Software Engineering Application

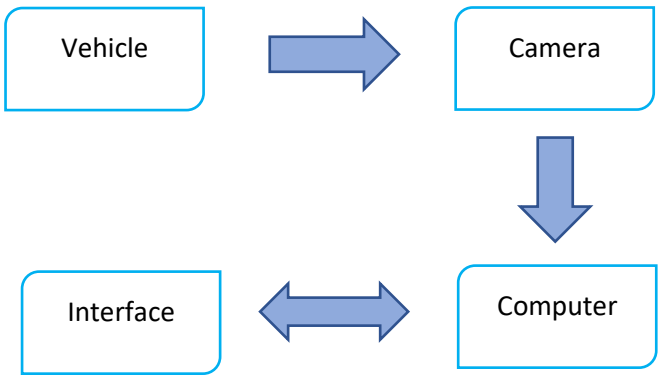
License Plate Recognition System

Zeynep Gizem Çetinci B1605.010034
Nurşah Demirpolat B1605.010034
Gülser Tuğçe Örke B1605.010022
Shereen Mohammad B1705.010074

Vehicle License Plate Recognition System

Turkey legally registered to the traffic supervisory authorities plates; civil, official, military, diplomatic, etc. There are different types such as and all types of different colors and formats (Traffic Control, 2010). In this study, it is aimed to recognize civilian plates that meet Turkish license plate standards. The general feature of these plates; It consists of black characters on a white background, the numbers in the first two characters indicating the city code, and the characters that come after it consist of random letters and numbers.

As seen in the block diagram, it basically consists of 4 main sections.



Vehicle license plate recognition system block diagram

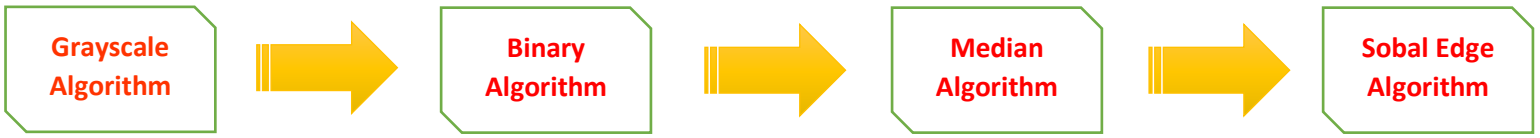
Vehicle: Recognition of the license plate is based on the passage of a vehicle by using algorithms written from the camera where the system is installed.

Camera: The image of the vehicle arriving at the place where the system is installed is taken by camera. The video resolution of the camera used should be 1024X768.

Computer: The performance of the computer used should be sufficient for the system to operate fast and the entrance port to the parallel port of the computer. It must be connected.

Interface Software: The interface program written for the system must be installed on the computer. The *.Net Framework* infrastructure must be installed on the computer for this interface to work.

Taking the Picture from the Camera



In the study, it has been converted to grayscale to work more comfortably and faster on color images taken from the camera.

1) *Grayscale* is the gray scale created by the fact that each color gets as many shades of gray as the tone. In this gray image, a gray scale is obtained by averaging the color value in each pixel. *Gray scaling;* P is given in Equation 1 to indicate an image, i and j in coordinates.

$$Avg = (P(i,j).Red+P(i,j).Green+P(i,j).Blue)/3$$

$$P(i,j).Red = Avg$$

$$P(i,j).Green = Avg$$

$$P(i,j).Blue = Avg$$

Color Image *Grayscale Image*

- It is a color image taken from the camera and converted to grayscale.

Grayscale Algorithm

```
private Bitmap grayLevel(Bitmap bmp)
{
        for (int i = 0; i < bmp.Height - 1; i++)
        {
           for (int j = 0; j < bmp.Width - 1; j++)
           {
              int value =  (bmp.GetPixel(j, i).R + bmp.GetPixel(j, i).G + bmp.GetPixel(j, i).B) / 3;
             Color renk;
              renk = Color.FromArgb(value, value, value);
             bmp.SetPixel(j, i, renk);
           }
        }
        return bmp;
}
```

2) *Binary Algorithm* is important in image processing to extract objects from their background into binary image. Binary image is used as input to feature extraction process and have an important role in generating unique feature to distinguish several classes in pattern recognition. This paper proposes an image processing algorithm to obtain a binary image from RGB. The results showed that the binary image of the proposed algorithm contained the desired object.

Grayscale Image *Binary Image*

- The image converted to grayscale and the binary image of that photo are as above.

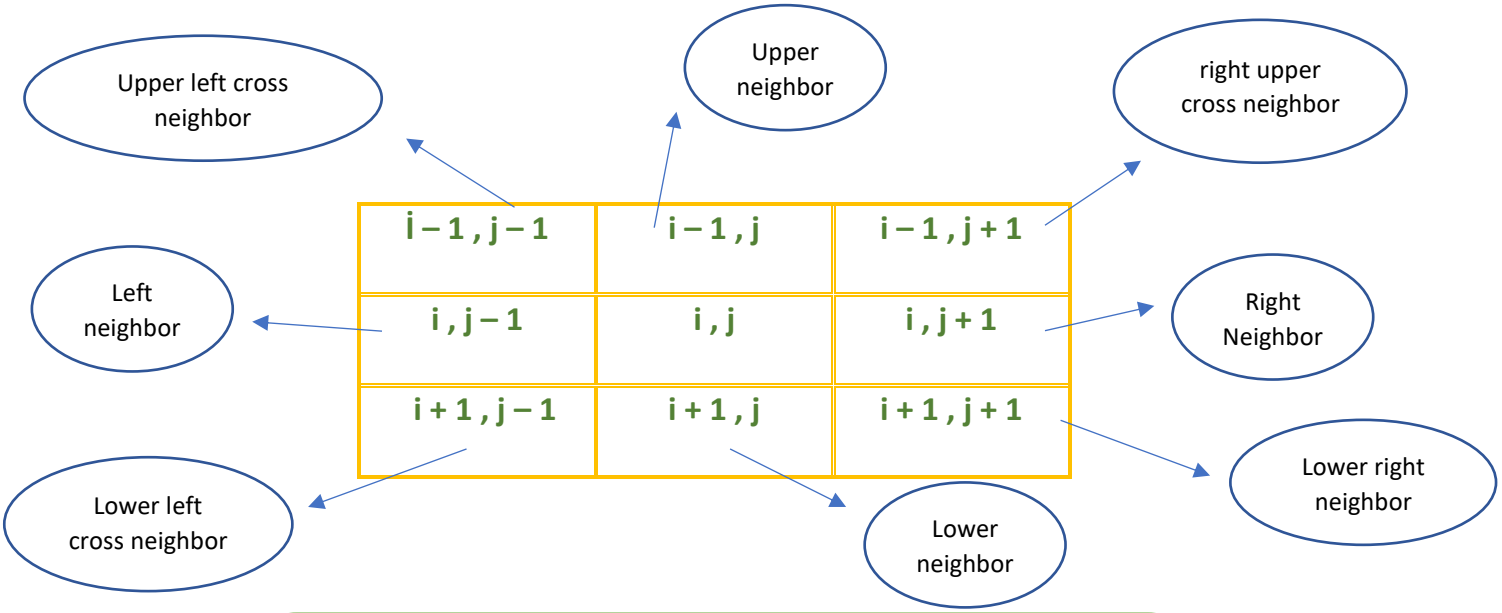
Binary Algorithm

```
private Bitmap Makebinary(Bitmap image)
{
        try
        {
           grayLevel(image);
           int temp;
           int level = 127;
           Color color;
           for (int i = 0; i < image.Height - 1; i++)   {
              for (int j = 0; j < image.Width - 1; j++){
                 temp = image.GetPixel(j, i).B;
                 if (temp < level){
                    color = Color.FromArgb(0, 0, 0);
                    image.SetPixel(j, i, color);
                 }
                 else{
                    color = Color.FromArgb(255, 255, 255);
                    image.SetPixel(j, i, color);
                 }
              } } return image;   }
}
```

3) *Median filters,* The average filter is the simplest filter that can be created with the help of a kernel. For each pixel, the surrounding pixels are averaged. Mean filters are ideal for restoring images with Gaussian noise. However, instead of taking the average here, it is assigned to the pixel, which is the median of the values in the neighborhood. Median filters are ideal for restoring images with impulsive noise.

Binary Image *Median Image*

- The image converted to binary image and the median image of that photo are as above.
- The median finding algorithm is like this. We try to find the median by crossing the right and left with i and j.




```
rightNeighbor = image.GetPixel(j + 1, i).R;
rightUpperCrossNeighbor = image.GetPixel(j + 1, i - 1).R;
upperNeighbor = image.GetPixel(j, i - 1).R;
upperLeftCrossNeighbor = image.GetPixel(j - 1, i - 1).R;
leftNeighbor = image.GetPixel(j - 1, i).R;
lowerLeftCrossNeighbor = image.GetPixel(j - 1, i + 1).R;
lowerNeighbor = image.GetPixel(j, i + 1).R;
lowerRightNeighbor = image.GetPixel(j + 1, i + 1).R;
```

Median Filter Algorithm

```
private Bitmap medianAlgorithm(Bitmap image)
        {
            Bitmap buffer = new Bitmap(image.Width, image.Height);
            Color color;

            for (int i = 0; i < image.Height; i++)  {
                for (int j = 0; j < image.Width; j++){
                    if ((i == 0) || (i == image.Height - 1) || (j == 0) || (j ==
image.Width - 1))
                        continue;
                    else{
                        int ortanca = medianFind(image, j, i);
                        color = Color.FromArgb(ortanca, ortanca, ortanca);
                        buffer.SetPixel(j, i, color);
                    }
                }
            }
            return buffer;
        }

        private int medianFind(Bitmap image, int j, int i)
        {
            int[] dizi = new int[9];
            Color color;
            int sagkomsu, sagustcaprazkomsu, ustkomsu, solustcapraz, solkomsu,
solaltcapraz, altkomsu, sagaltcapraz;

            sagkomsu = image.GetPixel(j + 1, i).R;
            sagustcaprazkomsu = image.GetPixel(j + 1, i - 1).R;
            ustkomsu = image.GetPixel(j, i - 1).R;
            solustcapraz = image.GetPixel(j + 1, i - 1).R;
            solkomsu = image.GetPixel(j - 1, i).R;
            solaltcapraz = image.GetPixel(j - 1, i + 1).R;
            altkomsu = image.GetPixel(j, i + 1).R;
            sagaltcapraz = image.GetPixel(j + 1, i + 1).R;

            dizi[0] = image.GetPixel(j, i).R;
            dizi[1] = sagkomsu;
            dizi[2] = sagustcaprazkomsu;
            dizi[3] = ustkomsu;
            dizi[4] = solustcapraz;
            dizi[5] = solkomsu;
            dizi[6] = solaltcapraz;
            dizi[7] = altkomsu;
            dizi[8] = sagaltcapraz;

            for (int x = 0; x < 8; x++){
                for (int y = x + 1; y < 9; y++){
                    if (dizi[x] < dizi[y])
                        continue;
                    else{
                        int temp = dizi[y];
                        dizi[y] = dizi[x];
                        dizi[x] = temp;
                    }
                }

          }
            return dizi[4];
        }
```

4) *Thresholding image*; The simplest thresholding methods replace each pixel in an image with a black pixel if the image intensity $I_{i,j}$ is less than some fixed constant T or a white pixel if the image intensity is greater than that constant. In the example image on the right, this results in the dark tree becoming completely black, and the white snow becoming completely white.

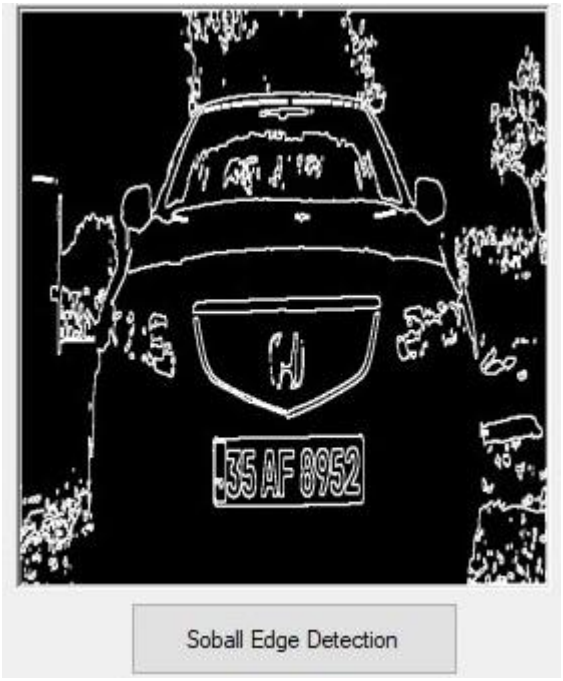
Thresholding Algorithm

```
public Bitmap ResmiEsiklemeYap(Bitmap GirisResmi)
        {
            Color OkunanRenk, DonusenRenk;
            int R = 0, G = 0, B = 0;
            int ResimGenisligi = GirisResmi.Width; //GirisResmi global tanımlandı.
            int ResimYuksekligi = GirisResmi.Height;
            Bitmap CikisResmi = new Bitmap(ResimGenisligi, ResimYuksekligi); //Cikis
resmini oluşturuyor. Boyutları giriş resmi ile aynı olur.
            int i = 0, j = 0; //Çıkış resminin x ve y si olacak.
            for (int x = 0; x < ResimGenisligi; x++)
            {
                j = 0;
                for (int y = 0; y < ResimYuksekligi; y++)
                {
                    OkunanRenk = GirisResmi.GetPixel(x, y);
                    if (OkunanRenk.R >= 128)
                        R = 255;
                    else
                        R = 0;
                    if (OkunanRenk.G >= 128)
                        G = 255;
                    else
                        G = 0;
                    if (OkunanRenk.B >= 128)
                        B = 255;
                    else
                        B = 0;
                    DonusenRenk = Color.FromArgb(R, G, B);
                    CikisResmi.SetPixel(i, j, DonusenRenk);
                    j++;
                }
                i++;
            }
            return CikisResmi;
        }
```

5) *Sobal Edge Algorithm* is used in image processing and computer vision, particularly within edge detection algorithms where it creates an image emphasising edges. It is named after Irwin Sobel and Gary Feldman, colleagues at the Stanford Artificial Intelligence Laboratory (SAIL). Sobel and Feldman presented the idea of an "Isotropic 3x3 Image Gradient Operator" at a talk at SAIL in 1968. Technically, it is a discrete differentiation operator, computing an approximation of the gradient of the image intensity function. At each point in the image, the result of the Sobel–Feldman operator is either the corresponding gradient vector or the norm of this vector. The Sobel–Feldman operator is based on convolving the image with a small, separable, and integer-valued filter in the horizontal and vertical directions and is therefore relatively inexpensive in terms of computations. On the other hand, the gradient approximation that it produces is relatively crude, in particular for high-frequency variations in the image.

$$\mathbf{G}_x = \begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix} * \mathbf{A} \quad \text{and} \quad \mathbf{G}_y = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} * \mathbf{A}$$

Sobal Edge Algorithm

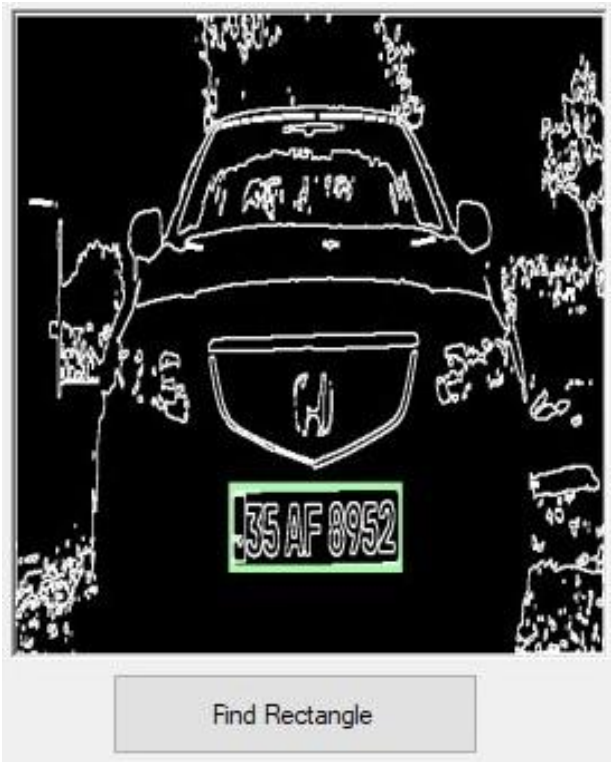



```
private Bitmap sobalEdgeDetection(Bitmap image)
{
  Bitmap buffer = new Bitmap(image.Width,
image.Height);
  Color renk;
  int valX, valY;
  int gradient;
  int[,] GX = new int[3, 3];
  int[,] GY = new int[3, 3];

  GX[0, 0] = -1; GX[0, 1] = 0; GX[0, 2] = 1;
  GX[1, 0] = -2; GX[1, 1] = 0; GX[1, 2] = 2;
  GX[2, 0] = -1; GX[2, 1] = 0; GX[2, 2] = 1;

  //Dikey yöndeki kenar için
  GY[0, 0] = -1; GY[0, 1] = -2; GY[0, 2] = -1;
  GY[1, 0] = 0; GY[1, 1] = 0; GY[1, 2] = 0;
  GY[2, 0] = 1; GY[2, 1] = 2; GY[2, 2] = 1;
```

```
for (int i = 0; i < image.Height; i++){
   for (int j = 0; j < image.Width; j++){
        if (i == 0 || i == image.Height - 1 || j == 0 || j == image.Width - 1){
           renk = Color.FromArgb(255, 255, 255);
           buffer.SetPixel(j, i, renk);
           valX = 0;
           valY = 0;
         }
         else{
           valX = image.GetPixel(j - 1, i - 1).R * GX[0, 0] +
           image.GetPixel(j, i - 1).R * GX[0, 1] +
           image.GetPixel(j + 1, i - 1).R * GX[0, 2] +
           image.GetPixel(j - 1, i).R * GX[1, 0] +
           image.GetPixel(j, i).R * GX[1, 1] +
           image.GetPixel(j + 1, i).R * GX[1, 2] +
           image.GetPixel(j - 1, i + 1).R * GX[2, 0] +
           image.GetPixel(j, i + 1).R * GX[2, 1] +
           image.GetPixel(j + 1, i + 1).R * GX[2, 2];

           valY = image.GetPixel(j - 1, i - 1).R * GY[0, 0] +
           image.GetPixel(j, i - 1).R * GY[0, 1] +
           image.GetPixel(j + 1, i - 1).R * GY[0, 2] +
           image.GetPixel(j - 1, i).R * GY[1, 0] +
           image.GetPixel(j, i).R * GY[1, 1] +
           image.GetPixel(j + 1, i).R * GY[1, 2] +
           image.GetPixel(j - 1, i + 1).R * GY[2, 0] +
           image.GetPixel(j, i + 1).R * GY[2, 1] +
           image.GetPixel(j + 1, i + 1).R * GY[2, 2];
```

```
            gradient = (int)(Math.Abs(valX) + Math.Abs(valY));

        if (gradient < 0)
           gradient = 0;
        if (gradient > 255)
           gradient = 255;

        renk = Color.FromArgb(gradient, gradient, gradient);
        buffer.SetPixel(j, i, renk);
    }
   }
   return buffer;
}
```

6) *Rectangular algorithm;* In the rectangular discovery algorithm after the image processing sequence is finished. Thanks to this algorithm, we will be able to easily trim the plate.

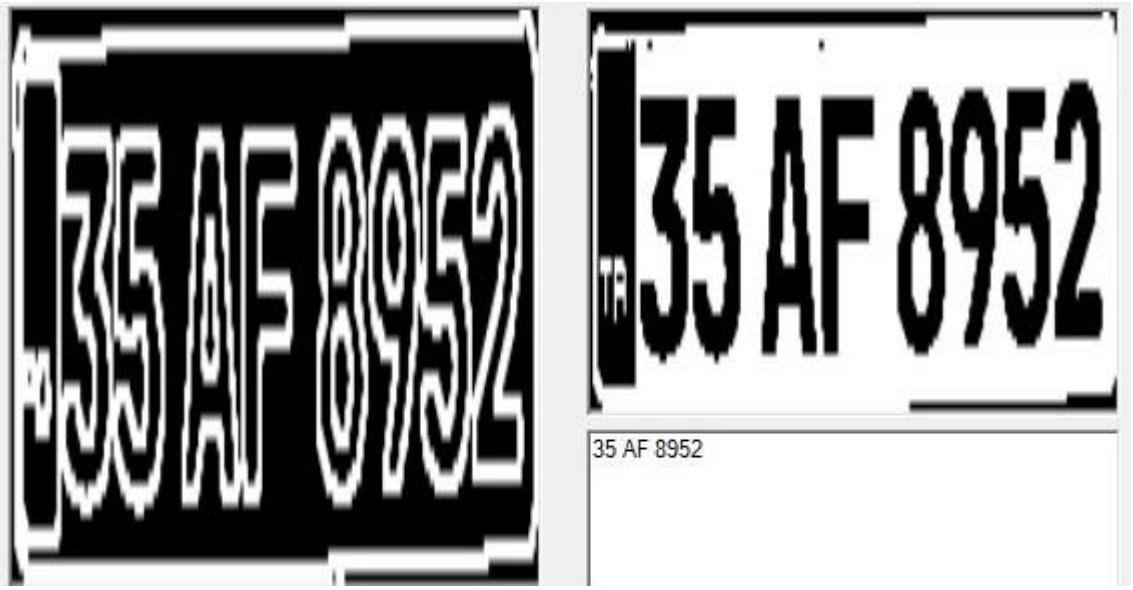


Rectangular algorithm

```
private Bitmap sobalEdgeDetection(Bitmap image){

        Bitmap İmage = (Bitmap)pictureBox5.Image;
         BlobCounter blobCounter = new BlobCounter();
         blobCounter.FilterBlobs = true;
         blobCounter.MinHeight = 10;
         blobCounter.MinWidth = 10;
         blobCounter.MaxWidth = 350;
         blobCounter.MaxHeight = 350;
         blobCounter.ProcessImage(İmage);
         Rectangle[] rects = blobCounter.GetObjectsRectangles();
```

```
Graphics g = Graphics.FromImage(İmage);
if (rects.Length > 1){
   var r2 = rects.OrderByDescending(r => r.Height * r.Width).ToList();
   Rectangle objectRect = r2[1];
   using (Pen pen = new Pen(Color.FromArgb(160, 255, 160), 5))
   {
      g.DrawRectangle(pen, objectRect);
   }
   g.Dispose();
  }
```

7) *Algorithm to crop and read the plate in the Picture,* The plate region was found at different angles and at different times of the day, and because of the difference in the resolution of the pictures depending on the intensity, direction of the light and some factors, clarification algorithms were applied to the images.



The areas outside the plate region, which we can call noise on the cropped image of the plate, were destroyed using the Hydrangea Filter. Some regions that cannot be destroyed at the end of the filter are also removed from the image by taking into account the characteristics of the characters such as pixel, alignment and ratio.

We made it possible to read the plate using the Tesseract library.Reading the characters, at this stage, the characters separated in the license plate region are compared with the samples found in a text file, and is found to be equivalent to the value closest to the character. The image above shows an image with the reserved character read.

Clipping Algorithm

```
private Bitmap sobalEdgeDetection(Bitmap image){

     Bitmap images = new Bitmap(pictureBox4.Image);
     Bitmap İmage = sobalEdgeDetection(images);
     BlobCounter blobCounter = new BlobCounter();
     blobCounter.FilterBlobs = true;
     blobCounter.MinHeight = 10;
     blobCounter.MinWidth = 10;
     blobCounter.MaxWidth = 350;
     blobCounter.MaxHeight = 350;
     blobCounter.ProcessImage(İmage);
     Rectangle[] rects = blobCounter.GetObjectsRectangles();
     if (rects.Length == 0)
     {
         System.Windows.Forms.MessageBox.Show("No rectangle found in image ");
     }
     else if (rects.Length > 1)
     {
         // get largets rect
         Console.WriteLine("Using largest rectangle found in image ");
         var r2 = rects.OrderByDescending(r => r.Height * r.Width).ToList();
         İmage = İmage.Clone(r2[1], İmage.PixelFormat);
     }
     else
     {
         Console.WriteLine("Huh? on image ");
     }
            pictureBox7.Image = İmage;
}
```

Read Algorithm

```
     Bitmap image = new Bitmap(pictureBox2.Image);
     Bitmap scr = ResmiEsiklemeYap(image);
     Bitmap sc = Select(scr);
     pictureBox8.Image = sc;
     var ocr = new TesseractEngine("./tessdata", "eng");
     var page = ocr.Process(sc);
     richTextBox1.Text = page.GetText();

 private Bitmap Select(Bitmap İmage)
 {
     BlobCounter blobCounter = new BlobCounter();
     blobCounter.FilterBlobs = true;
     blobCounter.ProcessImage(İmage);
     Rectangle[] rects = blobCounter.GetObjectsRectangles();
     if (rects.Length == 0)
     {
         System.Windows.Forms.MessageBox.Show("No rectangle found in image ");
     }
     else if (rects.Length > 1)
     {
         // get largets rect
         Console.WriteLine("Using largest rectangle found in image ");
         var r2 = rects.OrderByDescending(r => r.Height * r.Width).ToList();
         İmage = İmage.Clone(r2[3], İmage.PixelFormat);
     }
```

```
        else
        {
            Console.WriteLine("Huh? on image ");
        }
        return İmage;
    }
```

What is the Tesseract Library? How it works?

Tesseract is one of the most accurate open source OCR engines. Tesseract allows us to convert the given image into the text. Before going to the code we need to download the assembly and tessdata of the Tesseract. We can download the data from GitHub or NuGet.

Then add the following package.

```
1. using tessnet2;
2. using System.Drawing;
3. using System.Drawing.Drawing2D;
4. using System.Drawing.Imaging;
5.
6. // now add the following C# line in the code page
7. var image = new Bitmap(@ "Z:\NewProject\demo\image.bmp");
8. varocr = new Tesseract();
9. ocr.Init(@ "Z:\NewProject\How to use Tessnet2 library\C#\tessda
   ta", "eng", false);
10.     var result = ocr.DoOCR(image, Rectangle.Empty);
11.     foreach(tessnet2.Word word in result)
12.     {
13.         Console.writeline(word.text);
14.     }
```

Tesseract — is an optical character recognition engine with open-source code, this is the most popular and qualitative OCR-library.

OCR uses artificial intelligence for text search and its recognition on images.

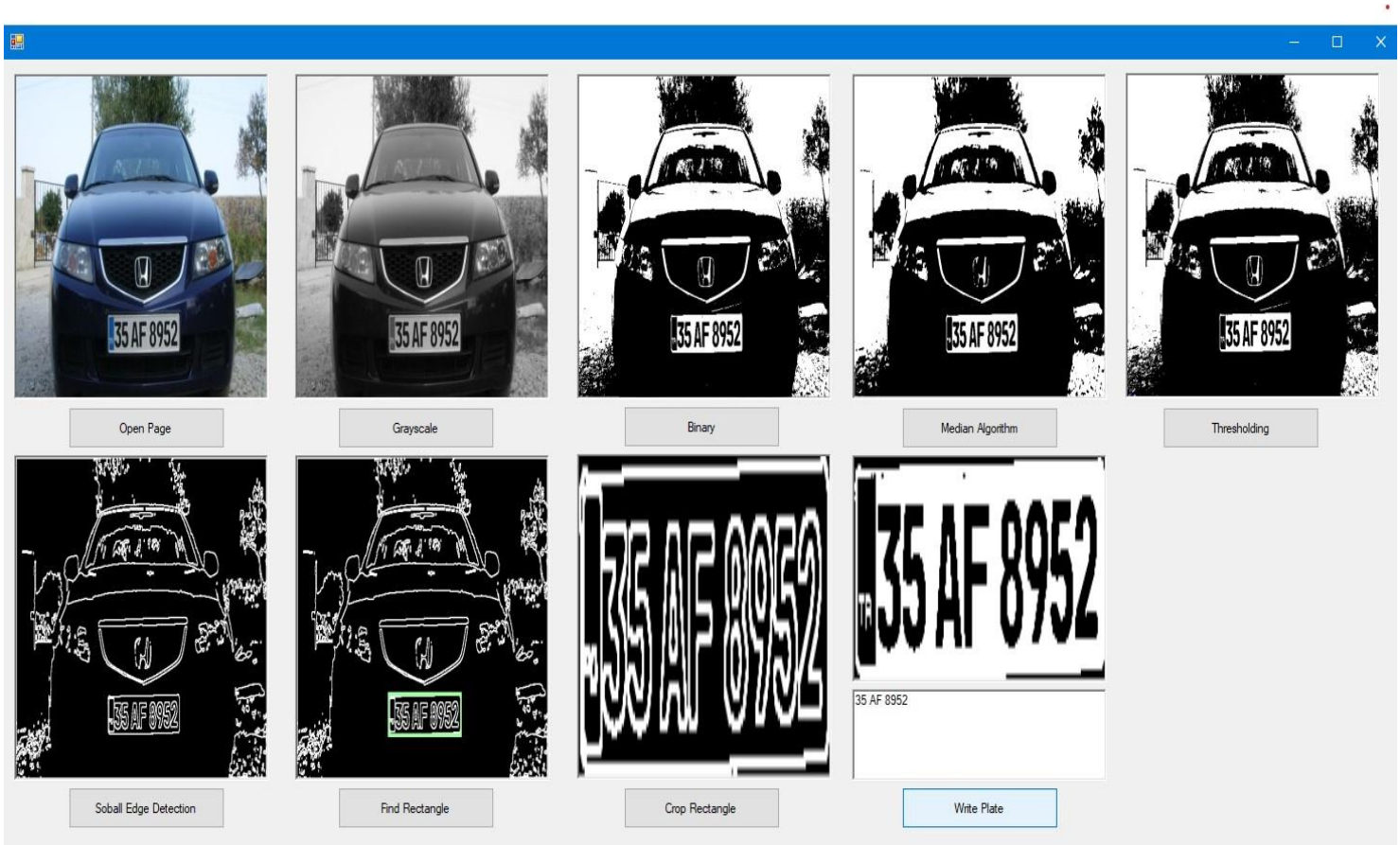
Tesseract is finding templates in pixels, letters, words and sentences. It uses two-step approach that calls adaptive recognition. It requires one data stage for character recognition, then the second stage to fulfil any letters, it wasn't insured in, by letters that can match the word or sentence context.

Conclusion and Suggestions

In this study, a license plate recognition system was developed using image processing techniques. Unlike previous studies, all algorithms have been rewritten in the software using C # programming language instead of ready systems. Tesseract is used to read the plates in this software. When the license plate recognition results were examined, it was confirmed by the applications that the plates were improved in reading speed and accuracy rates. The correct reading of the plates with 88.1%, with 98% success, supports this situation.

The study has been kept open to improvement, as it is in a position to perform basic image processing techniques. In subsequent studies, the success rate in character recognition and general success rates can be increased by using artificial intelligence methods in reading characters.

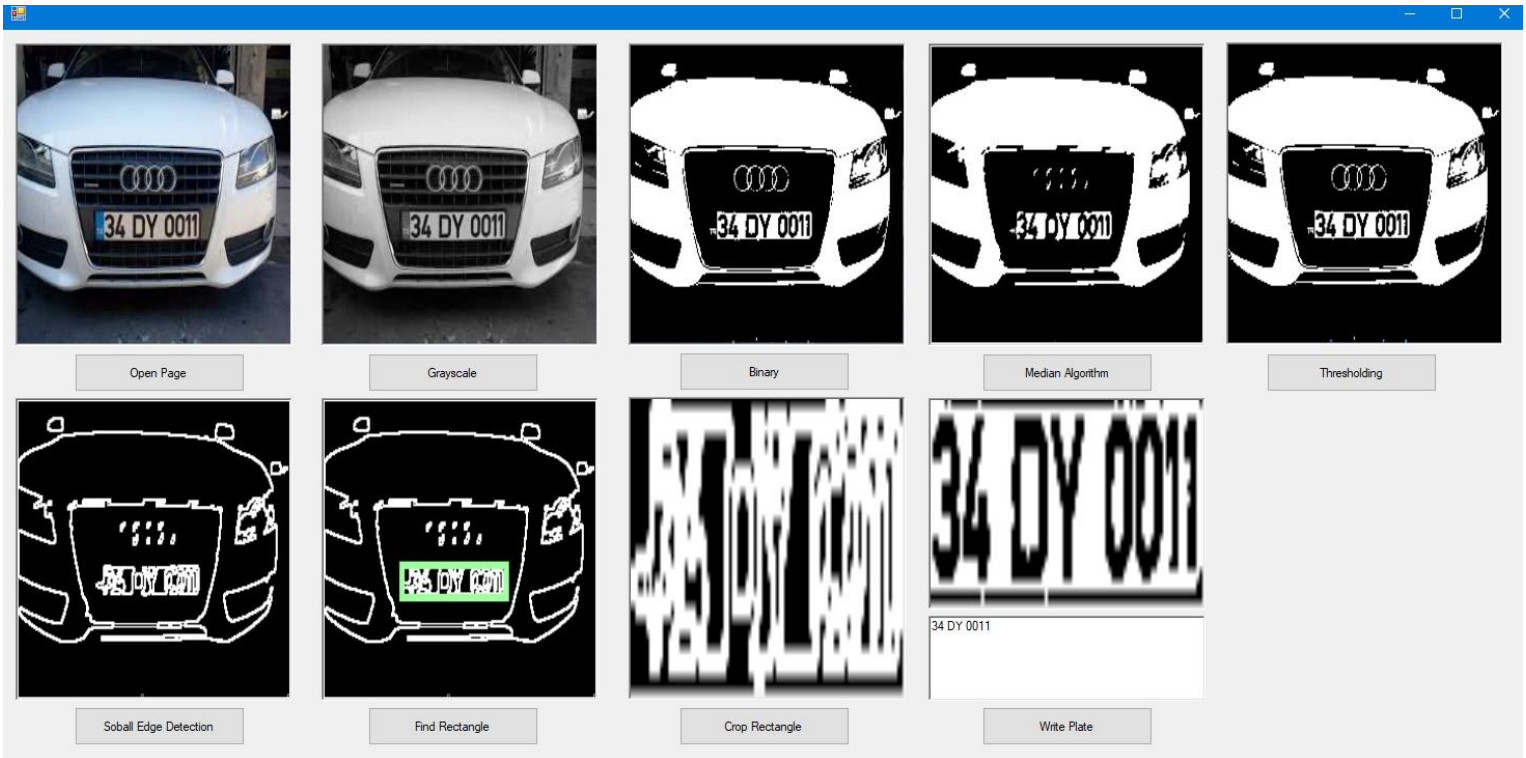
Other plate results we tried are below




Open Page
Grayscale
Binary
Median Algorithm
Thresholding
Soball Edge Detection
Find Rectangle
Crop Rectangle
07 GKN 62
Write Plate


Open Page
Grayscale
Binary
Median Algorithm
Thresholding
Soball Edge Detection
Find Rectangle
Crop Rectangle
HSK TR 249
Write Plate


Open Page
Grayscale
Binary
Median Algorithm
Thresholding
Soball Edge Detection
Find Rectangle
Crop Rectangle
06 YES 25
Write Plate


Open Page
Grayscale
Binary
Median Algorithm
Thresholding
Soball Edge Detection
Find Rectangle
Crop Rectangle
34 DY 0011
Write Plate