Elektrotehnički fakultet
Banjaluka

DIGITALNA OBRADA SLIKE

Projektni zadatak:
Softver za obradu slike

Asistent:
mr Vladimir Risojević

Studenti:
Ognjen Joldžić, 21/04
Zlatko Dejanović, 16/04
Daniel Kurtjak, 50/04

1. Zadatak

Glavni cilj projektnog zadatka iz predmeta Digitalna obrada slike je bila implementacija osnovnih algoritama za obradu i manipulaciju slikom. Algoritmi koje je bilo potrebno implementirati su obuhvatali jednostavne geometrijske operacije nad slikama, morfološke operacije, operacije filtriranja, te detekciju zakrivljenja. Spomenuti algoritmi su trebali da podrže rad sa binarnim slikama.

2. Način implementacije i funkcionalnost

U početnoj fazi rada na ovom projektnom radu odlučili smo se za izvjesna proširenja osnovne ideje, te samim tim i funkcionalnosti koje su zahtjevane osnovnom postavkom zadatka. Ovo se u prvom redu odnosi na rad sa slikama u boji, te rad sa grayscale slikama. Međutim, s obzirom da implementirani algoritmi nisu primjenljivi na sve tipove slika, pokušali smo u većini slučajeva programski ograničiti upotrebu programa nad slikama čiji tip nije odgovarajući za konkretnu funkciju (npr. meniji u kojima se nalaze opcije za rad sa grayscale slikama nisu dostupni – disabled – kad se radi sa slikama u boji ili sa binarnim slikama). Dalje, omogućili smo osnovne prikaze histograma i glavne operacije nad histogramom (za sive slike), te prelazak iz jednog moda rada u drugi, gdje je to moguće (moguća konverzija slike u boji u sivu sliku, te dalje u binarnu (obrnut postupak nije moguć, pa samim tim nije ni dozvoljen).

U tehničkoj realizaciji projekta koristili smo programski jezik C# (.NET Framework u verziji 2.0), zbog jednostavne činjenice da smo se tokom dosadašnjeg studiranja dobro upoznali sa njegovim principima, što nam je omogućilo da u kraćem vremenu napravimo korisnički interfejs u odnosu na drugo ponuđeno rješenje. Negativna strana ovakvog pristupa je nešto manje optimizovano izvršavanje jer smo sve funkcije koje su uključivale matematičke transformacije nad matricama morali napisati „od nule“. Ipak, s obzirom da su navedeni problemi izraženi samo u vrlo malom broju funkcija, smatramo da smo postigli dovoljan nivo optimizacije kojim bi se osiguralo nesmetano izvršavanje softvera na današnjem „prosječnom“ računaru.

Sam softver je sastavljen iz niza modula, koji se u finalnom proizvodu distribuiraju u vidu dinamičkih biblioteka (dll datoteka). Zajedničko za sve module je da koriste istu klasu (strukturu) kojom je slika zapisana u memoriji, i koja se i sama nalazi u odvojenom modulu. Klasa Slika ima nekoliko bitnih parametara, koji su prikazani u sljedećem kodu:

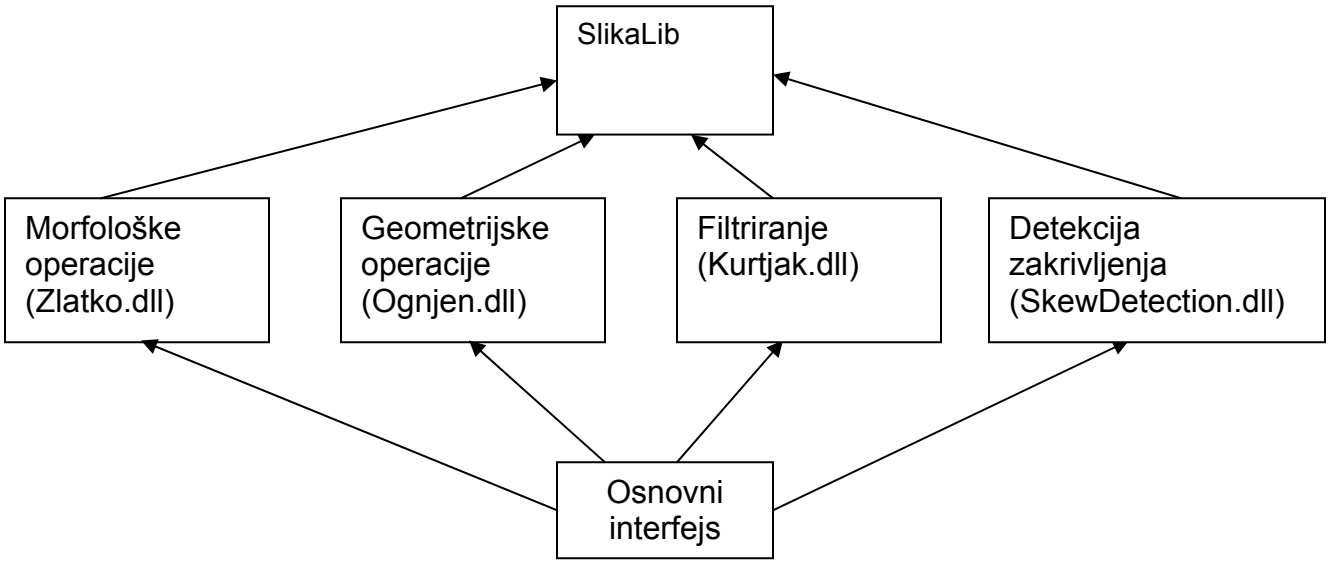
```
public enum imgType { binary, grayScale, rgb };

public float minValue;
public float maxValue;
public float[,] grayBuffer = null;

public Bitmap bmp;
private imgType imageType;
```

```
public int height;
public int width;
```

Objekat bmp tipa Bitmap, tip slike prestavljen enumeracijom imgType, te njene dimenzije su osnovni parametri svake slike i koriste se od strane svakog algoritma. Ostali parametri su vezani za specifične potrebe i implementacije, za koje je potrebno, recimo, sačuvati vrijednosti piksela slike u „buffer-u“ radi brže obrade. U ovom slučaju, podrazumijeva se da se vrijednosti piksela kreću između 0 i 1. Ukoliko to nije slučaj, potrebno je izvršiti normalizaciju.

Radi optimalnijih performansi, unutar ove klase smo iskoristili objekat klase Bitmap (ugrađen u .NET Framework). Ovakav pristup nam je omogućio da pojedinim pikselima slike i dalje pristupamo preko njihovih koordinata, ali da proces iscrtavanja bude znatno brži. Ovo je bitna osobina, pošto je potreba za ponovnim iscrtavanjem objekta mnogo češća od potrebe da se pristupa nekom pikselu unutar slike (naime, prilikom bilo kakve izmjene unutar prozora – promjena veličine, pozicije itd. – operativni sistem generiše zahtjev za ponovnim iscrtavanjem sadržaja, što je u našem slučaju slika). Ako se uzme u obzir da smo omogućili i selekciju nekog regiona u okviru slike, to dodatno komplikuje procese „housekeeping-a“, odnosno održavanja sadržaja prozora. Korištenje objekta klase Bitmap nam je omogućilo da brigu o iscrtavanju prepustimo operativnom sistemu (tačnije .NET Frameworku), gdje je taj postupak riješen mnogo optimalnije od bilo koje druge realizacije do koje bismo mi došli „ručno“.

Modularnost programa se vidi na sljedećem dijagramu.



Prednost ovakvog modularnog sistema je u činjenici da je bilo koji od ovih modula moguće zamijeniti ukoliko se dođe do funkcionalnije i bolje realizacije. Ovakva zamjena neće uticati na funkcionalnost programa. Dodavanje novih modula

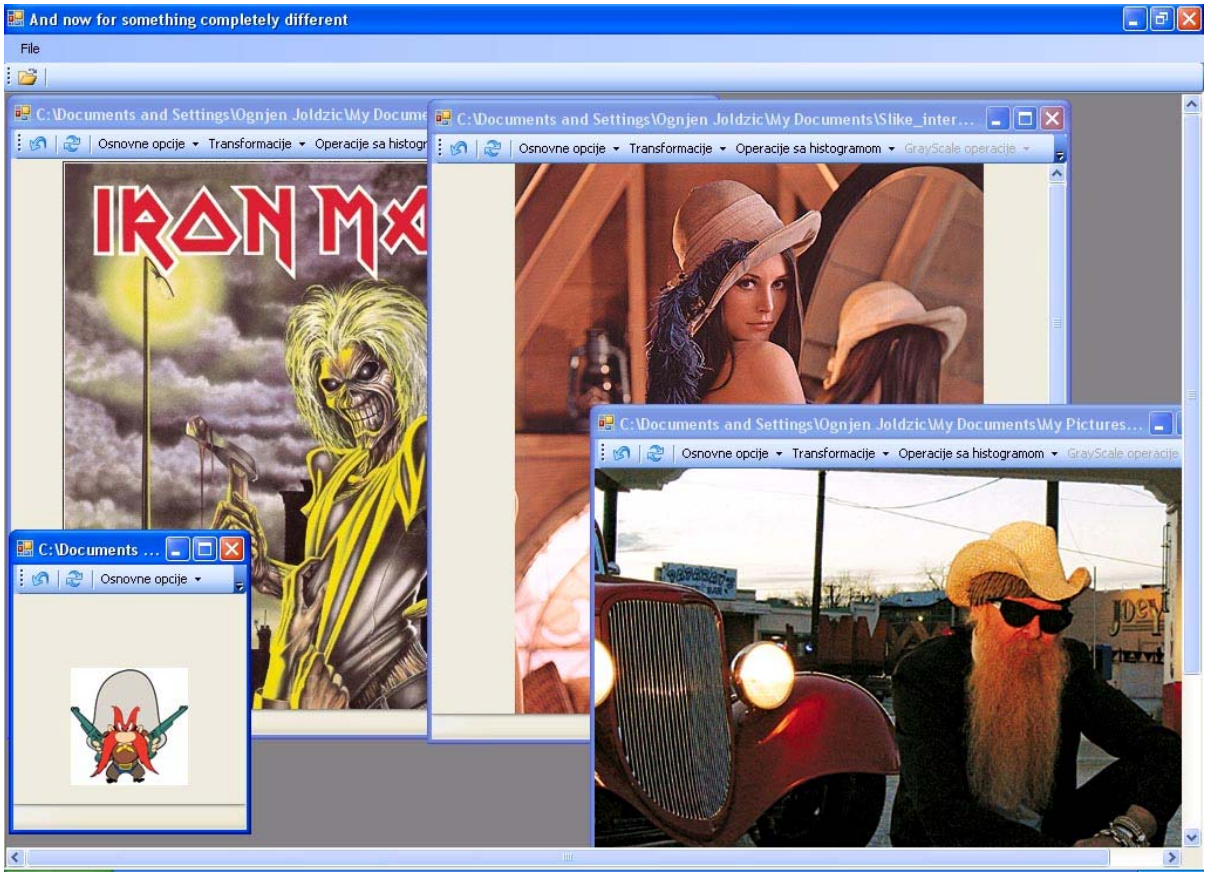
bi još tražilo i odgovarajuću izmjenu interfejsa kojim bi se omogućilo pozivanje date funkcije.

Pojedini moduli sadrže, osim funkcija, i dijaloge ukoliko je potrebno izvršiti unos nekih parametara za rad nekog algoritma. O svakom od modula će biti više riječi od strane članova tima koji su učestvovali u njegovoj realizaciji.

3. Interfejs

Interfejs se sastoji iz dva glavna dijela:

- MDI Parent prozor – ovaj prozor sadrži samo globalne funkcije (otvaranje, zatvaranje i izlaz iz programa). Njegova glavna funkcija je da obezbjedi „kontejner“ za klijentske prozore sa slikama. Jedna instanca programa može imati samo jedan ovakav prozor
- MDI Children – u svakom od ovih prozora se nalazi po jedna slika nad kojom se obrada vrši nezavisno od svih ostalih otvorenih slika. Svaki od njih sadrži kopiju menija za rad sa slikama. Postojanje nekog ovakvog prozora nije uslovljeno brojem drugih otvorenih prozora. Prozori koji sadrže slike nemaju svoj identitet u okviru Windows-ovog Task Managera, već su svi predstavljeni jednim MDI Parent prozorom, čiji okvir ne mogu da napuste. Broj otvorenih slika nije ograničen.



U slučaju da veličina slike prelazi predviđene dimenzije prozora, moguće je horizontalnim i vertikalnim scrollbar-ovima izvršiti pomicanje slike, ili proširiti prozor da veći dio slike bude obuhvaćen vidljivim segmentom.

Osnovni meniji na svakom prozoru sa slikom se sastoje iz sljedećih stavki:

- Undo (poništavanje prethodne akcije)
- osvježavanje
- osnovne opcije
  - sačuvaj izmjene
  - zatvori sliku bez čuvanja
- transformacije (više riječi u pojedinačnim opisima)
- operacije sa histogramom
- Grayscale operacije
- morfologija

Pri čuvanju slika, korisniku smo dali mogućnost izbora između nekoliko najčešće korištenih formata (BMP, JPEG, PNG, GIF, TIFF). Podrška za nabrojane tipove je ugrađena u .NET Framework, tako da se koristi najbolji kvalitet slike koji dati format dopušta.

4. Osnovne operacije

Funkcije iz ove grupe koje su dostupne korisniku su:

a) Crop

Ova funkcija djeluje u sprezi sa ugrađenom mogućnošću selekcije određenog regiona slike. Nakon izvršenja ove komande, selektovani dio originalne slike postaje rezultujuća slika. Programski kod koji je odgovoran za ovu operaciju izgleda ovako:

```
Slika rez = new Slika(sirina, visina, original.ImageType);
        for (int i = ax, m = 0; i < ax + sirina; i++, m++)
            for (int j = ay, n = 0; j < ay + visina; j++, n++)
                if (original.ImageType == imgType.rgb)
                    rez.setPixelRGBAt(m, n, original.getPixelRGBAt(i, j));
                else
                    rez.setPixelAt(m, n, original.getPixelAt(i, j));
           return rez;
```

b) Cut

Ova funkcija je na neki način suprotna funkciji opisanoj u prethodnom odjeljku. Rezultujuću sliku čini onaj region koji nije bio selektovan, dok je selektovani region popunjen bijelom bojom.

c) Negativ

Funkcija će kao rezultat vratiti sliku koja predstavlja negativ originalne slike. Ovdje nije bitno da li je slika u boji, siva ili binarna, jer će negativ biti formiran na osnovu parametra ImageType. Ovaj parametar se dodjeljuje slici prilikom otvaranja, te se ažurira prilikom svake promjene broja boja slike.

```
Slika rez = new Slika(original.Width, original.Height,
original.ImageType);
        for (int i = 0; i < original.Width; i++)
        {
            for (int j = 0; j < original.Height; j++)
            {
                if (original.ImageType == imgType.binary ||
original.ImageType == imgType.grayScale)
                {
                    float pixVal = original.getPixelAt(i, j);
                    rez.setPixelAt(i, j, (1 - pixVal));
                }
                else
                {
                    Color boja = original.getPixelRGBAt(i, j);
                    rez.setPixelRGBAt(i, j, Color.FromArgb(255 - boja.R,
255 - boja.G, 255 - boja.B));
                }
            }
        }
           return rez;
```

d) Rotacija

Rotacija za 90, odnosno za -90 stepeni su dvije varijacije istog algoritma koji svaki piksel originalne slike prenosi u odgovarajući piksel rezultujuće slike, pri čemu svojstva piksela (boja, odnosno intenzitet) ostaju ista.

e) Slika u ogledalu

Popularna funkcija Flip predstavlja zamjenu mjesta piksela sa različitih strana ose koja prolazi vertikalno ili horizontalno preko središnjeg reda ili središnje kolone slike. Zavisno od toga, postoje dvije varijacije ove funkcije: flip horizontal i flip vertical (u odnosu na horizontalnu ili vertikalnu osu).

```
public static Slika flip(Slika original, bool horizontalno)
    {
        Slika rez = new Slika(original.Width, original.Height,
original.ImageType);
        for (int x = 0; x < original.Width; x++)
        {
            for (int y = 0; y < original.Height; y++)
            {
                if (original.ImageType == imgType.binary ||
original.ImageType == imgType.grayScale)
                {
```

```
                    float pixVal = original.getPixelAt(x, y);
                    if (horizontalno) rez.setPixelAt(x, original.Height -
y - 1, pixVal);
                    else rez.setPixelAt(original.Width - x - 1, y,
pixVal);
                }
                else
                {
                    Color boja = original.getPixelRGBAt(x, y);
                    if (horizontalno) rez.setPixelRGBAt(x, original.Height
- y - 1, boja);
                    else rez.setPixelRGBAt(original.Width - x - 1, y,
boja);
                }
            }
        }
        return rez;
      }
```

f) Konverzija

Funkcije za konverziju u binarne, odnosno grayscale slike se moraju eksplicitno pozvati prije određenih funkcija za čije je korištenje potrebno da slike budu tačno odgovarajućeg tipa. Ovakve funkcije su morfološke operacije, funkcije za detekciju iskrivljenja itd. Realizovane su vrlo jednostavnim algoritmima. Pri konverziji u sive slike, pronalazi se srednja vrijednost intenziteta svih kanala (RGB), dok se pri konverziji u binarne slike vrši ispitivanje da li je ova srednja vrijednost veća od 0,5.

5. Matematička morfologija

1. Dilatacija

U slobodnom govoru možemo je nazvati kao proširivanje binarne slike crnim pikselima po odgovarajućem pravilu. To pravilo direktno određuje strukturni element i njegovo zadavanje. Strukturni element može biti proizvoljnog oblika. Naš softver neće dozvoliti izvršavanje nijedne morfološke operacije ako strukturni element nije prethodno zadan. Njegovo zadavanje se vrši opijom *Podesi strukturni element* gdje se prvo definiše broj piksela tog elementa, a zatim i koordinate tih piksela. Npr. osnovni strukturni element zadajemo na sljedeći način – prvo definišemo broj tačaka na 9, a zatim unesemo sljedeće koordinate: (-1,-1), (-1,0), (-1,1), (0,-1), (0,0), (0,1), (1,-1), (1,0), (1,1). Naravno, strukturni element ne mora da sadrži koordinatni početak. Strukturni element je predstavljen kao niz struktura, a svaka struktura predstavlja po jednu tačku i sastoji se od dva *int* polja.

Funkcija koja izvodi dilataciju izgleda ovako:

```
public static Slika dilatacija(Slika slika)
{
     if ((strElement != null) && (strElement.hasElement()))
     {
          Slika copy = slika.getImgCopy();
          Slika s = new Slika(copy.Width, copy.Height, imgType.binary);
             for (int i = 0; i < s.Width; i++)
                for (int j = 0; j < s.Height; j++)
                     s.setPixelAt(i, j, 1);
             for (int i = 0; i < copy.Width; i++)
                for (int j = 0; j < copy.Height; j++)
                   if (copy.getPixelAt(i, j) == 0)
                   {
                     element e = strElement.getFirst();
                     if (!(((e.x + i) < 0) || ((e.x + i) >= copy.Width)
                     || ((e.y + j) < 0) || ((e.y + j) >= copy.Height)))
                       s.setPixelAt((e.x + i), (e.y + j), 0);
                     while (strElement.hasNext())
                     {
                     e = strElement.getNext();
                     if (!(((e.x + i) < 0) || ((e.x + i) >= copy.Width)
                     || ((e.y + j) < 0) || ((e.y + j) >= copy.Height)))
                     s.setPixelAt((e.x + i), (e.y + j), 0);
                     }
                   }
                slika = s;
      }
      else
      {
           MessageBox.Show("Strukturni element je prazan skup! Postavite
strukturni element.", "Greška");
      }
      return slika;
}
```

Prvo se formira slika *s* koja je sastavljena samo od bijelih piksela. Onda prolazimo kroz kopiju glavne slike i zavisno od strukturnog elementa punimo sliku *s* sa crnim pikselima. U tome nam pomažu funkcije getFirst() koja uzima prvi piksel iz strukturnog elementa i hasNext() koja provjerava da li strukturni element ima još piksela i na osnovu nje se vrši terminacija petlje. Dobijeni rezultat smještamo u originalnu sliku i ona se vraća kao rezultat funkcije. U slučaju da strukturni element treba da postavi piksel van granica slike, to se zanemaruje pošto se prvo vrši provjera u okviru *if* izraza.

2. Erozija

Eroziju možemo shvatiti kao eliminisanje rubnih piksela. Sam proces eliminisanja opet direktno zavisi od strukturnog elementa koji se mora prethodno zadati da bi naš softver uopšte vršio eroziju na binarnoj slici.

Funkcija koja izvodi eroziju izgleda ovako:

```
public static Slika erozija(Slika slika)
{
     if ((strElement != null) && (strElement.hasElement()))
     {
          Slika copy = slika.getImgCopy();
          Slika s = new Slika(copy.Width, copy.Height, imgType.binary);
             for (int i = 0; i < s.Width; i++)
                for (int j = 0; j < s.Height; j++)
                   s.setPixelAt(i, j, 1);
             for (int i = 0; i < copy.Width; i++)
                for (int j = 0; j < copy.Height; j++)
                   if (copy.getPixelAt(i, j) == 0)
                    {
                    bool flag = true;
                    element e = strElement.getFirst();
                    if (!(((e.x + i) < 0) || ((e.x + i) >= copy.Width)
                     || ((e.y + j) < 0) || ((e.y + j) >= copy.Height)))
                    if (copy.getPixelAt((e.x + i), (e.y + j)) == 1)
                         flag = false;
                    while (strElement.hasNext())
                    {
                         e = strElement.getNext();
                     if (!(((e.x + i) < 0) || ((e.x + i) >= copy.Width)
                     || ((e.y + j) < 0) || ((e.y + j) >= copy.Height)))
                     if (copy.getPixelAt((e.x + i), (e.y + j)) == 1)
                     {
                          flag = false;
                          break;
                     }
                    }
                    if(flag) s.setPixelAt(i, j, 0);
                   }
       slika = s;
      }
      else
      {
                MessageBox.Show("Strukturni element je prazan skup!
Postavite strukturni element.", "Greška");
            }
     return slika;
}
```

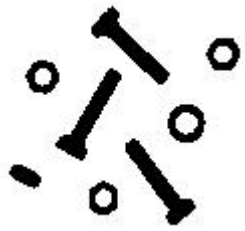
Princip je sličan kao kod dilatacije. Opet se postavlja slika *s* dimenzija kao originalna slika na kojoj su svi pikseli bijeli, pa se ona popunjava crnim pikselima zavisno od strukturnog elementa. Za svaki crni piksel na originalnoj slici provjeravamo sve susjedne piksele koje određuje strukturni element. U slučaju da postoji bar jedan bijeli, *flag* se postavlja na *false* i piksel sa originalne slike se neće pojaviti na slici *s*. Ako su svi “susjedni” pikseli crni, tada će flag ostati setovan na *true* i piksel će se iscrtati na slici *s*. Urađena je analogija sa dilatacijom u pogledu rezultata i piksela koji padaju van dimenzija slike.

3. Otvaranje i zatvaranje

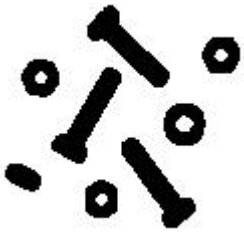
Obe funkcije su vrlo jednostavno implementirane zbog činjenice da otvaranje predstavlja eroziju pa dilataciju iste slike sa istim strukturnim elementom, a zatvaranje dilataciju pa eroziju. Funkcije su implementirane na sljedeći način:

```
public static Slika otvaranje(Slika s)
{
    return dilatacija(erozija(s));
}

public static Slika zatvaranje(Slika s)
{
    return erozija(dilatacija(s));
}
```

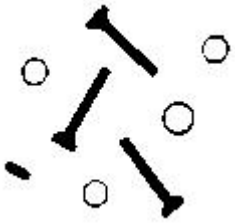
Primjer:



Originalna slika



Dilatacija osnovnim str.el.



Erozija osnovnim str.el.

6. Filtriranje

Implementirana su 4 filtera za uklanjanje šuma i jedan šum. Implementirani filtri su: uniformni, Gausov, median i Laplasov, a implementirani šum je salt & pepper. Gausov šum je ovog puta zaobiđen zbog izuzetno teške implementacije. Obraćena je posebna pažnja na rubne opcije funkcijom *getImgBoundary*. Primjenjena je analogija sa *Matlab*-om što se tiče funkcija *imFilter* i *FSpecial*, koje obavljaju analogne funkcije u našem softveru. Funkcija *FSpecial* je preklopljena 3 puta, zavisno od toga koji se parametri šalju, a koji su podrazumijevani. Prikazaćemo je kad se šalju svi parametri:

```
public static float[,] fSpecial(string type, int width, int height, float
sigma, float alpha)
        {
            float[,] f = new float[width, height];
            if (type.Equals("average"))
            {
                for (int i = 0; i < width; i++)
                    for (int j = 0; j < height; j++)
                        f[i, j] = 1f / (width * height);
            }
            else if (type.Equals("laplacian"))
            {
            alpha = Math.Max(0, Math.Min(alpha, 1));
            f[0, 0]=f[0, 2]=f[2, 0]= f[2, 2]= alpha / (alpha + 1);
            f[0, 1]=f[1, 0]=f[1, 2]= f[2, 1]= (1 - alpha) / (alpha + 1);
                f[1, 1] = -4 / (alpha + 1);
            }
            else if (type.Equals("gaussian"))
            {
                float[,] g = new float[width, height];
                for (int i = 0; i < width; i++)
                    for (int j = 0; j < height; j++)
                    {
                        f[i, j] = (-width / 2) + i;
                        g[i, j] = (-height / 2) + j;
                    }
                if (sigma > 0)
                    for (int i = 0; i < width; i++)
                        for (int j = 0; j < height; j++)
                            f[i, j] = (float)Math.Exp(-(f[i, j] * f[i, j]
+ g[i, j] * g[i, j]) / (2 * sigma * sigma));

                float sum = 0;
                for (int i = 0; i < width; i++)
                    for (int j = 0; j < height; j++)
                        sum += f[i, j];
                if (sum > 0)
                    for (int i = 0; i < width; i++)
                        for (int j = 0; j < height; j++)
                            f[i, j] = f[i, j] / sum;
            }
            Aux.printMatrix(f, width, height);
            return f;
           }
```

Parametar type je jedini obavezan u svakoj verziji ove funkcije i zavisno od njega se poziva odgovarajući filtar. Median filtar je implementiran u posebnoj funkciji, analogno sa *Matlab*-om:

```
public static Slika medfilt2(Slika slika, int sirina, int visina, string
boundaryType)
        {
            slika.makeBuffer();
            Slika copy = slika.getImgCopy();
            for (int i = 0; i < copy.Width; i++)
                for (int j = 0; j < copy.Height; j++)
                {
                    float[,] f = getImgBoundary(copy, sirina, visina, i,
j, boundaryType);
                    for (int ii = 0; ii < sirina * visina - 1; ii++)
                        for (int jj = ii; jj < sirina * visina; jj++)
                            if (f[ii / sirina, ii % sirina] > f[jj /
sirina, jj % sirina])
                            {
                                float tmp = f[ii / sirina, ii % sirina];
                                f[ii / sirina, ii % sirina] = f[jj /
sirina, jj % sirina];
                                f[jj / sirina, jj % sirina] = tmp;
                            }
                    slika.setPixelAt(i, j, f[sirina / 2, visina / 2]);
                }
            return slika;
        }
```

Funkcija koja implementira šum *salt & pepper* izgleda ovako:

```
        public static Slika imNoise(Slika s, string noiseType, float freq)
        {
            if (noiseType.Equals("salt & pepper"))
            {
                int f = (int)(freq * 100);
                Random r = new Random();
                for (int i = 0; i < s.Width; i++)
                    for (int j = 0; j < s.Height; j++)
                    {
                        int rdm = r.Next(0, 100);
                        if (rdm < f / 2)
                            s.setPixelAt(i, j, 0);
                        else if (rdm >= (100 - f / 2))
                            s.setPixelAt(i, j, 1);
                    }
                return s;
            }
            throw new Exception("Nije dobar parametar");
        }
```