GÖRÜNTÜ İŞLEME - (7.Hafta)

KENAR BULMA ALGORİTMALARI

Bu konuda bir çok algoritma olmasına rağmen en yaygın kullanılan ve etkili olan Sobel algoritması burada anlatılacaktır.

Sobel KenarBulma Algoritması

Görüntüyü siyah beyaza çevirdikten sonra eğer kenar bulma algoritmalarını kullanmak isterseniz bir kaç seçenekten en popüleri sobel kenar bulma filtresidir. Aşağıdaki çekirdek matrisler (konvolüsyon matrisleri) dikey, yatay ve köşegen şeklindeki kenarları bulmak için kullanılır. Sobel operatörü bir resmin kenarlarına karşılık gelen alansal yüksek frekans bölgelerini (keskin kenarları) ortaya çıkarır. Teorik olarak, operatör aşağıda gösterildiği gibi 3×3 konvolüsyon matrisinden oluşur.

-1	0	+1
-2	0	+2
-1	0	+1

Gx

+1	+2	+1
0	0	0
-1	-2	-1

Gy

Şekil. Sobel konvolüsyon matrisi

Bu matrisler, yatay ve dikey olarak görünen kenarlar için ayrı ayrı olacak şekilde düzenlenmiştir. Matrisler giriş görüntüsüne birbirinden bağımsız uygulanabilir. Böylece her bir yön için pikselin değeri ayrı ayrı ölçülmüş olur. Daha sonra çapraz duran kenarların mutlak değerini bulmak ve yönünü (açısını) bulmak için bu değerler aşağıdaki formüllerle birleştirilebilir. Piksel mutlak değeri şu şekilde hesaplanabilir.

$$|G| = \sqrt{Gx^2 + Gy^2}$$

Piksel değerini daha hızlı hesaplama için şu formülde kullanılabilir.

$$|G| = |Gx| + |Gy|$$

Ortaaya çıkan kenarın yön açısı (piksel ızgarasına göre) x ve y yönlerindeki değerlerine bakarak şu şekilde bulunur.

$$\theta = \arctan\left(\frac{Gy}{Gx}\right)$$

Bu durumda 0 derece yatay duran çizgileri, 90 derece ise dikey duran çizgileri gösterecektir. Eğik duran çizgilerde diğer açıları oluşturacaktır. Burada sıfır derece çizgide alt kısım siyahtan, üst kısım beyaza doğru geçişi gösterir. 0 derecenin bir benzeri olan yine yatay duran çizgi 180 derecede ise üst kısım siyah bölgeden alt kısım beyaz bölgeye doğru bir geçişi gösterecektir. Açılar saatin tersi yönüne göre ölçülür.

Burada Gx konvolüsyon matrisi tek başına kullanılırsa yatayda renk değişimini bulacağından, ortaya çıkan çizgileri dikey olarak görürüz. Benzer şekilde Gy konvolüsyon matrisi de tek başına kullanılırsa aşağıdan yukarıya doğru renk geçişini (siyahtan-beyaza) göstereceğinden ortaya çıkan çizgiler yatay olacaktır. Resim üzerindeki çizgileri yatay yada dikey görmek yerine kendi doğal duruşlarını görmek istersek, her iki konvolüsyon matrisini yukarıda formülü verilen hesaplama ile toplayabiliriz. Benzer şekilde yukarıda verilen karaköklü formülü de kullanabiliriz. Eğer resim üzerinde belli açılardaki çizgileri ortaya çıkarmak istersek verilen açı formülü ile hesaplama yaparak bu çizgileri de belirleyebiliriz.

İki matrisi aynı anda kullanmak için ve matris üzerindeki noktaları aşağıdaki şekilde temsil etmek için hesaplama yaparak deneyelim.

P_1	P_2	P_3
P_4	P_5	P_6
P_7	P_8	P_9

-1	0	+1
-2	0	+2
-1	0	+1

Gx

+1	+2	+1
0	0	0
-1	-2	-1

Gy

Buradaki her bir çekirdek matrisin hesabı şu şekilde hesaplanır.

$$|Gx| = |-P_1 + P_3 - 2P_4 + 2P_6 - P_7 + P_9|$$

$$|Gy| = |P_1 + 2P_2 + P_3 - P_7 - 2P_8 - P_9|$$

Bu çekirdek matris kullanılarak pikselin sobel değeri yaklaşık formül kullanılarak şu şekilde hesaplanır:

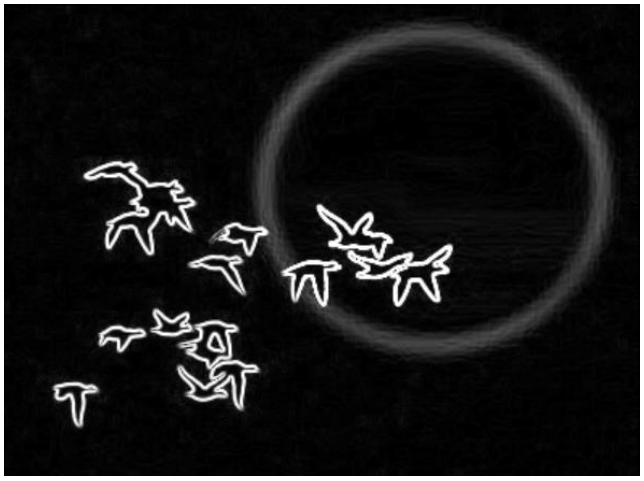
$$|G| = |Gx| + |Gy|$$

Diğer formül kullanıldığında ise şu şekilde olacaktır.

$$|G| = \sqrt{Gx^2 + Gy^2}$$

Burada renkli resimde renk değerleri 3 kanal olduğunda tek kanal üzerinden işlem yapmak için Gri renk değerleri üzerinden işlem yapmak gerekir. Aşağıdaki programda Gri renk için 3 kanalın ortalması alınmıştır. Gerekirse Gri renk formülleri de kullanılabilir (daha önceki konularda geçti).

Programlama (Sobel Filtresi)



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

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

    CikisResmi = new Bitmap(ResimGenisligi, ResimYuksekligi);

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

    int x, y;
```

```
    Color Renk;
    int P1, P2, P3, P4, P5, P6, P7, P8, P9;

    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++)
    {

    Renk = GirisResmi.GetPixel(x - 1, y - 1);
    P1 = (Renk.R + Renk.G + Renk.B) / 3;

    Renk = GirisResmi.GetPixel(x, y - 1);
    P2 = (Renk.R + Renk.G + Renk.B) / 3;

    Renk = GirisResmi.GetPixel(x + 1, y - 1);
    P3 = (Renk.R + Renk.G + Renk.B) / 3;

    Renk = GirisResmi.GetPixel(x - 1, y);
    P4 = (Renk.R + Renk.G + Renk.B) / 3;

    Renk = GirisResmi.GetPixel(x, y);
    P5 = (Renk.R + Renk.G + Renk.B) / 3;

    Renk = GirisResmi.GetPixel(x + 1, y);
    P6 = (Renk.R + Renk.G + Renk.B) / 3;

    Renk = GirisResmi.GetPixel(x - 1, y + 1);
    P7 = (Renk.R + Renk.G + Renk.B) / 3;

    Renk = GirisResmi.GetPixel(x, y + 1);
    P8 = (Renk.R + Renk.G + Renk.B) / 3;

    Renk = GirisResmi.GetPixel(x + 1, y + 1);
    P9 = (Renk.R + Renk.G + Renk.B) / 3;

    //Hesaplamayı yapan Sobel Temsili matrisi ve formülü.
    int Gx =Math.Abs( -P1 + P3 - 2 * P4 + 2 * P6 - P7 + P9); //Dikey çizgiler
    int Gy = Math.Abs(P1 + 2* P2 + P3 -  P7 - 2 * P8 - P9);  //Yatay Çizgiler

    int SobelDegeri =0
    //SobelDegeri = Gx;
    //SobelDegeri = Gy;
    //SobelDegeri = Gx + Gy;  //1. Formül

    SobelDegeri =Convert.ToInt16(Math.Sqrt( Gx* Gx + Gy* Gy));  //2.Formül

    //Renkler sınırların dışına çıktıysa, sınır değer alınacak. Negatif olamaz, formüllerde
mutlak değer vardır.
    if (SobelDegeri > 255) SobelDegeri = 255;

    //Eşikleme: örnek olarak 100 değeri kullanılmıştır.
    //if (SobelDegeri > 100)
    //    SobelDegeri = 255;
    //else
    //    SobelDegeri = 0;

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

    }
    }
    pictureBox2.Image = CikisResmi;
}
```

Prewitt Kenar Bulma Algoritması

Bu algoritma Sobel'e benzer ve aşağıdaki gibi biraz farklı çekirdek matris kullanır. Elde edilen sonuçlar resmin her yerinde aynı değildir.

-1	0	+1
-1	0	+1
-1	0	+1

Gx

+1	+1	+1
0	0	0
-1	-1	-1

Gy

Prewitt Kenar Bulma algoritmasının çekirdek matrisleri

Ödev: Bu algoritmalar Gri renk için ortalama formülü kullanılmıştır. Gri tonlama için daha gelişmiş olan formülleri kullanarak deneyin.

$$|G| = |Gx| + |Gy|$$

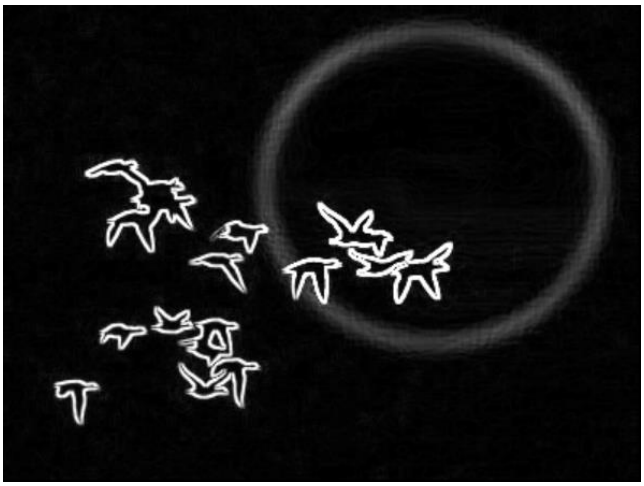
Piksel değerini daha hızlı hesaplama için şu formülde kullanılabilir.

$$|G| = \sqrt{Gx^2 + Gy^2}$$

Ortaaya çıkan kenarın yön açısı (piksel ızgarasına göre) x ve y yönlerindeki değerlerine bakarak şu şekilde bulunur.

$$\theta = \arctan\left(\frac{Gy}{Gx}\right)$$

Programlama (Prewitt Algoritması)



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

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

    CikisResmi = new Bitmap(ResimGenisligi, ResimYuksekligi);

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

```
    int x, y;

    Color Renk;
    int P1, P2, P3, P4, P5, P6, P7, P8, P9;

    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++)
    {

    Renk = GirisResmi.GetPixel(x - 1, y - 1);
    P1 = (Renk.R + Renk.G + Renk.B) / 3;

    Renk = GirisResmi.GetPixel(x, y - 1);
    P2 = (Renk.R + Renk.G + Renk.B) / 3;

    Renk = GirisResmi.GetPixel(x + 1, y - 1);
    P3 = (Renk.R + Renk.G + Renk.B) / 3;

    Renk = GirisResmi.GetPixel(x - 1, y);
    P4 = (Renk.R + Renk.G + Renk.B) / 3;

    Renk = GirisResmi.GetPixel(x, y);
    P5 = (Renk.R + Renk.G + Renk.B) / 3;

    Renk = GirisResmi.GetPixel(x + 1, y);
    P6 = (Renk.R + Renk.G + Renk.B) / 3;

    Renk = GirisResmi.GetPixel(x - 1, y + 1);
    P7 = (Renk.R + Renk.G + Renk.B) / 3;

    Renk = GirisResmi.GetPixel(x, y + 1);
    P8 = (Renk.R + Renk.G + Renk.B) / 3;

    Renk = GirisResmi.GetPixel(x + 1, y + 1);
    P9 = (Renk.R + Renk.G + Renk.B) / 3;

    int Gx = Math.Abs(-P1 + P3 - P4 + P6 - P7 + P9); //Dikey çizgileri Bulur
    int Gy = Math.Abs(P1 + P2 + P3 - P7 - P8 - P9);  //Yatay Çizgileri Bulur.

    int PrewittDegeri = 0;
    PrewittDegeri = Gx;
    PrewittDegeri = Gy;

    PrewittDegeri = Gx + Gy;  //1. Formül
    //PrewittDegeri = Convert.ToInt16(Math.Sqrt(Gx * Gx + Gy * Gy));  //2.Formül


    //Renkler sınırların dışına çıktıysa, sınır değer alınacak.
    if (PrewittDegeri > 255) PrewittDegeri = 255;

    //Eşikleme: Örnek olarak 100 değeri kullanıldı.
    //if (PrewittDegeri > 100)
    //PrewittDegeri = 255;
    //else
    //PrewittDegeri = 0;

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

    }
    }
    pictureBox2.Image = CikisResmi;
```

```
}
```

Robert Cross Kenar Bulma Algoritması

Basit ve hızlı bir algoritmadır. Resim üzerindeki 2 boyutlu geçişleri ölçer. Keskin kenarları ortaya çıkarır. Gri resim üzerinde işlem yapar. 2x2 lik matris kullandığından çok bulanık resimlerde kenarları bulamaz. Bulmuş olduğu kenarlarıda çok ince olarak gösterir.

Aşağıdaki 2x2 lik aralarında 90 açı bulunan iki matrisle işlemleri yürütür. Kullanım olarak Sobel'e benzer.

+1	0
0	-1

Gx

0	+1
-1	0

Gy

P_1	P_2
P_3	P_4

Robert Cross konvolisyon matrisleri

Sobel'de matrislerin yönleri x ve y eksenleri yönünde idi. Burada ise Resmin ızgarasına 45 açıyla durmaktadır. Kullanılacak formüller burada yine Gx ve Gy şeklinde gösterilmiştir fakat bu yönlerin ızgaraya 45 ve 135 derece ile durduğunu kabul edelim.

Bunun için aşağıdaki iki formülü kullanabiliriz. İkinci formül daha hızlı çalışır.

$$|Gx| = |P_1 - P_4|$$
$$|Gy| = |P_2 - P_3|$$

Olmak üzere

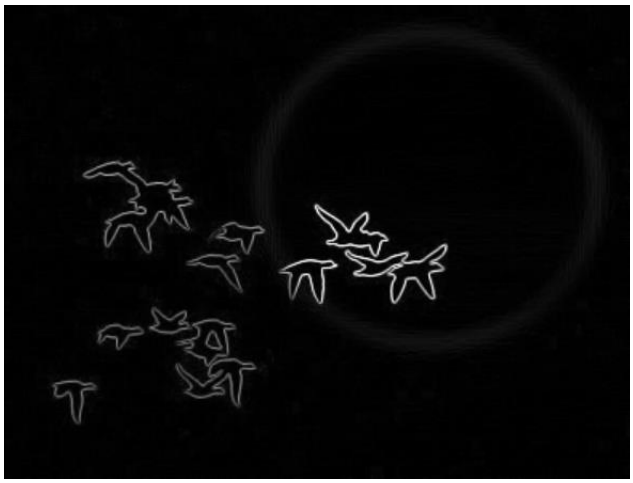
$$|G| = |Gx| + |Gy|$$
$$|G| = \sqrt{Gx^2 + Gy^2}$$

Formülleri kullanılabilir. Oluşan kenarın resim ızgarasına olan açısı aşağıdaki formülle bulunabilir (Dikkat formülü deneyerek teyid edin!).

$$\theta = \arctan\left(\frac{Gy}{Gx}\right) + 45$$

Bu operatörde 4 piksel olduğundan çıkış pikselin hangisine denk geldiği belirsizdir. Aslında yarım piksellik bir kaymadan söz edilebilir. Programlarken referans pikseli sol üst köşedeki ilk piksel alınmıştır.

Programlama (Robert Cross)



Dikkat bu uygulamada zeminle öndeki nesne arasındaki fark arttıkça kenarlar daha belirgin olmaktadır. Sebebini yorumlamaya çalışın.

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

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

    CikisResmi = new Bitmap(ResimGenisligi, ResimYuksekligi);

    int x, y;

    Color Renk;
    int P1, P2, P3, P4;

    for (x = 0; x < ResimGenisligi - 1; x++) //Resmi taramaya şablonun yarısı kadar dış
kenarlardan içeride başlayacak ve bitirecek.
    {
      for (y = 0; y < ResimYuksekligi - 1; y++)
      {

         Renk = GirisResmi.GetPixel(x , y);
         P1 = (Renk.R + Renk.G + Renk.B) / 3;

         Renk = GirisResmi.GetPixel(x +1, y);
         P2 = (Renk.R + Renk.G + Renk.B) / 3;

         Renk = GirisResmi.GetPixel(x , y + 1);
         P3 = (Renk.R + Renk.G + Renk.B) / 3;

         Renk = GirisResmi.GetPixel(x + 1, y + 1);
         P4 = (Renk.R + Renk.G + Renk.B) / 3;

         int Gx = Math.Abs(P1 - P4 ); //45 derece açı ile duran çizgileri bulur.
         int Gy = Math.Abs(P2 - P3); //135 derece açı ile duran çizgileri bulur.

         int RobertCrossDegeri = 0;
         RobertCrossDegeri = Gx;
         RobertCrossDegeri = Gy;

         RobertCrossDegeri = Gx + Gy;  //1. Formül
         //RobertCrossDegeri = Convert.ToInt16(Math.Sqrt(Gx * Gx + Gy * Gy));  //2.Formül

         //Renkler sınırların dışına çıktıysa, sınır değer alınacak.
```

```
            if (RobertCrossDegeri > 255) RobertCrossDegeri = 255; //Mutlak değer kullanıldığı için
negatif değerler oluşmaz.

            //Eşikleme
            //if (RobertCrossDegeri > 50)
            //    RobertCrossDegeri = 255;
            //else
            //    RobertCrossDegeri = 0;

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

        }
    }
    pictureBox2.Image = CikisResmi;

}
```

Compass Kenar Bulma Algoritması

Compass algoritması da Sobel ve Robert Cross gibi kenarların her iki tarafındaki renk geçiş farkını kullanan alternatif bir algoritmadır.

Bu algoritma 8 farklı yönü verecek şekilde çekirdek matrisi döndürüp en yüksek piksel değerini veren matris ile o pikselin değerini oluşturur. Böylece kenarın ilerleme yönünü en iyi bulan matrisi deneyerek kullanmış olmaktadır. Hangi açıdaki matrisin kullanıldığı ise o kenara ait ilerleme yönünü de vermiş olmaktadır.

Önceki algoritmalar Gx ve Gy şeklinde birbirine dik iki yönü kullanırken ve ara açıları bu iki yönün birleşimi ile bulurken, burada 8 adet yönün kullanılması (G_0, G_{45}, $G9_0$, G_{135}, G_{180}, G_{225}, G_{270}, G_{315}) kenarların yönünü daha hassas olarak vermekte ve o yöndeki kenarlar daha belirgin gösterilmektedir. Önceki Gx ve Gy algoritmaları bir 360 derecelik daireyi ancak dik kenarların geçtiği kısımlarda daha belirgin gösterirken, bu algoritma 45 derecelik açılar halinde yönleri bulduğu için daire tüm yönlerde çok daha belirgin olarak gözükecektir. Algoritmanın işlem süresi 8 adet matrisi resim üzerinde gezdirdiği için daha yavaştır. Algoritmanın matematiksel anlamış şu şekilde gösterilebilir.

$$|G| = \max\left(|G_i| : i = 1\ to\ n\right)$$

Burada n adet çekirdek matris için hesaplanan $|G_i|$ değerleri içinde en büyük değeri veren matris o pikselin değerini oluşturacaktır. Kullanılan 8 adet 45 açılarla duran matrisler şu şekilde belirlenebilir.

-1	-1	-1
+1	-2	+1
+1	+1	+1

0^0

-1	-1	+1
-1	-2	+1
+1	+1	+1

45^0

-1	+1	+1
-1	-2	+1
-1	+1	+1

90^0

+1	+1	+1
-1	-2	+1
-1	-1	+1

135^0

+1	+1	+1
+1	-2	+1
-1	-1	-1

180^0

+1	+1	+1
+1	-2	-1
+1	-1	-1

225^0

+1	+1	-1
+1	-2	-1
+1	+1	-1

270^0

+1	-1	-1
+1	-2	-1
+1	+1	+1

315^0

Buradaki kullanılan Compass algoritması olarak kullanılan çekirdek matrisin haricinde aşağıda 2 açıdaki örnekleri verilen algoritmalarda Compass mantığı ile kullanılabilir. Hatırlanırsa Sobel algoritması yukarıda 90^0 ve 0^0 derece olarak kullanılmıştı. Burada verilen 135^0 açı değerinde ve yukarıdaki örnek uygulamalarda nasıl yapıldığı incelenerek diğer 8 adet açıdaki matrisleri oluşturulabilir. Bunları kendiniz oluşturunuz. Ayrıca bu örneklerden Kirsch ve Robinson matrislerini yukarıda Sobel konusunda verildiği şekilde Gx ve Gy mantığı ile kenar bulma

işlemleri için deneyin. Sobel, Kirsch ve Robinson algoritmalarını iki eksende kullanıldığında hangisi daha iyi sonuç vermektedir gözlemleyin.

	0^0				135^0		
Sobel	-1	0	1		0	1	2
	-2	0	2		-1	0	1
	-1	0	1		-2	-1	0
Kirsch	-3	-3	5		-3	5	5
	-3	0	5		-3	0	5
	-3	-3	5		-3	-3	-3
Robinson	-1	0	1		0	1	1
	-1	0	1		-1	0	1
	-1	0	1		-1	-1	0

Dikkat! bu algoritmanın programlaması Ödevlerde istendiğinden buraya konulmamıştır. Diğer programlama örneklerine bakarak benzer şekilde programlanabilir. Fakat burada 8 tane matris olduğundan her matrisi yukarıdaki örneklerde verilen şekilde programlamak kodları uzatır. O nedenle matris değerlerini çift boyutlu bir dizide tutarak programlamak gerekir. Örneğin C# daki List (Listeler dizisi) kullanarak yapabilirsiniz. Yada 2 boyutlu bir dizi tutarak da olabilir.

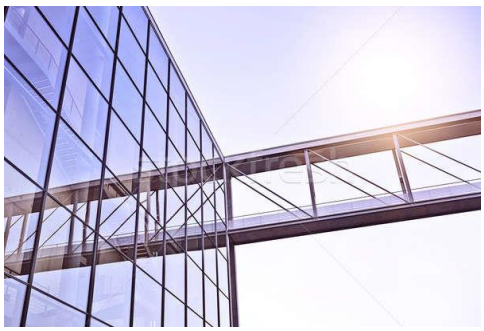
Aşındırma ve Genişletme Yöntemi ile Kenar Belirleme İşlemi

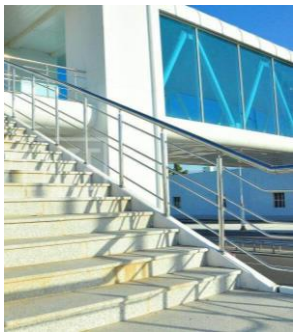
Bu yöntem bir sonraki notlarda Aşındırma ve Genişletme yöntemi anlatıldığından, oradaki notların sonuna konulmuştur.

Ödevler

Ödev 1: Compass algorirtmasının programlamasını yapınız. Algoritma içerisinde matrisleri kullanırken iki boyutlu Listeler dizisini kullanın. Listeler dizisi hakkında bilgi İnt.Tab.Prg dersinin 3 nolu notlarında vardır. İki boyutlu olan dizileri araştırın ve program içinde uygulayın.

Ödev 2: Çelik kontrüksiyon şeklinde aşağıdaki örneklere benzer (bunları kullanmayın kendiniz bulun) 3 tane farklı yapının üzerindeki çizgileri Sobel ve Compass algoritmalarını kullanarak ortaya çıkartın ve farklı açılarda duran çizgileri farklı renklerde boyatarak şeklin değişik bir görünümünü ortaya çıkartın.

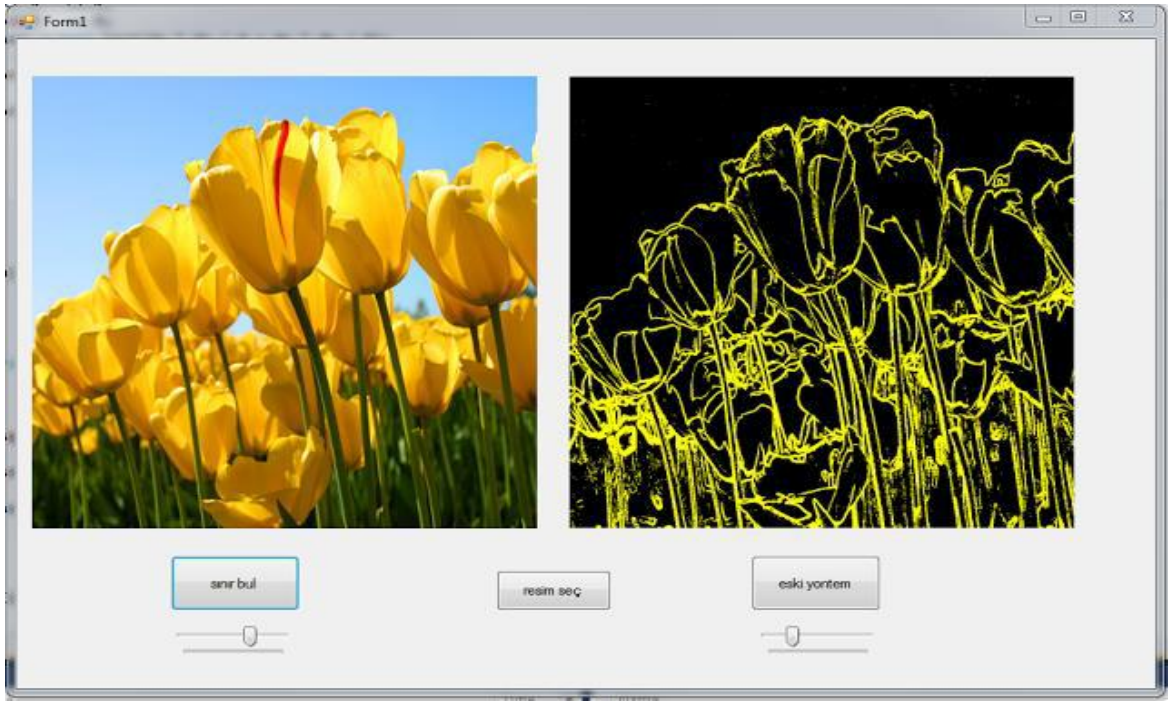




Ödev 3: 6 ve 7 nolu dökümanlardaki kodları hazır olarak verilmiş olan tüm algoritmaları kendi yazdığınız program içerisinde deneyin. Her algoritmanın en iyi sonuç verdiği örnek resimler bularak uygulamasını gösterin. Denediğiniz her algoritmayı hangi resimlerde daha iyi sonuç verdiğinin sebebini yorumlayarak birer cümle ile açıklayın.

Araştırma 1: Burada yer verilmeyen Canny Kenar Bulma Algoritmasını araştırıp programlamasını yapın ve güçlü taraflarını bulmaya çalışın.

Araştırma 2: Aşağıdaki örnek programın çalışma mantığını bulun ve kullanılan matris ve yapı ortaya çıkarın. Buradaki kodlarda değişken adlarını daha anlaşılır, kelimelerden oluşacak şekilde yeniden düzenleyin. Program içerisinde kullanılan Listeler dizi yapısının nasıl çalıştığını anlatın. (Tek boyutlu listeler İTP 3 nolu notlarda vardır). Matris değerleri için bu yapıyı diğer Ödevlerde de programlayarak, kullanabildiğinizi gösterin.



```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace goruntuisleme
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        Bitmap resim;
        private void button1_Click(object sender, EventArgs e)
        {
            openFileDialog1.Filter = "Resim Dosyaları|" +
"*.bmp;*.jpg;*.gif;*.wmf;*.tif;*.png";


          if (openFileDialog1.ShowDialog() == DialogResult.OK)
          {
              pictureBox2.Image = null;
              pictureBox1.Image = Image.FromFile(openFileDialog1.FileName);
          }


public Bitmap sinirbul(Bitmap resim)
       {
           Bitmap ret = new Bitmap(resim.Width, resim.Height);
           for (int i = 1; i < resim.Width - 1; i++)
           {
               for (int j = 1; j < resim.Height - 1; j++)
               {
                   Color cr = resim.GetPixel(i + 1, j);
                   Color cl = resim.GetPixel(i - 1, j);
                   Color cu = resim.GetPixel(i, j - 1);
                   Color cd = resim.GetPixel(i, j + 1);
                   Color cld = resim.GetPixel(i - 1, j + 1);
                   Color clu = resim.GetPixel(i - 1, j - 1);
                   Color crd = resim.GetPixel(i + 1, j + 1);
                   Color cru = resim.GetPixel(i + 1, j - 1);
                   int power = getMaxD(cr.R, cl.R, cu.R, cd.R, cld.R, clu.R,
cru.R, crd.R);
```

```
                    if (power > 50)
                        ret.SetPixel(i, j, Color.Yellow);
                    else
                        ret.SetPixel(i, j, Color.Black);
                }
            }
            return ret;
        }
        private int getD(int cr, int cl, int cu, int cd, int cld, int clu, int
cru, int crd, int[,] matrix)
        {
            return Math.Abs(matrix[0, 0] * clu + matrix[0, 1] * cu + matrix[0, 2]
* cru
               + matrix[1, 0] * cl + matrix[1, 2] * cr
                  + matrix[2, 0] * cld + matrix[2, 1] * cd + matrix[2, 2] * crd);
        }
        private int getMaxD(int cr, int cl, int cu, int cd, int cld, int clu, int
cru, int crd)
        {
            int max = int.MinValue;
            for (int i = 0; i < templates.Count; i++)
            {
                int newVal = getD(cr, cl, cu, cd, cld, clu, cru, crd,
templates[i]);
                if (newVal > max)
                    max = newVal;
            }
            return max;
        }

            private List<int[,]> templates = new List<int[,]>
   {
      new int[,] {{ -3, -3, 5 }, { -3, 0, 5 }, { -3, -3, 5 } },


    new int[,] {{ -3, 5, 5 }, { -3, 0, 5 }, { -3, -3, -3 } },
    new int[,] {{ 5, 5, 5 }, { -3, 0, -3 }, { -3, -3, -3 } },
    new int[,] {{ 5, 5, -3 }, { 5, 0, -3 }, { -3, -3, -3 } },
    new int[,] {{ 5, -3, -3 }, { 5, 0, -3 }, { 5, -3, -3 } },
    new int[,] {{ -3, -3, -3 }, { 5, 0, -3 }, { 5, 5, -3 } },
    new int[,] {{ -3, -3, -3 }, { -3, 0, -3 }, { 5, 5, 5 } },
    new int[,] {{ -3, -3, -3 }, { -3, 0, 5 }, { -3, 5, 5 } }
 };
```

Araştırma 3:

Aşağıdaki linkte verilen gelişmiş kırpma işleminde Sobel Filtresi kullanılmaktadır. Bu kırpma işleminde kullanılan algoritmayı programlayın.

http://www.cescript.com/2015/07/icerik-tabanli-imge-olcekleme.html