UNIVERSIDADE
LUSÓFONA

Deteção e Identificação de **Sinais de Trânsito**

Miguel Carreta – 21901101
Ricardo Gonçalves – 22005012
Eduardo Miranda – 22002197

Computação Gráfica | LEI | 25/06/2022
www.ulusofona.pt

Índice

Conteúdo

Introdução

Pretende-se recriar no laboratório um sistema de deteção e identificação de sinais de trânsito presentes na berma da estrada, idêntico aos que equipam alguns automóveis. Neste projeto pretende-se dar mais enfase aos sinais de limite de velocidade fazendo a leitura do valor por reconhecimento de carateres. Caso existam outros sinais na imagem será feita apenas a sua deteção não havendo identificação.

Figura 1 – Exemplo de imagens a analisar

Do ponto de vista do utilizador, o procedimento deverá ser: abrir na aplicação a imagem contendo um ou mais sinais e processar a imagem, mostrando como resultado final o valor do limite (texto) e localização dos sinais na imagem (quadrado sobreposto à imagem). Embora sejam disponibilizados pelos docentes alguns exemplos de teste, é recomendável que os alunos testem os seus trabalhos com outros exemplos para garantir que a aplicação desenvolvida é suficientemente robusta.

Implementação da solução

Criou-se duas variáveis do tipo inteiro com a largura (width) e altura (height) da imagem original. Criou-se também uma lista de imagens que irá guardar as imagens dos limites de velocidade; um array bidimensional de inteiros para guardar as etiquetas; um Dicionário de chave e valor inteiro, para auxiliar nos cálculos das etiquetas; e uma lista de arrays bidimensionais de inteiros para guardar os pontos extremos XY das etiquetas dos limites de velocidade.

```
unsafe
{

    MIplImage m = img.MIplImage;
    MIplImage mCopy = imgCopy.MIplImage;

    int width = img.Width;
    int height = img.Height;

    List<Image<Bgr, byte>> digitsListFromDataBase = getDigitsFromDataBase();
    List<Image<Bgr, byte>> velocityDigitsList = new List<Image<Bgr, byte>>();

    int[,] etiquetas = new int[height, width];

    Dictionary<int, int> countEtiquetasDic = new Dictionary<int, int>();
    List<int[,]> extremePointsXYEtiquetasVelocity = new List<int[,]>();
    limitSign = new List<string[]>();
    warningSign = new List<string[]>();
    prohibitionSign = new List<string[]>();


    //IMAGE HSV


    //Transforma o vermelho em branco e tudo o resto em Preto
    imgCopy = redToBlackAndWhite(imgCopy);

    etiquetas = counterEtiquetas(imgCopy);

    etiquetas = connectedComponents(etiquetas, width, height);

    // SaveEtiquetasAsCSV(etiquetas, width, height);

    countEtiquetasDic = generateEtiquetasInDictionary(etiquetas, width, height);

    countEtiquetasDic = eliminateNoiseByPercentage(countEtiquetasDic);


    List<int[,]> extremePointsXYEtiquetas = discoverExtremeXYEtiquetas(countEtiquetasDic, etiquetas, width, height);

    drawRectanglesOnSigns(imgCopy, extremePointsXYEtiquetas);
```

Figura 2 - 1ºa parte do código

Na figura 2 encontra-se a primeira parte do projeto que tem como fim descobrir os pontos extremos XY das etiquetas.

De seguida iremos explicar cada uma das funções.

Retornar os dígitos da base de dados:

```
1 referência
public static List<Image<Bgr, byte>> getDigitsFromDataBase()
{
    unsafe
    {
        List<Image<Bgr, byte>> digits = new List<Image<Bgr, byte>>();

        int imagemDigitos;

        for (imagemDigitos = 0; imagemDigitos <= 9; imagemDigitos++)
        {
            Image<Bgr, byte> img = new Image<Bgr, byte>(@"C:\Users\belgi\OneDrive\Documents\Faculdade\2ANO\2SEMESTRE\ComptGrafica\digitos\" + imagemDigitos + ".png");
            Bitmap bitMapImg = new Bitmap(img.Bitmap, new Size(160, 250));
            bitMapImg.SetResolution(300, 300);

            Image<Bgr, byte> digito = new Image<Bgr, byte>(bitMapImg);

            digito = invertColorsDigitsFromDataBase(digito);
            digits.Add(digito);
        }
        return digits;
    }
}
```

Figura 3 - Retornar uma lista de imagens

Iterou-se imagem a imagem, e com a ajuda que o Prof deu, redimensionamos a imagem para um tamanho fixo, com fim a agilizar o processo de comparação de dígitos.

No fim invertemos as cores Preto -» Branco, e Branco -» Preto e adicionámos à lista.

RGB para HSV e inverter as cores Vermelho -» Branco , e tudo o resto a preto

```
public static double[] RgbToHsv(double r, double g, double b)
{
    double[] hsv = new double[3];

    r = r / 255.0;
    g = g / 255.0;
    b = b / 255.0;

    double cmax = Math.Max(r, Math.Max(g, b));
    double cmin = Math.Min(r, Math.Min(g, b));
    double diff = cmax - cmin;
    double h = 0, s = 0;

    if (cmax == r && g >= b)
        h = 60 * ((g - b) / diff) + 0;

    else if (cmax == r && g < b)
        h = 60 * ((g - b) / diff) + 360;

    else if (cmax == g)
        h = (60 * ((b - r) / diff) + 120);

    else if (cmax == b)
        h = (60 * ((r - g) / diff) + 240);

    if (cmax == 0)
    {
        s = 0;
    }
    else if (cmax > 0)
    {
        s = diff / cmax;
    }

    double v = cmax;

    hsv[0] = h;
    hsv[1] = s;
    hsv[2] = v;

    return hsv;
```

Figura 4 - Transformar RGB em HSV

Esta função recebe cada um dos canais de cor (R G B) e converte em HSV retornando um array de Double.

```
1 referência
public static Image<Bgr, byte> redToBlackAndWhite(Image<Bgr, byte> img)
{
    unsafe
    {
        MIplImage m = img.MIplImage;
        byte* dataPtr = (byte*)m.imageData.ToPointer(); // Pointer to the image
        int width = img.Width;
        int height = img.Height;
        int nChan = m.nChannels;
        int step = m.widthStep;
        int x, y;
        double[] hsv = new double[3];


        for (y = 0; y < height; y++)
        {
            for (x = 0; x < width; x++)
            {
                hsv = RgbToHsv((dataPtr + nChan * x + step * y)[2], (dataPtr + nChan * x + step * y)[1], (dataPtr + nChan * x + step * y)[0]);

                if ((hsv[0] >= 250 || hsv[0] <= 7) && hsv[1] >= 0.3 && hsv[2] >= 0.3)
                {
                    (dataPtr + nChan * x + step * y)[0] = (byte)255;
                    (dataPtr + nChan * x + step * y)[1] = (byte)255;
                    (dataPtr + nChan * x + step * y)[2] = (byte)255;
                }
                else
                {
                    (dataPtr + nChan * x + step * y)[0] = (byte)0;
                    (dataPtr + nChan * x + step * y)[1] = (byte)0;
                    (dataPtr + nChan * x + step * y)[2] = (byte)0;
                }
            }
        }
        return img;
    }
}
```

Figura 5 - Converter o vermelho em branco e o resto em preto

Como podemos visualizar, depois de converter em HSV, criámos a condição adequada para converter o vermelho em branco e o resto em preto.

Criar etiquetas

```
2 referências
public static int[,] counterEtiquetas(Image<Bgr, byte> img)
{
    unsafe
    {
        MIplImage m = img.MIplImage;
        byte* dataPtr = (byte*)m.imageData.ToPointer(); // Pointer to the image
        int width = img.Width;
        int height = img.Height;
        int nChan = m.nChannels;
        int step = m.widthStep;

        int[,] etiquetas = new int[height, width];
        int count = 1;

        for (int y = 0; y < height; y++)
        {
            for (int x = 0; x < width; x++)
            {
                // Se encontrar branco , conta como etiqueta
                if ((dataPtr + nChan * x + step * y)[0] == 255)
                {
                    etiquetas[y, x] = count;
                    count++;
                }

            }
        }
        return etiquetas;
    }
}
```

Figura 6 - Criação de etiquetas

Nesta função são criadas etiquetas. Futuramente as etiquetas servirão para distinguir os componentes interligados, de modo a que os maiores componentes interligados irão corresponder ao sinal e não ao ruído.

Aqui criámos uma variável bidimensional de inteiros chamada “Etiquetas”. E percorrendo a imagem, criámos a condição, se o pixel for branco, a etiqueta na posição Y X leva um incremento.

No fim retornamos a variável etiquetas.

Componentes interligados

```
2 referências
public static int[,] connectedComponents(int[,] etiquetas, int width, int height)
{
    int x, y;
    bool changed = true;

    while (changed)
    {
        changed = false;
        for (y = 2; y < height - 2; y++)
        {
            for (x = 2; x < width - 2; x++)
            {
                if (etiquetas[y, x] != 0)
                {
                    int min;

                    List<int> valorEtiqueta = new List<int>();

                    if (etiquetas[y - 1, x - 1] != 0)
                    {
                        valorEtiqueta.Add(etiquetas[y - 1, x - 1]);
                    }


                    if (etiquetas[y - 1, x] != 0)
                    {
                        valorEtiqueta.Add(etiquetas[y - 1, x]);
                    }

                    if (etiquetas[y - 1, x + 1] != 0)
                    {
                        valorEtiqueta.Add(etiquetas[y - 1, x + 1]);
                    }


                    if (etiquetas[y, x - 1] != 0)
                    {
                        valorEtiqueta.Add(etiquetas[y, x - 1]);
                    }

                    if (etiquetas[y, x] != 0)
                    {
                        valorEtiqueta.Add(etiquetas[y, x]);
                    }

                    if (etiquetas[y, x + 1] != 0)
                    {
                        valorEtiqueta.Add(etiquetas[y, x + 1]);
                    }

                    if (etiquetas[y + 1, x - 1] != 0)
                    {
                        valorEtiqueta.Add(etiquetas[y + 1, x - 1]);
                    }

                    if (etiquetas[y + 1, x] != 0)
                    {
                        valorEtiqueta.Add(etiquetas[y + 1, x]);
                    }

                    if (etiquetas[y + 1, x + 1] != 0)
                    {
                        valorEtiqueta.Add(etiquetas[y + 1, x + 1]);
                    }


                    min = valorEtiqueta.Min();


                    if (etiquetas[y, x] != min)
                    {
                        changed = true;
                        etiquetas[y, x] = min;
                    }
```

Figura 7 - Percorrer de cima para baixo

```
                }
            }
        }

        if (!changed) break;

        changed = false;

        for (y = height - 2; y > 2; y--)
        {
            for (x = width - 2; x > 2; x--)
            {
                if (etiquetas[y, x] != 0)
                {
                    int min;

                    List<int> valorEtiqueta = new List<int>();

                    if (etiquetas[y - 1, x - 1] != 0)
                    {
                        valorEtiqueta.Add(etiquetas[y - 1, x - 1]);
                    }


                    if (etiquetas[y - 1, x] != 0)
                    {
                        valorEtiqueta.Add(etiquetas[y - 1, x]);
                    }

                    if (etiquetas[y - 1, x + 1] != 0)
                    {
                        valorEtiqueta.Add(etiquetas[y - 1, x + 1]);
                    }


                    if (etiquetas[y, x - 1] != 0)
                    {
                        valorEtiqueta.Add(etiquetas[y, x - 1]);
                    }

                    if (etiquetas[y, x] != 0)
                    {
                        valorEtiqueta.Add(etiquetas[y, x]);
                    }

                    if (etiquetas[y, x + 1] != 0)
                    {
                        valorEtiqueta.Add(etiquetas[y, x + 1]);
                    }

                    if (etiquetas[y + 1, x - 1] != 0)
                    {
                        valorEtiqueta.Add(etiquetas[y + 1, x - 1]);
                    }

                    if (etiquetas[y + 1, x] != 0)
                    {
                        valorEtiqueta.Add(etiquetas[y + 1, x]);
                    }

                    if (etiquetas[y + 1, x + 1] != 0)
                    {
                        valorEtiqueta.Add(etiquetas[y + 1, x + 1]);
                    }


                    min = valorEtiqueta.Min();

                    if (etiquetas[y, x] != min)
                    {
                        changed = true;
                        etiquetas[y, x] = min;
                    }

                }
```

Figura 8 - Percorrer de baixo para cima

A partir das etiquetas vamos procurar a etiqueta com o menor valor de um conjunto de etiquetas interligadas. Essa procura é feita através de observar os vizinhos do pixel da imagem, mais precisamente a sua etiqueta. Assim, a um conjunto de etiquetas interligadas vamos atribuir o menor valor do conjunto de forma a que essa etiqueta se refira a um conjunto de pixéis interligados.

Criação de um Dicionário

```
2 referências
public static Dictionary<int, int> generateEtiquetasInDictionary(int[,] etiquetas, int width, int height)
{
    //  List<int> valorEtiqueta = new List<int>();
    Dictionary<int, int> dic = new Dictionary<int, int>();
    // int i = 0;

    for (int y = 0; y < height; y++)
    {
        for (int x = 0; x < width; x++)
        {
            if (etiquetas[y, x] != 0)
            {
                if (!dic.ContainsKey(etiquetas[y, x]))
                {
                    dic.Add(etiquetas[y, x], 1);
                }
                else
                {
                    if (dic.TryGetValue(etiquetas[y, x], out int valor))
                    {
                        dic.Remove(etiquetas[y, x]);
                        dic.Add(etiquetas[y, x], valor + 1);
                    }

                }
            }

        }
    }

    return dic;
}
```

Figura 9 - Criação de dicionário

Foi criado um dicionário para guardar as etiquetas depois da equalização feita na função “ConnectComponents”, de modo a que seja mais pratico fazer outros processos com as etiquetas. E a quantidade de pixéis que pertence a essa etiqueta

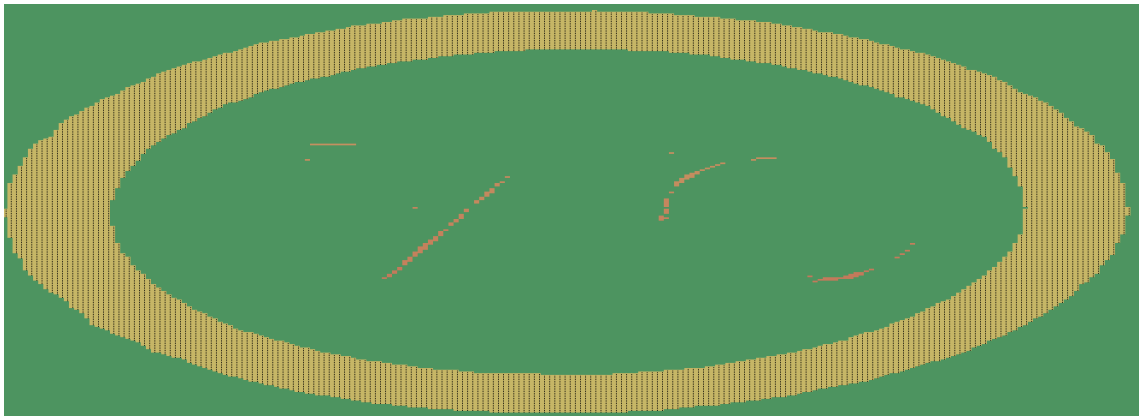
Função para guardar as etiquetas em CSV

```
1 referência
public static void SaveEtiquetasAsCSV(int[,] arrayToSave, int width, int height)
{

    using (StreamWriter file = new StreamWriter("etiquetas.csv"))
    {
        for (int y = 0; y < height; y++)
        {
            for (int x = 0; x < width; x++)
            {
                file.Write(arrayToSave[y, x] + ";");
            }
            file.Write(Environment.NewLine);
        }

    }
}
```

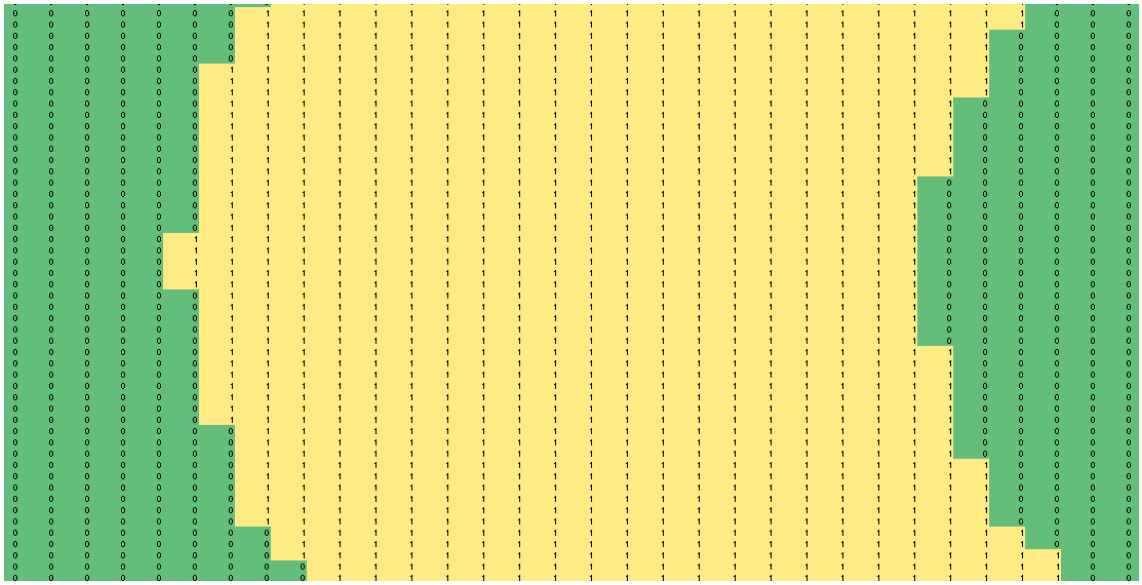
Figura 10 - Guardar etiquetas em CSV

Nesta função criamos um ficheiro chamado “etiquetas.csv”, e com a nossa largura e altura da imagem , escrevemos la dentro as etiquetas separadas por “ ; ”.

Etiquetas guardadas em CSV e visualizado em Excel



Como podemos visualizar, a partir do HSV vermelho foi isolada a parte circular do sinal, depois com a atribuição das etiquetas foi estabelecida uma etiqueta que engloba o conteúdo do sinal.



Neste exemplo a etiqueta, que interliga a borda do sinal, identificada foi a número 1.

Eliminação de ruído

```
1 referência
public static Dictionary<int, int> eliminateNoiseByPercentage(Dictionary<int, int> dictionaryEtiquetas)
{

    int maiorNumeroDeEtiquetasContado = 0;
    int keyMaisContada = 0;
    double percentagem = 0.3;

    foreach (var item in dictionaryEtiquetas.OrderByDescending(key => key.Value))
    {
        if (item.Value >= maiorNumeroDeEtiquetasContado)
        {
            keyMaisContada = item.Key;
            maiorNumeroDeEtiquetasContado = item.Value;
        }
    }


    foreach (var item in dictionaryEtiquetas.OrderByDescending(key => key.Value))
    {
        double percentagemPeloMaior = (double)item.Value / maiorNumeroDeEtiquetasContado;

        if (maiorNumeroDeEtiquetasContado > 25000)
        {
            percentagem = 0.1;
        }

        if (percentagemPeloMaior < percentagem)
        {
            if (dictionaryEtiquetas.TryGetValue(item.Key, out int value))
            {
                dictionaryEtiquetas.Remove(item.Key);

            }
        }
```

Figura 11 - Elimina o ruido

Esta função tem como objetivo eliminar o ruido da imagem.

Para tal, ordenámos o dicionário descendentemente e foi definido um threshold.

Caso as etiquetas sejam menores que esse threshold, são eliminadas do dicionário.

Localizar os pontos extremos e de seguida guardá-los numa lista de inteiros bidimensionais

```
1 referência
public static int[,] locateExtremePoints(int[,] etiquetas, int etiqueta, int width, int height)
{
    int[,] extremoDasEtiquetas = new int[4, 2];

    int xMenor = width;
    int xMaior = -1;
    int yMenor = height;
    int yMaior = -1;

    int pontoYMenor = 0, pontoYMaior = 0;

    /*
     ------------------------------------------------------------------      y Menor

     ------------------------------------------------------------      x Menor                  x Maior

     ------------------------------------------------------------------      y Maior

     */

    for (int y = 0; y < height - 1; y++)
    {
        for (int x = 0; x < width - 1; x++)
        {
            if (etiquetas[y, x] == etiqueta)
            {
                if (y < yMenor)
                {
                    yMenor = y;

                    extremoDasEtiquetas[0, 0] = y;
                    extremoDasEtiquetas[0, 1] = x;

                    pontoYMenor = y;
                }

                if (x < xMenor)
                {
                    xMenor = x;
                    extremoDasEtiquetas[1, 0] = y;
                    extremoDasEtiquetas[1, 1] = x;
                }
```

Figura 13 - Localiza extremos parte 1

```
                }

                if (y > yMaior)
                {
                    yMaior = y;
                    extremoDasEtiquetas[2, 0] = y;
                    extremoDasEtiquetas[2, 1] = x;

                    pontoYMaior = y;
                }

                if (x > xMaior)
                {
                    xMaior = x;
                    extremoDasEtiquetas[3, 0] = y;
                    extremoDasEtiquetas[3, 1] = x;

                }
            }
        }
    }

    return extremoDasEtiquetas;
}
```

Figura 12 - Localiza extremos parte 2

Nesta função localizamos os pontos extremos X e Y das etiquetas.

Os extremos são determinados dentro do conjunto de pixéis que pertencem a uma etiqueta. Temos quatro condições possíveis:

O pixel com o Y mais pequeno guardamos em [0,0], corresponde ao extremo de cima.

O pixel com o X mais pequeno guardamos em [1,1], corresponde ao extremo da esquerda.

O pixel com o Y maior guardamos em [2,0], corresponde ao extremo debaixo.

O pixel com o X maior guardamos em [3,1], corresponde ao extremo da direita.

As análises entre pixéis são feitas pela comparação dos pixéis anteriormente comparados e através das condições e substituições obtêm-se os extremos.

```
2 referências
public static List<int[,]> discoverExtremeXYEtiquetas(Dictionary<int, int> etiquetasDic, int[,] etiquetas, int width, int height)
{
    List<int[,]> pontosExtremosXY = new List<int[,]>();

    foreach (var item in etiquetasDic)
    {
        pontosExtremosXY.Add(locateExtremePoints(etiquetas, item.Key, width, height));

    }
    return pontosExtremosXY;
}
```

E agora que já localizámos os pontos através da função “LocateExtremePoints”, guardamos esses mesmos pontos numa lista bidimensional de inteiros.

Desenhar um retângulo nos sinais a partir dos pontos extremos

```
public static void drawRectanglesOnSigns(Image<Bgr, byte> img, List<int[,]> extremePoints)
{

    foreach (var item in extremePoints)
    {
        int topY = item[0, 0];
        int leftX = item[1, 1];
        int bottomY = item[2, 0];
        int rightX = item[3, 1];

        int differenceBottomTop = bottomY - topY;
        int differenceRightLeft = rightX - leftX;

        img.Draw(new Rectangle(leftX, topY, differenceRightLeft, differenceBottomTop), new Bgr(0, 0, 235), 3);

    }
}
```

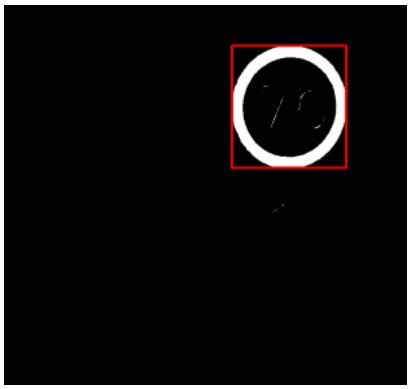


Figura 14 - Sinal com retângulo desenhado

Nesta função percorremos a lista de pontos extremos que obtivemos e desenhamos na nossa imagem cópia um retângulo com o “.Draw“ . Usando o ponto X mais à esquerda e o ponto Y mais a cima como referência, e do lado direito usamos a diferença entre o ponto X mais à direita e o ponto X mais à esquerda. E repetimos o processo para o ponto mais a baixo, fazemos a diferença do ponto Y mais abaixo e o ponto Y mais a cima.

E na figura 14 temos o nosso resultado final.

```
//Imagem dos sinais de velocidade
Image<Bgr, byte> imgVelocitySigns = setImageVelocitySigns(img, extremePointsXYEtiquetas);

int[,] etiquetasVelocity = counterEtiquetas(imgVelocitySigns);

etiquetasVelocity = connectedComponents(etiquetasVelocity, imgVelocitySigns.Width, imgVelocitySigns.Height);

//  SaveEtiquetasVelocidadesAsCSV(etiquetasVelocity, extremePointsXYEtiquetas);

Dictionary<int, int> countEtiquetasVelocityDic = generateEtiquetasInDictionary(etiquetasVelocity, imgVelocitySigns.Width, imgVelocitySigns.Height);

countEtiquetasVelocityDic = eliminateNoiseVelocityByPercentage(countEtiquetasVelocityDic);
```

Nesta segunda fase, fazemos o tratamento de imagem em relação aos dígitos que existem dentro dos sinais. Passaremos a explicar apenas a primeira função pois as outras já foram todas explicadas.

Criar uma imagem para os dígitos dentro dos sinais de velocidade e inverter as cores

```
public static Image<Bgr, byte> setImageVelocitySigns(Image<Bgr, byte> img, List<int[,]> extremePointsXYEtiquetas)
{
    unsafe
    {
        foreach (var item in extremePointsXYEtiquetas)
        {
            if (!isTriangulo(item))
            {
                int topY = item[0, 0];
                int leftX = item[1, 1];
                int bottomY = item[2, 0];
                int rightX = item[3, 1];


                MIplImage m = img.MIplImage;
                byte* dataPtr = (byte*)m.imageData.ToPointer(); // Pointer to the image
                int nChan = m.nChannels;
                int step = m.widthStep;

                for (int y = topY + 25; y < bottomY - 25; y++)
                {
                    for (int x = leftX + 20; x < rightX - 20; x++)
                    {

                        if ((dataPtr + nChan * x + step * y)[0] < 85 && (dataPtr + nChan * x + step * y)[1] < 85 && (dataPtr + nChan * x + step * y)[2] < 85)
                        {
                            (dataPtr + nChan * x + step * y)[0] = 255;
                            (dataPtr + nChan * x + step * y)[1] = 255;
                            (dataPtr + nChan * x + step * y)[2] = 255;
                        }
                        else
                        {
                            (dataPtr + nChan * x + step * y)[0] = 0;
                            (dataPtr + nChan * x + step * y)[1] = 0;
                            (dataPtr + nChan * x + step * y)[2] = 0;
                        }

                    }

                }
            }
        }

    }
    return img;
}
```

Nesta função percorremos os pontos extremos X e Y das etiquetas, e se o sinal não tiver um formato em triângulo, restringimos a área dos dígitos e invertemos a cor dos dígitos para branco e o resto a preto.

Com os dígitos a branco, agilizamos o processo para compara-los com os dígitos da base de dados.

Descobrir se é um triângulo

```
3 referências
public static bool isTriangulo(int[,] extremoEtiquetas)
{
    int pontoYMenor = extremoEtiquetas[0, 0];
    int pontoXMenor = extremoEtiquetas[1, 0];
    int pontoYMaior = extremoEtiquetas[2, 0];
    int pontoXMaior = extremoEtiquetas[3, 0];



    int pontoMedio = (pontoYMaior + pontoYMenor) / 2;

    int pontoDiferenca = pontoYMaior - pontoYMenor;



    // Tringulo piramide                                    tringulo invertido   (cedencia de passagem)
    if (pontoXMenor > pontoMedio + (0.2 * pontoDiferenca) || pontoXMenor < pontoMedio - (0.2 * pontoDiferenca))
    {

        return true;
    }

    return false;
}
```

Nesta função fazemos a distinção dos sinais de perigo e de cedência de passagem.

Utilizamos o ponto extremo mais à esquerda como referência à forma geométrica do sinal. Caso esse ponto tenha um Y muito alto corresponde a um triângulo (sinal de perigo), caso tenha um Y muito baixo corresponde a um triângulo invertido (cedência de passagem).

A distinção é feita através do valor da coordenada Y, e por quanto excede ou fica aquém do ponto médio.

Adição às listas Signs, Warning e Prohibition

Adição à lista WarningSign e validação de dígitos

```
foreach (var item in extremePointsXYEtiquetas)
{
    // SE for triangulo guarda
    if (isTriangulo(item))
    {
        string[] warningVector = new string[5];
        warningVector[0] = "-1";
        warningVector[1] = item[1, 1].ToString();    // left X
        warningVector[2] = item[0, 0].ToString();    // topY Y
        warningVector[3] = item[3, 1].ToString();    // right X
        warningVector[4] = item[2, 0].ToString();    // bottom Y

        warningSign.Add(warningVector);
    }
    else
    {
        List<int> numerosValidos = new List<int>();

        extremePointsXYEtiquetasVelocity = discoverExtremeXYEtiquetas(countEtiquetasVelocityDic, etiquetasVelocity, imgVelocitySigns.Width, imgVelocitySigns.Height);

        drawRectanglesOnSigns(imgVelocitySigns, extremePointsXYEtiquetasVelocity);


        foreach (var extremePoints in extremePointsXYEtiquetasVelocity)
        {
            int topY = extremePoints[0, 0];
            int leftX = extremePoints[1, 1];
            int bottomY = extremePoints[2, 0];
            int rightX = extremePoints[3, 1];

            int differenceBottomTop = bottomY - topY;
            int differenceRightLeft = rightX - leftX;

            velocityDigitsList.Add(imgVelocitySigns.Copy(new Rectangle(leftX, topY, differenceRightLeft, differenceBottomTop)));

            //   imgVelocitySigns.Draw(new Rectangle(leftX, topY, differenceRightLeft, differenceBottomTop), new Bgr(0, 0, 235), 3);

        }


        bool isVelocityDigit = false;


        numerosValidos = validDigitsToLimitSigns(velocityDigitsList, digitsListFromDataBase);
```

Nesta fase percorremos os pontos extremos X e Y das etiquetas. Se for um triangulo, introduzimos as coordenadas do sinal no vetor e adicionamos o mesmo ao WarningSign.

Se não for um triângulo, adicionamos a uma lista de imagens os dígitos existentes na imagem e de seguida comparamos com os dígitos da base de dados.

Comparar os digitos do sinal com os digitos da base de dados

```
public static List<int> validDigitsToLimitSigns(List<Image<Bgr, byte>> velocityDigitsList, List<Image<Bgr, byte>> digitsList)
{
    List<int> numerosValidos = new List<int>();
    int count;
    int[] pixeisSemelhantesEmDigitos = new int[10];

    foreach (var extremePoints in velocityDigitsList)
    {
        unsafe
        {
            Bitmap bitMapImg = new Bitmap(extremePoints.Bitmap, new Size(160, 250));
            bitMapImg.SetResolution(300, 300);

            Image<Bgr, byte> imagem = new Image<Bgr, byte>(bitMapImg);

            MIplImage digito = imagem.MIplImage;
            byte* dataPrtDigito = (byte*)digito.imageData.ToPointer();
            int stepDigito = digito.widthStep;
            int nChanDigito = digito.nChannels;

            for (int imagemBaseDados = 0; imagemBaseDados < digitsList.Count(); imagemBaseDados++)
            {
                count = 0;

                MIplImage digitoBaseDados = digitsList[imagemBaseDados].MIplImage;
                byte* dataPrtDigitoBaseDados = (byte*)digitoBaseDados.imageData.ToPointer();
                int stepDigitoBaseDados = digitoBaseDados.widthStep;
                int nChanDigitoBaseDados = digitoBaseDados.nChannels;


                for (int y = 0; y < 250; y++)
                {
                    for (int x = 0; x < 160; x++)
                    {
                        if ((dataPrtDigito + nChanDigito * x + stepDigito * y)[0] == (dataPrtDigitoBaseDados + nChanDigitoBaseDados * x + stepDigitoBaseDados * y)[0]
                            && (dataPrtDigito + nChanDigito * x + stepDigito * y)[1] == (dataPrtDigitoBaseDados + nChanDigitoBaseDados * x + stepDigitoBaseDados * y)[1]
                            && (dataPrtDigito + nChanDigito * x + stepDigito * y)[2] == (dataPrtDigitoBaseDados + nChanDigitoBaseDados * x + stepDigitoBaseDados * y)[2])
                        {
                            count++;
                        }
                    }
                }


                Console.WriteLine(imagemBaseDados + " " + count);

                pixeisSemelhantesEmDigitos[imagemBaseDados] = count;
            }
```

```
            Console.WriteLine("_______________");


            int max = pixeisSemelhantesEmDigitos.Max();
            for (int i = 0; i < pixeisSemelhantesEmDigitos.Count(); i++)
            {

                if (pixeisSemelhantesEmDigitos[i] == max)
                {
                    // abaixo de 18000 é ruido
                    if (max < 18000)
                    {
                        continue;
                    }
                    numerosValidos.Add(i);
                    break;

                }
            }

            resetPixeisSemelhantes(pixeisSemelhantesEmDigitos);
        }
    }
    return numerosValidos;
```

Para conseguir uma comparação fidedigna incrementámos uma variável sempre que um pixel fosse igual em ambas as imagens.

Para uma explicação mais simples, vamos apresentar uma imagem da consola com os valores para um sinal de 70.

```
0 27447
1 12605
2 18546
3 20297
4 13670
5 21196
6 20240
7 15554
8 20293
9 21941

_______________
0 14634
1 16255
2 19100
3 15941
4 15847
5 13791
6 10763
7 29052
8 12470
9 14664
```

Como podemos visualizar na imagem, o primeiro digito mais semelhante está posição 0 com 27447 pixéis semelhantes, e o segundo digito está na posição 7 com 29052 pixéis semelhantes.

Assim percebemos facilmente que o sinal em questão é de limite 70.

Atribuir velocidade

```
String velocity = "";
if (numerosValidos.Count() == 2)
{
    numerosValidos.Sort();
    numerosValidos.Reverse();

    foreach (var i in numerosValidos)
    {
        velocity += i.ToString();
        if (i == 0)
        {
            isVelocityDigit = true;
        }
    }
}


else if (numerosValidos.Count() == 3)
{

    numerosValidos.Sort();
    numerosValidos.Reverse();

    velocity += "1";

    foreach (var i in numerosValidos)
    {
        if (i == 1)
        {
            isVelocityDigit = true;
            continue;
        }

        velocity += i.ToString();
    }
}
```

Depois de várias tentativas falhadas, chegámos à conclusão que o melhor era ordenar os números e fazer um reverse, pois, o dígito “0” vem em todos os limites de velocidade. E assim ficaria sempre ordenado.

Quando são três números, estamos na presença de um limite de “100” ou “120” , portanto, optamos por escrever logo o “1” , ignora-lo de seguida e escrever o resto dos números.

Adicionar à lista Signs e Prohibition

```
if (isVelocityDigit)

{
    // Os sinais de proibicao vêm SEMPRE por cima dos de limite
    if (temProibicaoPorCimaDoLimite)
    {
        foreach (var extremePoints in extremePointsXYEtiquetas)
        {
            if (counterHelper == 0)
            {
                counterHelper++;
                temProibicaoELimite = true;
                continue;
            }

            string[] velocityVector = new string[5];
            velocityVector[0] = velocity;
            velocityVector[1] = extremePoints[1, 1].ToString();    // left X
            velocityVector[2] = extremePoints[0, 0].ToString();    // topY Y
            velocityVector[3] = extremePoints[3, 1].ToString();    // right X
            velocityVector[4] = extremePoints[2, 0].ToString();    // bottom Y

            limitSign.Add(velocityVector);
            break;
        }
    }
    else
    {
        string[] velocityVector = new string[5];
        velocityVector[0] = velocity;
        velocityVector[1] = item[1, 1].ToString();    // left X
        velocityVector[2] = item[0, 0].ToString();    // topY Y
        velocityVector[3] = item[3, 1].ToString();    // right X
        velocityVector[4] = item[2, 0].ToString();    // bottom Y

        limitSign.Add(velocityVector);
    }
}
else
{
    if (temProibicaoELimite)
    {
        foreach (var extremePoints in extremePointsXYEtiquetas)
        {
            string[] prohibitionVector = new string[5];
            prohibitionVector[0] = "-1";
            prohibitionVector[1] = extremePoints[1, 1].ToString();    // left X
            prohibitionVector[2] = extremePoints[0, 0].ToString();    // topY Y
            prohibitionVector[3] = extremePoints[3, 1].ToString();    // right X
            prohibitionVector[4] = extremePoints[2, 0].ToString();    // bottom Y

            prohibitionSign.Add(prohibitionVector);
            break;
        }
    }
    else
    {
        string[] prohibitionVector = new string[5];
        prohibitionVector[0] = "-1";
        prohibitionVector[1] = item[1, 1].ToString();    // left X
        prohibitionVector[2] = item[0, 0].ToString();    // topY Y
        prohibitionVector[3] = item[3, 1].ToString();    // right X
        prohibitionVector[4] = item[2, 0].ToString();    // bottom Y

        prohibitionSign.Add(prohibitionVector);
    }
}
```

Devido a vários problemas que tivemos, nomeadamente o facto de o algoritmo gravar os sinais com as coordenadas trocadas, criamos umas quantas condições para o algoritmo perceber quando é que havia um sinal de proibição por cima do de limite de velocidade.

Conclusão

Este trabalho, embora todo o esforço que exigiu, foi bastante gratificante. Com a ajuda do professor em aula e todas as noites mal dormidas, classificamos este projeto como um sucesso.

Tivemos a oportunidade de aplicar conhecimentos teóricos abordados em aulas como a binarização de imagens, componentes interligados e etiquetação.

Estamos bastantes satisfeitos com o resultado final e gostámos imenso de trabalhar com pixéis.