GÖRÜNTÜ İŞLEME - (7.Hafta)

KENAR BELİRLEME ALGORİTMALARI

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

SOBEL FİLTRESİ

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 çalışan kenarlara en üst düzeyde yanıt vermek için tasarlanmıştır. Matrisler giriş görüntüsüne ayrı ayrı uygulanabilir. Böylece her bir yön için pikselin değeri ayrı ayrı ölçülmüş olur. Daha sonra bu değerler her bir noktada mutlak büyüklüğü ve yönü bulmak üzere birleştirilebilir. Piksel değeri şu şekilde verilir.

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

Alışılmış olarak daha hızlı hesaplama için yaklaşık değer şu şekilde de hesaplanabilir.

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

Kenarın yön açısı (piksel ızgarasına göre) alansal piksel değerine bakarak şu şekilde bulunur.

$$\theta = \arctan(Gy/Gx)$$

Bu durumda 0 derece yönlendirme açısı, resim üzerinde soldan sağa doğru siyahtan beyaza doğru kontras değişimini gösterir. Bunun dışındaki tüm açılar saatin tersi yönüne göre ölçülür.

Çoğu zaman, bu mutlak büyüklük kullanıcının gördüğü çıktıyı verir; İki bileşeninin hesaplanması Şekil'de gösterilen temsili konvolüsyon operatörünü kullanarak yapılır ve girdi görüntüsü üzerinden tek bir kez geçilir.

P_1	P_2	P_3
P_4	P_5	P_6
P_7	P_8	P_9

Şekil. Yaklaşık piksel değerini hızla hesaplamak için kullanılan temsili konvolüsyon matrisi.

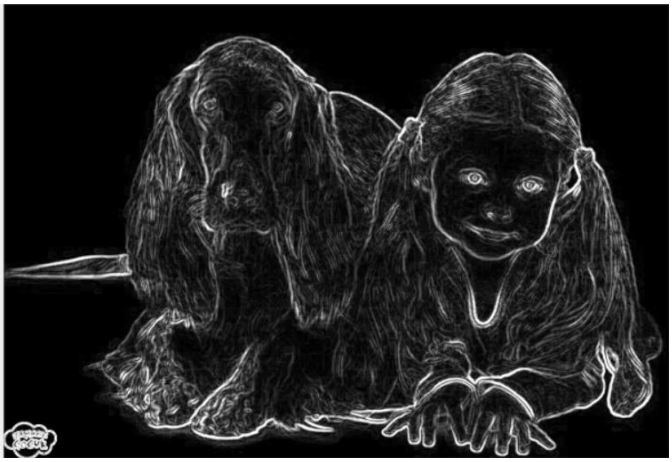
Bu çekirdek matris kullanılarak yaklaşık büyüklük şu şekilde hesaplanır:

$$|G| = |(P_1 + 2 \times P_2 + P_3) - (P_7 + 2 \times P_8 + P_9)| + |(P_3 + 2 \times P_6 + P_9) - (P_1 + 2 \times P_4 + P_7)|$$

Sobel operatörü Roberts Coross operatörüne benzer fakat ondan daha yavaş hesaplar. Sobelde çekirdek matris daha büyük olduğu için gürültüden daha az etkilenir. Operatör aynı zamanda Roberts Cross'a kıyasla benzer kenarlar için daha yüksek çıkış değerleri üretir.

Görüntülerdeki doğal kenarlıklar, Sobel operatörünün yumuşatma etkisinden dolayı çıkıştaki birkaç piksellik hatlara neden olur. Buna karşı koymak için biraz inceltme arzulanabilir. Başarısız olursa Canny operatöründe olduğu gibi bir çeşit histeresis sırtının izlenmesi de kullanılabilir.

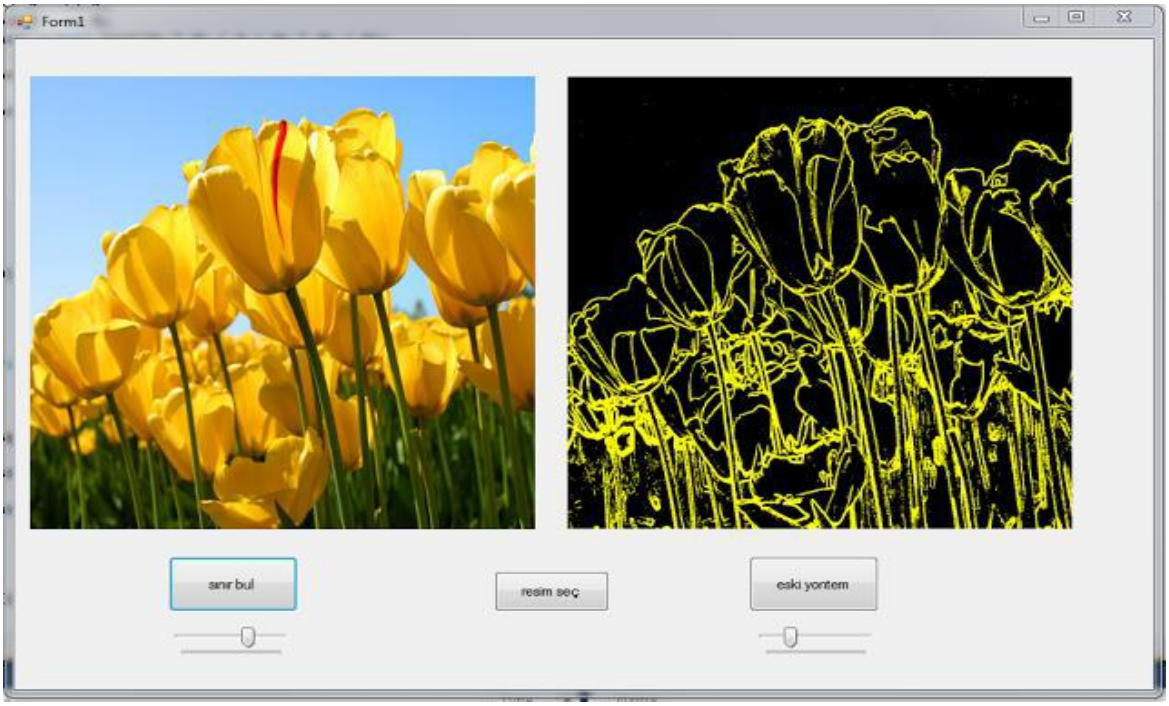
Programlama (Sobel Filtresi)



```
        private void sobelToolStripMenuItem_Click_1(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;

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

                    Renk = GirisResmi.GetPixel(x -1, y -1);
                    P1 = Renk.R;
```

```
                Renk = GirisResmi.GetPixel(x , y - 1);
                P2 = Renk.R;

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

                Renk = GirisResmi.GetPixel(x - 1, y );
                P4 = Renk.R;

                Renk = GirisResmi.GetPixel(x , y );
                P5 = Renk.R;

                Renk = GirisResmi.GetPixel(x + 1, y );
                P6 = Renk.R;

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

                Renk = GirisResmi.GetPixel(x , y + 1);
                P8 = Renk.R;

                Renk = GirisResmi.GetPixel(x + 1, y + 1);
                P9 = Renk.R;

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

                //Renkler sınırların dışına çıktıysa, sınır değer alınacak.
                if (RenkDegeri > 255) RenkDegeri = 255;
                CikisResmi.SetPixel(x, y, Color.FromArgb(RenkDegeri, RenkDegeri, RenkDegeri));
            }
        }
        pictureBox2.Image = CikisResmi;

    }
```

Örnek başka bir çalışma:




```
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 } }
};
```