SVEUČILIŠTE U RIJECI

TEHNIČKI FAKULTET

Preddiplomski sveučilišni studij računarstva

Završni rad

Postupci detekcije ruba pri obradi digitalne slike

Patricija Zubalić

0069065136

Rijeka, rujan 2016.

SVEUČILIŠTE U RIJECI

TEHNIČKI FAKULTET

Preddiplomski sveučilišni studij računarstva

Završni rad

Postupci detekcije ruba pri obradi digitalne slike

Mentor: doc.dr.sc.Jerko Škifić

Patricija Zubalić

0069065136

Rijeka, rujan 2016.

Izjava o samostalnoj izradi rada

Izjavljujem da sam samostalno izradila ovaj rad.

Rijeka, rujan 2016. _____

Patricija Zubalić

Zahvala

Zahvaljujem se mami što mi je bila podrška.

Zahvaljujem se tati što je bio spreman financirati moje studiranje koliko god trajalo.

Zahvaljujem se tatinoj sestrični što mi je pomogla kad je bilo najteže.

Zahvaljujem se mentoru koji je rekao da je volja najvažnija, a sve ostalo se nauči.

Zahvaljujem se prijateljima zbog kojih je studiranje bilo lakše.

Sadržaj

1 Uvod

Slika je dvodimenzionalni signal i u računalu je pohranjena kao matrica. Slika se obrađuje na način da se pojedinačno mijenja svaki njen piksel, prema određenoj formuli kojom se dobiva željeni efekt. Obrada digitalne slike može se podijeliti u dvije vrste obrade, a to su operacije koje uzimaju u obzir samo piksel na kojem se u tom trenutku funkcija nalazi, te filteri - operacije koji uzimaju u obzir i okolne piksele pri izmjeni trenutnog piksela. Ovaj završni rad opisuje implementaciju detektora ruba slike, a detektori pri izmjeni pojedinog piksela uzimaju u obzir i vrijednosti okolnih piksela.

Detekcija rubova spada u obradu digitalne slike, koja spada u područje računalne grafike. Detekcija rubova prvi je korak razumijevanja i analize slike. Jedna je od osnovnih koraka u obradi slike, analizi slike, prepoznavanju uzorka slike i računalnom vidu.

2 Slika i obrada slike

Slika je dvodimenzionalan signal. U računalu je pohranjena kao matrica u kojoj se svaki broj sastoji od jednog ili tri kanala, ovisno o tome je li slika u boji ili siva slika. Pri detekciji rubova uvijek se koriste sive slike, odnosno slike s jednim kanalom. Pojedinom pikselu slike pristupa se na slijedeći način;

```
uchar piksel;

for (int x = 0; x < input.rows; ++x)
{
  for (int y = 0; y < input.cols ; ++y)
  {
    piksel = input.at<uchar>(Point (x, y));
  }
}
```

gdje je *input* naziv matrice u kojoj je spremljena slika, a x i y označavaju red, odnosno stupac piksela na kojem se funkcija u određenom trenutku nalazi.

2.1 Obrada slike

Primjena filtera temelji se na konvoluciji. U matematici, konvolucija je matematička operacija nad dvije funkcije (f i g), a producira treću funkciju, koja je obično gledana kao modificirana verzija jedne originalne funkcije, dajući integral množenja točaka dvije funkcije.

2.2 Konvolucija pri obradi slike

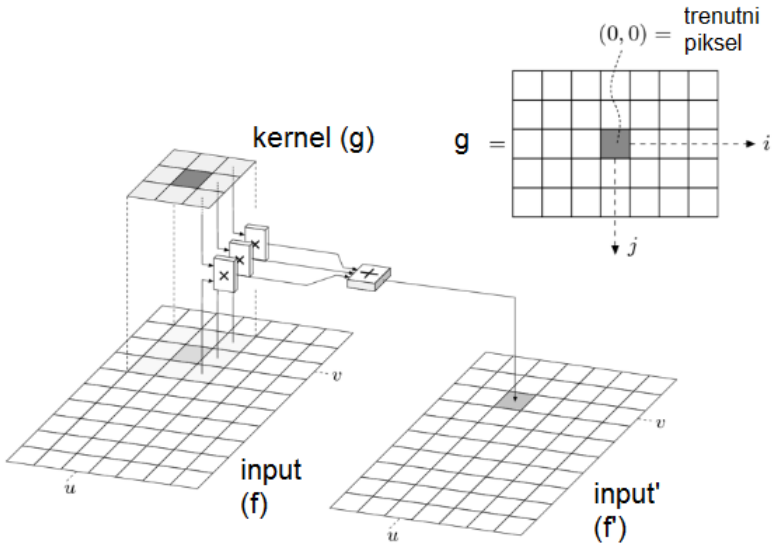
Na području obrade slika, konvolucija označava množenje piksela izvorne slike s pikselima kernela (konvolucijske maske). Izvorna slika predstavlja prvu funkciju (ili funkciju f), a kernel predstavlja drugu funkciju (ili funkciju g). Rezultat konvolucije te dvije funkcije je filtrirana slika, odnosno funkcija $f * g$.

Prikazan je dio programskog kôda kojim se izvršava konvolucija.

```
//iteracija po slici
for ( int x = 0; x < input.rows; ++x )
{
    for ( int y = 0; y < input.cols; ++y )
    {
        tmpX = x; tmpY = y;

        //iteracija po kernelu
        for ( int xk = 0; xk <size; ++xk )
        {
            for ( int yk = 0; yk < size; ++yk )
           {

                //rubni uvjeti
                if ( x == 0 ) tmpX =  x + 1;
                if ( y == 0 ) tmpY =  y + 1;
                if ( x == ( input.rows-1 ) ) tmpX = x - 1;
                if ( y == ( input.cols-1 ) ) tmpY = y - 1;

                //konvolucija
                value = input.at<uchar> ( Point ( xk+tmpX-1, yk+tmpY-1 )
   ) * kernel.at<float> ( Point ( xk, yk ) );
                sum = sum +  value;
            }

            output.at<uchar> ( Point ( x, y ) ) = sum;
            sum = 0;

```

```
28              }
29          }
30  }
```

U matrici *kernel* pohranjen je kernel Gaussovog filtera. Trenutni piksel, kao i pikseli okolne slike, množe se s vrijednostima kernela te se spremaju u varijablu *sum*. Koliko okolnih piksela se uzima ovisi o veličini kernela, odnosno vrijednostima *xk* i *yk*. Vrijednost te varijable postaje vrijednost trenutnog piksela. Taj proces vizualno je prikazan na slici 2.1.



Slika 2.1: Vizualni prikaz konvolucije pri obradi slike

3 Softver i knjižnica korišteni za implementaciju

3.1 KDevelop 4

Za razvojno okruženje korišten je softver KDevelop 4 (Version 4.7.1, Using KDE Development Platform 4.14.6). Kdevelop je besplatan softver integriranog development okruženja (eng. *integrated development environment, IDE*) za KDE Platformu na računalnim operacijskim sustavima baziranim na Unixu. KDevelop ne uključuje kompajler, već koristi vanjski kompajler kao što je GCC ili Clang, da bi producirao izvršive binarne datoteke. [1]

3.2 OpenCV

Za rad sa slikama korištena je OpenCV (eng. *Open Source Computer Vision Library*) softver knjižnica. OpenCV je softver knjižnica otvorenog koda namjenjena područjima računalnog vida i strojnog učenja. Napravljena je kako bi omogućila zajedničku infrastukturu za aplikacije računalnog vida, te za povećanje korištenja strojne percepcije u komercijalnim proizvodima. Knjižnica je BSD- licenciran proizvod (što znači da ima minimalnu destrikciju na redistribuciju) te iz tog razloga omogućava laku upotrebu i izmjenu.

OpenCV ima više od 2500 optimiziranih algoritama. Neki od algoritama su algoritmi za detekciju i prepoznavanje lica, identificiranje objekata, prepoznavanje ljudskih radnji na videu, praćenje objekata koji se miču, generiranje 3D modela objekata, spajanje slika kako bi se dobila visoka rezolucija cijele scene, pronalaženje sličnih slika iz baze podataka, otklanjanje efekta crvenih očiju i sl.

Neke od primjena OpenCV knjižnice su pomaganje robotima locirati i podići objekte, pokretanje interaktivne umjetnosti, inspekcija oznaka na proizvodima u tvrtkama diljem svijeta, te brzo prepoznavanja lica. [2]

3.2.1 Korištene funkcije iz OpenCV knjižnice

- IMREAD - učitavanje slike

  Mat imread *(const String& filename, int flags=IMREAD_COLOR)*

- FILTER2D - primjena filtera

  void filter2D *(InputArray src, OutputArray dst, int ddepth, InputArray kernel, Point anchor=Point(-1,-1), double delta=0, int borderType=BORDER_DEFAULT)*

- GAUSSIANBLUR - primjena Gaussovog filtera

  void GaussianBlur *(InputArray src, OutputArray dst, Size ksize, double sigmaX, double sigmaY=0, int*

- NAMEDWINDOW i IMSHOW - prikaz slike

  void namedWindow *(const string& winname, int flags=WINDOW_AUTOSIZE)*

  void imshow *(const string& winname, InputArray mat)*

4 Faze pri detekciji rubova

4.1 Uglađivanje slike

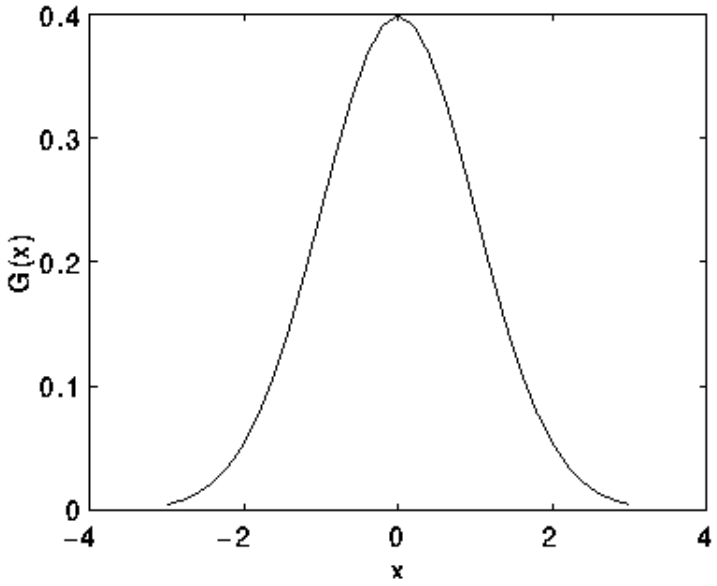
Prije primjene algoritma za detekciju rubova otklanjaju se smetnje pomoću filtera za uglađivanje slike. Najčešće se koristi Gaussov filter.

U elektronici i obradi signala, Gaussov filter je filter čiji je impulsni odziv Gaussova funkcija (ili njena aproksimacija). [3]

Gaussova distribucija u jednoj dimenziji ima slijedeći oblik;

$$g(x) = \frac{1}{\sqrt{2\pi\sigma}} exp(\frac{-x^2}{2\sigma^2}) \tag{4.1}$$

gdje je *sigma(σ)* standardna devijacija. [4] Na slici 4.1 je vizualno prikazana distribucija Gaussa, uz σ = 1 i pretpostavku da distribucija ima središte u nuli.



Slika 4.1: Vizualni prikaz 1D distribucije Gaussa [4]

Kako je slika 2D signal, za njeno uglađivanje potrebna je 2D distribucija Gaussove funkcije, prema formuli prikazanoj u izrazu 4.2.

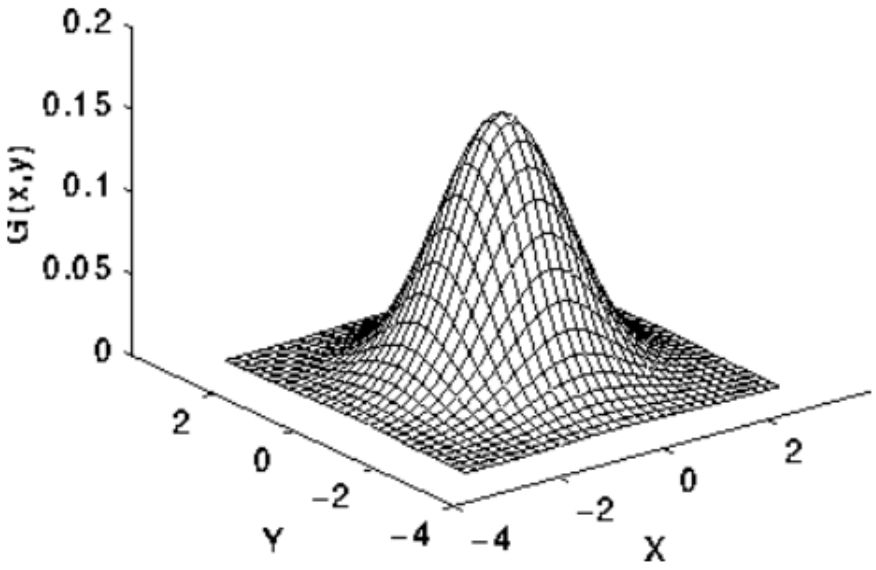
$$g(x,y) = \frac{1}{2\pi\sigma^2} exp(\frac{-x^2+y^2}{2\sigma^2}) \tag{4.2}$$

Na temelju te formule kreira se kernel s kojim se konvoluira izvorna slika. Na slici 4.2 prikazan je kernel veličine 5x5 sa sigmom vrijednosti 1.

$$\frac{1}{273}\begin{bmatrix} 1 & 4 & 7 & 4 & 1 \\ 4 & 16 & 26 & 16 & 4 \\ 7 & 26 & 41 & 26 & 7 \\ 4 & 16 & 26 & 16 & 4 \\ 1 & 4 & 7 & 4 & 1 \end{bmatrix}$$

Slika 4.2: Kernel Gaussovog filtera veličine 5x5

Gaussova funkcija u dvije dimenzije prikazana je na slici 4.3.

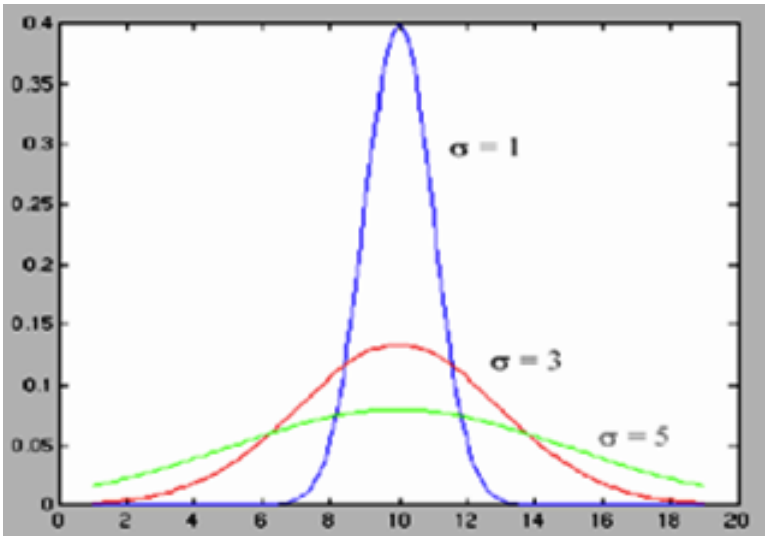


Slika 4.3: Vizualni prikaz 2D distribucije Gaussa [4]

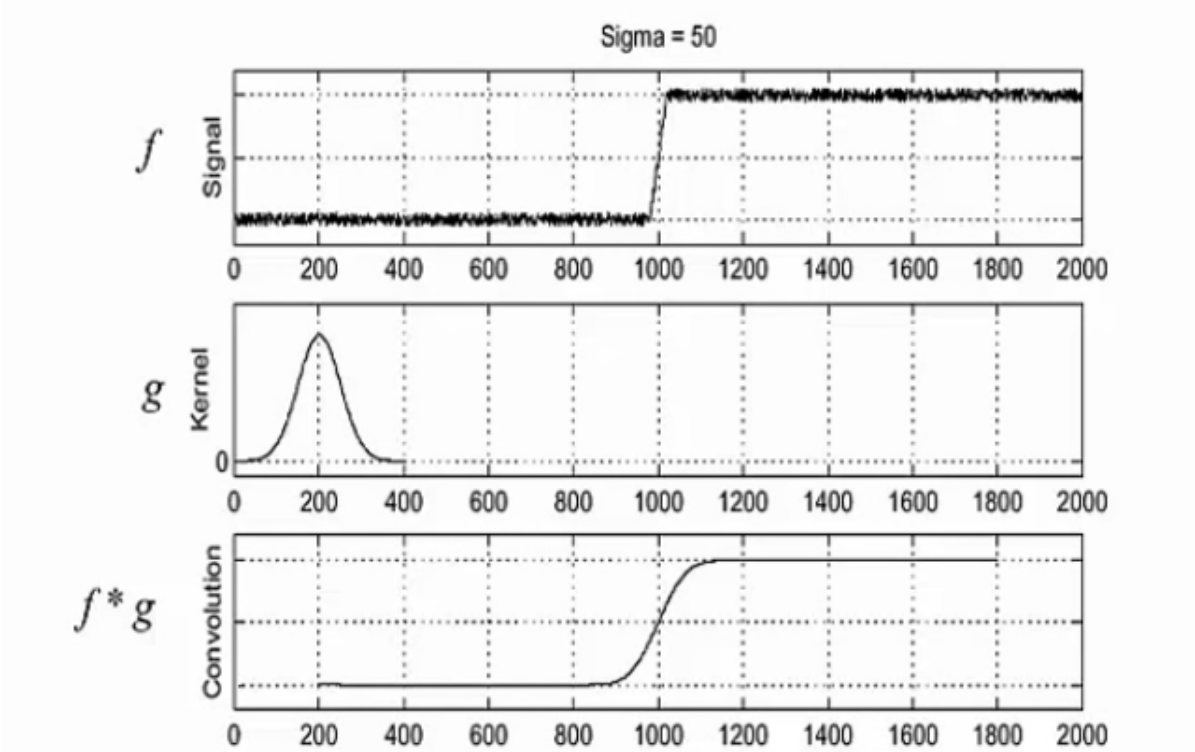
Gaussov filter također uklanja i detalje sa slike, te i tu činjenicu treba uzeti u obzir

pri definiranju jačine filtera. Jačinu filtera definira se putem sigme, te što je ona veća, slika ima manje detalja i manje smetnji.

Na slici 4.4 prikazana je ovisnost veličine kernela o vrijednosti sigme, a na slici 4.5 prikazana je konvolucija signala izvorne slike s kernelom Gaussovog filtera, odnosno proces otklanjanja smetnji.



Slika 4.4: Ovisnost veličine kernela o sigmi



Slika 4.5: Konvolucija izvorne slike s kernelom Gaussovog filtera

4.2 Diferencijacija

Nakon što su sa slike otklonjene smetnje, rubovi se mogu naći pomoću prve derivacije. [5] Izraz 4.3 prikazuje derivaciju kontinuirane funkcije *f*.

$$f' = \frac{df}{dx} = \lim_{\Delta x \to 0} \frac{f(x) - f(x - \Delta x)}{\Delta x} \tag{4.3}$$

Kako se slike sastoje od diskretnih vrijednosti, minimalna vrijednost Δx može biti 1. Prethodna formula se reducira na oblik koji je prikazan u izrazu 4.4.

$$f' = \frac{df}{dx} = f(x) - f(x - 1). \tag{4.4}$$

Taj oblik naziva se diskretna aproksimacija derivacije. Npr. imamo red piksela unutar slike koji izgleda ovako:

f(x)	10	10	10	10	10	20	20	20	20	20

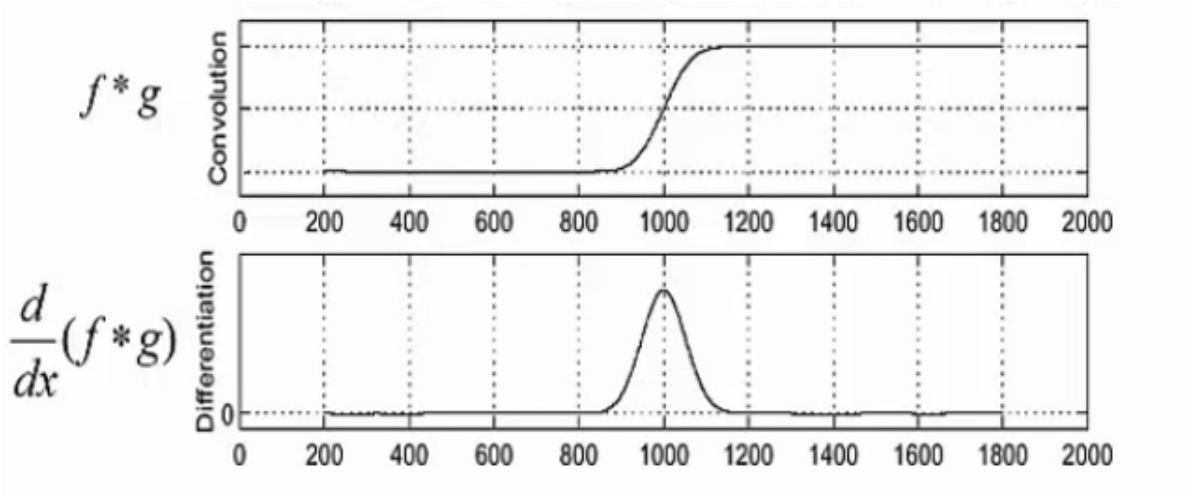
Koristeći prethodno navedenu formulu dobivamo njegovu derivaciju:

f(x) 0 0 0 0 0 10 0 0 0 0

Derivacija može biti izračunata primjenom konvolucijske maske (kernela) na funkciju f, slično filtriranju. U ovom slučaju kernel se sastoji samo od dva elementa (-1 i 1).

Diferencijacija jednog reda piksela vizualno je prikazana na slici 4.6.

Funkcija f je stepenična funkcija i njenom se derivacijom na mjestu ruba dobije visoka vrijednost, što je prikazano na slici 4.6. Još neke od maski za računanje prve derivacije 1D signala su -1 0 1 i -1 0 0 1.



Slika 4.6: Diferencijacija uglađene slike

Vektor gradijenta (promjene intenziteta slike) piksela $f(x, y)$ je (f_x, f_y), gdje je f_x derivacija u smjeru osi x, a f_y derivacija u smjeru osi y. [4] Gradijent ima magnitudu M i smjer θ, čije su formule prikazane u izrazima 4.5 i 4.6.

$$\theta = arctan\frac{f_y}{f_x} \tag{4.5}$$

$$M = \sqrt{f_x^2 + f_y^2} \tag{4.6}$$

Ponekad se za detekciju slike koristi i druga derivacija, a taj će slučaj biti naknadno spomenut.

4.3 Detekcija

Rubovi su nagle promjene intenziteta slike, te postoji više vrsta rubova. Idealan rub je signal s oblikom stepenice, međutim najčešće se pojavljuju signali oblika rampe. Navedene vrste rubova prikazane su na slici 4.3, a rubovi mogu nastati kao:

- granice između objekata
- promjene osvjetljenja
- promjena u orijentaciji površine objekta

(a) Stepenica - nagla promjena

(b) Rampa - postepena promjena

Slika 4.7: Vrste rubova [6]

Rubovi se detektiraju pomoću gradijenta magnitude, a gradijent magnitude piksela (x, y) definiran je u izrazu 4.7. [5]

$$M(x, y) = \sqrt{f^2x(x, y) - f^2y(x, y)} \tag{4.7}$$

5 Razni algoritmi za detekciju rubova

Algoritmi za detekciju rubova dijele se u dvije grupe:

- detektori bazirani na gradijentu - metoda gradijenta detektira rubove tako da pronalazi maksimum ili minimum u prvoj derivaciji slike. U ovu skupinu spadaju Prewitt, Sobel i Robert's cross.
- detektori bazirani na Laplasijanu - metoda Laplasijana detektira rubove tako da pronalazi sjecište u nuli u drugoj derivaciji slike. U tu skupinu spadaju Laplasijan i Gausov Laplasijan.

5.1 Prewitt

5.1.1 Definicija

Rubovi se detektiraju pomoću razlike između intenziteta piksela unutar slike. Sve maske koje se koriste za detekciju rubova još se nazivaju i derivacijske maske. Kako je već prethodno navedeno, slika je dvodimenzionalni signal, a promjene u signalu mogu biti izračunate samo pomoću diferencijacije. [7]

Svaka derivacijska maska mora imati slijedeće značajke:

- mora sadržavati i pozitivne i negativne brojeve
- zbroj elemenata unutar maske mora biti nula
- veća vrijednost elementa maske znači veća detekcija ruba

Prewitt detektor se sastoji od dvije maske - jedna za detekciju rubova u horizontalnom smjeru (odnosno u smjeru osi x) i jedna za detekciju rubova u vertikalnom smjeru (odnosno u smjeru osi y). Elementi derivacijske maske su standardizirani i ne mogu se mjenjati.

$$Gx = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} Gy = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}$$

Primjena derivacijske maske na sliku izračunava prvu derivaciju i razliku između intenziteta piksela u pojedinoj regiji. Kako je centralni red, odnosno stupac, nula, ne uključuje originalne vrijednosti slike, već izračunava razliku između desnog i lijevog, odnosno gornjeg i donjeg piksela oko ruba. To povećava intenzitet ruba i postaje pojačan s obzirom na originalnu sliku.[7]

Formula za izračun magnitude gradijenta prikazana je u izrazu 5.8, međutim, najčešće se upotrebljava formula iz izraza 5.9, koja omogućuje brži izračun. U izrazu 5.10 prikazana je formula za izračun kuta orjentacije ruba. [8]

$$|G| = \sqrt{(Gx^2 + Gy^2)} \tag{5.8}$$

$$|G| = |Gx| + |Gy| \tag{5.9}$$

$$\theta = arctan(\frac{Gx}{Gy}) \tag{5.10}$$

5.1.2 Implementacija

Implementirana je funkcija Prewitt koja kao ulaz uzima kernel za horizontalnu i vertikalnu detekciju, izvornu sliku i veličinu kernela te izvršava pronalaženje gradijenta, odnosno detektira rubove.

```
1  Mat Prewitt ( Mat Gx, Mat Gy, Mat& input, int size )
2  {
3      Mat output = Mat::zeros ( input.size(), input.type() );
4
5      double pixelX, pixelY;
6      double Gx_sum = 0, Gy_sum = 0;
7      double tmpX, tmpY;
8
9      //iteracija po slici
10     for ( int x = 0; x < input.rows; ++ )
11     {
12         for ( int y = 0; y < input.cols; ++y )
```

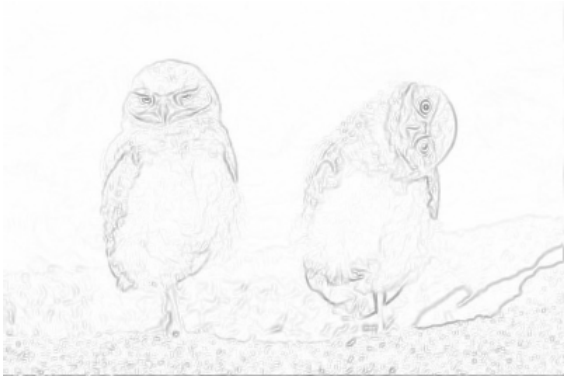
```
13        {
14             tmpX = x; tmpY = y;
15
16
17            //iteracija po kernelu
18             for ( int xk = 0; xk <size; ++xk )
19             {
20                  for ( int yk = 0; yk < size; ++yk )
21                  {
22                  //rubni uvjeti
23                       if ( x == 0 ) tmpX =  x + 1;
24                       if ( y == 0 ) tmpY =  y + 1;
25                       if ( x == ( input.rows-1 ) ) tmpX = x - 1;
26                       if ( y == ( input.cols-1 ) ) tmpY = y - 1;
27
28                       pixelX = input.at<uchar> ( Point ( xk+tmpX-1, yk+
    tmpY-1 ) ) * Gx.at<float> ( Point ( xk, yk ) );
29                       Gx_sum = Gx_sum + pixelX;
30
31                   pixelY = input.at<uchar> ( Point ( xk+tmpX-1, yk+tmpY-1
    ) ) * Gy.at<float> ( Point ( xk, yk ) );
32                   Gy_sum = Gy_sum + pixelY;
33                  }
34             }
35
36             //magnituda gradijenta
37             output.at<uchar> ( Point (x, y) )= abs(Gx_sum) + abs(Gy_sum);
38           //sqrt(ceil((Gx_sum*Gx_sum) + (Gy_sum*Gy_sum)));
39
40             Gx_sum = 0;
41           Gy_sum = 0;
42
43       }
44
45    }
46
```

```
47      return output;
48
49  }
```

5.1.3 Rezultati implementacije

Na slici 5.1 prikazana je izvorna slika, a na slici 5.2 prikazan je rezultat prethodno prikazanog kôda, odnosno efekt detekcije rubova pomoću Prewitt detektora.

Slika 5.1: Izvorna slika



Slika 5.2: Slika nakon primjene Prewitt detektora

5.2 Sobel operator

5.2.1 Definicija

Operator se sastoji od dvije derivacijske maske veličine 3x3. Maske su dizajnirane na način da budu maksimalno osjetljive na vertikalne i horizontalne rubove, kao kod Prewitt detektora. Velika razlika s obzirom na Prewit detektor je to što se kod Sobel detektora elementi maske mogu mijenjati, odnosno prilagođavati, sve dok ne krše neko od pravila maski za derivaciju. [9] Elementi derivacijske maske Prewitta omogućuju usrednjavanje gradijenta na području od tri reda, odnosno stupca. Kada se ta maska primjenjuje na određeni red ili stupac, gubi se puno detalja s tog reda ili stupca. Iz tog razloga se na centralni red, odnosno stupac dodaje veću "težinu", što dovodi do elemenata derivacijske matrice Sobel detektora. Derivacijska matrica Sobela omogućuje otklanjanje smetnji s uzimanjem u obzir što se dešava u prethodnom i sljedećem redu, ali i dalje se zadržavaju posebnosti trenutnog reda (kod detekcije u smjeru osi y). [10]

$$Gx = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} Gy = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

Maske mogu biti primijenjene odvojeno za prikazivanje rubova samo u horizontalnom ili samo u vertikalnom smjeru, a njihova zajednička primjena rezultira pronalaženjem apsolutne vrijednosti magnitude gradijenta na svakoj točki i orijentaciji tog gradijenta.

Formula prema kojoj se izračunava magnituda gradijenta je prikazana u izrazu 5.11,

$$|G| = \sqrt{(Gx^2 + Gy^2)} \tag{5.11}$$

međutim najčešće se upotrebljava formula iz izraza 5.12, koja omogućuje brži izračun. [11]

$$|G| = |Gx| + |Gy| \tag{5.12}$$

Kut orijentacije ruba u kojoj se povećava lokalni gradijent računa se putem formule

prikazane u izrazu 5.13.

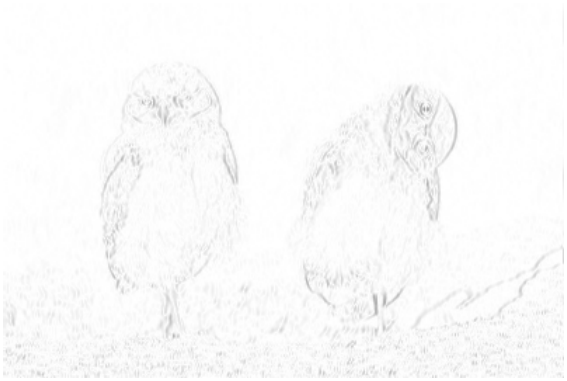
$$\theta = arctan(\frac{Gx}{Gy}) \tag{5.13}$$

5.2.2 Implementacija

Implementacija je ista kao kod Prewitt detektora, razlika je samo u derivacijskoj matrici.

5.2.3 Rezultat implementacije

Na slici 5.3 prikazana je primjena Sobel detektora ruba u smjeru osi x, a na slici 5.4 u smjeru osi y.

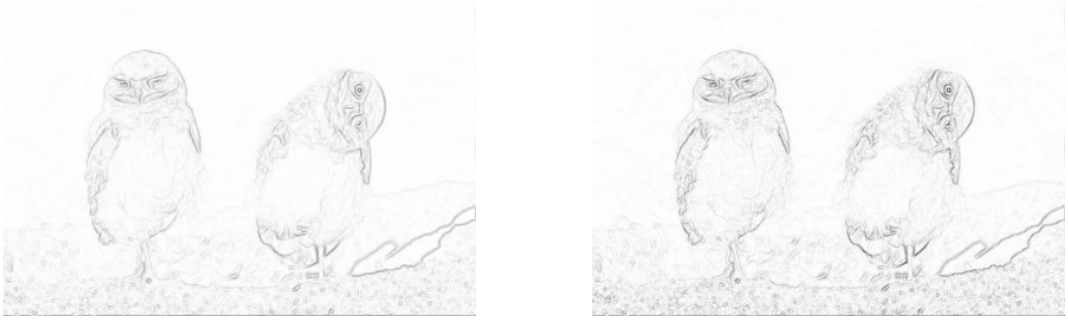


Slika 5.3: Sobel detektor u smjeru osi x

Slika 5.4: Sobel detektor u smjeru osi y

5.2.4 Usporedba Sobel i Prewitt detektora

Kao što je prethodno navedeno, derivacijska maska Prewitt detektora usrednjuje gradijent na području od tri reda, odnosno tri stupca. Kod Sobel detektora radi se razlika između trenutnog reda i ostalih redova, odnosno trenutnog stupca i ostalih stupaca na način da se trenutnom redu ili stupcu dodaje veća "težina". To rezultira većim stupnjem detalja, što se vidi na slici 5.5.



(a) Prewitt detektor *(b) Sobel detektor*

Slika 5.5: Usporedba Prewitt i Sobel detektora

5.3 Robert's cross

5.3.1 Definicija

Operator Robert's cross izvršava jednostavan i brz izračun 2D lokalnih gradijenta slike. Vrijednosti piksela izlazne slike u svakoj točki predstavljaju procjenjenu apsolutne vrijednosti magnitude lokalnog gradijenta ulazne slike u toj točki. [12]

Operator se sastoji od dvije konvolucijske maske veličine 2x2. Maske su maksimalno osjetljive na rubove pod kutem od 45 stupnjeva, svaka od maski za jednu od okomitih smjerova.

$$Gx = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} Gy = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$$

Magnituda gradijenta izračunava se na isti način kao i kod Sobel operatora, a kut povećanja lokalnog gradijenta izračunava se prema formuli u izrazu 5.14.

$$\theta = arctan(\frac{Gx}{Gy}) - \frac{3\pi}{4} \tag{5.14}$$

5.3.2 Implementacija

Funkcija za primjenu Robert's cross operatora ista je kao i funkcija za primjenu Prewitt operatora, čiji je kod prethodno naveden. Što se tiče cijelokupnog koda programa, jedina je razlika u matricama, odnosno konvolucijskim maskama koje se uzimaju u funkciju.

5.3.3 Rezultat implementacije

Na slici 5.6 prikazana je primjena Robert's cross detektora.

Slika 5.6: Slika nakon primjene Robert's cross operatora

5.3.4 Razlike između Sobel operatora i Robert's cross operatora

- Utjecaj smetnji

  Kako Robert's cross operator ima manje dimenzije matrica osjetljiviji je na smetnje, što se vidi iz priloženih slika. Na izvornu sliku punu smetnji Sobel operator ipak uspijeva detektirati rubove, dok se na slici koja je izlazna slika Robert's cross algoritma rubovi veoma slabo ocrtavaju.

Slika 5.7: Izvorna slika sa smetnjama

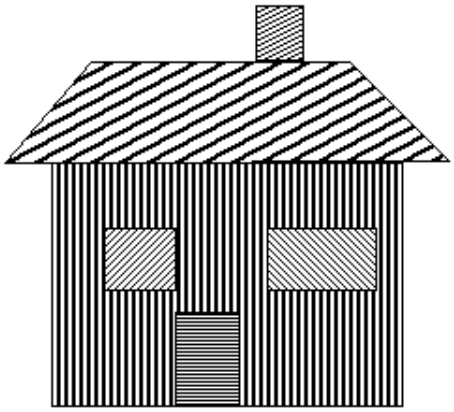
(a) Sobel detektor

(b) Robert's cross detektor

Slika 5.8: Usporedba s obzirom na smetnje

- Detekcija rubova

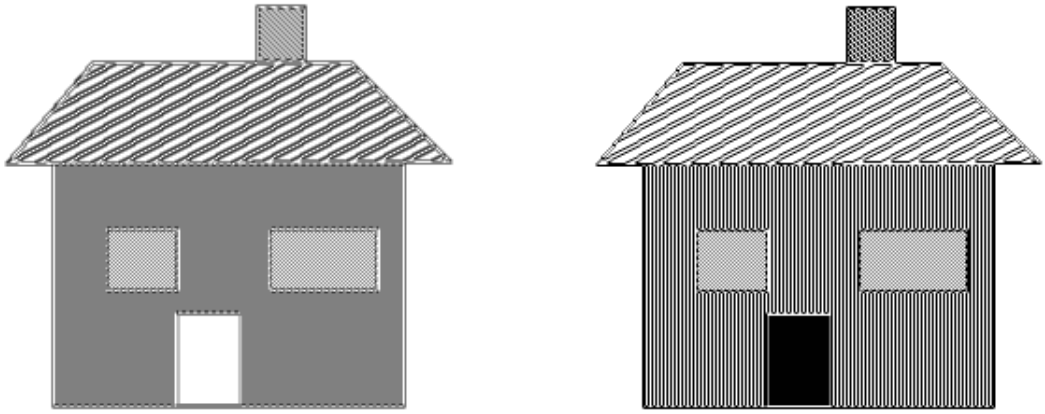
  Sobel operator bolje detektira horizontalne i vertikalne rubove, a Robert's cross bolje detektira dijagonalne rubove.



Slika 5.9: Izvorna slika

- Brzina

  Što se tiče brzine, Robert's cross operator je puno brži od Sobel operatora iz razloga što koristi manje dimenzije matrica. Izvršavanje konvolucije nad prethodno prikazanom izvornom slikom iznosi 98.014 ms kod Robert's cross operatora te 217.897 ms kod



(a) Sobel detektor *(b) Robert's cross detektor*

Slika 5.10: Usporedba s obzirom na smjerove detekcije

Sobel operatora.

5.4 Canny detektor ruba

Canny operator za detekciju rubova sastoji se od 5 koraka [14] :

1. Uglađivanje
2. Pronalaženje gradijenta
3. Suzbijanje vrijednosti koje nisu maksimalne
4. Dupli prag
5. Praćenje rubova putem histereze

5.4.1 Uglađivanje

Prva faza Canny detektora ruba je uglađivanje slike, postupak koji je objašnjen u poglavlju o fazama detekcije. Na slici 5.11 nalazi se izvorna slika, a na slici 5.12 nalazi se slika nakon primjene filtera za uglađivanje slike.

Slika 5.11: Izvorna slika

Slika 5.12: Slika nakon primjene Gausovog filtera

5.4.2 Pronalaženje gradijenta

Pronalaženje gradijenta provodi se pomoću Sobel operatora, čija je implementacija prethodno prikazana. Na slici 5.13 nalazi se izgled fotografije nakon primjene Sobel operatora za detekciju rubova.

Slika 5.13: Fotografija nakon primjene Sobel detektora

5.4.3 Odstranjivanje vrijednosti koje nisu lokalno maksimalne

Svrha ovog koraka je pretvoriti zamućene rubove slike magnituda gradijenta u oštre rubove. To se radi očuvanjem svih lokalnih maksimuma gradijenta slike i brisanjem svega ostalog. [14] Algoritam za svaki piksel sastoji se od sljedećih koraka:

- zaokruživanje orijentacije gradijenta na najbližih 45 stupnjeva s obzirom na osam susjednih komponenata, odnosno piksela
- uspoređivanje jačine ruba trenutnog piksela s jačinom ruba piksela u pozitivnom i negativnom smjeru gradijenta (npr. ako je kut ruba 90, trenutni piksel uspoređuje se s lijevim i desnim pikselom)
- ako je jačina ruba trenutnog piksela najveća, sačuva se vrijednost jačine ruba a u suprotnom se vrijednost piksela odbacuje

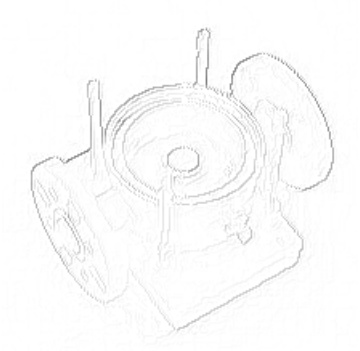
5.4.4 Implementacija

```
int find_angle (float angle) {
  if (angle > 0 && angle <=33.75) return 0;
  else if (angle > 33.75 && angle <=78.75) return 45;
  else if (angle > 78.75 && angle <=123.75) return 90;
  else return 135;
}

void Non_maximum_suppression (Mat &output, Mat& direction) {
  int theta;
  uchar piksel;

    for (int x = 0; x < output.rows; ++x)  {
  for (int y = 0; y < output.cols; ++y) {

    piksel = output.at<uchar>(Point (x, y));


              if (x!=0 && y!=0 && x!=output.rows && y!=output.cols) {
```

```
19      theta = find_angle (direction.at<float>(Point (x, y)));
20
21      if (theta == 0) {
22          if (!(piksel > output.at<uchar>(Point (x-1, y)) && piksel >
    output.at<uchar>(Point (x+1, y))))
23          {
24            output.at<uchar>(Point (x, y)) = 0;
25          }
26      }
27
28      else if (theta == 45) {
29          if (!(piksel > output.at<uchar>(Point (x, y-1)) && piksel >
    output.at<uchar>(Point (x, y+1))))
30          {
31            output.at<uchar>(Point (x, y)) = 0;
32          }
33      }
34
35      else if (theta == 90) {
36          if (!(piksel > output.at<uchar>(Point (x-1, y+1)) && piksel >
    output.at<uchar>(Point (x+1, y-1))))
37          {
38            output.at<uchar>(Point (x, y)) = 0;
39          }
40      }
41
42      else {
43         if (!(piksel > output.at<uchar>(Point (x-1, y-1)) && piksel >
    output.at<uchar>(Point (x+1, y+1))))
44          {
45            output.at<uchar>(Point (x, y)) = 0;
46          }
47      }
48
49
50          }
```

```
51
52
53    }
54        }
55
56 }
```

Na slici 5.14 prikazan je efekt odstranjivanja vrijednosti koje nisu lokalno maksimalne.



Slika 5.14: Fotografija nakon odstranjivanja lokalno ne-maksimalnih vrijednosti

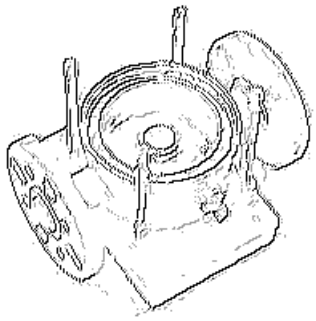
5.4.5 Dvostruki prag

Pikseli rubova koji ostaju nakon otklanjanja vrijednosti koje nisu maksimalne su i dalje označeni s njihovim jačinama. Mnogi od njih će vjerojatno biti pravi rubovi, ali neki od njih su možda smetnje ili varijacije boja, npr. zbog grube površine. Najjednostavniji način za raspoznavanje je korištenje pragova, kako bi se zadržali samo pikseli koji su jači od određene vrijednosti. U Canny operatoru koristi se dvostruki prag. Pikseli koji su jači od visokog praga su označeni kao jaki, pikseli koji su slabiji od niskog praga se odbacuju, a pikseli između visokog i niskog praga su označeni kao slabi.

```
1 Mat Double_threshold ( Mat& output, Mat& strength_of_pixel, int low, int
      high )
2 {
3     for ( int y = 0; y < output.rows; ++y )
4     {
5         for ( int x = 0; x < output.cols; ++x )
```

```
6         {
7
8     //0-not edge, 1-weak, 2-strong
9     if (output.at<uchar>(Point (x, y)) > high)
10    {
11        strength_of_pixel.at<uchar>(Point (x, y)) = 2;
12        output.at<uchar>(Point (x, y)) = 255;
13    }
14
15    else if (output.at<uchar>(Point(x,y)) < high && output.at<uchar>(
      Point (x, y)) > low)
16    {
17      strength_of_pixel.at<uchar>(Point (x, y)) = 1;
18      output.at<uchar>(Point (x, y)) = 100;
19    }
20
21    else {
22      strength_of_pixel.at<uchar>(Point (x, y)) = 0;
23      output.at<uchar>(Point(x, y)) = 0;
24    }
25
26  }
27    }
28
29
30    return output;
31
32 }
```

Slika 5.15 prikazuje izgled fotografije nakon primjenjivanja dvostrukog praga.



Slika 5.15: Fotografija nakon primjene dvostrukog praga

5.4.6 Praćenje rubova putem histereze

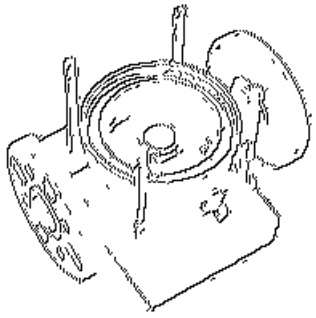
Jaki rubovi su interpretirani kao "sigurni" rubovi i mogu biti direktno uključeni u finalnu sliku rubova, dok su slabi rubovi uključeni samo ako su spojeni s jakim rubovima. Logika koja stoji iza toga je da smetnje i male varijacije vrlo vjerojatno neće rezultirati jakim rubom (s ispravno određenim vrijednostima pragova). Jaki rubovi će (većinom) biti samo pravi rubovi izvorne slike. Slabi rubovi mogu biti ili pravi rubovi ili smetnje, odnosno varijacije boje.

5.4.7 Implementacija

```
Mat Hysteresis ( Mat& output, Mat& strength_of_pixel)
{
    uchar strength;

    for (int x = 0; x < output.rows; ++x) {
      for (int y = 0; y < output.cols; ++y) {

      strength = strength_of_pixel.at<uchar>(Point (x,y));
      if (strength == 1) {
       for (int i = -1; i <= 1 ; i++) {
         for (int j = -1; j <= 1 ; j++) {

          if (strength_of_pixel.at<uchar>(Point (x+i,y+j)) == 2)
          {
```

```
15              output.at<uchar>(Point (x, y)) = 255;
16            }
17            else output.at<uchar>(Point (x, y)) = 0;
18           }
19         }
20
21       }
22     }
23
24
25     return output;
26
27 }
```

Na slici 5.16 prikazana je slika nakon primjene histereze.



Slika 5.16: Finalna slika detekcije rubova

5.5 Laplasijan detektor ruba

Prethodno navedeni detektori rubova za detekciju koriste prvu derivaciju, dok Laplasijan koristi drugu derivaciju. Osim toga, ne detektira rubove u određenom smjeru, već rubove dijeli na vanjske i unutrašnje.[15]

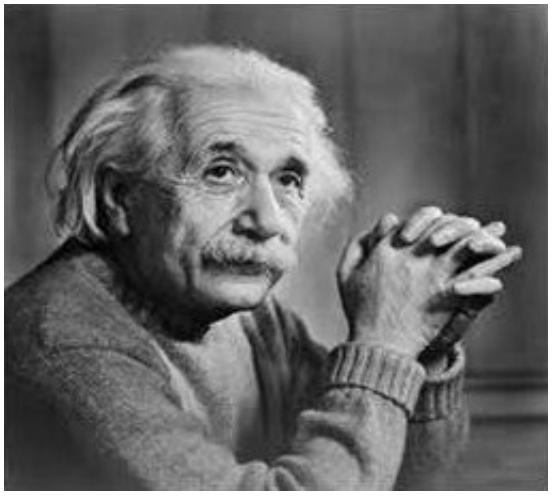
Laplasijan se dijeli na pozitivne i negativne derivacijske maske.

5.5.1 Pozitivni Laplasijan

U pozitivnom Laplasijanu imamo standardnu masku u čijem je središtu negativni element, a ostali su elementi pozitivni.

Pozitivni Laplasijan koristi se za detekciju vanjskih rubova na slici. Na slici 5.17 prikazana je izvorna slika, a na slici 5.18 prikazan je pozitivni Laplasijan.

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$



Slika 5.17: Izvorna slika

Slika 5.18: Slika nakon primjene pozitivnog Laplasijana

5.5.2 Negativni Laplasijan

U negativnom Laplasijanu imamo standardnu masku u čijem je središtu pozitivan element te rubovi moraju biti nula, ali svi ostali elementi unutar maske moraju biti negativni.

Negativni Laplasijan koristi se za detekciju unutarnjih rubova slike. Na slici 5.19 prikazan je negativni Laplasijan.

$$\begin{bmatrix} 0 & 1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

Slika 5.19: Slika nakon primjene negativnog Laplasijana

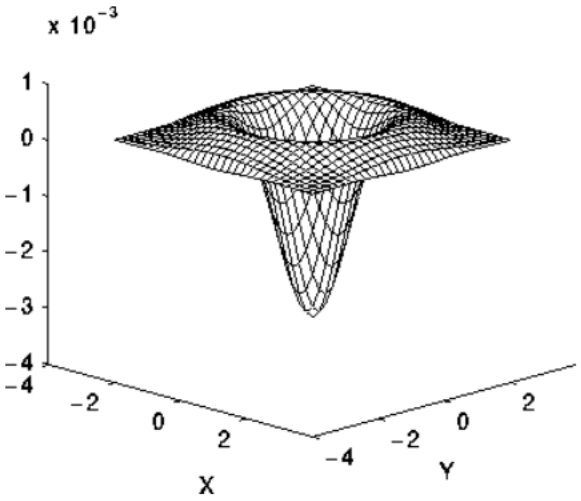
Implementacija Laplasijana je "jeftinija" od implementacije metode gradijenta, iz razloga što ima samo jednu masku. Kako Laplasijan koristi drugu derivaciju, veoma je osjetljiv na smetnje i prije primjene njegove maske slika se uglađuje s Gausovim filterom.

5.6 Gausov Laplasijan

Kako su kerneli Gausa i Laplasijana obično puno manji od slike, ova metoda zahtjeva puno manje aritmetičkih operacija. LoG (*eng.Laplacian of Gaussian*) kernel može biti prethodno izračunat, tako da se u vrijeme izvršavanja programa mora izvršiti samo jedna konvolucija.

2D funkcija Gausovog Laplasijana centrirana u nuli, sa standardnom devijacijom σ prikazana je u izrazu 5.15, a vizualno je prikazana na slici 5.20.

$$LoG(x,y) = -\frac{1}{\pi * \sigma^4}[1 - \frac{x^2+y^2}{2\sigma^2}] \exp -(\frac{x^2+y^2}{2\sigma^2}) \tag{5.15}$$



Slika 5.20: 2D funkcija Gausovog Laplasijana [15]

Gausov Laplasijan može biti aproksimiran kernelom veličine 5x5 poput [16]:

$$\begin{bmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 2 & 1 & 0 \\ 1 & 2 & -16 & 2 & 1 \\ 0 & 1 & 2 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

Na slici 5.21 prikazan je efekt detekcije ruba pomoću LoG detektora.

Slika 5.21: Detekcije rubova pomoću Gausovog Laplasijana

6 Zaključak

Algoritmi za detekciju rubova baziraju se na primjeni derivacijskih matrica. Svaki algoritam za detekciju ima svoju određenu derivacijsku matricu i svaka ima određene prednosti i nedostatke u odnosu na ostale. Smatram da je veoma važno shvatiti na koji način različite matrice utječu na izmjenu slike, ne samo iz razloga da bi znali odabrati detektor ruba koji nam najviše odgovara za određeni problem, već da bi znali i napraviti svoju derivacijsku matricu. Derivacijske matrice moguće je izmjenjivati i prilagođavati te njihovom rotacijom mijenjati smjer u kojem se detektiraju rubovi.

Detekcija rubova podloga je za mnogo drugih, složenijih operacija nad slikom - kao što su segmentacija i prepoznavanje objekata. Bilo mi je zanimljivo pisati ovaj završni rad i nadam se da ću ovo znanje o postupcima detekcije rubova pri obradi digitalne slike nadograditi i nadopuniti s novim znanjima na području obrade digitalne slike.

Literatura

[1] Wikipedia - Kdevelop, URL: https://en.wikipedia.org/wiki/KDevelop

[2] OpenCV, URL: http://opencv.org/about.html

[3] Wikipedia - Gaussian filter, URL: https://en.wikipedia.org/wiki/Gaussian_filter

[4] Gaussian smoothing, URL: http://homepages.inf.ed.ac.uk/rbf/HIPR2/gsmooth.htm

[5] Mubarak Shah: "Fundamentals of computer vision", University of Central Florida, December 7, 1997.

[6] Youtube video - Edge detection in image processing, URL: https://www.youtube.com/watch?v=V2z_x80xP

[7] Prewitt Operator, URL: http://www.tutorialspoint.com/dip/prewitt_operator.htm

[8] Wikipedia - Prewitt, URL: https://en.wikipedia.org/wiki/Prewitt_operator

[9] Sobel Operator, URL: http://www.tutorialspoint.com/dip/sobel_operator.htm

[10] Stackoverflow - Sobel, URL: http://stackoverflow.com/questions/17078131/why-sobel-operator-looks-that-way

[11] Sobel, URL: http://homepages.inf.ed.ac.uk/rbf/HIPR2/sobel.htm

[12] Robert's cross, URL: http://homepages.inf.ed.ac.uk/rbf/HIPR2/roberts.htm

[13] Canny Tutorial, URL: http://www.pages.drexel.edu/ nk752/cannyTut2.html

[14] Raman Maini & Himanshu Aggarwal: "Study and Comparison of Various Image Edge Detection Techniques"

[15] Laplacian Operator, URL: http://www.tutorialspoint.com/dip/laplacian_operator.htm [16

LoG, URL: http://fourier.eng.hmc.edu/e161/lectures/gradient/node8.html

Sažetak

Cilj ovog završnog rada bio je analizirati i usporediti razne algoritme za detekciju rubova te implementirati iste. U ovom završnom radu definirani su i analizirani Prewitt, Sobel, Robert's cross, Canny, Laplasijan i Gausov Laplasijan. Prva tri detektora ruba spadaju u metodu pronalaženja gradijenta i koriste prvu derivaciju za detektiranje rubova. Zadnja dva detektora koriste drugu derivaciju za detektiranje rubova i spadaju u Laplasijan detektore.

Prewitt i Sobel koriste se za detekciju rubova u vertikalnom i horizonatalnom smjeru, a Robert's cross koristi se za detekciju rubova pod kutem od 45 stupnjeva. Ukoliko želimo uz usrednjavanje gradijenta zadržati i što više detalja, koristimo Sobel detektor umjesto Prewitta. Ako radimo sa slikom na kojoj ima puno smetnji, Prewitt je malo bolji od Sobel detektora i puno bolji od Robert's cross detektora. Za Canny možemo reći da je poboljšana verzija nekog od prethodna tri detektora, iako se najčešće kao temelj uzima Sobel. Na temeljni algoritam se dodaju još neke operacije koje omogućavaju precizniju detekciju i smanjivanje debljine rubova. Laplasijan i Gausov Laplasijan bolji su u odnosu na ostale navedene algoritme po pitanju brzine izvršavanja iz razloga što imaju samo jednu konvolucijsku matricu (a ostali imaju dvije).

Ključne riječi - detekcija rubova, derivacijske maske, gradijent

Abstract

The purpse of this final thesis was to analyze and compare various edge detection algorithms and to implement them. Prewitt, Sobel, Robert's cross, Canny, Laplacian and Laplacian of Gaussian edge detectors were defined and analyzed in this thesis. First three are based on gradient method and they are first derivative edge detectors, last two are based on Laplacian method and they are second derivative edge detectors.

Prewitt and Sobel are used for edge detection in vertical and horizontal direction and Robert's cross is used for detecting an edge with angle of 45 degrees. If with averaging gradient magnitude we want to keep the details too, we need to use Sobel detector instead of Prewitt. If we are processing a picture with a lot of noise, Prewitt will be a little bit better then Sobel detector and a lot better than Robert's cross detector. For Canny we can say that it is improved version of some of these three edge detectors, even Sobel is most used as a basis for Canny detector. On that basis algorithm Canny adds a few more operations which add more precision for detecting an edge and also provides edge thinning. Laplacian and Gaussian of Laplacian have only one convolution matrix (and other edge detecion algorithms have two) and that is the reason they are faster than the others.

Keywords - edge detecion, derivative masks, gradient