GÖRÜNTÜ İŞLEME - (8.Hafta)

RESMİ ALT BÖLGELERE AYIRMA

BAĞLANTILI BİLEŞEN ETİKETLEME (Çift Geçiş Metodu)

Bir resim üzerindeki aynı renk koduna sahip bölgelerin ortaya çıkarılması, birbirinden ayrılması o resim üzerindeki alan özelliklerini tespit edilmesini sağlayacaktır. Tespit edilen her bölge, yada grup aynı renge boyanabilir yada aynı numara ile etiketlenebilir. Böylece sonraki işlemlerde bu alanlar üzerinde gerekli tanımlama işlemleri yapılabilir.

Bu amaçla "bağlantılı bileşen etiketleme" algoritması kullanılabilir. Bu algoritmanın bir çok farklı yöntemleri olsa da burada kolay olan Çift-Geçiş metodu incelenecektir. Bu amaçla komşuluk ilişkisi olarak ele alınan pikselin çevresindeki 8 piksel (3x3 matris şeklinde) yada 4 piksel (+ şeklinde, köşeler yok) incelemeye alınabilir. Siyah ve beyaz iki renkten oluşan bir resim için algoritmayı şu şekilde tanımlayabiliriz.

---------- 000 ----------

Tüm beyaz piksellere atanan değerler değişmeyene kadar dön

{

Eğer ele alınan piksel rengi (ortadaki piksel) beyaz ise burayı işlet (siyah ise pas geçecektir)

{

- Ortadaki pikselin tüm komuşlarına bak (8 adet yada 4 adet komşu, hangisi uygulanıyorsa)

- Tüm komşular siyah ise ve bu piksele etiket atanmamışsa bu yeni bir pikseldir. Piksele yeni bir etiket ata. Diğer piksele geç. (Yani etraf ada şeklinde her taraf deniz, ortadaki beyaz alanda etikette yoksa yeni bir etiket atanır (aşağıdaki resimde bu örnek gözükmüyor)).

- Komşulardan en az biri beyaz ise ve etiket atanmamışsa, o zaman kendine yeni bir etiket numarası ata. (Aşağıdaki ilk resimde sarının etrafında 4 tane beyaz var ve hiç birinde etiket yok. O zaman ilk numara olan 1 bir sonraki resimde atanmış oldu)

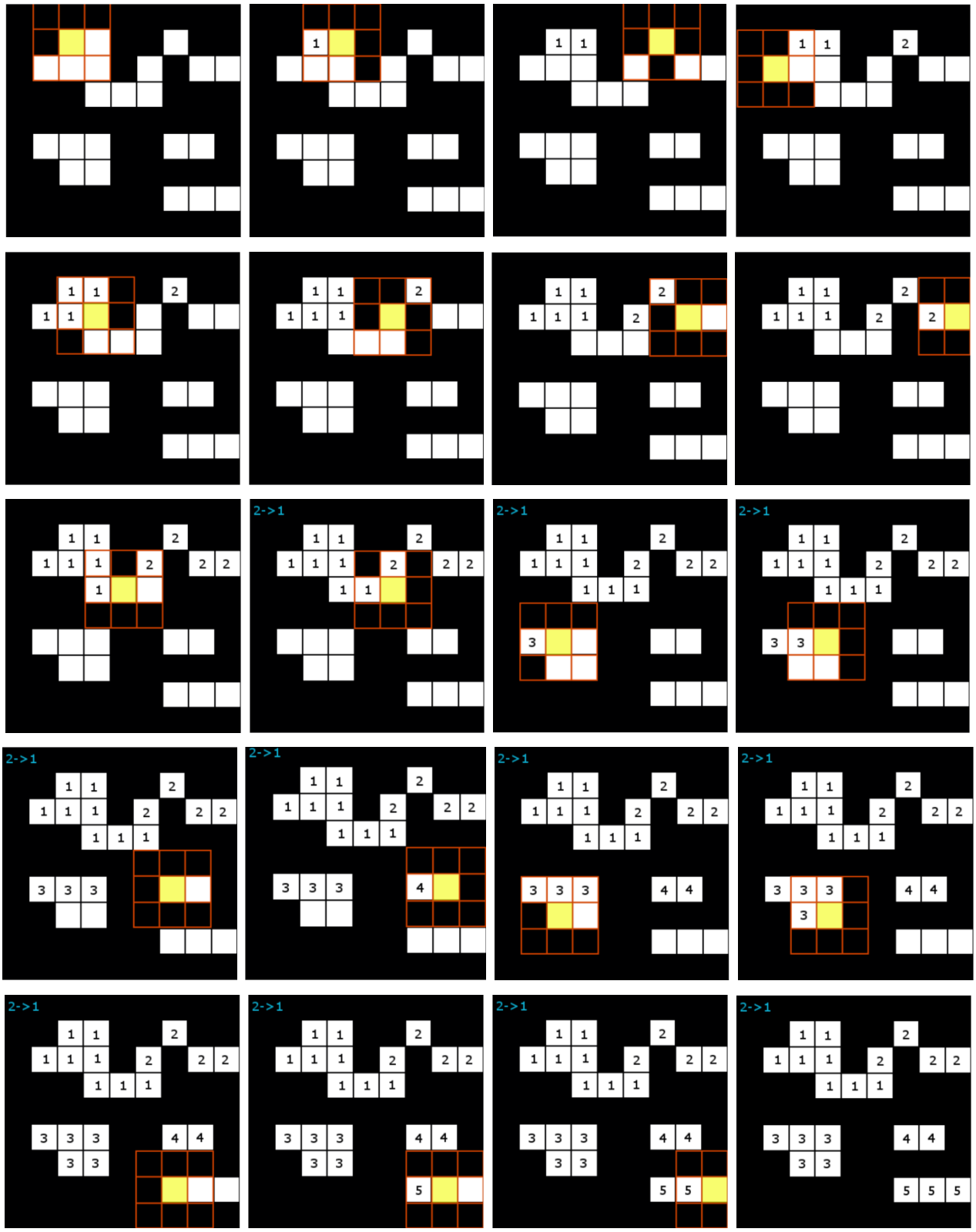
- Kendisi etiketsiz ve komşulardan en az biri etiketli ise (etiketli olanlar beyaz renklidir) bu yeni piksele (ortadaki piksele) çevredeki en küçük değere sahip etiketi ata. (Aşağıdaki 2. resimde sarının etrafında numara olan en küçük hücre 1 dir. O zaman en küçük olan bu değer kendisine atanacaktır. 3. Resimde bu göserilmiş oluyor.

- Kendisi etiketli ise ve komşuları da etiketli ise, kendisine en düşük komşunun değerini ata. Yani her durumda komşuların en küçük değeri atanıyor. Kendisinin etiketli olabilmesi için resim bir defa baştan sona taranmış olmalı (ilk geçiş gerçekleşmiş olmalı). Aşağıdaki resimler ilk geçişi göstermektedir (sarının her seferinde ortası boş).

}

}

---------- 000 ----------



Burada verilen örnekler ilk geçişi göstermektedir. Dikkat edilirse üst sağ köşedeki piksellerde 1 ve 2 rakamları komşuluk olarak yanyana gelmiştir. İkinci bir geçiş daha yapılacak olursa buradaki piksellerden 2 rakamları 1 rakamına dönüşecektir ve böylece üstteki birbirine komşu pikseller aynı numara ile etiketlenmiş olacaktır. Aynı numaralı etiketler farklı renklere boyanabilir yada etiketleme yapmak yerine yeni renk kodu atanabilir. Böylece işlem sonunda bölgeler birbirinden net olarak ayrılmış olacaktır.

Program (Bağımsız alanları tespit etme)

a) Orjinal resim

b)Beyaz kenarlar ortaya çıksın diye negatifi alındı

c) 200 eşik değeri ile siyah beyaz yapıldı

d) Her yaprak alanı ayrı ayrı tespit edildi.

```
//RESMİ BÖLGELERE AYIRMA*****************************
private void bolgelereAyirmaToolStripMenuItem_Click(object sender, EventArgs e)
{
    Bitmap GirisResmi, CikisResmi;

    int KomsularinEnKucukEtiketDegeri = 0;

    GirisResmi = new Bitmap(pictureBox1.Image);
    int ResimGenisligi = GirisResmi.Width;
    int ResimYuksekligi = GirisResmi.Height;
    int PikselSayisi = ResimGenisligi * ResimYuksekligi;

    GirisResmi = ResmiGriTonaDonustur(GirisResmi); //Resmi önce gri tona dönüştürüyor.
    GirisResmi = ResmiEsiklemeYap(GirisResmi); //Resmi 128 ile eşikleme siyah beyaz yapıyor.
    pictureBox1.Image = GirisResmi; //Resmin son halini gösteriyor.

    CikisResmi = new Bitmap(ResimGenisligi, ResimYuksekligi);
```

```
        int x, y, i, j, EtiketNo = 0;

        int[,] EtiketNumarasi = new int[ResimGenisligi, ResimYuksekligi]; //Resmin her pikselinin etiket numarası tutulacak.

        //Tüm piksellerin Etiket numarasını başlangıçta 0 olarak atayacak. Siyah ve beyaz farketmez. Zaten ileride beyaz olanlara numara verilecek.
        for (x = 0; x < ResimGenisligi; x++)
        {
            for (y = 0; y < ResimYuksekligi; y++)
            {
                EtiketNumarasi[x, y] = 0;
            }
        }

        int IlkDeger = 0, SonDeger = 0;
        bool DegisimVar = false; //Etiket numaralarında değişim olmayana kadar dönmesi için sonsuz döngüyü kontrol edecek.

        do //etiket numaralarında değişim kalmayana kadar dönecek.
        {
            DegisimVar = false;
            //------------------------ Resmi tarıyor ----------------------------
            for (y = 1; y < ResimYuksekligi - 1; y++) //Resmin 1 piksel içerisinden başlayıp, bitirecek. Çünkü çekirdek şablon en dış kenardan başlamalı.
            {
                for (x = 1; x < ResimGenisligi - 1; x++)
                {
                    //Resim siyah beyaz olduğu için tek kanala bakmak yeterli olacak. Sıradaki piksel beyaz ise işlem yap. Beyaz olduğu 255 yerine 128 kullanarak yapıldı.
                    if (GirisResmi.GetPixel(x, y).R > 128)
                    {

                        //işlem öncesi ele alınan pikselin etiket değerini okuyacak. İşlemler bittikten sonra bu değer değişirse, sonsuz döngü için işlem yapılmış demektir.
                        IlkDeger = EtiketNumarasi[x, y];

                        //Komşular arasında en küçük etiket numarasını bulacak.
                        KomsularinEnKucukEtiketDegeri = 0;
                        for (j = -1; j <= 1; j++) //Çekirdek şablon 3x3 lük bir matris. Dolayısı ile x,y nin -1 den başlayıp +1 ne kadar yer kaplar.
                        {
                            for (i = -1; i <= 1; i++)
                            {

                                if (EtiketNumarasi[x + i, y + j] != 0 && KomsularinEnKucukEtiketDegeri == 0)  //hücrenin etiketi varsa ve daha hiç en küçük atanmadı ise ilk okuduğu bu değeri en küçük olarak atayacak.
                                {
                                    KomsularinEnKucukEtiketDegeri = EtiketNumarasi[x + i, y + j];
                                }
                                else if (EtiketNumarasi[x + i, y + j] < KomsularinEnKucukEtiketDegeri && EtiketNumarasi[x + i, y + j] != 0 && KomsularinEnKucukEtiketDegeri != 0)  //En küçük değer ve okunan hücreye etiket atanmışsa, içindeki değer en küçük değerden küçük ise  o zaman en küçük o hücrenin değeri olmalıdır.
                                {
                                    KomsularinEnKucukEtiketDegeri = EtiketNumarasi[x + i, y + j];
                                }
                            }
                        }

                        if (KomsularinEnKucukEtiketDegeri != 0) //Beyaz komşu buldu ve içlerinde en küçük etiket değerine sahip numara da var.  O zaman orta piksele o numarayı ata.
```

```
                    {
                        EtiketNumarasi[x, y] = KomsularinEnKucukEtiketDegeri;
                    }
                    else if (KomsularinEnKucukEtiketDegeri == 0) //Komşuların hiç birinde etiket numarası yoksa o zaman yeni bir numara ata
                    {
                        EtiketNo = EtiketNo + 1;
                        EtiketNumarasi[x, y] = EtiketNo;
                    }

                    SonDeger = EtiketNumarasi[x, y]; //İşlem öncesi ve işlem sonrası değerler aynı ise ve butün piksellerde hep aynı olursa artık değişim yok demektir.

                    if (IlkDeger != SonDeger)
                        DegisimVar = true;

                }

            }
        }
    } while (DegisimVar == true); // Etiket numarlarında değişik kalmayana kadar dön.


    // Etiket değerine bağlı resmi renklendirecek----------------------
    // Pikseller üzerine yazılmış numaraları diziye atıyor. Dizi boyutu resimdeki piksel sayısınca oluyor.
    int[] DiziEtiket = new int[PikselSayisi];
    i = 0;
    for (x = 1; x < ResimGenisligi - 1; x++)
    {
        for (y = 1; y < ResimYuksekligi - 1; y++)
        {
            i++;
            DiziEtiket[i] = EtiketNumarasi[x, y];
        }
    }

    //Dizideki etiket numaralarını sıralıyor. Hazır fonksiyon kullanıyor.
    Array.Sort(DiziEtiket);

    //Tekrar eden etiket numaraarını çıkarıyor. Hazır fonksiyon kullanıyor. Tekil numaraları diziye atıyor.
    int[] TekrarsizEtiketNumaralari = DiziEtiket.Distinct().ToArray();

    //DİKKAT BURADA RenkDizisi ihtiyaç değil gibi. Renk adedi direk Tekrarsız numaralardan alınabilir.
    int[] RenkDizisi = new int[TekrarsizEtiketNumaralari.Length]; //Tekil numaralar aynı boyutta renk dizisini oluşturuyor.

    for (j = 0; j < TekrarsizEtiketNumaralari.Length; j++)
    {
        RenkDizisi[j] = TekrarsizEtiketNumaralari[j]; //sıradaki ilk renge, ait olacağı etiketin kaç numara olacağını atıyor.
    }

    int RenkSayisi = RenkDizisi.Length; //kaç tane numara varsa o kadar renk var demektir.

    Color[] Renkler = new Color[RenkSayisi];
    Random Rastgele = new Random();
    int Kirmizi, Yesil, Mavi;

    for (int r = 0; r < RenkSayisi; r++) //sonraki renkler.
    {
        Kirmizi = Rastgele.Next(5, 25)*10; //Açık renkler elde etmek ve 10 katları şeklinde olmasını sağlıyor. yani 150-250 arasındaki sayıları atıyor.
        Yesil = Rastgele.Next(5, 25) * 10;
```

```
            Mavi = Rastgele.Next(5, 25) * 10;

            Renkler[r] = Color.FromArgb(Kirmizi, Yesil, Mavi); //Renkler dizisi Color tipinde renkleri tutan bir dizidir.
        }

            //Color[] Renkler= { Color.Black, Color.Blue, Color.Red, Color.Orange, Color.LightPink,
Color.LightYellow, Color.LimeGreen, Color.MediumPurple, Color.Olive, Color.Magenta, Color.Maroon,
Color.AliceBlue, Color.AntiqueWhite, Color.Aqua, Color.LightBlue, Color.Azure, Color.White  };

            for (x = 1; x < ResimGenisligi - 1; x++) //Resmin 1 piksel içerisinden başlayıp, bitirecek. Çünkü çekirdek
şablon en dış kenardan başlamalı.
            {
                for (y = 1; y < ResimYuksekligi - 1; y++)
                {
                    int RenkSiraNo = Array.IndexOf(RenkDizisi, EtiketNumarasi[x, y]); //Dikkat: önemli bir komut.
Dizinin değerinden sıra numarasını alıyor. int[] array = { 2, 3, 5, 7, 11, 13 }; int index = Array.IndexOf(array, 11); //
returns 4

                    if (GirisResmi.GetPixel(x, y).R < 128) //Eğer bu pikselin rengi siyah ise aynı pikselin CikisResmi
resmide siyah yapılacak.
                    {
                        CikisResmi.SetPixel(x, y, Color.Black);
                    }
                    else
                    {
                        CikisResmi.SetPixel(x, y, Renkler[RenkSiraNo]);
                    }
                }
            }
        pictureBox2.Image = CikisResmi;
    }
```

ALAN AŞINDIRMA YADA GENİŞLETME (Morfolojik İşlemler/Biçimsel İşlemler)

Bir resim üzerindeki alanları bulmak bazen bölgeleri net olarak çıkarmak için yetmez. Birbirine çok ince bir çizgi ile bağlı olan alanlar aslında farklı alanlar olabilir. Yada bir bölge üzerinde çok sayıda hatalı duran küçük alanlar aslında bölgenin içerisinde olabilir. Gürültülü bir görüntü olabilir. Böyle bir durumda çok ince bağlanmış bölgeleri birinden ayırmak istiyorsak o ince çizgileri eritmek, kaybetmek isteyebiliriz. Yada alan içindeki çok küçük kısımları kapatıp büyük olan alana dahil etmek isteyebiliriz. Bu tür bir uygulama için Morfolojik işlemler dediğimiz, şekil yada biçimi değişmek için yaptığımız uygulamalara ihtiyaç vardır.

Morfolojik görüntü işleme teknikleri, şekillerin biçimsel yapısı ile ilgilenerek nesneleri ayırt etmemize ve gruplayabilmemize olanak sağlar. Yöntem gri seviye görüntüler üzerinde de çalışsa da genellikle siyah-beyaz (ikili) görüntüler üzerinde kullanılır. Morfolojik filtreler genelde iki temel işlemden türetilmiştir. Bunlar **erosion (aşındırma, daraltma)** ve **dilation (genişletme)** işlemleridir. Aşındırma ikili bir görüntüde bulunan nesnelerin boyutunu seçilen yapısal elemente bağlı olarak küçültürken, genişletme nesnenin alanını artırır.

Bu işlemlerden aşındırma işlemi birbirine ince bir gürültü ile bağlanmış iki veya daha fazla nesneyi birbirinden ayırmak için kullanılırken, genişletme işlemi ise aynı nesnenin bir gürültü ile ince bir şekilde bölünerek ayrı iki nesne gibi görünmesini engellemek için kullanılır. Aslında bu iki işlem birbirinin tersidir. Resim üzerindeki alanlarda bu işlemlerden birini uyguladığımızda komşu diğer alanlar zıttı olan işleme tabi tutulmuş olur. Yani aşındırma uygularken komşu alanda genişletme uygulanmış olur.

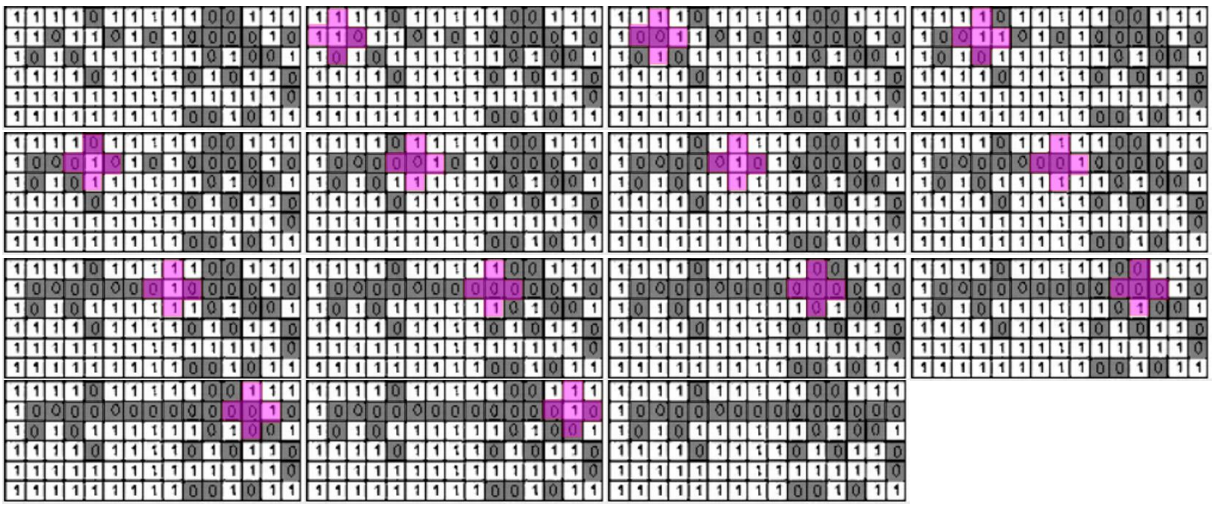
İşlemin nasıl yapıldığını anlamak için aşağıdaki resimler üzerinde anlatalım. Bunun için resmin üzerinde + şeklinde 5 tane pikselden oluşan bir şablon gezdireceğiz. Ortadaki 5. piksel, resim üzerinde işlem yaptığımız piksele karşılık gelir.

- Bu 5 tane piksel resim üzerine konulduktan sonra, beş pikselin tamamı beyaz alanlara basıyorsa, yani 5 pikselin hepsinin karşılığı olan beyaz ise o zaman üzerinde işlem yapılan piksel beyaz olarak işaretlenir.
- Eğer bu 5 pikselden herhangi biri siyah bir pikselin üzerine denk geldiyse o zaman ortadaki pikselin değeri siyah yapılır.

Dikkat edilirse bu işlem ile siyah bölge genişletilirken (dilation), beyaz bölge aşındırılmış (erosion) olmaktadır. Bu işlemde piksellerin tamamının beyazla örtüşmesi Fit, herhangi birinin örtüşmesi ise Hit olarak literatürde adlandırılır. Özetle bu işlem (aşındırma ve genişletme işlemi) bölger üzerinde Açma ve Kapama işlemlerine neden olacaktır. Tekrar anlatacak olursak şu sonuçlara ulaşılır.

Açma İşlemi(Opening): Burada anlatılan yöntem kullanılarak görüntü üzerinde küçük parçaların kaybolması sağlanabilir. Bu işlem için önce Aşındırma (erosion) uygulanırsa görüntünün kenarlarındaki çok küçük parçalar kaybolacaktır. Ama bu durumda genel alan küçülmüş olur. Küçülen alanı tekrar eski haline getirmek için Genişletme (dilation) işlemi uygulanır. Böylece resmin genelindeki birinci seviye küçük parçalar kaybolmuş olur. Yeterli olmaz ise bir kez daha aynı işlem uygulanır. Bu işlem gürültü nedeniyle birbirine bitişik olarak bulunan alanları ayrıştırmak için uygun olacaktır.

Kapama İşlemi(Closing): Açma işleminde uygulanan adımların tersten uygulanmasıdır. Böylece görüntü içerisindeki ayrık parçalar birbirine yaklaşır. Önce genişletme işlemi uygulanarak birbirine yakın alanlar birleştirilmiş olur. Aradaki gürültülü alanlar kaybolmuş olur. Fakat bu genel alanın büyümesine yol açar. Eski haline getirmek için tekrar aşındırma uygulanırsa kenarlardan kırpılacaktır. Ortadaki kapanan bölgeler büyük alana birleştiği için tekrar ayrılamayacaktır.



Dikkat. Burada gezici şablonun bir satırlık hareketi gösterilmiştir. Her satır bittiğinde bir alt satıra inilmeli. Bir alt satıra indiğinde piksellerin yeni değerleri üzerinden işlem yaparsa siyah bölge gittikçe genişler ve her taraf siyah olur. Bu olayın olmaması için daima piksellerin eski değerleri üzerinden işlem yapılmalıdır.

Aşındırma ve Genişletme Yöntemi ile Kenar Belirleme İşlemi

Bu yöntemlerden herhangi biri kullanılarak bir resmin üzerindeki kenarlar ortaya çıkarılabilir. Bunun için resmi önce siyah-beyaz iki renge dönüştürürsek (burada eşik belirlenerek ne kadarı siyah, ne kadarının beyaz olacağına karar verilir) Ardından örneğin beyaz kısımlara genişletme uygulanırsa beyaz alan dışarıya doğru 1 piksellik genişlemiş olacaktır. Ardından elde edilen bu resim, bir önceki orjinal siyah-beyaz resimden çıkarılırsa, aynı renk değerine sahip noktalar 0 sonucunu verirken, farklı renk değerine sahip pikseller o anki değerini verecektir. En dış hattaki genişleyen pikseller ancak farklı renk oluşturacağından bu kısımlar beyaz olarak ortaya çıkacaktır.

Burada dikkat edilirse renkli resim üzerinde cismin alanlarının kenarı bulunmuyor. Sadece siyah beyaza dönüşmüş resmin kenarları bulunuyor. Bu nedenle resim üzerindeki her bölgenin kenarını bulamaz. Renkli

resimdeki tüm kontraslı kenarlar bulunması isteniyor, daha önceki notlarda anlatılan Sobel gibi farklı algoritmaları kullanmak gerekir.

Bu uygulamada genişletme işlemi uygulanırken, 1 piksellik genişletme yapılırsa ortaya çıkan kenar görüntüsüde 1 piksellik olacaktır. Eğer genişletme 2 piksel olarak yapılırsa bu sefer kenarlar daha belirgin 2 piksel olarak gözükecektir. Yada 1 piksellik kenar görüntüsüne sahip resim, genişletme işlemine tabi tutulursa, kenarlar daha kalın olarak gözükecektir.

Peki bu uygulamada kenar bulmak için Genişletme yerine Aşındırma uyugulansaydı ne olurdu? Bu durumda tek değişen ortaya çıkan kenarlar orjinal kenarın iç kısmında oluşurdu. Genişletme işleminde kenarlar alının dışında oluşur. Yani kenar görüntüsü orjinal görüntüden bir 1 piksel daha büyüktür.

Programlama

Orijinal Resim

Gri Resim

128 Eşikle oluşturulmuş Siyah Beyaz Resim

Genişletme uygulanmış Resim.

Genişleyen kenarlar kırmızı gösterilmiş.


Orijinal siyah beyaz resimle genişleyen resmi çıkardığımızda sınırları bulmuş oluruz. İkinci resim Orijinal siyah beyaz resme Sobel uygulandığında elde edilen görüntüdür. Genişletme ve Çıkarma yöntemi ile kenarlar daha net olarak elde edilmiştir.

Orijinal renkli resme Sobel uyguladığında elde edilen görüntü.

```
private void kenarGenisletmeToolStripMenuItem_Click(object sender, EventArgs e)
{
    Color OkunanRenk;
    Bitmap GirisResmi, CikisResmi;
    GirisResmi = new Bitmap(pictureBox1.Image);

    int ResimGenisligi = GirisResmi.Width;
    int ResimYuksekligi = GirisResmi.Height;

    CikisResmi = new Bitmap(ResimGenisligi, ResimYuksekligi);

    int x, y, i, j;

    int R, G, B;

    int SablonBoyutu = 3;
    int ElemanSayisi = SablonBoyutu * SablonBoyutu;


    for (x = (SablonBoyutu - 1) / 2; x < ResimGenisligi - (SablonBoyutu - 1) / 2; x++) //Resmi
taramaya şablonun yarısı kadar dış kenarlardan içeride başlayacak ve bitirecek.
```

```
    {
        for (y = (SablonBoyutu - 1) / 2; y < ResimYuksekligi - (SablonBoyutu - 1) / 2; y++)
        {


            bool RenkBeyaz = false;

            for (i = -((SablonBoyutu - 1) / 2); i <= (SablonBoyutu - 1) / 2; i++)
            {
                for (j = -((SablonBoyutu - 1) / 2); j <= (SablonBoyutu - 1) / 2; j++)
                {
                    OkunanRenk = GirisResmi.GetPixel(x + i, y + j);

                    if (OkunanRenk.R > 128) //Beyaz ise
                        RenkBeyaz = true;

                }

            }
            if (RenkBeyaz == true )
            {
                Color KendiRengi = GirisResmi.GetPixel(x, y);

                if(KendiRengi.R>128)
                    CikisResmi.SetPixel(x, y, Color.FromArgb(255, 255, 255));
                else
                    CikisResmi.SetPixel(x, y, Color.FromArgb(255, 0, 0));
            }
            else //siyahsa
                CikisResmi.SetPixel(x, y, Color.FromArgb(0, 0, 0));


        }
    }

    pictureBox2.Image = CikisResmi;


}

private void orjinalResimdenCikarToolStripMenuItem_Click(object sender, EventArgs e)
{
    Color OkunanRenk;
    Bitmap OrjinalResim, GenislemisResim, CikisResmi;
    OrjinalResim = new Bitmap(pictureBox1.Image);

    int ResimGenisligi = OrjinalResim.Width;
    int ResimYuksekligi = OrjinalResim.Height;

    CikisResmi = new Bitmap(ResimGenisligi, ResimYuksekligi);

    GenislemisResim = new Bitmap(pictureBox2.Image);

    int x, y, i, j;

    int R, G, B;

    int SablonBoyutu = 3;
    int ElemanSayisi = SablonBoyutu * SablonBoyutu;


    for (x = (SablonBoyutu - 1) / 2; x < ResimGenisligi - (SablonBoyutu - 1) / 2; x++) //Resmi
taramaya şablonun yarısı kadar dış kenarlardan içeride başlayacak ve bitirecek.
    {
        for (y = (SablonBoyutu - 1) / 2; y < ResimYuksekligi - (SablonBoyutu - 1) / 2; y++)
        {
```

```
                Color OrjinalRenk = OrjinalResim.GetPixel(x, y);
                Color GenislemisResimRenk = GenislemisResim.GetPixel(x, y);

            R = Math.Abs( OrjinalRenk.R - GenislemisResimRenk.R);

            CikisResmi.SetPixel(x, y, Color.FromArgb(R, R, R));


        }
    }

    pictureBox2.Image = CikisResmi;

}
```