Elektrotehnički fakultet
Banja Luka

Seminarski rad iz predmeta Digitalna obrada slike

Obrada slika dokumenata

detekcija iskošenja teksta korištenjem Houghove transformacije i ispravljanje teksta (za binarne slike)

Momić Zlatan, 31/00

Sadržaj

1. Uvod

Aplikacija „Obrada slika digitalizovanih dokumenata“ je u osnovi urađena kao modularna aplikacija koja treba da podrži osnovne funkcionalnosti obrade slika kao i neke od naprednih kao što su uklanjanje šuma korištenjem filtara zasnovanih na matematičkoj morfologiji te detekcija iskošenja i ispravljanje neporavnatog teksta.

Implementacija aplikacije je izvedena u .NET okruženju korištenjem programskog jezika C#. Osnovna ideja koncepta modularnosti je u jednostavnosti proširenja aplikacije te laka mogućnost unaprijeđenja postojećih funkcionalnosti bez potrebe za velikom izmjenom programskog koda. Tako ova aplikacija daje veoma dobru osnovu za daljnje usavršavanje i implementaciju novih funkcionalnosti.

U daljnjem dijelu teksta obratićemo pažnju na implementaciju detekcije iskošenja teksta korištenjem Bairdovog i Houghovog algoritma te implementaciju ispravljanja za detektovani ugao iskošenja korištenjem pomenutih algoritama.

2. Definicija problema

U svakodnevnom životu se susrećemo sa velikim brojem dokumenata koje je potrebno digitalizovati. Razlozi za digitalizaciju su potreba za brzim i sigurnim prenosom dokumenata, poboljšanje i restauracija čitljivosti dokumenata, arhiviranje te brza pretraga već arhiviranih dokumenata, i mnogi drugi.

Korištenje slika dobijenih putem faks aparata zadaje poteškoće kao što su nekvalitetna štampa ili iskošenje teksta uslijed loše uvučenog papira. Dok smo kod skenera u mogućnosti da na lak način dođemo do digitalizovane slike koja je u okviru od tri stepena u odnosu na pravu orijentaciju stranice teksta koju skeniramo kod faks aparata često dolazi do većih iskošenja zbog uvlačenja papira u svrhu štampanja primljene stranice teksta. Šum je takođe velik problem. Često se dešava da se slika šalje hiljadama kilometara preko loših telefonskih parica, a uz to faks aparati koriste standarde za kompresiju slika sa gubicima. Ispravljanjem iskošenja bi se znatno unaprijedila iskoristivost dokumenata prenešenih faks aparatima.

3. Objašnjenje izbora teorije i tehnologije za rješavanje problema

Prilikom korištenja skenera, pažljivi korisnik može postaviti dokument tako da linije teksta budu u granicama tri stepena od prave horizontale. Kada koristimo faks ili njemu sličan aparat nemamo nikakve garancije za tim. Tekst na stranici može ali i ne mora biti dobro poravnat sa ivicom stranice. Tako, jedan od prvih koraka u cilju čitanja slika dobijenih sa faks aparata jeste procjena ugla iskošenja linija teksta. Metoda analize slike čiji izlaz predstavlja ugao iskošenja se naziva detekcija ugla iskošenja. Detekcija ugla iskošenja se može postići na razne načine i ovdje će biti naveden samo jedan od mogućih rješenja.

3.1. Bairdov algoritam

Ako je približno poznat ugao iskošenja, horizontalna projekcija nam može pomoći prilikom nalaženja ugla iskošenja. Projekcija predstavlja sumu vrijednosti piksela u određenom pravcu, pa horizontalna projekcija predstavlja sumu piksela u pojedinom redu. Ako je približno poznat ugao iskošenja, slika se može rotirati za taj iznos. Tada se projekcije računaju za male uglove dok se ne pronađe maksimalna vrijednost horizontalne projekcije odnosno dok visina linije teksta ne bude minimalna ili dok bijeli prostor između linija teksta ne bude maksimalan. Dobijeni ugao je pravi ugao iskošenja.

Za sve karaktere koji nemaju produžetak (svi karakteri osim g, j, p, q i y) važi da su pikseli donjeg dijela karaktera kolinearni za svaku liniju teksta. Tako ako pronađemo okvir svakog karaktera, srednji pikseli donje ivice većine okvira bi trebali biti kolinearni za svaku liniju teksta.

Polazeći od ove spoznaje dolazimo do sljedećeg algoritma (Baird 1987):

- Identifikovati sve povezane regione, podrazumijevati da svaki predstavlja karakter osim ako ne prelazi određenu maksimalnu veličinu,
- Pronaći okvir svakog regiona i locirati centralni piksel donje ivice okvira,
- Za dati ugao θ izračunati projekciju tačaka dobijenih u prethodnom koraku, čime dobijamo jednodimenzionalni projekcijski niz $P(\theta)$. Baird koristi takozvane „bin“-ove, odnosno prozore određene veličine, pa projekcija sada predstavlja sumu vrijednosti „bin“-ova $P_i(\theta)$ pronađenih za ugao θ. Vrijednost „bin“-a predstavlja sumu vrijednosti piksela unutar njega.
- Pronaći maksimum funkcije projekcije:

$$A(\theta) = \sum_{i=1}^{n} P_i(\theta)^2$$

gdje je n broj „bin“-ova. Ugao koji daje maksimum je korektni ugao iskošenja.

Procjena veličine ugla iskošenja je korisna jer redukuje vrijeme potrebno za pretragu za pronalazak maksimuma. Ako ugao iskošenja nije približno poznat može se raditi detaljna pretraga ali je vremenski prezahtjevna.

3.2. Houghov algoritam

Umjesto određivanja ugla iskošenja korištenjem projekcija i određivanjem maksimuma probajmo nešto drugačiji pristup.
Houghova transformacija (Hough 1962) je metod za detekciju pravih linija u rasterskoj slici. Kroz svaki crni piksel slici možemo provući beskonačno mnogo pravih linija, jedna za svaki mogući ugao. Svaka od ovih linija se može prikazati u eksplicitnoj formi:

$$Y = mX + b$$

gdje su koordinate piksela kroz koji pravac prolazi (X, Y). Naklon pravca je m, a presjek sa y osom je b.

Ako ovu jednačinu posmatramo drugačije, tako da su X i Y konstante a m i b koordinate, jednačina se može prikazati u obliku:

$$b = -Xm + Y$$

što predstavlja pravac u (m, b) prostoru. Tako, jedna tačka u prostornom domenu slike (X, Y) odgovara pravcu u (m, b) koordinatama.

Svaki piksel u prostornom domenu slike u (m, b) koordinatnom prostoru odgovra pravcu. Još bitnije je primjetiti da mjesta u (m, b) prostoru, koga još zovemo Houghov prostorni domen, u kojima se dvije linije sijeku odgovaraju kolinearnim pikselima u originalnom prostornom domenu slike. Ovo i nije naročito korisno jer su bilo koja dva piksela kolinearna, ali isto važi i za višestruke presjeke.

Ovo vodi do sljedećeg zaključka:

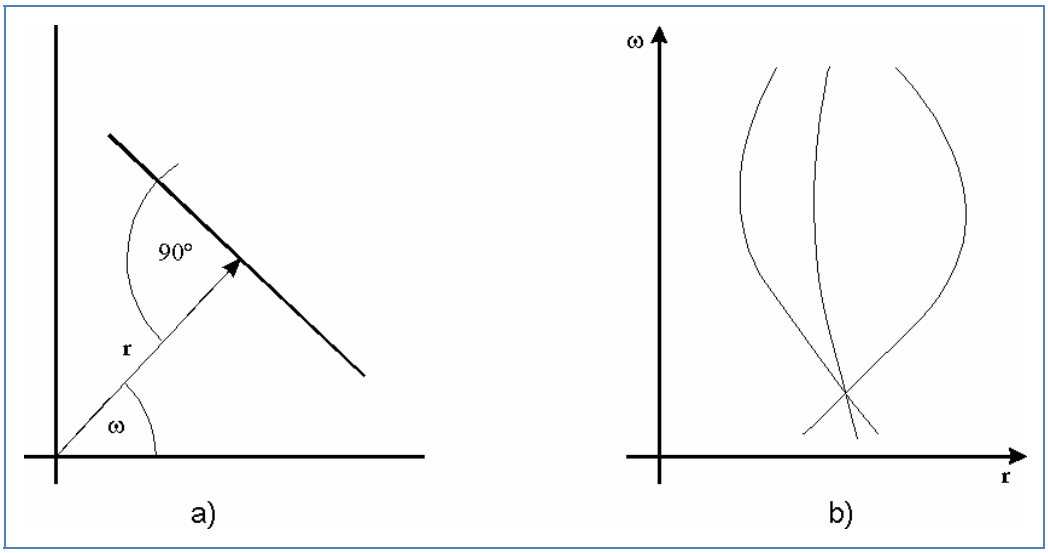
Ako se N pravaca u Houghovom prostoru, koji odgovaraju pikselima u prostornom domenu slike, kojih takođe ima N, sijeku u jednoj tački onda tih N piksela leže na istom pravcu. Parametri m i b odgovaraju parametrima tačke presjeka (m, b) u Houghovom domenu.

Ovo je osnova Houghove transformacije. Svi pikseli se konvertuju u linije u (m, b) prostor, a onda se tačke presjeka najviše linija označavaju. Pošto u stvarnosti na pravcu postoji beskonačno mnogo tačaka, implementacija je voma slična implementaciji histograma. Stepen kvantizacije u (m, b) koordinatama se odlučuje unaprijed i Houghova slika se kreira. Za svaki piksel u originalnoj slici, linija u Houghovom domenu se izračunava i svaki piksel na toj liniji u Houghovom prostoru se inkrementira. Nakon što se procedura obavi za sve piksele, pikseli u Houghovom domenu koji imaju najveće vrijednosti odgovaraju najvećem broju kolinearnih piksela originalne slike.

Korištenje prethodno navedenog ekslicitnog oblika reprezentacije prave dovodi do problema kada se radi sa vertikalnim linijama. Tada naklon pravca m postaje beskonačan. Postoje druge matematičke reprezentacije pravca sa kojim možemo izbjeći navedeni problem:

$$r = x\cos(\omega) + y\sin(\omega)$$

gdje je r ortogonalna udaljenost od koordinatnog početka do tog pravca, a ω je ugao koji zaklapa ortogonalni pravac najkraće udaljenosti r sa x osom. Sada su koordinate Houghovog prostornog domena (r, ω).



Slika 1 – a) Normalna forma, način predstavljanja pravca za ugao ω,
b) (r, ω) Houghova transformacija tri kolinearna piksela

Veza između Houghove transformacije i detekcije iskošenja bi trebala da bude jasna. Prvi koraci Bairdovog algoritma za detekciju iskošenja daju sliku sa velikim brojem grupa kolinearnih piksela. Houghova slika od ovoga bi trebala imati vršnu vrijednost piksela u presječnim tačkama i to baš za ugao ω koji odgovara uglu iskošenja.

4. Detalji rješenja na nivou struktura podataka i algoritma

Modul aplikacije koji implementira prethodno obrađenu teoretsku pozadinu koristeći .NET razvojno okruženje se nalazi u okviru SkewDetection dll fajla. U okviru tog modula postoje tri osnovna dijela koda složena u tri osnovne klase:

- Baird.cs
- Hough.cs
- HoughTransform.cs

Klasa HoughTransform.cs je glavna klasa koja sadrži sljedeće public funkcije:

- `public static Slika HoughSpaceImage(Slika original)`
  - vraća sliku u Houghovom domenu
- `public static Slika BairdImage(Slika original)`
  - vraća sliku nakon primjene Bairdovog algoritma
- `public static Slika rotateSkewedImage(Slika sOriginal)`
  - vraća ispravljenu sliku za detektovani ugao iskošenja
- `public static Slika rotateImage(Slika original, float angle)`
  - vraća rotiranu sliku za određeni ugao angle

Ove funkcije će se pozivati iz glavnog programa u cilju ispravaljanja iskošenja, rotiranja i demonstracije rada Houghove transformacije.

Klase Baird.cs i Hough.cs predstavljaju jezgro i implementaciju teoretskog, prethodno opisanog, rješenja.

Klasa Baird.cs sadrži sljedeće:

- `public Baird(Slika sOriginal)`
  - proslijeđivanjem originalne slike ovom konstruktoru originalna slika prolazi kroz korake identifikacije povezanuh regiona, pronalaženja okvira svakog regiona i lociranje centralnih piksela donje ivice okvira.
- `public Slika BairdImage`
  - nakon pozivanja konstruktora rezultat Bairdovog algoritma u obliku slike možemo pročitati iz ovog property-a

Klasa Hough.cs sadrži sljedeće:

- `public Hough(Slika original)`
  - proslijeđivanjem originalne slike ovom konstruktoru postavlja se opseg ugla ω na 180 a zatim se izvršava funkcija Transform().
- `public void Transform(Slika original)`
  - izvršiće se analiza slike i izračunati ugao iskošenja za prethodno postavljen opseg i kvantizaciju ugla ω.
- `public int Omega`
  - postavljanje opsega ugla ω.
- `public int RMax`
  - maksimalna ortogonalna udaljenost koordinatnog početka do pravca
- `public int TMax`
  - ugao iskošenja tj. ugao ω za koju pikseli slike u Houghovom domenu imaju vršnu vrijednost
- `public float TMVal`
  - vršna vrijednost piksela u Houghovom domenu gdje se desio najveći broj presjeka

5. Detalji rješenja na nivou implementacije

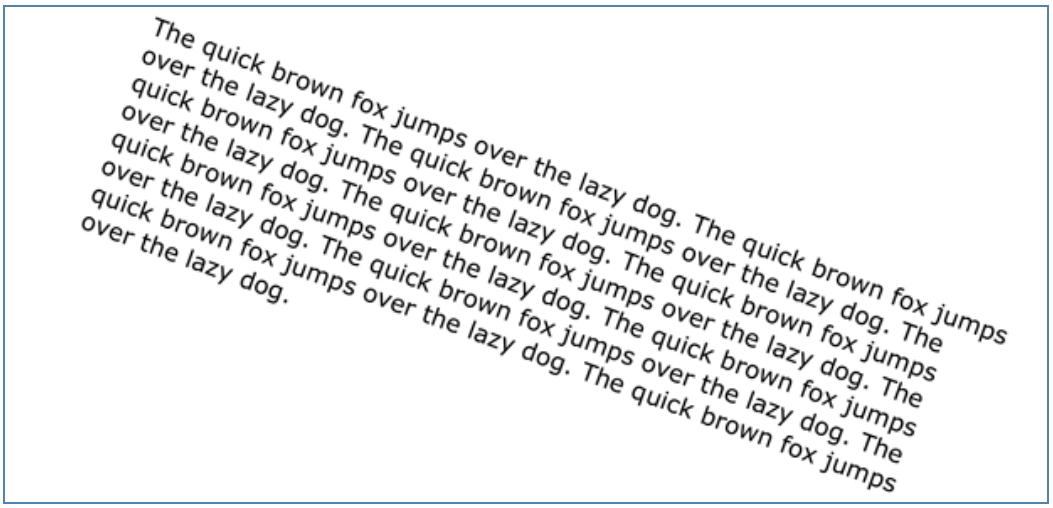
Ovdje će biti navedena detaljnije sama implementacija pojedinih važnijih funkcija i prethodno opisanih algoritama.

```
/* ------------------------------------------------------------------ */
//Implementacija konstruktora Baird klase
/* ------------------------------------------------------------------ */
public Baird(Slika sOriginal)
{
   objekat = 0;
   pozadina = 1;
   MARK = 2;
   velicina = 20;

   original = sOriginal;
   nc = original.Width;
   nr = original.Height;

   dataOriginal = new float[nc, nr];
   for (int i = 0; i < nc; i++)
      for (int j = 0; j < nr; j++)
         dataOriginal[i, j] = original.getPixelAt(i,j);

   bairdImage = new Slika(nc, nr, original.ImageType);

   for (int i = 0; i < nc; i++)
      for (int j = 0; j < nr; j++)
         bairdImage.setPixelAt(i, j, 0);

   markcc();

   for (int i = 0; i < nc; i++)
      for (int j = 0; j < nr; j++)
         if (bairdImage.getPixelAt(i, j) > 0)
            bairdImage.setPixelAt(i, j, 1);
}

/* ------------------------------------------------------------------ */
//Implementacija identifikacije povezanih regiona, pronalaženja okvira
//svakog regiona i lociranje centralnog piksela donje ivice regiona
/* ------------------------------------------------------------------ */
private void markcc()
{
   for (int i=0; i < nc; i++)
   {
      for (int j=0; j < nr; j++)
         if (dataOriginal[i, j] == objekat)
         {
            mark (i, j);
            bbox (i, j);
            bairdImage.setPixelAt((lrc + ulc) / 2, lrr, 1);
            unmark(ulc, ulr, lrc, lrr);
         }
    }
}
```

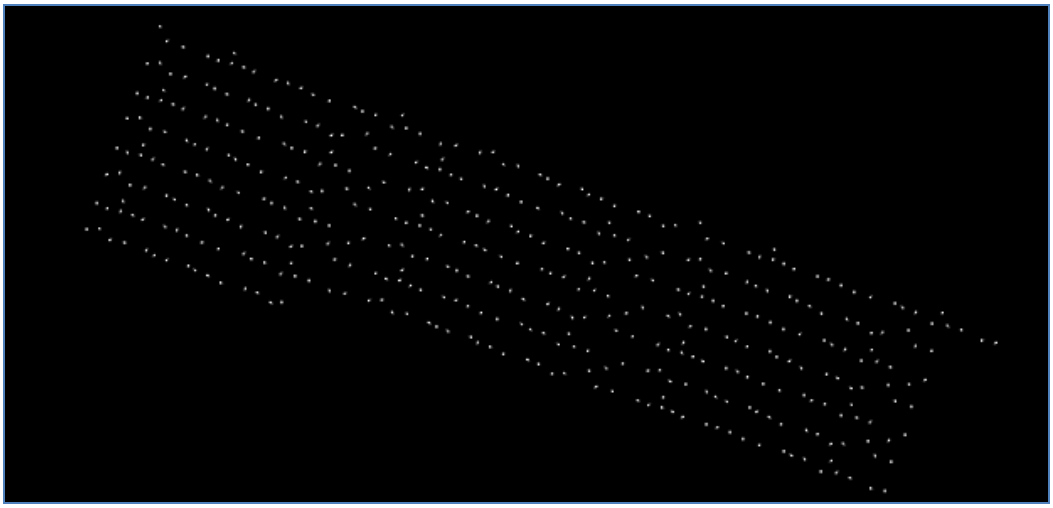
```
/* ------------------------------------------------------------------ */
//Implementacija konstruktora Hough klase
/* ------------------------------------------------------------------ */
public Hough(Slika original)
{
   omega = 180;
   Transform(original);
}


/* ------------------------------------------------------------------ */
//Implementacija Houghove transformacije
/* ------------------------------------------------------------------ */
public void Transform(Slika original)
{
    int centerX, centerY, r, w, i, j, nc, nr;
    long w2, h2;
    double conv = Math.PI / 180.0;

    nc = original.Width; //broj kolona
    nr = original.Height; //broj redova

    //pronađi centar slike (koordinatni početak)
    centerX = nc / 2;
    centerY = nr / 2;

    w2 = nc * nc;
    h2 = nr * nr;

    //pronađi maksimalnu ortogonalnu udaljenost od koordinatnog početka
    //do pravca
    rmax = (int)(Math.Sqrt((double)(w2 + h2)) / 2.0);

    //kreiranje slike za Hough domen - izaberite svoju kvantizaciju
    data = new float[omega, 2 * rmax + 1];

    //postavljanje vrijednosti piksela Hough slike na nula
    for (r = 0; r < 2 * rmax + 1; r++)
       for (w = 0; w < omega; w++)
          data[w, r] = 0;

    //postavljanje ugla iskošenja i maksimalne vrijednosti piksela
    //Hough slike na nula
    tmax = 0; tmval = 0;

    //Za svaki piksel u originalnoj slici, izračunava se linija u Hough
    //prostoru, i svaki piksel Hough slike na toj liniji se inkrementira
    for (i = 0; i < nc; i++)
       for (j = 0; j < nr; j++)
       {
          float pixVal = original.getPixelAt(i, j);
          if (pixVal > 0)
             for (w = 0; w < omega; w++)
             {
                 r = (int)((i - centerX) * Math.Cos((double)(w * conv))
                         + (j - centerY) * Math.Sin((double)(w * conv))
                           );
                 data[w, rmax + r] += 1;
              }
        }
```

```
    //Nakon što su svi pikseli obrađeni, pikseli u Hough slici
    //koji imaju najveće vrijednosti odgovaraju najvećem broju
    //kolinearnih piksela u originalnoj slici.
    for (i = 0; i < omega; i++)
       for (j = 0; j < 2 * rmax + 1; j++)
          if (data[i, j] > tmval)
          {
             tmval = data[i, j]; //maksimalna vrijednost piksela Hough
                                 //slike
             tmax = i; //maksimalni ugao theta tj. ugao iskošenja
          }
}

/* ------------------------------------------------------------------ */
//Implementacija funkcije ispravljanja iskošenja kroz pozive
//gore navednih funkcija
/* ------------------------------------------------------------------ */
public static Slika rotateSkewedImage(Slika sOriginal)
{
   Slika original = sOriginal.getImgCopy();
   Baird baird = new Baird(original);
   Hough hough = new Hough(baird.BairdImage);
   float angle = -(hough.TMax-90);
   return rotateImage(sOriginal, angle);
}
```

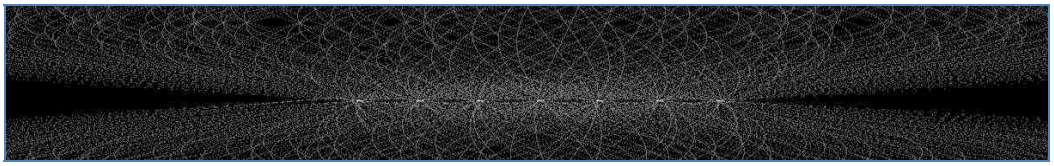
6. Demonstracija funkcionalnosti



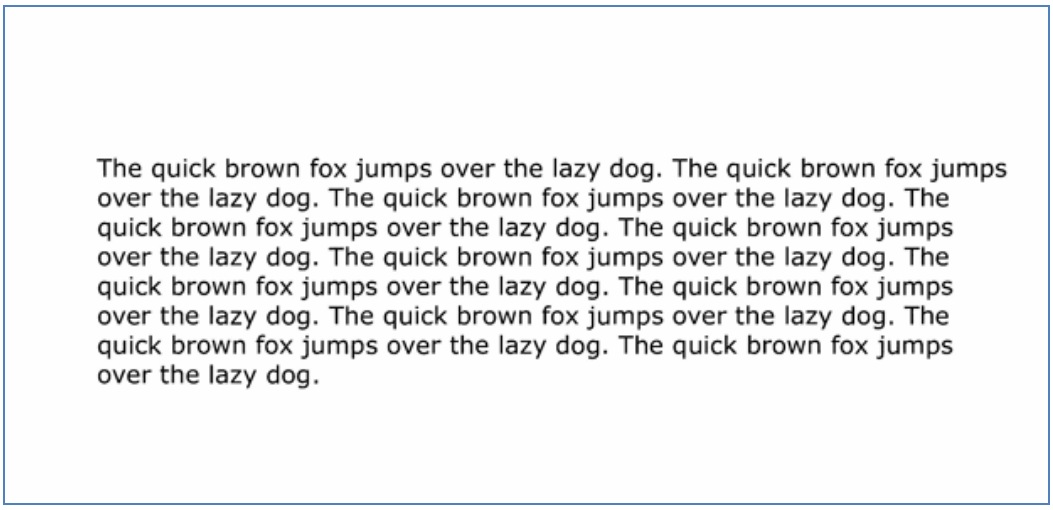
Slika 2 – Slika sa iskošenjem od 20°



Slika 3 – Nakon primjene Bairdovog algoritma



Slika 4 – Houghova transformacija i detekcija vršne vrijednosti



Slika 5 – Slika nakon rotacije za detektovani ugao iskošenja

7. Diskusija i prijedlozi daljeg poboljšanja

Algoritam iako dobro odrađuje posao detekcije i ispravljanja iskošenja bi mogao biti poboljšan.

Povećanjem stepena kvantizacije koordinate ω Hougovog prostora, bi povećali rezoluciju detekcije iskošenja ispod cijelog stepena što bi rezultovalo mnogo ljepšim ispravljanjem iskošenja kod slika koje su iskošene za uglove manje od jednog stepena.

Optimizacijom koda detekcije iskošenja bi se moglo poraditi na brzini izvođenja same aplikacije npr. poboljšanjem načina izdvajanja povezanih regiona.

Rotacija slike se može implementirati operacijom matričnog množenja matrice slike i matrice transformacije čime bi se izvela potpuno nezavisna implementacija od funkcija rotiranja .NET razvojnog okruženja.

8. Literatura

- Algorithms for image processing and computer vision
  J.R. Parker

- Digital Image Processing
  Rafael C. Gonzalez, Richard E. Woods

- Skripta u okviru predmeta digitalna obrada slike
  Prof. dr Zdenka Babić, Asistent: mr Vladimir Risojević
  http://dsp.etfbl.net/dip/

- Rotacija slike u C# programskom jeziku
  http://blog.paranoidferret.com/index.php/2007/06/13/csharp-tutorial-image-editing-rotate/

- MSDN Library – Microsoft developer network
  http://msdn.microsoft.com/