UNIVERSIDADE LUSÓFONA

DETEÇÃO E IDENTIFICAÇÃO DE SINAIS DE TRÂNSITO

Trabalho realizado por:

a22007578, Alexandre Costa

a22007237, João Eleutério

a22006526, Ricardo Cleto

OBJETIVO DO PROJETO:

O projecto proposto na cadeira de Computação Gráfica, visa recriar um sistema de detecção de sinais de trânsito e, com isto, pretende-se dar mais ênfase aos sinais de limite de velocidade fazendo a leitura do valor por reconhecimento de caracteres. Caso existam outros sinais na imagem será feita apenas a sua detecção não havendo identificação das características do mesmo.

O trabalho é composto por 5 níveis de dificuldade, sendo cada um representado por diversas imagens estáticas, que irão ser “corridas” através de um executável denominado por Eval, que tem como objectivo testar a credibilidade do projeto. Este é composto por sinais nos quais se pretende testar a deteção do mesmo - como sinais de Limite de Velocidade, Sinais de Proibição ou Sinais de Perigo.

Para o projeto não ter uma complexidade muito elevada, este foi simplificado tendo apenas que se desenvolver o algoritmo para detecção de sinais compostos por uma gama de cores de vermelho.

IMPLEMENTAÇÃO DO PROJETO:

Para dar início à implementação do projeto, o grupo começou por se dividir na execução das funções, evitando assim a repetição de código e proporcionando a simplificação do mesmo.

Em seguida, começámos por pesquisar técnicas de conversão de cores (RGB), tendo sido utilizada a transformação de RGB em HSV.

```
public static double[] TranformaRGBEmHSV(double red, double blue, double
green)
```

A utilização desta função foi fundamental, pois pudemos detetar as cores circulares internas do sinal de trânsito e, ainda, foi-nos facilitada a binarização da imagem, podendo assim ser mais fácil separar o sinal do restante.

HSV[0] = Gama de tonalidade de vermelho

HSV[1] = Saturação da imagem

HSV[2] = Brilho da imagem

```
for (int y = 0; y < height - 1; y++)
{
    for (int x = 0; x < width - 1; x++)
    {
        blue = ((double)(dataPtr + nChan * x + step * y)[0]) / 255;
        green = ((double)(dataPtr + nChan * x + step * y)[1]) / 255;
        red = ((double)(dataPtr + nChan * x + step * y)[2]) / 255;

        resultado = TranformaRGBEmHSV(red, blue, green);

        // valir de H esta no range dos vermelhos
        // valor de s esta no range dos vermelhos
        // valor do v esta no range dos vermelhos
        if ((resultado[0] < 15 || resultado[0] > 310) && resultado[1] > 0.45 && resultado[2] > 0.30)
        {

            (dataPtrcopy + nChan * x + stepcopy * y)[0] = 255;
            (dataPtrcopy + nChan * x + stepcopy * y)[1] = 255;
            (dataPtrcopy + nChan * x + stepcopy * y)[2] = 255;

        }
        else
        {
            (dataPtrcopy + nChan * x + stepcopy * y)[0] = 0;
            (dataPtrcopy + nChan * x + stepcopy * y)[1] = 0;
            (dataPtrcopy + nChan * x + stepcopy * y)[2] = 0;
        }
    }
}
```

De seguida, para conseguirmos realizar a correta detecção dos sinais, tivemos que recorrer a um Fecho (Dilatação e depois Erosão) da imagem, permitindo minimizar erros produzidos na binarização.

```
public static void dilatacao(int height, int width, Image<Bgr, byte> img)
```

Isto consiste em verificar através de uma máscara 3x3. Se existir apenas 1 pixel branco, e caso a condição esteja verdadeira, o pixel central será branco (RGB = 255 nas 3 camadas) e, consequentemente, o pixel central também irá ficar a branco. Caso contrário, ficará a preto.

Seguidamente realizámos a função da erosão:

```
public static void erosao(int height, int width, Image<Bgr, byte> img)
```

Esta consiste em saber se os 9 pixéis da máscara 3x3 têm o mesmo valor da gama de RGB. Caso esta máscara esteja "correta", manterá a cor do pixel central (branco). Caso contrário, manterá o pixel central com a mesma cor (gama de RGB).

Por fim, utilizámos a função fecho:

```
public static void fecho(int nrCiclos, int height, int width, Image<Bgr,
byte> img
```

A função do fecho é composta pela função da dilatação e depois uma erosão, permitindo assim unir os objetos da imagem binarizada.

Caso queiramos realizar o processo inverso ao Fecho:

```
public static void abertura(int nrCiclos, int height, int width,
Image<Bgr, byte> img)
```

Ainda realizámos uma função denominada de *Abertura*, que irá separar objetos que tenham ligações, ou seja, consiste em chamarmos primeiro a função de erosão e em seguida a dilatação.

Depois da binarização da imagem, chamámos o Fecho, com a quantidade de ciclos desejável.

```
int nrCiclos = 2;
fecho(nrCiclos, height, width, imgCopy);
```

Posteriormente realizámos a função das etiquetas, que tem como objectivo colocar um número de etiqueta em cada pixel presente na imagem.

```
public static int[,] EtiquetasFuncao(int height, int width, Image<Bgr, byte> img, int xInicial, int yInicial, int xFinal, int yFinal)
```

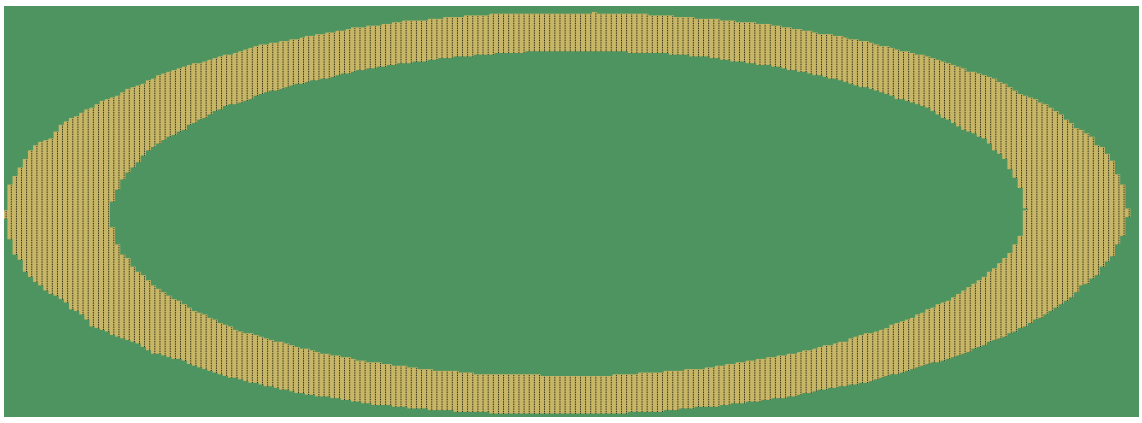
Este algoritmo irá percorrer a imagem de cima para baixo e da esquerda para a direita, no primeiro ciclo *for*. Passando ao segundo *for*, a função que irá realizar o mesmo, só que numa orientação contrária. Sendo, por fim, a imagem percorrida de baixo para cima e da direita para a esquerda, mapeando assim todas as etiquetas necessárias (Esta função também será utilizada futuramente na descoberta dos conteúdos interiores do sinal).

Seguidamente, realizámos uma função denominada por SaveArrayAsCSV:

```
public static void SaveArrayAsCSV(int[,] jaggedArrayToSave, string fileName, int height, int width)
```

Este tem como objetivo criar um ficheiro externo ao programa em .csv com todas as etiquetas realizadas anteriormente. Podemos usar esta ferramenta para realizar DEBUG das etiquetas, tentando perceber se estas foram realizadas corretamente ou não (Esta função também será utilizada futuramente na descoberta dos conteúdos interiores do sinal).

O CSV demonstra a seguinte imagem:



Ainda realizámos outra função denominada por DescobrirLocalizaçãoEtiquetasArea,

```
public static List<int[,]> descobrirLocalizacaoEtiquetasEAreas(int[,] Etiquetas, int width, int height, int trashHold)
```

Que irá descobrir quantas vezes aparece uma determinada etiqueta e, após este processo, seguir-se-á a descoberta da localização das mesmas. De seguida verificamos quantas vezes aparece cada etiqueta em relação à imagem total e, desta maneira, removemos etiquetas que tenham uma área muito reduzida da imagem (através de um threshold). Assim é possível manter, somente numa lista, os valores de etiquetas com uma área superior ao threshold, que indicará que são sinais válidos da imagem.

Após a criação destas 3 funções, chamamo-las dentro da função Signs da seguinte forma:

```
// descobrir e mapear todos os pixeis a branco marcando como etiquetas
Etiquetas = EtiquetasFuncao(height, width, imgCopy, 0, 0, width, height);
SaveArrayAsCSV(Etiquetas, "etiquetas.csv", height, width);
localizacaoEtiquetas = descobrirLocalizacaoEtiquetasEAreas(Etiquetas, width, height, 7);
```

Seguidamente, percorremos a lista da localização das etiquetas e começámos a calcular os pontos extremos do sinal. Depois, utilizámos uma função do C# chamada de Rectangle que irá recortar a área do sinal.

```
for (int i = 0; i < localizacaoEtiquetas.Count(); i++)
{
    int[] dimensoes = new int[2];
    int heightSinal = localizacaoEtiquetas.ElementAt(i)[3, 0] - localizacaoEtiquetas.ElementAt(i)[0, 0];
    int widthSinal = localizacaoEtiquetas.ElementAt(i)[2, 1] - localizacaoEtiquetas.ElementAt(i)[1, 1];

    int yInicial = localizacaoEtiquetas.ElementAt(i)[0, 0];
    int xInicial = localizacaoEtiquetas.ElementAt(i)[1, 1];

    Rectangle rectBounding = new Rectangle(xInicial, yInicial, widthSinal, heightSinal);
    Bgr color = new Bgr(0, 255, 255);
    imgCopy.Draw(rectBounding, color, 5);
```

De seguida, preenchemos 4 strings respetivamente com os pontos extremos do sinal (Mais à esquerda, Mais à direita, Mais acima e mais a baixo).

```
vetor1 = localizacaoEtiquetas.ElementAt(i)[1, 1] + ""; // Left-x
vetor2 = localizacaoEtiquetas.ElementAt(i)[0, 0] + ""; // Top-y
vetor3 = localizacaoEtiquetas.ElementAt(i)[2, 1] + ""; // Right-x
vetor4 = localizacaoEtiquetas.ElementAt(i)[3, 0] + ""; // Bottom-y
```

Depois, realizámos uma condição composta pelos pontos extremos do sinal, em conjunto com um threshold para validarmos se o sinal em causa é um triângulo ou um círculo.

Caso seja um triângulo,

Preenchemos os vectores de DummyVector2 que irá ser adicionado da lista de WarningSigns.

```
// se for um triangulo fazer
if ((localizacaoEtiquetas.ElementAt(i)[3, 0] < localizacaoEtiquetas.ElementAt(i)[2, 0] + 40) ||
(localizacaoEtiquetas.ElementAt(i)[0, 0] + 40 > localizacaoEtiquetas.ElementAt(i)[2, 0]))
{
    string[] dummy_vector2 = new string[5];

    dummy_vector2[0] = "-1";  // value -1
    dummy_vector2[1] = vetor1;
    dummy_vector2[2] = vetor2;
    dummy_vector2[3] = vetor3;
    dummy_vector2[4] = vetor4;
    warningSign.Add(dummy_vector2);


}
```

Caso o sinal seja um círculo,

Recomeçamos todo o processo novamente (Produção de etiquetas, localização das etiquetas e guardamos em CSV novamente).

```
EtiquetasSinal = EtiquetasFuncao(heightSinal, widthSinal, NAOSEINOME, widthSinal / 5, heightSinal / 5, 4 * widthSinal / 5, 4 * heightSinal / 5);

SaveArrayAsCSV(EtiquetasSinal, "etiquetasSinal.csv", heightSinal, widthSinal);
localizacaoEtiquetasSinal = descobrirLocalizacaoEtiquetasEAreas(EtiquetasSinal, widthSinal, heightSinal, 30);

// ver se tem vermelho no meu do sinal
for (int y = heightSinal / 4; y < 3 * heightSinal / 4; y++)
{
    for (int x = widthSinal / 4; x < 3 * widthSinal / 4; x++)
    {
        blue = ((double)(dataPtrSinalCopy + nChanSinalCopy * x + stepSinalCopy * y)[0]) / 255;
        green = ((double)(dataPtrSinalCopy + nChanSinalCopy * x + stepSinalCopy * y)[1]) / 255;
        red = ((double)(dataPtrSinalCopy + nChanSinalCopy * x + stepSinalCopy * y)[2]) / 255;

        resultado = TranformaRGBEmHSV(red, blue, green);

        // valir de H esta no range dos vermelhos
        // valor de s esta no range dos vermelhos
        // valor do v esta no range dos vermelhos
        if ((resultado[0] < 15 || resultado[0] > 310) && resultado[1] > 0.45 && resultado[2] > 0.30)
        {
            souVelocidade = false;
            break;
        }
    }
    if (souVelocidade == false)
    {
        break;
    }
}
```

```
int valor = 80;
for (int y = 0; y < heightSinal - 1; y++)
{
    for (int x = 0; x < widthSinal - 1; x++)
    {

        if ((dataPtrSinal + nChanSinal * x + stepSinal * y)[0] < valor && (dataPtrSinal + nChanSinal * x + stepSinal * y)[1] < valor &&
        (dataPtrSinal + nChanSinal * x + stepSinal * y)[2] < valor)
        {

            (dataPtrSinal + nChanSinal * x + stepSinal * y)[0] = 255;
            (dataPtrSinal + nChanSinal * x + stepSinal * y)[1] = 255;
            (dataPtrSinal + nChanSinal * x + stepSinal * y)[2] = 255;

        }
        else
        {
            (dataPtrSinal + nChanSinal * x + stepSinal * y)[0] = 0;
            (dataPtrSinal + nChanSinal * x + stepSinal * y)[1] = 0;
            (dataPtrSinal + nChanSinal * x + stepSinal * y)[2] = 0;
        }
    }
}
int[,] EtiquetasSinal = new int[heightSinal, widthSinal];
```

Após a realização do mesmo processo, corremos o interior do sinal e realizamos a binarização novamente, para que seja mais fácil a comparação dos números / objectos do sinal com as fotos da base de dados (números de 0-9).

Agora, utilizamos uma função do C# que permite redimensionar as imagens da base de dados com as que temos provenientes do sinal (também redimensionadas), e binarizamos os números provenientes da base de dados:

```
for (int imagens = 0; imagens < 10; imagens++)
{
    Image<Bgr, Byte> imgNumero = new Image<Bgr, Byte>(@"C:\Users\PC\Desktop\faculdade\2 ano\2 semestre\computação grafica\numeros-projeto\" + imagens + ".png");
    Bitmap bmpImage = new Bitmap(imgNumero.Bitmap, new Size(XNumeros, YNumeros));

    bmpImage.SetResolution(300, 300);

    Image<Bgr, byte> ImagemBaseDeDados = new Image<Bgr, byte>(bmpImage);

    MIplImage mBaseDeDados = ImagemBaseDeDados.MIplImage;
    byte* dataPtrBaseDeDados = (byte*)mBaseDeDados.imageData.ToPointer(); // Pointer to the imagecopy
    int stepBaseDeDados = mBaseDeDados.widthStep;
    int nChanBaseDeDados = mBaseDeDados.nChannels; // number of channels - 3

    int valor = 90;
    for (int y = 0; y < YNumeros - 1; y++)
    {
        for (int x = 0; x < XNumeros - 1; x++)
        {

            if ((dataPtrBaseDeDados + nChanBaseDeDados * x + stepBaseDeDados * y)[0] < valor && (dataPtrBaseDeDados + nChanBaseDeDados * x + stepBaseDeDados * y)
            && (dataPtrBaseDeDados + nChanBaseDeDados * x + stepBaseDeDados * y)[2] < valor)
            {

                (dataPtrBaseDeDados + nChanBaseDeDados * x + stepBaseDeDados * y)[0] = 255;
                (dataPtrBaseDeDados + nChanBaseDeDados * x + stepBaseDeDados * y)[1] = 255;
                (dataPtrBaseDeDados + nChanBaseDeDados * x + stepBaseDeDados * y)[2] = 255;

            }
            else
            {
                (dataPtrBaseDeDados + nChanBaseDeDados * x + stepBaseDeDados * y)[0] = 0;
                (dataPtrBaseDeDados + nChanBaseDeDados * x + stepBaseDeDados * y)[1] = 0;
                (dataPtrBaseDeDados + nChanBaseDeDados * x + stepBaseDeDados * y)[2] = 0;
            }
        }
    }
```

Depois de realizarmos a comparação dos números do sinal com a base de dados:

```
for (int imagemNumero = 0; imagemNumero < numerosBaseDeDados.Count(); imagemNumero++)
{
    soma = 0;

    MIplImage mBaseDeDados = numerosBaseDeDados[imagemNumero].MIplImage;
    byte* dataPtrBaseDeDados = (byte*)mBaseDeDados.imageData.ToPointer(); // Pointer to the imagecopy
    int stepBaseDeDados = mBaseDeDados.widthStep;
    int nChanBaseDeDados = mBaseDeDados.nChannels; // number of channels - 3

    for (int y = 0; y < YNumeros; y++)
    {
        for (int x = 0; x < XNumeros; x++)
        {
            // COMPARAR AS IMAGENS
            double blueNumero = ((double)(dataPtrNumero + nChanNumero * x + stepNumero * y)[0]);
            double greenNumero = ((double)(dataPtrNumero + nChanNumero * x + stepNumero * y)[1]);
            double redNumero = ((double)(dataPtrNumero + nChanNumero * x + stepNumero * y)[2]);

            double blueBaseDeDados = (dataPtrBaseDeDados + nChanBaseDeDados * x + stepBaseDeDados * y)[0];
            double greenBaseDeDados = (dataPtrBaseDeDados + nChanBaseDeDados * x + stepBaseDeDados * y)[1];
            double redBaseDeDados = (dataPtrBaseDeDados + nChanBaseDeDados * x + stepBaseDeDados * y)[2];

            if (blueNumero == blueBaseDeDados && greenNumero == greenBaseDeDados && redNumero == redBaseDeDados)
            {
                soma++;
            }

        }
    }
    maisParecido[imagemNumero] = soma;
}
```

Após este trecho de código, adicionamos os números numa lista, e verificamos se é um sinal de 2 ou 3 dígitos:

```
// escrever velocidade
String velocidade = "";
if (numerosFinal.Count() == 2)
{
    numerosFinal.Sort();
    numerosFinal.Reverse();
    for (int naoSei = 0; naoSei < numerosFinal.Count(); naoSei++)
    {
        velocidade += numerosFinal[naoSei] + "";
        // tem um 0 com 2 digitos é valido
        if (numerosFinal[naoSei] == 0)
        {
            valido = true;
        }
    }
}

if (numerosFinal.Count() == 3)
{
    numerosFinal.Sort();
    numerosFinal.Reverse();
    velocidade += "1";
    for (int naoSei = 0; naoSei < numerosFinal.Count(); naoSei++)
    {
        if (numerosFinal[naoSei] == 1)
        {
            continue;
        }
        // tem um 0 com 3 digitos é valido
        if (numerosFinal[naoSei] == 0)
        {
            valido = true;
        }
        velocidade += numerosFinal[naoSei] + "";

    }
}
```

Por fim, adicionamos novamente os extremos das posições dos sinais em conjunto com a velocidade do sinal na lista de DummyVectors1, que corresponde à lista de LimitSigns.

```
// preenche o vetor das velocidades e adiciona-o
if (valido)
{
    string[] dummy_vector1 = new string[5];
    dummy_vector1[0] = velocidade;   // Speed limit
    dummy_vector1[1] = vetor1;
    dummy_vector1[2] = vetor2;

    dummy_vector1[3] = vetor3;
    dummy_vector1[4] = vetor4;
    limitSign.Add(dummy_vector1);

}
```

Caso o sinal não seja um sinal de velocidade, este irá entrar numa condição que valida que o sinal em questão é um sinal de proibição. Porém só existem 2

condições que validam esta opção, sendo a primeira, através dos números do sinal (Caso não tenha números válidos será apenas um sinal de proibição), ou caso o sinal não seja um sinal de velocidade.

```
// preenche o vetor das velocidades e adiciona-o
if (valido)
{
    string[] dummy_vector1 = new string[5];
    dummy_vector1[0] = velocidade;    // Speed limit
    dummy_vector1[1] = vetor1;
    dummy_vector1[2] = vetor2;

    dummy_vector1[3] = vetor3;
    dummy_vector1[4] = vetor4;
    limitSign.Add(dummy_vector1);

}
```

```
// preenche o vetor dos proibidos e adiciona-o
else
{
    string[] dummy_vector3 = new string[5];
    dummy_vector3[0] = "-1";
    dummy_vector3[1] = vetor1;
    dummy_vector3[2] = vetor2;

    dummy_vector3[3] = vetor3;
    dummy_vector3[4] = vetor4;

    prohibitionSign.Add(dummy_vector3);
}
```


```
// preenche o vetor dos proibidos e adiciona-o
else
{
    string[] dummy_vector3 = new string[5];
    dummy_vector3[0] = "-1";
    dummy_vector3[1] = vetor1;
    dummy_vector3[2] = vetor2;

    dummy_vector3[3] = vetor3;
    dummy_vector3[4] = vetor4;

    prohibitionSign.Add(dummy_vector3);
}
```


Sendo verificada uma destas condições, o vetor DummyVector3 será preenchido com as coordenadas extremas da figura e este será adicionado à lista de ProhibitionSign.

CONCLUSÃO:

Concluindo, este projeto, feito no âmbito da cadeira Computação Gráfica, foi um trabalho de grupo que foi feito na totalidade com a discussão conjunta de quais seriam as melhores respostas e execução de problemas.

Como produto final, conseguimos executar um sistema de deteção e identificação de sinais de trânsito. Entre estes os sinais de perigo, proibição e limites de velocidade. Desta forma, o sistema está preparado e feito na sequência de analisar imagens estáticas, em vez de recorrer a um cenário real, que é o do nosso quotidiano.

Podemos afirmar que com a discussão em grupo concordámos que o projeto deveria ser feito de forma simplificada. Ou seja, tendo apenas que se desenvolver o algoritmo para a deteção de sinais composto por uma gama de cores de vermelho. Ainda, antes de começar a execução do trabalho, houve a responsabilidade do grupo de se dividir para que se focassem em determinadas funções para que houvesse uma maior produtividade e desenvolvimento.

O produto final foi conseguido em trabalho de equipa e a maior dificuldade na execução do mesmo foi ao adicionar as etiquetas, levando assim a todos uma maior concentração do tempo para a resolução deste problema.

Por fim, desenvolvemos um projeto discutindo todas as suas funcionalidades em equipa e executando cada passo com produtividade. Apenas com uma dificuldade, que posteriormente foi ultrapassada, conseguimos obter um produto final com todos os objetivos pretendidos no enunciado que nos foi facultado.