



Uludağ Üniversitesi

Bilgisayar Mühendisliği Tezli Yüksek Lisans Programı

Görüntü İşleme ve Uygulamaları

**1. Dönem – Proje 1
Uygulama Raporu**

Hazırlayan

Nisanur BULUT – 501731014

İçindekiler

1. Plaka Tanıma Yazılımı Nedir
2. Plaka Tanıma Yazılımının Hedefleri
3. Kullanılan Yöntemler
 - 3.1 Gri Seviye İndirgeme
 - 3.2 Median Filtreleme
 - 3.3 Sobel Filtreleme
 - 3.4 Otsu Algoritması
4. Morfolojik İşlemler
 - 4.1 Aşındırma İşlemi
 - 4.2 Genişletme İşlemi
 - 4.3 Gürültü Yok Etme
 - 4.4 Opening/Closing İşlemi
5. Aforge Kütüphanesi ve Damla Filtreleme İşlemi
6. Filtre Uygulama İşlemlerinin Özeti
7. Plaka bölgesinin belirlenmesi
8. Özet

1. Plaka Tanıma Yazılımı Nedir?

Plaka Tanıma Sistemi kameralardan elde edilen araç görüntüsünün üzerinde plaka bölgesi tespit edilerek ve ayrıştırılarak, plaka üzerinde bulunan karakterlerin optik karakter tanıma (Görüntü İşleme) yöntemleri ile okunması işlemidir. Dijital görüntülerin alınması sayesinde plaka okumasının gerçekleştiriliyor olması sayesinde, yazılım tabanlı olarak çalışırlar. Projeye göre değişik uygulamaları olup, uygulamaya özel algoritma ve donanım yapısının bir araya getirilmesi ile oluşturulan bir sistemdir.

2. Plaka Tanıma Yazılımının Hedefleri

Araçların plaka bilgisinin, görüntülerden elde edilmesiyle aslında aracın kimlik bilgisi elde edilir. Bu sayede bir araca dair aşağıda belirtilen durumlar takip edilebilir.

- a) Otoyol, köprü ve gişelerden geçen araçların çalıntı veya ihlal bilgilerine ulaşabilmek ve analizlerini yapabilmek
- b) Otopark, site, nizamiye ve buna benzer kontrollü geçiş noktalarının giriş ve çıkışlarında insan gücünü hafifletmek, kolay, hızlı, güvenli giriş ve çıkışlarını sağlamak
- c) Kantar, araç muayene istasyonları ve buna benzer yerlerde plaka bilgilerine otomatik olarak ulaşabilmek
- d) Şehir giriş ve çıkışlarının kontrol altında tutulması, şehir güvenliğinin sağlanması
- e) Karayollarının araç yoğunluk ölçümlerinin yapılması
- f) Karayollarının stratejik ulaşım analizlerinin yapılması
- g) Trafik düzenleme işlemleri
- h) İki nokta arasında araçların ulaşım süresinin belirlenmesi, bilgi panosuna aktarılması
- i) Akıllı trafik sistemlerinin kurulması
- j) Otopark kontrolü, yoğunluk ölçümleri
- k) Kayıtlı olan ve ya geçiş yetkisi bulunan araçlar için otomatik bariyer devreye girmesi ya da yasaklı, izinsiz araçların geçmeye çalışma durumunda alarmin devreye girmesi

3. Kullanılan Yöntemler

Aracın plaka bilgisinin elde edilebilmesi için plakanın koordinat bilgisi bilinmelidir. Koordinatların yer tespitinin ardından karakter tanıma işlemi yapılır. Genel anlamda süreç adım adım ifade edilecek olursa aşağıdaki gibi listelenir.

- a) Kamera görüntüsü üzerinden plaka yerinin tespit edilmesi ve ayrıştırılması
- b) Plakanın sonraki algoritmalara uygun şekilde yeniden konumlandırılması ve boyutlandırılması
- c) Parlaklık, zıtlık gibi görüntü özelliklerinin normalizasyonu
- d) Karakter ayırma ile görüntüden karakterlerin çıkarılması
- e) Optik karakter tanıma
- f) Ülkeye özgü söz dizimi ve geometrik kontroller

3.1 Gri Seviye İndirgeme

Uygulama sürecinde görüntü işleme hızını artırabilmek için RGB formattaki görüntü, ortalama değer yöntemiyle gri seviyeye indirgenmiştir. Byte veri tipi dönüşümü yapılır.

- ❖ Resim üzerinde yapılan her işlemin tamamlanma süreci, progressbar ile gösterilir.

```

progressBar1.Visible = true;
int i, j;
Color ort; //Color sınıfından bir renk nesne tanımlıyoruz.

//int r,g,b;
progressBar1.Maximum = bmp.Width * bmp.Height; //İşlem çubuğunun maksimum olduğu yer for döngüsünün sonundaki piksel değerine erişmemiz durumundadır.
for (i = 0; i <= bmp.Width - 1; i++) //dikey olarak görüntümüzü tarıyoruz.
{
    for (j = 0; j <= bmp.Height - 1; j++) //yatay olarak görüntümüzü tarıyoruz.
    {
        ort = bmp.GetPixel(i, j);
        ort = Color.FromArgb((byte)((ort.R + ort.G + ort.B) / 3), (byte)((ort.R + ort.G + ort.B) / 3), (byte)((ort.R + ort.G + ort.B) / 3));
        bmp.SetPixel(i, j, ort);
        if ((i % 10) == 0) //her on satırda bir göstergelyi güncelle
        {
            progressBar1.Value = i * bmp.Height + j;
            Application.DoEvents();
        }
    }
}

```

Resim 1- Gri Seviye İndirgeme C# Kod Bloğu



Resim2-Gri Seviye İndirgeme Sonucu

3.2 Median Filtreleme

Gri seviyeye indirgenmiş olan görüntü üzerinde keskin geçişleri en az seviyeye indirmek için median filtre uygulanacaktır. Median filtre 3x3, 5x5, 7x7 gibi tek sayı boyutlu filtrelerden oluşur. Görüntüyü yumuşatır. Kullanılan çekirdek şablonun yani filtrenin boyutu arttıkça yumuşama yani bulanıklıklaşma da artar.

Median filtrenin tercih edilme sebebi ortalama alıcı filtreyle kıyaslandığında daha sağlıklı sonuçlar vermesidir. Görüntü üzerindeki detay kaybı daha az olur. Bir pikselin değerini değiştirirken komşularının ve kendisinin ortalamasını almak yerine komşuları içinde ortanca değer ile değişim yapar.

Temsil yeteneği uzak bir piksel sıralanan dizinin uçlarında kalacağından (hiç bir zaman ortada bulunmayacaktır) oradaki komşuların genel temsilini etkilemesi imkansız hale gelmiş olur. Komşu piksellerin birinin değeri olması gerektiği için, kenar boyunca hareket ettiğinde gerçekçi olmayan piksel değerleri oluşturmaz. Bu nedenle, medyan filtre, keskin kenarları ortalama filtreden daha iyi korur

```

public static Bitmap MedianFilter(this Bitmap sourceBitmap, int matrixSize)
{
    BitmapData sourceData = sourceBitmap.LockBits(new Rectangle(0, 0, sourceBitmap.Width,
sourceBitmap.Height), ImageLockMode.ReadOnly, PixelFormat.Format32bppArgb);

    byte[] pixelBuffer = new byte[sourceData.Stride * sourceData.Height];

    byte[] resultBuffer = new byte[sourceData.Stride * sourceData.Height];

    Marshal.Copy(sourceData.Scan0, pixelBuffer, 0, pixelBuffer.Length);

    sourceBitmap.UnlockBits(sourceData);

    int filterOffset = (matrixSize - 1) / 2;
    int calcOffset = 0;

    int byteOffset = 0;

    List<int> neighbourPixels = new List<int>();
    byte[] middlePixel;

    for (int offsetY = filterOffset; offsetY < sourceBitmap.Height - filterOffset; offsetY++)
    {
        for (int offsetX = filterOffset; offsetX < sourceBitmap.Width - filterOffset; offsetX++)
        {
            byteOffset = offsetY * sourceData.Stride + offsetX * 4;

            neighbourPixels.Clear();

            for (int filterY = -filterOffset;
                filterY <= filterOffset; filterY++)
            {
                for (int filterX = -filterOffset; filterX <= filterOffset; filterX++)
                {
                    calcOffset = byteOffset + (filterX * 4) + (filterY * sourceData.Stride);

                    neighbourPixels.Add(BitConverter.ToInt32(pixelBuffer, calcOffset));
                }
            }

            neighbourPixels.Sort();

            middlePixel = BitConverter.GetBytes(neighbourPixels[filterOffset]);

            resultBuffer[byteOffset] = middlePixel[0];
            resultBuffer[byteOffset + 1] = middlePixel[1];
            resultBuffer[byteOffset + 2] = middlePixel[2];
            resultBuffer[byteOffset + 3] = middlePixel[3];
        }
    }

    Bitmap resultBitmap = new Bitmap(sourceBitmap.Width, sourceBitmap.Height);

    BitmapData resultData = resultBitmap.LockBits(new Rectangle(0, 0, resultBitmap.Width,
resultBitmap.Height), ImageLockMode.WriteOnly, PixelFormat.Format32bppArgb);

    Marshal.Copy(resultBuffer, 0, resultData.Scan0, resultBuffer.Length);

    resultBitmap.UnlockBits(resultData);

    return resultBitmap;
}

```

Filtreleme işlemleri ExtBitmap C# sınıfı içerisinde tanımlanmıştır. 3x3 Matrix boyutu kullanılmıştır

```
namespace akilgicesistemleri
{
    public static class ExtBitmap
    {
        public static Bitmap CopyToSquareCanvas(this Bitmap sourceBitmap, int canvasWidthLength)...
        public static Bitmap ConvolutionFilter(this Bitmap sourceBitmap, double[,] xFilterMatrix, double[,] yFilterMatrix, int bias = 0, bool grayscale = false)...
        public static Bitmap Sobel3x3Filter(this Bitmap sourceBitmap, bool grayscale = true)...
        public static Bitmap MedianFilter(this Bitmap sourceBitmap, int matrixSize)...
        public static Bitmap OpenMorphologyFilter(this Bitmap sourceBitmap, int matrixSize, bool applyBlue = true, bool applyGreen = true, bool applyRed = true)...
        public static Bitmap CloseMorphologyFilter(this Bitmap sourceBitmap, int matrixSize, bool applyBlue = true, bool applyGreen = true, bool applyRed = true)...
        public static Bitmap DilateAndErodeFilter(this Bitmap sourceBitmap, int matrixSize, MorphologyType morphType, bool applyBlue = true, bool applyGreen = true, bool applyRed = true)...
        public enum MorphologyType...
    }
}
```

Resim3-ExtBitmap sınıfı genel görünümü



Resim4- Median filtrenin uygulaması

3.3 Sobel Filtreleme

Median filtre uygulanarak gürültüsü azaltılmış, keskin geçişler azaltılacaktır. Görüntü üzerinde kenar bulma işlemi için sobel filtresi kullanılacaktır. Dikey, yatay ve köşegen şeklindeki kenarları bulmak için kullanılacaktır. Filtreleme için gerekli matrisler aşağıdadır.

```
class Matrix
{
    public static double[,] Sobel3x3Horizontal
    {
        get
        {
            return new double[,]
            {
                { -1, 0, 1 },
                { -2, 0, 2 },
                { -1, 0, 1 },
            };
        }
    }

    public static double[,] Sobel3x3Vertical
    {
        get
        {
            return new double[,]
            {
                { -1, -2, -1 },
                { 0, 0, 0 },
                { 1, 2, 1 },
            };
        }
    }
}
```

////////////////////// SOBEL ////////////////////////

```
bmpsobel = ExtBitmap.Sobel3x3Filter(bmpmedian, true);
Bitmap bmpsobe1 = (Bitmap)bmpsobe.Clone();
```

```
public static Bitmap Sobel3x3Filter(this Bitmap sourceBitmap, bool grayscale = true)
{
    Bitmap resultBitmap = ExtBitmap.ConvolutionFilter(sourceBitmap,
    Matrix.Sobel3x3Horizontal, Matrix.Sobel3x3Vertical, 1.0, 0, grayscale);
    return resultBitmap;
}
```

```

public static Bitmap ConvolutionFilter(this Bitmap sourceBitmap, double[,]  

xFilterMatrix,double[,] yFilterMatrix, double factor = 1,int bias = 0, bool grayscale  

= false)  

{  

    BitmapData sourceData = sourceBitmap.LockBits(new Rectangle(0, 0,sourceBitmap.Width,  

    sourceBitmap.Height),ImageLockMode.ReadOnly,PixelFormat.Format32bppArgb);  

    byte[] pixelBuffer = new byte[sourceData.Stride * sourceData.Height];  

    byte[] resultBuffer = new byte[sourceData.Stride * sourceData.Height];  

    Marshal.Copy(sourceData.Scan0, pixelBuffer, 0, pixelBuffer.Length);  

    sourceBitmap.UnlockBits(sourceData);  

    if (grayscale == true)  

    {  

        float rgb = 0;  

        for (int k = 0; k < pixelBuffer.Length; k += 4)  

        {  

            rgb = pixelBuffer[k] * 0.11f;  

            rgb += pixelBuffer[k + 1] * 0.59f;  

            rgb += pixelBuffer[k + 2] * 0.3f;  

            pixelBuffer[k] = (byte)rgb;  

            pixelBuffer[k + 1] = pixelBuffer[k];  

            pixelBuffer[k + 2] = pixelBuffer[k];  

            pixelBuffer[k + 3] = 255;  

        }  

    }  

    double blueX = 0.0;  

    double greenX = 0.0;  

    double redX = 0.0;  

    double blueY = 0.0;  

    double greenY = 0.0;  

    double redY = 0.0;  

    double blueTotal = 0.0;  

    double greenTotal = 0.0;  

    double redTotal = 0.0;  

    int filterOffset = 1;  

    int calcOffset = 0;  

    int byteOffset = 0;  

    for (int offsetY = filterOffset; offsetY <sourceBitmap.Height - filterOffset;  

    offsetY++)  

    {  

        for (int offsetX = filterOffset; offsetX <  

        sourceBitmap.Width - filterOffset; offsetX++)  

        {  

            blueX = greenX = redX = 0;  

            blueY = greenY = redY = 0;  

            blueTotal = greenTotal = redTotal = 0.0;  

            byteOffset = offsetY *sourceData.Stride +offsetX * 4;  

            for (int filterY = -filterOffset;filterY <= filterOffset; filterY++)  

            {  

                for (int filterX = -filterOffset; filterX <= filterOffset; filterX++)

```

```

        {
            calcOffset = byteOffset + (filterX * 4) + (filterY * sourceData.Stride);

            blueX += (double)(pixelBuffer[calcOffset]) * xFilterMatrix[filterY + filterOffset,
                                                                    filterX + filterOffset];
            greenX += (double)(pixelBuffer[calcOffset + 1]) * xFilterMatrix[filterY +
                                                                    filterOffset, filterX + filterOffset];

            redX += (double)(pixelBuffer[calcOffset + 2]) *
                    xFilterMatrix[filterY + filterOffset,
                                    filterX + filterOffset];

            blueY += (double)(pixelBuffer[calcOffset]) *
                    yFilterMatrix[filterY + filterOffset,
                                    filterX + filterOffset];

            greenY += (double)(pixelBuffer[calcOffset + 1]) *
                    yFilterMatrix[filterY + filterOffset,
                                    filterX + filterOffset];

            redY += (double)(pixelBuffer[calcOffset + 2]) *
                    yFilterMatrix[filterY + filterOffset,
                                    filterX + filterOffset];
        }
    }

    blueTotal = Math.Sqrt((blueX * blueX) + (blueY * blueY));
    greenTotal = Math.Sqrt((greenX * greenX) + (greenY * greenY));
    redTotal = Math.Sqrt((redX * redX) + (redY * redY));

    if (blueTotal > 255)
    { blueTotal = 255; }
    else if (blueTotal < 0)
    { blueTotal = 0; }

    if (greenTotal > 255)
    { greenTotal = 255; }
    else if (greenTotal < 0)
    { greenTotal = 0; }

    if (redTotal > 255)
    { redTotal = 255; }
    else if (redTotal < 0)
    { redTotal = 0; }

    resultBuffer[byteOffset] = (byte)(blueTotal);
    resultBuffer[byteOffset + 1] = (byte)(greenTotal);
    resultBuffer[byteOffset + 2] = (byte)(redTotal);
    resultBuffer[byteOffset + 3] = 255;
}

}

Bitmap resultBitmap = new Bitmap(sourceBitmap.Width, sourceBitmap.Height);

BitmapData resultData = resultBitmap.LockBits(new Rectangle(0, 0, resultBitmap.Width,
resultBitmap.Height), ImageLockMode.WriteOnly, PixelFormat.Format32bppArgb);

Marshal.Copy(resultBuffer, 0, resultData.Scan0, resultBuffer.Length);
resultBitmap.UnlockBits(resultData);

return resultBitmap;
}

```




Resim5-Sobel filtrelerinin uygulaması

3.4 Otsu Algoritması

Gri seviyeli görüntünün ikili seviyeye dönüştürülebilmesi için otsu algoritması kullanılacaktır. İkili seviyeye dönüştürmek için bir eşik değeri belirlenir. Bu eşik değerinin altında kalan pikseller siyaha dönüştürülürken üstünde kalan pikseller beyaza dönüştürülür. Ancak her görüntü aynı niteliklere sahip değildir dolayısıyla her görüntünün kendine has eşik değerinin hesaplanmasına ihtiyaç duyulur. Uygulamanın bu adımında, eşik değerinin tespit edilmesi için Otsu Algoritması kullanılacaktır.

Otsu Algoritması eşik değeri yöntemi mümkün olan bütün eşik değerleri için (255'e kadar yani) bütün eşik değerleri için sınıf-*i* varyansı denilen bir değeri hesaplar ve bu değerin en düşük olduğu indeksi döndürür. Sınıf-*i* varyansın minimize edilmesi derken kast edilen sınıflar önyüz (foreground) ve arkayüz (background) pikselleridir. O an incelenen renk değerinden büyük olan piksellere önyüz pikselleri; küçük olanlar arkayüz pikselleri denir. Bir renk için histogramda (*buckets*) o renkten büyük olan renklerin (*i:*) frekanslarının toplamının (*np.sum*) toplam piksel sayısına (*image_size*) bölümü bize önyüz renklerinin ağırlığını verir. Bu değerin 1 sayısından çıkarılmasıyla arkayüz ağırlığı elde edilir.

Varyans hesabı yapabilmek için önce sınıfların ortalama değerleri bulunmalıdır. Bunun için her sınıf için o sınıfa üye olan renklerin kendi değerleri ile o renkteki piksel sayısı çarpılır ve sonuç, kümeye ait toplam piksel sayısına bölünür.

Otsu algoritmasının kullanımı için .dll dosyasından yararlanılmıştır. Bu method birebir proje içerisinde kodlanmamıştır.

```
// Otsu Eşikleme

[DllImport("kernel32.dll")]
public static extern int GetTickCount();

[DllImport(@"OtsuEsikleme.dll")]
public static extern void OtsuEsikleme(ref byte pixelDizisi, ref byte esikDeger, int genislik, int yukseklik);

private System.Drawing.Point[] ToPointsArray(List<IntPoint> points)
{
    return points.Select(p => new System.Drawing.Point(p.X, p.Y)).ToArray();
}
}
```

Resim6-Otsu algoritması için dll dosyası çağırısı.

```
////////// otsu
int x, y;
int genislik = bmpsobe1.Width;
int yukseklik = bmpsobe1.Height;
byte[] pixeller = new byte[(int)genislik * yukseklik];
Bitmap resim = (Bitmap)bmpsobe1.Clone();
for (y = 0; y < yukseklik; y++)
    for (x = 0; x < genislik; x++)
        // Pixelleri kütüphanenin işleyebileceği tek boyutlu bir diziye atıyoruz.
        // Gri seviyede tüm ana renkler eşit olduğu için sadece kırmızıyı okumak gri seviye için yeterli.
        pixeller[y * genislik + x] = resim.GetPixel(x, y).R;

byte esikDeger = 0;
OtsuEsikleme(ref pixeller[0], ref esikDeger, genislik, yukseklik);
int renkk;
for (y = 0; y < yukseklik; y++)
    for (x = 0; x < genislik; x++)
    {
        renkk = pixeller[y * genislik + x]; // gri
        resim.SetPixel(x, y, Color.FromArgb(renkk, renkk, renkk)); // Gri seviyeyi argb moduna dönüştürüp resme aktarıyoruz.
    }

otsu = (Bitmap)resim.Clone();
```

Resim7-otsu algoritması çalıştırılması



Resim8-Otsu algoritması uygulaması

4. Morfolojik İşlemler

Görüntü üzerinde iskelet, imgedeki sınırlar gibi yapıların tanımlanması ve bilgi çıkarımı yapılması ve gürültü giderimi, bölütleme için matematiksel morfoloji işlemlerine ihtiyaç vardır. Morfolojik görüntü işleme 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) ve dilation (genişletme) işlemleridir. Aşındırma ikili bir görüntüde bulunan nesnelerin boyutunu seçilen yapısal elemente bağlı olarak küçültürken, genişletme nesnenin alanını artırır.

- ❖ Morfolojik işlemler enum olarak kodlama içerisinde yer almıştır.

```
public enum MorphologyType
{
    Dilation,
    Erosion
}
```

- ❖ Uygulama için otsu görüntü üzerinde morfolojik işlemler olan yayma ve aşındırma işlemleri yapılacaktır.

Morfolojik İmge İşleme – Yayma ve Aşındırma

	<p>Aşındırma (Erosion): Matematiksel morfolojinin temel operasyonlarından biridir. Ele alınan bölgenin sınır bölgelerinin aşındırılmasında kullanılmaktadır.</p>
	<p>Yayma (Dilation): Diğer bir temel morfolojik işlemdir. Ele alınan bölgenin sınırlarının genişletilmesinde kullanılmaktadır.</p>

```
otsu = (Bitmap)resim.Clone();
Bitmap bmperosion;
Bitmap bmpdilation;
Bitmap bmpclosing;

bmperosion = ExtBitmap.DilateAndErodeFilter(otsu, 3, akıllıgecissistemleri.ExtBitmap.MorphologyType.Erosion, true, true, true);
Bitmap bmperosionn = (Bitmap)bmperosion.Clone();

bmpdilation = ExtBitmap.DilateAndErodeFilter(bmperosionn, 7, akıllıgecissistemleri.ExtBitmap.MorphologyType.Dilation, true, true, true);
Bitmap bmpdilationn = (Bitmap)bmpdilation.Clone();
```

Resim9-Dilation ve Erosion İşlemlerinin Extmap sınıfından çağırılması

```

public static Bitmap DilateAndErodeFilter(this Bitmap sourceBitmap, int matrixSize, MorphologyType morphType,
bool applyBlue = true, bool applyGreen = true, bool applyRed = true)
{
    BitmapData sourceData = sourceBitmap.LockBits(new Rectangle(0, 0, sourceBitmap.Width,
sourceBitmap.Height), ImageLockMode.ReadOnly, PixelFormat.Format32bppArgb);

    byte[] pixelBuffer = new byte[sourceData.Stride * sourceData.Height];

    byte[] resultBuffer = new byte[sourceData.Stride * sourceData.Height];

    Marshal.Copy(sourceData.Scan0, pixelBuffer, 0, pixelBuffer.Length);

    sourceBitmap.UnlockBits(sourceData);

    int filterOffset = (matrixSize - 1) / 2;
    int calcOffset = 0;

    int byteOffset = 0;

    byte blue = 0;
    byte green = 0;
    byte red = 0;

    byte morphResetValue = 0;

    if (morphType == MorphologyType.Erosion)
    {
        morphResetValue = 255;
    }

    for (int offsetY = filterOffset; offsetY < sourceBitmap.Height - filterOffset; offsetY++)
    {
        for (int offsetX = filterOffset; offsetX < sourceBitmap.Width - filterOffset; offsetX++)
        {
            byteOffset = offsetY * sourceData.Stride + offsetX * 4;

            blue = morphResetValue;
            green = morphResetValue;
            red = morphResetValue;

            if (morphType == MorphologyType.Dilation)
            {
                for (int filterY = -filterOffset;
                    filterY <= filterOffset; filterY++)
                {
                    for (int filterX = -filterOffset; filterX <= filterOffset; filterX++)
                    {
                        calcOffset = byteOffset + (filterX * 4) + (filterY * sourceData.Stride);

                        if (pixelBuffer[calcOffset] > blue)
                        {
                            blue = pixelBuffer[calcOffset];
                        }

                        if (pixelBuffer[calcOffset + 1] > green)
                        {
                            green = pixelBuffer[calcOffset + 1];
                        }
                        if (pixelBuffer[calcOffset + 2] > red)
                        {
                            red = pixelBuffer[calcOffset + 2];
                        }
                    }
                }
            }
        }
    }
}

```

```

else if (morphType == MorphologyType.Erosion)
{
    for (int filterY = -filterOffset; filterY <= filterOffset; filterY++)
    {
        for (int filterX = -filterOffset; filterX <= filterOffset; filterX++)
        {
            calcOffset = byteOffset +
                (filterX * 4) +
                (filterY * sourceData.Stride);

            if (pixelBuffer[calcOffset] < blue)
            {
                blue = pixelBuffer[calcOffset];
            }

            if (pixelBuffer[calcOffset + 1] < green)
            {
                green = pixelBuffer[calcOffset + 1];
            }

            if (pixelBuffer[calcOffset + 2] < red)
            {
                red = pixelBuffer[calcOffset + 2];
            }
        }
    }

    if (applyBlue == false)
    {
        blue = pixelBuffer[byteOffset];
    }

    if (applyGreen == false)
    {
        green = pixelBuffer[byteOffset + 1];
    }

    if (applyRed == false)
    {
        red = pixelBuffer[byteOffset + 2];
    }

    resultBuffer[byteOffset] = blue;
    resultBuffer[byteOffset + 1] = green;
    resultBuffer[byteOffset + 2] = red;
    resultBuffer[byteOffset + 3] = 255;
}
}

Bitmap resultBitmap = new Bitmap(sourceBitmap.Width, sourceBitmap.Height);

BitmapData resultData = resultBitmap.LockBits(new Rectangle(0, 0, resultBitmap.Width,
resultBitmap.Height),
ImageLockMode.WriteOnly, PixelFormat.Format32bppArgb);

Marshal.Copy(resultBuffer, 0, resultData.Scan0, resultBuffer.Length);

resultBitmap.UnlockBits(resultData);

return resultBitmap;
}

```

4.1 Aşındırma İşlemi(Erosion)

Bu işlem için görüntü üzerinde n boyutlu bir çekirdek gezdirilecektir. Ortadaki n . piksel, resim üzerinde işlem yaptığımız piksele karşılık gelir. Bu n tane piksel resim üzerine konulduktan sonra, piksellerin tamamı beyaz alanla örtüştü ise yani, n tane pikselin hepsinin karşılığı olan alan beyaz ise o zaman üzerinde işlem yapılan piksel beyaz olarak işaretlenir. Eğer bu n tane pikselden herhangi biri siyah bir pikselin üzerine denk geldiyse o zaman ortadaki pikselin değeri siyah yapılır. Dikkat edilirse bu işlem ile siyah bölge genişletilirken, beyaz bölge aşındırılmış olmaktadır.

- ❖ Uygulama için birbirine ince gürültülerle bağlanmış nesneleri birbirinden ayırarak gürültü temizleme yapılmak amaçlanmıştır.

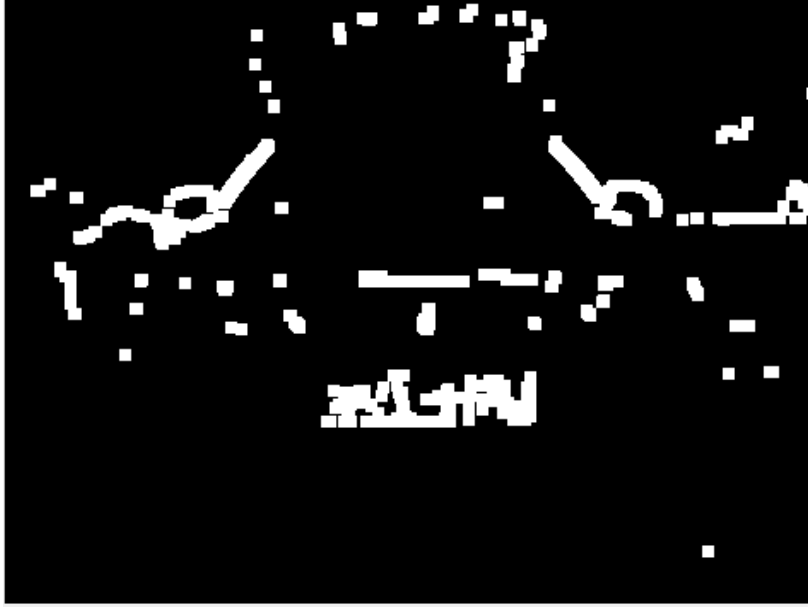


Resim9-Erozyon işleminin uygulanması

4.2 Genişletme İşlemi(Dilation)

Genişletme işlemi aynı nesnenin bir gürültü ile ince bir şekilde bölünerek ayrı iki nesne gibi görünmesini engellemek için kullanılır. Aslında aşındırma ve genişletme işlemleri birbirinin tersidir. Görüntü üzerindeki alanlarda bu işlemlerden birini uygulandığında komşu diğer alanlar zıttı olan işleme tabi tutulmuş olur. Yani aşındırma uygularken komşu alanda genişletme uygulanmış olur.

- ❖ Uygulama için genişletme işlemi aynı nesnenin bir gürültü ile ince bir şekilde bölünerek ayrı iki nesne gibi görünmesini engellemek için kullanılır.



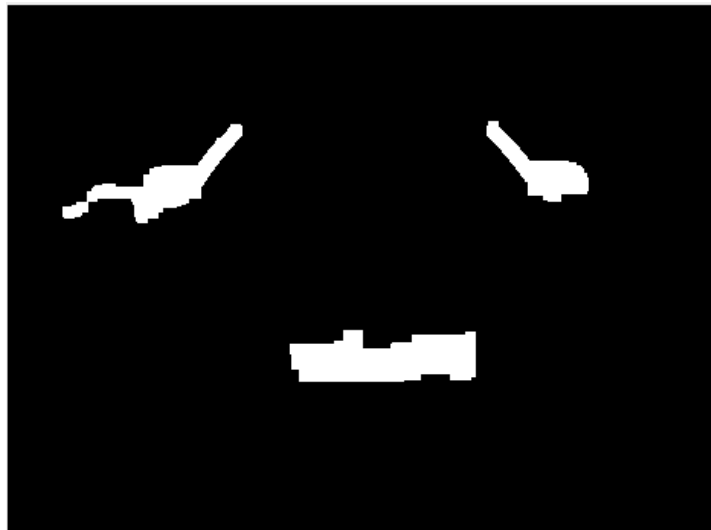
Resim10-Dilation İşlemi

4.3 Opening/Closing İşlemi

Aşındırma ve genişletme işlemlerinin ardından sırasıyla opening ve closing işlemleri yapılır. Buradaki amaç detayları ele geçirmektir. Kodlama aşamasında opening ve closing işlemlerinin yapıldığı methodlar Extmap sınıfı içinde tanımlanmıştır.

```
public static class ExtBitmap
{
    public static Bitmap CopyToSquareCanvas(this Bitmap sourceBitmap, int canvasWidthLenght)[...]
    public static Bitmap ConvolutionFilter(this Bitmap sourceBitmap, double[,] xFilterMatrix, double[,] yFilterMatrix, double factor = 1, int bias = 0, bool grayscale = false)[...]
    public static Bitmap Sobel3x3Filter(this Bitmap sourceBitmap, bool grayscale = true)
    {
        Bitmap resultBitmap = ExtBitmap.ConvolutionFilter(sourceBitmap, Matrix.Sobel3x3Horizontal, Matrix.Sobel3x3Vertical, 1.0, 0, grayscale);
        return resultBitmap;
    }
    public static Bitmap MedianFilter(this Bitmap sourceBitmap, int matrixSize)[...]
    public static Bitmap OpenMorphologyFilter(this Bitmap sourceBitmap, int matrixSize, bool applyBlue = true, bool applyGreen = true, bool applyRed = true)
    {
        Bitmap resultBitmap = sourceBitmap.DilateAndErodeFilter(matrixSize, MorphologyType.Erosion, applyBlue, applyGreen, applyRed);
        resultBitmap = resultBitmap.DilateAndErodeFilter(matrixSize, MorphologyType.Dilation, applyBlue, applyGreen, applyRed);
        return resultBitmap;
    }
    public static Bitmap CloseMorphologyFilter(this Bitmap sourceBitmap, int matrixSize, bool applyBlue = true, bool applyGreen = true, bool applyRed = true)
    {
        Bitmap resultBitmap = sourceBitmap.DilateAndErodeFilter(matrixSize, MorphologyType.Dilation, applyBlue, applyGreen, applyRed);
        resultBitmap = resultBitmap.DilateAndErodeFilter(matrixSize, MorphologyType.Erosion, applyBlue, applyGreen, applyRed);
        return resultBitmap;
    }
}
```

Resim11-Opening&Closing işleminin çağırısı



Resim 12-Closing işlem uygulamasının sonucu

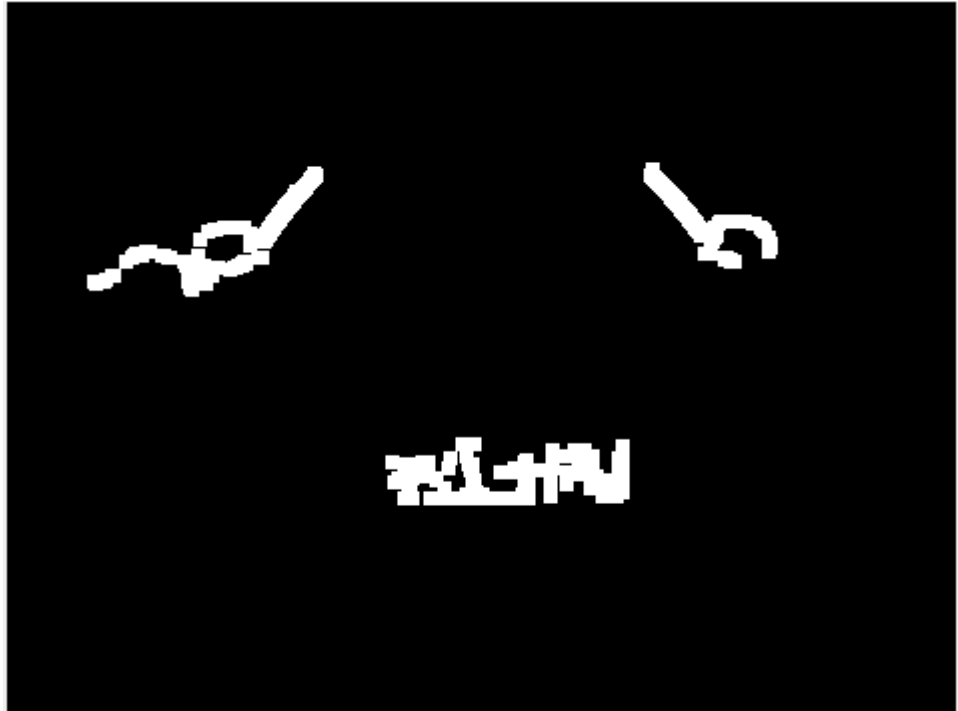
4.4 Gürültü Yok etme

Görüntü üzerinde filtreleme işlemlerinin ve morfolojik işlemlerin yapılmasının ardından kalan gürültüleri temizlemek için Aforge kütüphanesinden yararlanıldı. Görüntü üzerinde filtreleme ve morfolojik işlemlerimizi yaptıktan sonra görüntü üzerinde kalan gürültüleri temizlemek için AForge kütüphanesinin BlobsFiltering yani damla filtreleme işlemini uygulayarak verilen boyutlar dışında kalan bölgeleri temizledik.

- ❖ Filtre genişliği 70 pixel, yüksekliği 40 pixel den küçük olan gürültüleri yok etmek için kullanılmıştır.
- ❖ Üzerinde işlem yapılan görüntüde kalan noktasal gürültülerden kurtulmak amaçlanır.

```
Bitmap one = (Bitmap)bmpdilationn.Clone();
BlobsFiltering filter0 = new BlobsFiltering();
// filtre ayarlaması
filter0.CoupledSizeFiltering = true;
filter0.MinWidth = 70;
filter0.MinHeight = 40;
// filtre uygula
filter0.ApplyInPlace(one);
```

Resim13-Gürültü temizleme kod parçasığı



Resim14-Gürültü temizleme işlem sonucu

5. Aforge Kütüphanesi ve Damla Filtreleme İşlemi

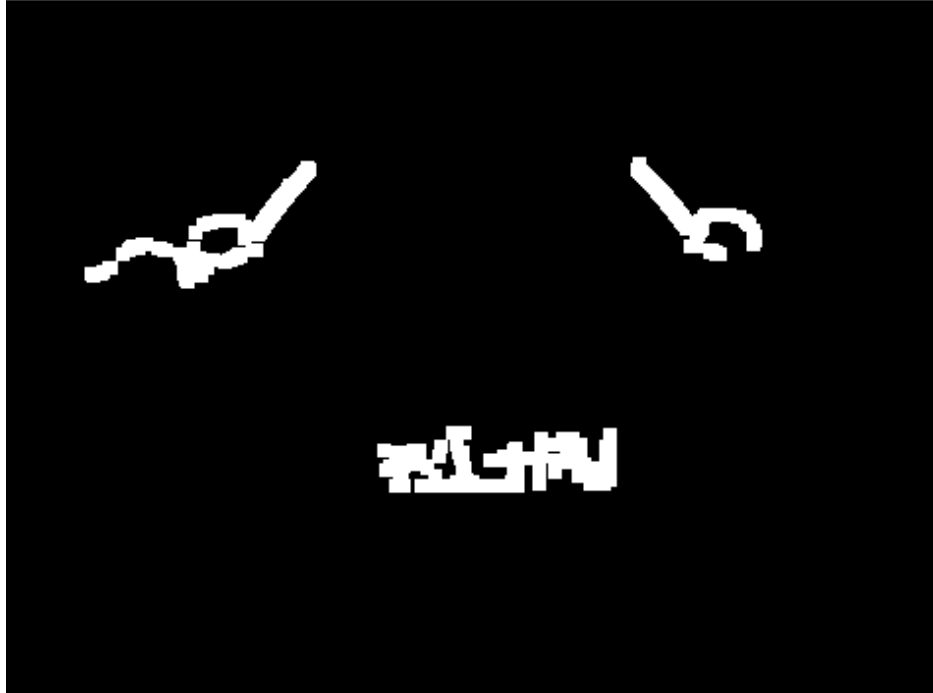
Aforge kütüphanesi görüntü işleme alanında kullanılan açık kaynak kodlu bir .NET kütüphanesidir. Görüntü üzerinde manuel olarak ya da otomatik olarak matematik işlemler yapılmasına olanak sağlar. Filtrelemeden sinir ağı hesaplamalarına değin pek çok alanda kolaylıklar sağlar.

- ❖ Uygulama içinde, Aforge kütüphanesi yardımıyla gürültü temizleme işlemi için damla filtre metodu kullanılacaktır.

Damla filtre yardımıyla, verilen boyutlardan küçük olan gürültüler temizlenir. Kütüphane bunun için BlobsFiltering metodunu kullanıma sunar.

```
bmperosion = ExtBitmap.DilateAndErodeFilter(otsu, 3, akıllıgecissistemleri.ExtBitmap.MorphologyType.Erosion, true, true, true);  
Bitmap bmperosionn = (Bitmap)bmperosion.Clone();  
  
bmpdilation = ExtBitmap.DilateAndErodeFilter(bmperosionn, 7, akıllıgecissistemleri.ExtBitmap.MorphologyType.Dilation, true, true, true);  
Bitmap bmpdilationn = (Bitmap)bmpdilation.Clone();  
  
Bitmap one = (Bitmap)bmpdilationn.Clone();  
BlobsFiltering filter0 = new BlobsFiltering();  
// filtre ayarlaması  
filter0.CoupledSizeFiltering = true;  
filter0.MinWidth = 70;  
filter0.MinHeight = 40;  
// filtre uygula  
filter0.ApplyInPlace(one);
```

Resim11-Dilation uygulanan resme damla filtresinin uygulanması



Resim 12-Damla filtresinin resim üzerindeki etkisi

6. Filtre Uygulama İşlemlerinin Özeti

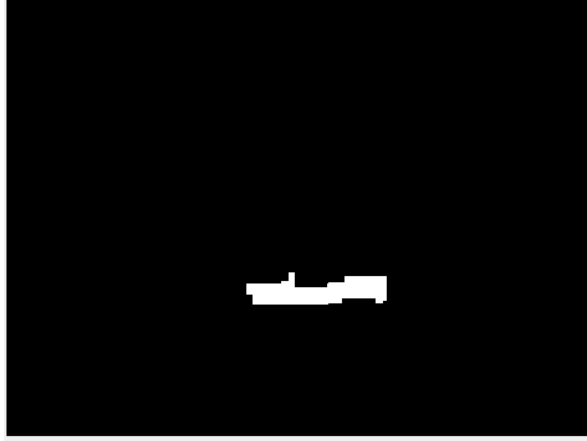
Aforge kütüphanesinin yardımıyla, otsu algoritması için kullanılan dll dosyasının yardımıyla ve sobel, median filtrelerinin tanımlayıcı olarak kodlanmasıyla ardarda pek çok filtreleme işlemi yapılmıştır. Yapılan işlemler özetle sıralanacak olursa aşağıdaki gibidir. Ardarda pek çok filtreleme işlemi yaparak plaka görüntüsünün elde edilmesine uygun bir taslak hazırlanmaya çalışılmıştır.

7. Plaka Bilgisinin Araştırılması

Görüntü üzerinde gürültü temizleme ve filtreleme işlemlerinin yapılmasının ardından plaka yerinin araştırılması işlemine geçilir. Bunun için Aforge kütüphanesinden yararlanılır ve plakanın koordinat bilgileri elde edilmeye çalışılacaktır.

Türkiye’deki araç plakalarının birçoğunun görünümü dikdörtgen şeklindedir. Aforge kütüphanesi yardımıyla elde edilen plaka bölgesinin gerçekten plaka boyutlarına uygun olup olmadığını anlayabilmek için boyut kontrolü yapılmalıdır. Plaka boyutları ön taraf için genellikle 11x52 iken arka taraf için 21x32’dir. Sonuç olarak elde edilen görüntü plaka bilgisi olarak kullanıcıya sunulacaktır.

❖ Uygulama sürecinde plaka tespiti yalnızca aracın ön tarafındaki plaka için yapılmıştır.



Resim13-Plaka bölgesinin çizimi

```

ConnectedComponentsLabeling filter = new ConnectedComponentsLabeling();
// filtre uygula
Bitmap newImage = filter.Apply(one1);
Bitmap newImage1 = (Bitmap)newImage.Clone();
// obje sayısını kontrol et
int objectCount = filter.ObjectCount;

BlobCounter blobCounter = new BlobCounter();
blobCounter.ProcessImage(rect);
Blob[] blobs = blobCounter.GetObjectsInformation();
// Görüntüye çizmek için Graphic nesnesi ve bir kalem oluşturun
Graphics g = Graphics.FromImage(newImage);
Pen bluePen = new Pen(Color.Blue, 2);
// her nesne kptrol edilir ve etrafında daire çizilir.

for (int i = 0, n = blobs.Length; i < n; i++)
{
    /*
    * x1=0
    * x2=1
    * y1=2
    * y2=3
    */
    List<IntPoint> edgePoints = blobCounter.GetBlobsEdgePoints(blobs[i]);
    List<IntPoint> corners = PointsCloud.FindQuadrilateralCorners(edgePoints);
    int[] sinirlar = sinir(corners);
    sinirlar[0] = sinirlar[0] - 2;
    sinirlar[1] = sinirlar[1] + 2;
    sinirlar[2] = sinirlar[2] - 2;
    sinirlar[3] = sinirlar[3] + 2;
    int en = sinirlar[1] - sinirlar[0];
    int boy = sinirlar[3] - sinirlar[2];
    float ort = (float)en / (float)boy;

    List<IntPoint> ucnoktalar = new List<IntPoint>();
    ucnoktalar.Add(new IntPoint(sinirlar[0], sinirlar[2]));
    ucnoktalar.Add(new IntPoint(sinirlar[1], sinirlar[2]));
    ucnoktalar.Add(new IntPoint(sinirlar[1], sinirlar[3]));
    ucnoktalar.Add(new IntPoint(sinirlar[0], sinirlar[3]));
    g.DrawPolygon(bluePen, ToPointsArray(ucnoktalar));
    g.DrawString("Plaka Koordinatları : (x,y): (" + sinirlar[0].ToString() + "," + sinirlar[2].ToString() + ") \n en, boy, ort: " + (sinirlar[1] - sinirlar[0]).ToString() + ", "
    + (sinirlar[3] - sinirlar[2]).ToString() + ", " + ort.ToString() + " blob sayisi: " + blobs.Length.ToString(), new Font("Arial", 8), Brushes.White, new System.Drawing.Point(sinirlar[0], sinirlar[3] + 4));

    bluePen.Dispose();
    g.Dispose();
}

```

Resim14- Plaka bölgesinin çizilmesi için gerekli kod bloğu

```

Bitmap rect1 = (Bitmap)pictureBox1.Image.Clone();
Graphics g1 = Graphics.FromImage(rect1);
Pen bluePen2 = new Pen(Color.Red, 2);
// her nesne kptrol edilir ve etrafında daire çizilir.

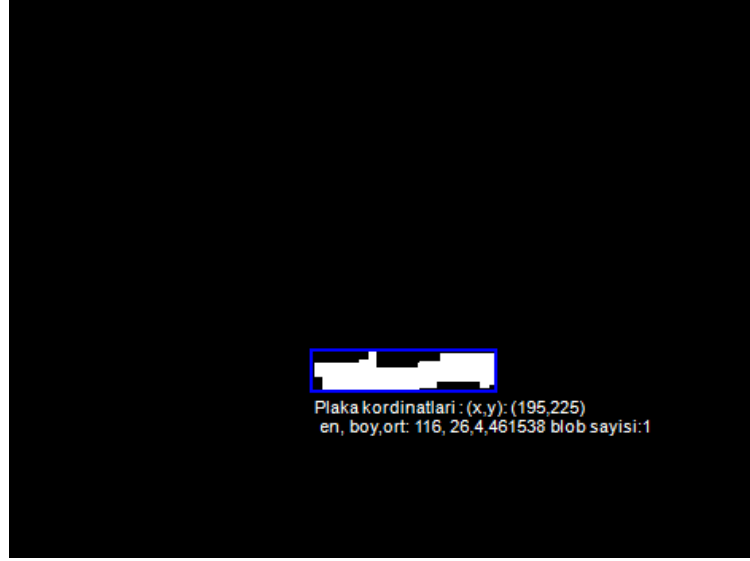
List<Blob> bloplar = new List<Blob>();
for (int i = 0, n = blobs.Length; i < n; i++)
{
    /*
    * x1=0          x1,y1-----x2,y1
    * x2=1          |               |
    * y1=2          x1,y2-----x2,y2
    * y2=3
    */
    List<IntPoint> edgePoints = blobCounter.GetBlobsEdgePoints(blobs[i]);
    List<IntPoint> corners = PointsCloud.FindQuadrilateralCorners(edgePoints);
    int[] sinirlar = sinir(corners);
    sinirlar[0] = sinirlar[0] - 5;
    sinirlar[1] = sinirlar[1];
    sinirlar[2] = sinirlar[2] - 5;
    sinirlar[3] = sinirlar[3] + 5;
    int en = sinirlar[1] - sinirlar[0];
    int boy = sinirlar[3] - sinirlar[2];
    float ort = (float)en / (float)boy;
    if (ort >= 3 && ort <= 5.7)
    {
        g1.DrawLine(bluePen2, new System.Drawing.Point[] { new System.Drawing.Point(sinirlar[0], sinirlar[2]),
        new System.Drawing.Point(sinirlar[1], sinirlar[2]), new System.Drawing.Point(sinirlar[1], sinirlar[3]),
        new System.Drawing.Point(sinirlar[0], sinirlar[3]), new System.Drawing.Point(sinirlar[0], sinirlar[2]) });

        g1.DrawString("Plaka Koordinatları : (x,y): (" + sinirlar[0].ToString() + "," + sinirlar[2].ToString() + ") \n en, boy, ort: " + (sinirlar[1] - sinirlar[0]).ToString() + ", "
        + (sinirlar[3] - sinirlar[2]).ToString() + ", " + ort.ToString() + " blob sayisi: " + blobs.Length.ToString(), new Font("Arial", 8), Brushes.White, new System.Drawing.Point(sinirlar[0], sinirlar[3] + 4));

        bluePen2.Dispose();
        g1.Dispose();
    }
}

```

Resim15-Plaka bölge sınırlarının belirlenmesi için gerekli kod bloğu



Resim16-Plaka bölgesinin dikdörtgen şekilde gösterimi

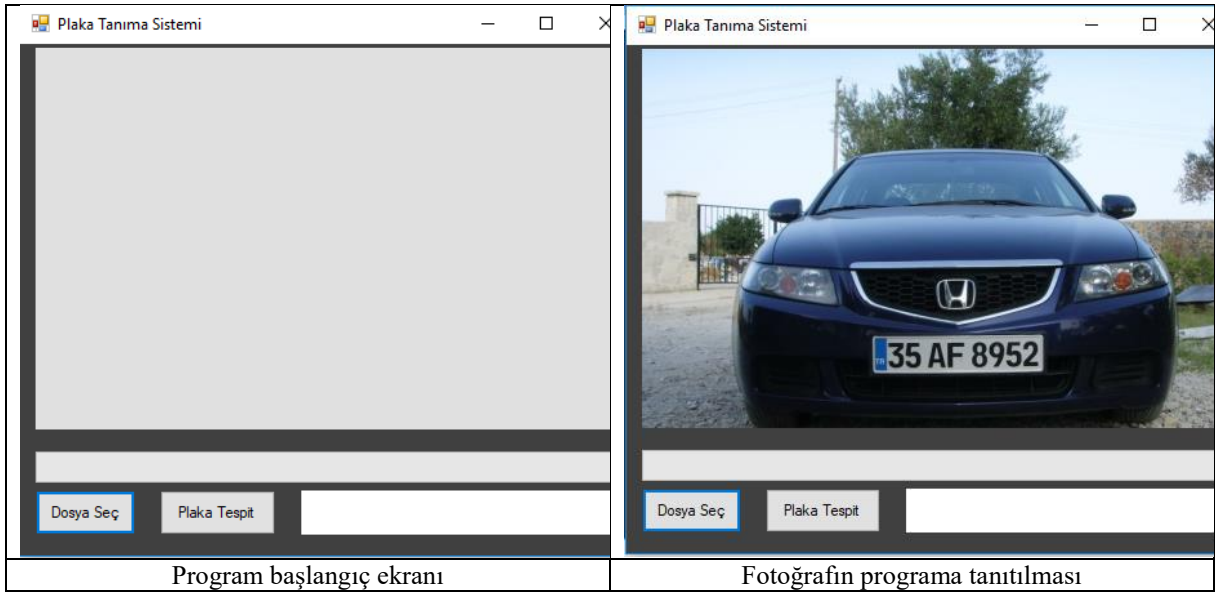
8. Özet

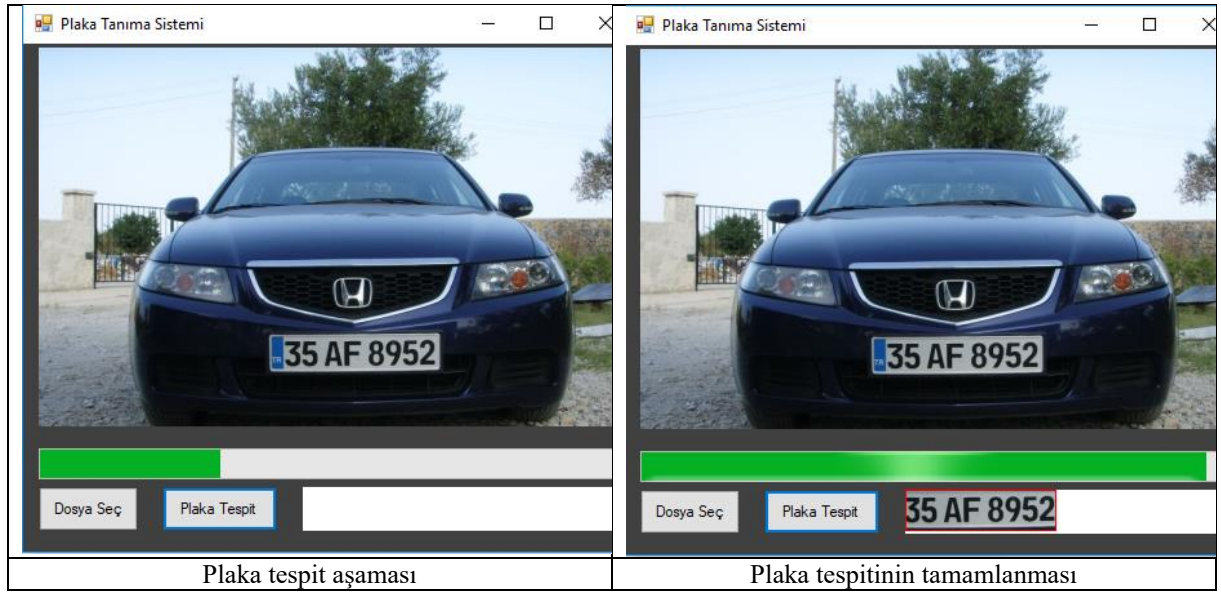
Görüntünün gri seviyeye indirgenmesi, yumuşatma işlemi, kenar bulma filtesinin(sobel) kullanımı, aşındırma ve genişletme işlemlerinin yapılmasının ardından Aforge kütüphanesi yardımıyla plaka bölgesinin araştırılması ve belirlenmesi gerçekleştirilmiştir.

❖ Proje C# programlama dili kullanılarak masaüstü uygulaması olarak gerçekleştirilmiştir.

Programın Kullanım Adımları

- Kullanıcı Dosya seç butonunu kullanarak, plakasını tanımak istediği fotoğrafın fiziksel yolunu programa tanıtır.
- Fotoğrafın yüklenmesinin ardından, plaka tespit butonu tıklanarak plaka tanıma işlemi başlatılır.
- Plaka tanıma işlemi tamamlandığında, kullanıcıya plaka gösterimi yapılır.





Tablo1-Programın çalışma sürecinin gösterilmesi

Filtre işlemlerinin gösterimi





Proje Kaynak Kodu: <https://github.com/NisanurBulut/PlakaTanimaSistemi>

Kaynakça

1. <https://trafik.net.tr/plaka-tanima-sistemi-nedir/>
2. <http://www.hobibilisim.com/pts-nedir/>
3. <http://www.barissamanci.net/Makale/20/goruntu-isleme-teknikleri-gercek-zamanli-plaka-tanima-sistemi/>
4. <http://www.aforgenet.com/framework/docs/html/4a83d944-d776-ba2c-9847-3254fe3dbfdd.htm>
5. <https://www.bulentsiyah.com/goruntunun-sinir-egrisini-cikaran-filtreler-matlab/>
6. <https://yavuzbugra.wordpress.com/2011/05/01/goruntu-islemeye-filtreleme/>
7. <http://bilgisayarkavramlari.sadievrenseker.com/2007/11/26/ortanca-filtresi-median-filter/>
8. <http://bilgisayarkavramlari.sadievrenseker.com/2008/11/17/aritmetik-ortalama-average-mean/>
9. <http://huseyinasoy.com/Otsu-Esik-Belirleme-Metodu>
10. <http://embedded.kocaeli.edu.tr/otsu-metodu/>
11. <https://medium.com/@sddkal/python-ile-g%C3%B6r%C3%BCnt%C3%BC-%C4%B0%C5%9Fleme-mean-ve-median-filtreler-1891cdebef632>
12. <http://www.plaka.com.tr/plaka-gorunumu.html>
13. <http://guraysonugur.aku.edu.tr/wp-content/uploads/sites/22/2017/05/G%C3%B6r%C3%BCnt%C3%BC-%C4%B0%C5%9Fleme-Ders-9.1.pdf>
14. <https://slideplayer.biz.tr/slide/10376908/>
15. <http://www.cescript.com/2012/08/morfolojik-goruntu-isleme.html>