog praga (threshold), pikselu p dodeliti crnu ili belu boju.

Pošto smo videli kratak opis algoritma, u nastavku dajemo FCM kod radi boljeg razumevanja istog.

```
#include <opencv2/highgui/highgui.hpp>
#include <iostream>

int main(int argc, const char *argv[])
{
    if (argc != 3) {
        std::cerr << "Usage: ./binarization path_to_img threshold" <<
        std::endl;
        return 1;
    }

    // read image
    cv::Mat img = cv::imread(argv[1], cv::IMREAD_GRAYSCALE);
```

```

float threshold = std::atof(argv[2]);
const int L = 256;
std::vector<std::vector<float>> mi(img.rows,
→    std::vector<float>(img.cols, 0));
auto output = gaussian_blur(img);

float maxm = 0;
for (int i = 1; i < output.rows-1; i++) {
    for (int j = 1; j < output.cols-1; j++) {
        unsigned s = 0;
        for (int x = -1; x <= 1; x++) {
            for (int y = -1; y <= 1; y++) {
                s += std::abs(output.at<unsigned char>(i,j)-output.at<unsigned
→    char>(i+x,j+y));
            }
        }
        mi[i][j] = (1.0*s)/(s+(L-1));
        maxm = std::max(maxm, mi[i][j]);
    }
}

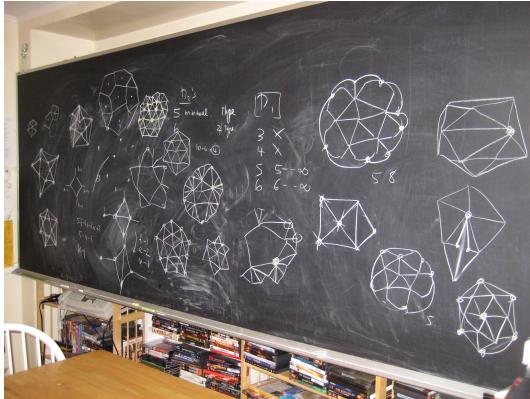
for (int i = 1; i < output.rows-1; ++i) {
    for (int j = 1; j < output.cols-1; j++) {
        output.at<unsigned char>(i,j) = (mi[i][j]/maxm < threshold) ? 0 :
→    255;
    }
}

namedWindow("Display window", cv::WINDOW_AUTOSIZE);
imshow("Display window", output);
cv::waitKey(0);
imwrite("edge_detection.png", output);

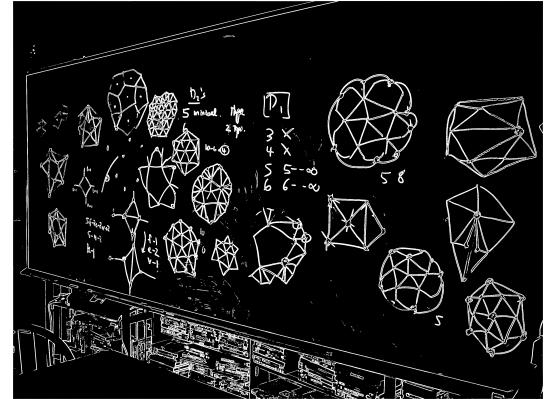
return 0;
}

```

Na Slici 10 je prikazan rezultat rada algoritma.

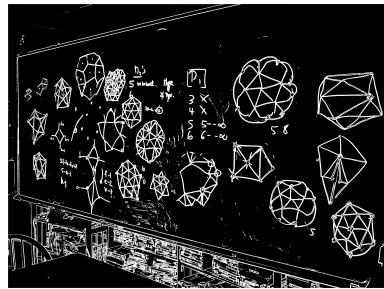


Slika 9: Original

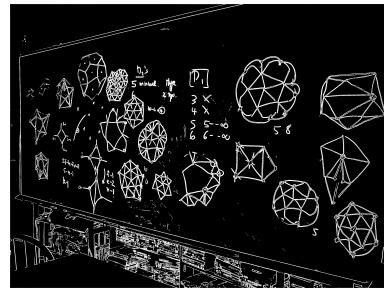


Slika 10: Rezultat

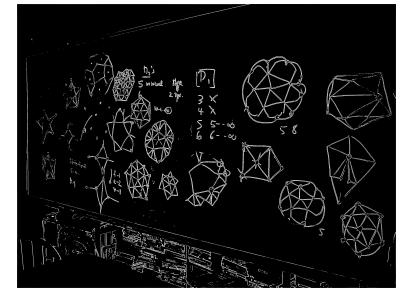
Na sledećim slikama možemo videti kako se rezultat menja u zavisnosti od praga koji se zadaje.



Slika 11: threshold = 0.25



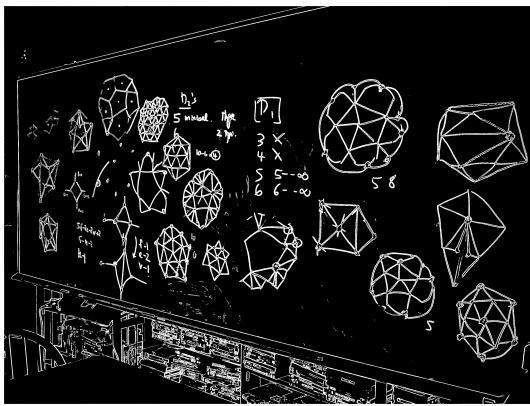
Slika 12: threshold = 0.35



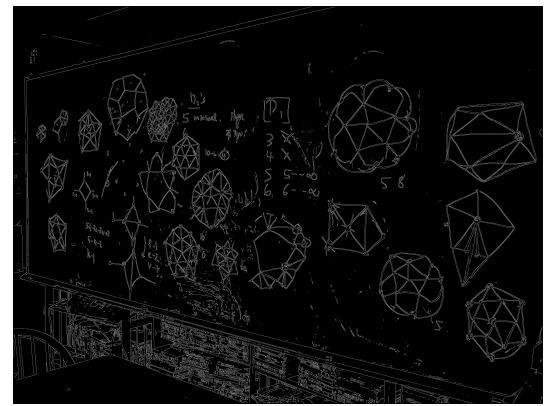
Slika 13: threshold = 0.5

3.2 FED i CED

Na Slikama 14 i 15 možemo videti izlaze ovih algoritama.

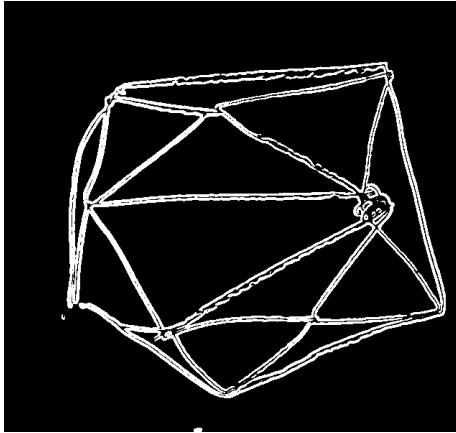


Slika 14: FED rezultat

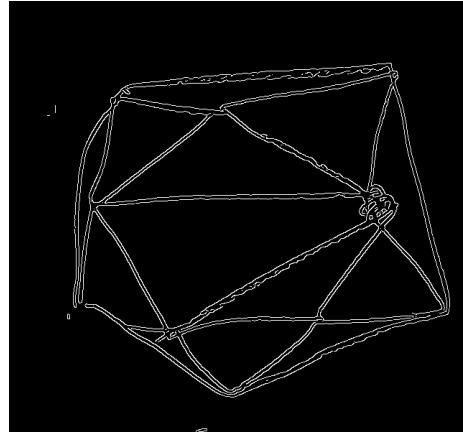


Slika 15: Canny rezultat

Iako naizgled izgleda da FED daje bolje rezultate, ivice kod Canny algoritma su tanje (sto je poželjno) i to možemo videti tako što ćemo prikazati uvećane delove Slika 14 i 15.



Slika 16: FED rezultat



Slika 17: Canny rezultat

U poređenju sa CED, FED je dosta brži. Naime, za obradu Slike 9 sto puta, FED algoritmu je bilo potrebno 60 sekundi, dok CED algoritmu 110. Naravno, ograničavamo se na ručnu implementaciju CED-a autora koja je najverovatnije sporija od implementacije istog algoritma u bibliotekama.

Još jedna prednost FED u odnosu na CED je to što se mnogo lakše implementira.

4 Poboljšavanje kvaliteta slike korišćenjem fazi logike

Poboljšavanje kvaliteta slike (eng. image enhancement) predstavlja jedan od osnovnih zadataka obrade slika. Cilj je obraditi ulaznu sliku tako da izlazna bude pogodnija od ulazne za neku specifičnu upotrebu.

Slike sa lošim kontrastom, loše osvetljene i zamućene slike se dobijaju usled nesavršenosti uređaja koji ih prave (senzori, fotoaparati, itd.), neodgovarajućih uslova za vreme pravljenja slike (loše svetlo) i tokom prenosa slika.

Sa druge strane, slike sa dobrim kontrastom, dobro osvetljene slike, one koje daju dovoljno informacija posmatraču, potrebne su za veliki broj drugih oblasti (medicina, analiza prostora, autonomna navigacija, obrada slika dobijenih satelitima) ili za dalju obradu slika.

Metodi za poboljšavanje kvaliteta slike se mogu podeliti u 3 grupe [4]:

1. Prostorni metod (eng. spatial domain method)
2. Metod domena frekvencija (eng. frequency domain method)
3. Fazi metod (eng. fuzzy method)

Prostorni metodi primenjuju određenu transformaciju nad svakim pikselom slike pojedinačno. Daju dosta dobre rezultate, ali se može desiti da rezultujuća slika ne izgleda uvek adekvatno (npr. da izgleda isprano). Neke od prostornih metoda zasnivaju svoj rad na histogramu slike (pomeranje histograma (eng. histogram sliding), razvlačenje histograma (eng. histogram stretching), podešavanje histograma (eng. histogram equali-

zation)). Rezultati metoda podešavanja histograma će biti poređeni sa rezultatima koje daje predstavljena fazi metoda.

Metod domena frekvencija se pokazuje kao dosta skup [3], čak i uz korišćenje brzih transformacija (npr. brza Furijeova transformacija), tako da nije pogodan za obradu slika u realnom vremenu.

Fazi metod je sposoban da imitira ponašanje eksperta uz korišćenje baze znanja. Postoji veliki broj metoda koji kombinuje metodu podešavanja histograma sa metodom zasnovanom na fazi logici. Mnoge fazi metode prate standardan sled koraka (fazifikacija, promena vrednosti funkcije pripadnosti, defazifikacija), dok neke ne prolaze kroz sve ove korake.

U nastavku su predstavljene dve metode za poboljšavanje kvaliteta slike zasnovane na fazi logici: jedna za slike u boji, druga za crno-bele slike.

4.1 Tehnika za poboljšavanje kvaliteta crno-belih slika sa lošim kontrastom [3]

Vrednosti ulazne slike X , dimenzija $M \times N$, fazifikuju se primenom funkcije μ :

$$\mu_{ij} = \frac{x_{ij} - x_{min}}{x_{max} - x_{min}},$$

gde je $0 \leq i < M, 0 \leq j < N$, x_{max} maksimalna, a x_{min} minimalna vrednost piksela na slici X i x_{ij} vrednost piksela na mestu ij slike X . Za 8-bitnu sliku, vrednost piksela može biti u intervalu $[0, 2^8 - 1]$.

μ_{ij} predstavlja stepen osvetljenosti koju ima piksel ij na ulaznoj slici X . U fazi terminologiji, μ je funkcija pripadnosti i omogućava prevodenje nefazi ulaza (slika X) u njen odgovarajući fazi oblik. Primetimo da je $0 \leq \mu_{ij} \leq 1$.

Kako bi se dobio što bolji rezultat, za modifikaciju funkcije pripadnosti se uzima poznati INT operator, koji su predložili King i Pal 1981. godine [5].

INT operator primenjen na vrednostima funkcije pripadnosti μ :

$$\bar{\mu}_{ij}(\mu_{ij}) = \begin{cases} 2 \cdot \mu_{ij}^2 & 0 \leq \mu_{ij} \leq \mu_c \\ 1 - (2 \cdot (1 - \mu_{ij})^2) & \mu_c < \mu_{ij} \leq 1 \end{cases} \quad (8)$$

Tačka μ_c je granična vrednost koja se može menjati. Najčešće uzimana vrednost za nju je 0.5 (takođe korišćenja i u našoj implementaciji).

Formula kojom je predstavljen INT operator, u stvari, predstavlja matematičku formulaciju naredna 3 if-then pravila [9]:

1. Ako je piksel svetao, onda će biti svetiji.
2. Ako je piksel siv, onda će biti siv.
3. Ako je piksel taman, onda će biti tamniji.

Praktično, nakon primene INT operatora, vrednost piksela sa funkcijom pripadnosti manjom od 0.5 će se smanjiti. Oni pikseli sa većom vrednošću funkcije pripadnosti od 0.5 imaće veću vrednost.

Nakon modifikacije funkcije pripadnosti, dobijene vrednosti treba defazifikovati i na taj način dobiti nove nijanse sive, izlaznu sliku.

Izlazna slika Y se dobija primenom inverzne funkcije μ^{-1} :

$$y_{ij} = \mu^{-1}(\bar{\mu}_{ij}) = x_{min} + \bar{\mu} \cdot (x_{max} - x_{min}).$$

U nastavku je kod koji implementira gore navedeni algoritam.

```
#include <iostream>
#include <vector>
#include <opencv2/highgui/highgui.hpp>

int main(int argc, const char *argv[])
{
    if (argc != 2) {
        std::cerr << "Usage: ./fuzzy_grayscale path_to_image" << std::endl;
        return 1;
    }

    // input image
    cv::Mat input = cv::imread(argv[1], cv::IMREAD_GRAYSCALE);
    // output image
    cv::Mat output = cv::Mat(input.rows, input.cols, CV_8UC1,
                           cv::Scalar(255));

    unsigned min_gray = *std::min_element(input.begin<unsigned char>(),
                                         input.end<unsigned char>());
    unsigned max_gray = *std::max_element(input.begin<unsigned char>(),
                                         input.end<unsigned char>());
    unsigned diff = max_gray - min_gray;

    for (int i = 0; i < input.rows; i++) {
        for (int j = 0; j < input.cols; ++j) {
            // fuzzification
            double mem_func = (input.at<unsigned char>(i,j) - min_gray) /
                           double(diff);
```

```

// modification of membership function by INT operator
if (mem_func <= 0.5 and mem_func > 0) {
    mem_func = 2 * pow(mem_func, 2);
}
else if (mem_func <= 1 and mem_func > 0.5) {
    mem_func = 1 - 2 * pow(1 - mem_func, 2);
}

// defuzzification
output.at<unsigned char>(i,j) = min_gray + mem_func * diff;
}

// Show and save
namedWindow("Display window", cv::WINDOW_AUTOSIZE);
imshow("Display window", output);
cv::waitKey(0);
imwrite("fuzzy_grayscale.jpg", output);

return 0;
}

```

Na Slici 19, 21 i 23 su prikazani rezultati rada algoritma.



Slika 18: Original



Slika 19: Rezultat



Slika 20: Original



Slika 21: Rezultat



Slika 22: Original



Slika 23: Rezultat

Primetimo da je na sve tri slike kontrast adekvatno poboljšan. Rezultati su dobijeni za $\mu_c = 0.5$. Očigledno je da su u velikoj meri zavisni od μ_c , što potvrđuju primeri:



Slika 24: $\mu_c = 0.234$



Slika 25: $\mu_c = 0.7$



Slika 26: $\mu_c = 1$



Slika 27: $\mu_c = 0.3$



Slika 28: $\mu_c = 0.6$



Slika 29: $\mu_c = 0.9$

Rezultati nisu dobri, sem oni gde je vrednost μ_c bliska 0.5 (Slika 25 i 28). Na taj način je eksperimentalno pokazano da je poželjno da μ_c uzima vrednosti bliske 0.5.

4.2 Tehnika za poboljšavanje kvaliteta slika u boji sa lošim kontrastom [4]

RGB model boja je aditivni model boja koji opisuje boju dodavanjem odgovarajućeg stepena crvene, zelene i plave boje. Taj stepen, za 8-bitnu sliku, može imati vrednosti u intervalu $[0, 2^8 - 1]$. Metod opisan u nastavku poboljšava kontrast slike opisane u RGB modelu boja.

Fazifikacija, izmena vrednosti funkcije pripadnosti i defazifikacija se rade posebno na svakom kanalu učitane slike X , dimenzija $M \times N$.

Pre nego što uvedemo funkciju pripadnosti, definisamo količničku fazi konstantu F_d (eng. denominational fuzzifier) kao

$$F_d = \frac{x_{max} - x_{mid}}{0.5^{-\frac{1}{F_e}} - 1},$$

gde je F_e eksponencijalna fazi konstanta, x_{max} maksimalna vrednost trenutno posmatranog kanala piksela na slici X , x_{mid} srednja vrednost trenutno posmatranog kanala piksela na slici X .

Eksperimentalno je pokazano da su dobro izabrane vrednosti za F_e 1 i 2.

Funkciju pripadnosti za jedan kanal (R, G ili B) definišemo na sledeći način:

$$\mu_{ij} = \left(1 + \frac{x_{max} - x_{ij}}{F_d} \right)^{-F_e},$$

gde je $0 \leq i < M, 0 \leq j < N$. Primetimo da važi $0 \leq \mu_{ij} \leq 1$.

Nakon fazifikacije se vrši modifikacija funkcije pripadnosti za svaki kanal svakog piksela korišćenjem INT operatora definisanog u formuli (8).

Defazifikacija se vrši na sledeći način:

$$y_{ij} = \mu^{-1}(\bar{\mu}_{ij}) = x_{max} - F_d \cdot (\bar{\mu}_{ij})^{-F_e} + F_d,$$

gde je $\bar{\mu}_{ij}$ rezultat primene INT operatora na μ_{ij} .

Defazifikacijom dobijamo vrednost trenutnog kanala piksela (y_{ij}) koji će činiti izlaznu sliku Y .

Kombinovanjem novodobijenih vrednosti crvene, zelene i plave boje svakog piksela ulazne slike X , dobija se izlazna slika Y .

U nastavku je prikazan kod kao reprezentacija prikazanog algoritma.

```
#include <iostream>
#include <vector>
#include <opencv2/highgui/highgui.hpp>

#define NUM_CHANNELS 3

int main(int argc, const char *argv[])
{
    if (argc != 2) {
        std::cerr << "Usage: ./fuzzy_color path_to_image" << std::endl;
        return 1;
    }

    // input image
    cv::Mat input = cv::imread(argv[1], CV_LOAD_IMAGE_COLOR);
    // output image
    cv::Mat output = cv::Mat(input.rows, input.cols, CV_8UC3);

    unsigned max_R, min_R;
    unsigned max_G, min_G;
    unsigned max_B, min_B;
    unsigned R, G, B;

    max_R = min_R = input.at<cv::Vec3b>(0, 0)[2];
    max_G = min_G = input.at<cv::Vec3b>(0, 0)[1];
    max_B = min_B = input.at<cv::Vec3b>(0, 0)[0];

    for (int i = 1; i < input.rows; i++) {
        for (int j = 1; j < input.cols; ++j) {
            R = input.at<cv::Vec3b>(i, j)[2];
            G = input.at<cv::Vec3b>(i, j)[1];
            B = input.at<cv::Vec3b>(i, j)[0];

            if (max_R < R) max_R = R;
            else if (min_R > R) min_R = R;

            if (max_G < G) max_G = G;
            else if (min_G > G) min_G = G;

            if (max_B < B) max_B = B;
            else if (min_B > B) min_B = B;
        }
    }
}
```

```

}

std::vector<unsigned> max_RGB {max_R, max_G, max_B};
std::vector<double> mid_RGB { (max_R - min_R) / 2.0, (max_G - min_G) /
→ 2.0, (max_B - min_B) / 2.0};

std::vector<unsigned> RGB;
RGB.reserve(3);
std::vector<unsigned> curr_RGB;
curr_RGB.reserve(3);

double Fe = 2.0;
double Fd;

for (int i = 0; i < input.rows; i++) {
    for (int j = 0; j < input.cols; ++j) {
        curr_RGB.resize(0);
        curr_RGB.push_back(input.at<cv::Vec3b>(i, j)[2]);
        curr_RGB.push_back(input.at<cv::Vec3b>(i, j)[1]);
        curr_RGB.push_back(input.at<cv::Vec3b>(i, j)[0]);
        RGB.resize(0);

        for (int c = 0; c < NUM_CHANNELS; ++c) {
            Fd = (max_RGB[c] - mid_RGB[c]) / 0.41421356237309515;
            // fuzzification
            double mem_func = pow(1 + (max_RGB[c] - curr_RGB[c]) /
→ Fd, -Fe);

            // modification of membership function by INT operator
            if (mem_func <= 0.5 and mem_func > 0) {
                mem_func = 2 * pow(mem_func, 2);
            }
            else if (mem_func <= 1 and mem_func > 0.5) {
                mem_func = 1 - 2 * pow(1 - mem_func, 2);
            }

            // defuzzification
            RGB.push_back(max_RGB[c] - Fd * (pow(mem_func, -1 / Fe)) +
→ + Fd);
        }
    }
    cv::Vec3f BGR(RGB[2], RGB[1], RGB[0]);
    output.at<cv::Vec3b>(i, j) = BGR;
}
}

```

```

    // Show and save
    namedWindow("Display window", cv::WINDOW_AUTOSIZE);
    imshow("Display window", output);
    cv::waitKey(0);
    imwrite("fuzzy_color.jpg", output);

    return 0;
}

```

Na Slici 31, 33 i 35 su prikazani rezultati rada algoritma.



Slika 30: Original



Slika 31: Rezultat



Slika 32: Original



Slika 33: Rezultat



Slika 34: Original



Slika 35: Rezultat

4.3 Poređenje rezultata metoda za poboljšavanje kvaliteta slika u boji, crno-belih slika i metoda podešavanja histograma

Metoda podešavanja histograma (eng. histogram equalization - HE) je jedna od najpopularnijih metoda za obradu slika. Zbog toga što su njeni rezultati dobri, upoređeni su sa rezultatima predstavljenih fazi metoda.



Slika 36: Originalna slika



Slika 37: Rezultat metode HE



Slika 38: Rezultat fazi metode za crno-bele slike



Slika 39: Rezultat fazi metode za slike u boji

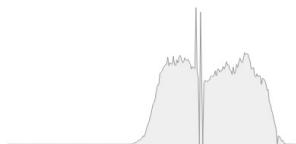
Na osnovu dobijenih rezultata se uveravamo u to da HE metod daje rezultate vredne pažnje. Ponekad (Slika 37) rezultat HE metoda nije ni blizu onoga što bi uradila pandan fazi metoda. Ipak, to ne znači da je rezultat lošiji - zavisi šta je posmatraču potrebno od slike. U slučaju da je to možda veća količina informacija, Slika 37 daje najbolje rezultate. Sudeći po tome koliko uspešno je poboljšan kontrast, definitivno su pobednici fazi metode.



Slika 40: Originalna slika



Slika 41: Rezultat metode HE



Slika 42: Histogram originalne slike



Slika 43: Histogram nakon HE



Slika 44: Rezultat fazi metode za crno-bele slike



Slika 45: Rezultat fazi metode za slike u boji



Slika 46: Histogram nakon fazi metode za crno-bele slike



Slika 47: Histogram nakon fazi metode za slike u boji

Kada je kontrast mera kvaliteta, fazi metode uspevaju da proizvedu ubedljivo bolje rezultate. Vidno najbolje rezultate daje metoda za poboljšavanje kvaliteta slika u boji.

5 Zaključak

Teorija fazi skupova i fazi logika imaju značajnu primenu u procesiranju slika. Neke od tih primena videli smo i u ovom radu - binarizacija slike, detektovanje ivica, podešavanje kontrasta i osvetljenosti.

Opisana su dva algoritma, Fuzzy C-means (FCM) za binarizaciju slike i Fuzzy Edge Detection (FED) za detekciju ivica. Ovi algoritmi su poređeni sa algoritmima koji ne koriste fazi logiku. Konkretno, poredili smo FCM sa k-means-om i FED sa Canny Edge Detection (CED) algoritmom. Slike dobijene FCM-om i k-means-om se skoro i ne razlikuju, dok to nije slučaj sa FED-om i CED-om, čiji su izlazi primetno drugačiji. U oba slučaja su algoritmi koji koriste fazi logiku bila brža.

Takođe, opisana su i dva metoda za poboljšavanje kvaliteta slika (slika u boji i crno-belih) koji su, u poređenju sa tradicionalnim tehnikama zasnovanim na histogramu slike, dali mnogo bolje rezultate, na uštrb prostorne i vremenske složenosti.

Glavna prednost fazi pristupa jeste jednostavnost njihove implementacije i sposobnost da opišu neodređenosti (eng. vagueness) na slikama.

Pored prednosti, ovi pristupi imaju i mane. Nekad to može biti brzina izvršavanja, ili pak memorija. Takođe, i sami rezultati se mogu dosta razlikovati od rezultata algoritama koji nisu bazirani na fazi logici, kao što je to bio slučaj kod detekcije ivica i fazi metode za poboljšavanje kvaliteta crno-belih slika.

Literatura

- [1] Nebojša Perić, "Neke primene teorije fazi skupova i fazi logike u procesiranju slika", Matematički fakultet u Beogradu, 2014.
- [2] Mario I. Chacon M, "Fuzzy binarization and segmentation of text images for opcr", Mexico New Mexico State University
- [3] Ajay Kumar Gupta, Siddharth Singh Chouhan, Manish Shrivastava, "Fuzzy based Low Contrast Image Enhancement Technique by using Pal and King Method", International Journal of Computer Applications, May 2016.
- [4] Pooja Mishra, Khom Lal Sinha, "A Highly Efficient Color Image Contrast Enhancement using Fuzzy Based Contrast", Advanced Research in Electrical and Electronic Engineering, 2014
- [5] Sankar K. Pal, Robert A. King, "Image Enhancement using Smoothing with Fuzzy Sets", IEEE Transactions on Systems, Man and Cybernetics, July 1981.
- [6] S. N. Sivanandam, S. Sumathi, S. N. Deepa, "Introduction to Fuzzy Logic using MATLAB Springer", Verlag Berlin Heidelberg 2007.
- [7] Hamid R. Tizhoosh, "Fuzzy Image Enhancement: An Overview", University of Magdeburg, Department for Technical Computer Science
- [8] Shital B.Nikam, Dnyaneshwar D. Ahire, "A Survey on Fuzzy Based Techniques For Contrast Enhancement", International Research Journal of Engineering and Technology, November 2016
- [9] Tarun Mahashwari, Amit Asthana, "Image Enhancement Using Fuzzy Technique", International Journal of Research Review in Engineering Science and Technology, June 2013