UNIVERSIDADE FEDERAL DE SANTA CATARINA
CAMPUS BLUMENAU
CURSO DE GRADUAÇÃO EM ENGENHARIA DE CONTROLE E AUTOMAÇÃO

Leonardo dos Santos Schmitt

Trabalho Final: Visão Computacional em Robótica

Blumenau
2024

1 DESCRIÇÃO DO PROBLEMA

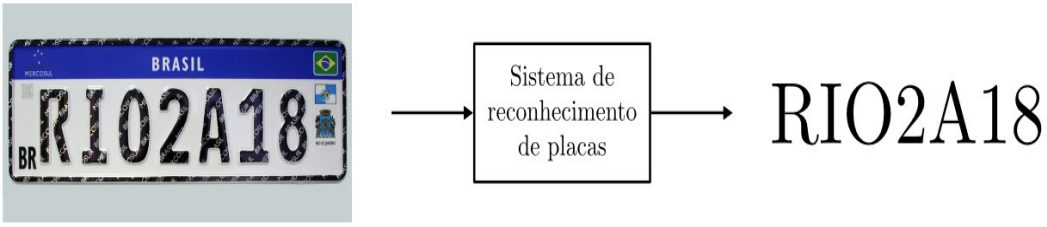
Este trabalho tem como objetivo desenvolver um sistema para o reconhecimento de placas veiculares. As placas a serem reconhecidas pelo sistema seguem o padrão Mercosul, conforme ilustrado na Figura 1.

Figura 1 – Uma placa veicular que deve ser reconhecida através do sistema.

Fonte: Dr.Prof.Marcos Matsuo, 2024.

O sistema deve ser capaz de reconhecer os caracteres das placas veiculares, retornando os sete principais caracteres de identificação, conforme mostrado na Figura 2. A apresentação dos caracteres identificados pode ser realizada de diversas formas, mas neste trabalho o autor optou por exibi-los no terminal do *VScode*, conforme será discutido adiante.

Figura 2 – Diagrama básico apresentando a entrada e a saída de um sistema de reconhecimento de placas veiculares.



Fonte: Dr.Prof.Marcos Matsuo, 2024.

Além disso, as imagens são fornecidas pelo Dr. Prof. Marcos Matsuo e estão organizadas em uma pasta denominada `banco_de_imagens`, onde estão separadas por níveis. Essa organização permite o desenvolvimento de um modelo genérico para o sistema, bem como a avaliação de seu desempenho. Na Figura 3, é possível visualizar duas imagens de placas veiculares, no qual a Figura 3a representa uma placa do nível um, em que as placas

estão aproximadamente paralelas ao plano da câmera. Já no nível dois, apresentado na Figura 3b, é possível observar distorções de perspectiva nas região das placas.

(a) Imagem do nível um. (b) Imagem do nível dois.

Figura 3 – Duas imagens de diferentes níveis.

Em conjunto com as imagens das placas, também é fornecida uma imagem que contém todos os caracteres de ‘A’ a ‘Z’ e de ‘0’ a ‘9’, utilizando a mesma fonte das placas veiculares do padrão Mercosul, conforme apresentado na Figura 4. Essa imagem pode ser usada para extrair os caracteres de referência, que serão empregados na etapa de reconhecimento de caracteres.

Figura 4 – Imagem contendo os caracteres utilizados nas placas veiculares.

Fonte: Dr.Prof.Marcos Matsuo, 2024.

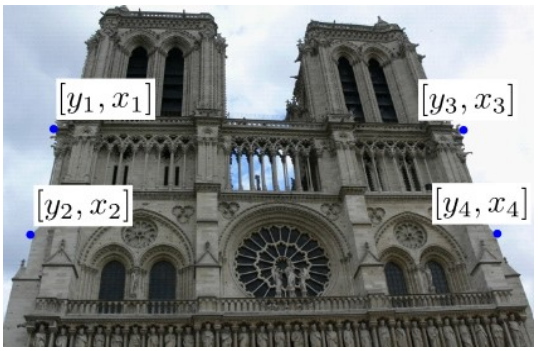
1.1 SOLUÇÃO DA ATIVIDADE 1

Conforme elucidado anteriormente, esta atividade solicita o desenvolvimento de um sistema para o reconhecimento de caracteres em placas veiculares. Esta seção apresentará a solução desenvolvida pelo autor, abrangendo um contexto geral do processo e destacando detalhadamente cada fase e sua implementação. Para fins práticos e de entendimento, este tópico será dividido em subseções, as quais estarão conectadas entre si.

1.1.1 Ajuste das Imagens

De início, é necessário realizar uma identificação prévia das placas em relação à sua orientação, pois, conforme mostrado na Figura 3, há dois níveis no banco de imagens, nos quais cada placa pode apresentar diferentes perspectivas. Para desenvolver um algoritmo genérico capaz de identificar todas as placas, independentemente de sua orientação, é imprescindível realizar uma homografia planar.

Este método consiste na relação entre dois planos em diferente perspectivas. mantendo a relação de pontos, linhas e formas. Um exemplo pode ser visto na Figura 5, no qual a Figura 5a, representa um foto tirada mais abaixo, fazendo com que o objeto em questão da cena tenha a sua base maior que a parte superior, contudo, dependendo da aplicação essa perspectiva pode não ser a mais correta de se analisar a imagem. Sendo assim é necessário realizar uma transformação para que a base do objeto fique de acordo com sua parte superior, conforme mostrado na Figura 5b.



(a) Imagem sem transformação de perspectiva.

(b) Imagem com transformação de perspectiva.

Figura 5 – Transformação de perspectiva entre imagens.

De início, é necessário realizar uma identificação prévia das placas em relação à sua orientação, pois, conforme mostrado na Figura 3, há dois níveis no banco de imagens, nos quais cada placa pode apresentar diferentes perspectivas. Para desenvolver um algoritmo genérico capaz de identificar todas as placas, independentemente de sua orientação, é imprescindível realizar uma homografia planar.

$$\begin{bmatrix} \tilde{x} \\ \tilde{y} \\ \tilde{z} \end{bmatrix} = \begin{bmatrix} h_{1,1} & h_{1,2} & h_{1,3} \\ h_{2,1} & h_{2,2} & h_{2,3} \\ h_{3,1} & h_{3,2} & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}. \tag{1}$$

É importante ressaltar que na matriz de transformação de homografia, existem 8 coeficientes que devem ser determinados. Sendo assim, é necessário definir pelo menos quatro pares de pontos na imagem original e na desejada, conforme mostrado na Figura 5a, onde foram selecionados os pontos desejados na imagem original. Estes por sua vez, podem

ser relacionados à Equação (1), gerando assim, uma nova representação, conforme mostrado nas equações abaixo (Equação (2) e Equação (3)).

$$x'_i = \frac{\tilde{x}_i}{\tilde{z}_i} = \frac{h_{1,1}x_i + h_{1,2}y_i + h_{1,3}}{h_{3,1}x_i + h_{3,2}y_i + 1}. \tag{2}$$

$$y'_i = \frac{\tilde{y}_i}{\tilde{z}_i} = \frac{h_{2,1}x_i + h_{2,2}y_i + h_{2,3}}{h_{3,1}x_i + h_{3,2}y_i + 1}. \tag{3}$$

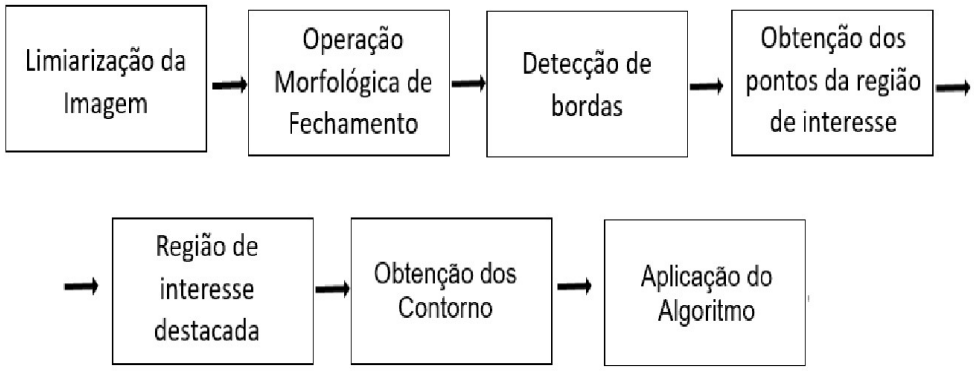
As equações acima podem ser reescritas de forma a obter um sistema de equações lineares, resultando em uma matriz 8×8 denominada A e em vetores coluna h e b, correspondendo, respectivamente, ao vetor de incógnitas e ao vetor de variáveis independentes, conforme apresentado na Equação (4). Para realizar a implementação do algoritmo, o autor criou uma função chamada `homografia()`, que está contida no módulo `visaoComputacional` criado em sala de aula. A função utiliza um laço de repetição para obter os valores de todas as incógnitas e retorná-los para uma variável, que neste trabalho foi denominada de **H**.

$$\begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & -x'_1x_1 & -x'_1y_1 \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -y'_1x_1 & -y'_1y_1 \\ x_2 & y_2 & 1 & 0 & 0 & 0 & -x'_2x_2 & -x'_2y_2 \\ 0 & 0 & 0 & x_2 & y_2 & 1 & -y'_2x_2 & -y'_2y_2 \\ x_3 & y_3 & 1 & 0 & 0 & 0 & -x'_3x_3 & -x'_3y_3 \\ 0 & 0 & 0 & x_3 & y_3 & 1 & -y'_3x_3 & -y'_3y_3 \\ x_4 & y_4 & 1 & 0 & 0 & 0 & -x'_4x_4 & -x'_4y_4 \\ 0 & 0 & 0 & x_4 & y_4 & 1 & -y'_4x_4 & -y'_4y_4 \end{bmatrix} \begin{bmatrix} h_{1,1} \\ h_{1,2} \\ h_{1,3} \\ h_{2,1} \\ h_{2,2} \\ h_{2,3} \\ h_{3,1} \\ h_{3,2} \end{bmatrix} = \begin{bmatrix} x'_1 \\ y'_1 \\ x'_2 \\ y'_2 \\ x'_3 \\ y'_3 \\ x'_4 \\ y'_4 \end{bmatrix}. \tag{4}$$

No caso da aplicação da homografia planar para este trabalho, os quatro pontos selecionados na imagem original devem estar aproximadamente nos extremos da placa, conforme será apresentado adiante. Esses pontos serão mapeados para as coordenadas `[0, 0]`, `[0, n_linhas-1]`, `[n_colunas-1, n_linhas-1]` e `[n_colunas-1, 0]` da nova imagem, onde as variáveis `n_linhas` e `n_colunas` representam o número de linhas e colunas definidas pelo autor, que, neste caso, foram 300 e 1550, respectivamente.

Após o conceito de homografia ser definido, antes de sua aplicação, é necessário utilizar processos intermediários para identificar automaticamente os quatro pontos extremos da placa. Esses processos incluem desde a isolação da região de interesse na imagem, preservando as dimensões de linhas e colunas da imagem original, até a aplicação do algoritmo *Ramer-Douglas-Peucker*, conforme apresentado na Figura 6.

Figura 6 – Fluxograma para o processo de obtenção dos 4 pontos de interesse.



Fonte: Autor, 2024.

Para realizar a extração da região de interesse, é necessário entender que, neste caso, trata-se de isolar na imagem original apenas o elemento mais importante: a placa. Os demais *pixels* que não pertencem a essa região possuem valor igual a 0 (cor preta). Nesse processo, inicialmente, realiza-se uma limiarização global com um limiar definido, sendo este o valor de 135. Em outras palavras, a limiarização global permite segmentar a cor do espaço de tons de cinza para um espaço binário. Assim, *pixels* com valores acima do limiar definido são convertidos para 255 (cor branca), enquanto os demais são convertidos para 0 (cor preta), conforme mostrado na Equação (5) e Figura 7a.

A limiarização global está disponível no módulo `visaoComputacional` e pode ser chamada pela função `limiarizacao_global_1()`. Essa função recebe como parâmetros a matriz **I**, que representa a imagem, e o valor do limiar. Além disso, ressalta-se que o valor do limiar foi definido de maneira empírica pelo autor, por meio de simulações. Também foi realizada a conversão da imagem, que estava no espaço de cor BGR, para o espaço de tons de cinza utilizando a função `cvtColor()` do módulo *OpenCV*.

$$I_{\text{bin}}(y,x) = \begin{cases} 255, & \text{se } I(y,x) > \text{limiar} \\ 0, & \text{caso contrário} \end{cases} \tag{5}$$

Com a imagem no espaço de cor binário, é necessário realizar a conversão dos *pixels*, na qual os que possuem valor 0 passam a ter valor 255, e os que possuem valor 255 passam a ter valor 0, conforme apresentado na Figura 7b. Esse procedimento é indispensável, pois facilita a realização de operações morfológicas, bem como a detecção de bordas para identificar as regiões de interesse. Ressalta-se que essa conversão pode ser realizada como `255 -` I_{bin}.

(a) Imagem no espaço de cor binário.

(b) Imagem no espaço de cor binário na forma negativa.

Figura 7 – Imagens convertidas para o espaço de cor binário.

Conforme abordado anteriormente e seguindo o fluxograma apresentado na Figura 6, após a limiarização, deve-se realizar a operação morfológica de fechamento, tendo em vista que a transição entre bordas pode não estar bem definida devido à presença de ruídos que afetam o contorno. Essa operação consiste na aplicação de uma dilatação na imagem de entrada **I** utilizando um elemento estruturante (EL) denotado por **S**. Em seguida, realiza-se a erosão na imagem resultante, utilizando o mesmo elemento estruturante. Esse procedimento permite que a dilatação preencha os espaços em branco na imagem binária, alterando minimamente as dimensões externas. Posteriormente, a erosão reduz o objeto dilatado ao tamanho original, mas com os espaços preenchidos. É possível visualizar a aplicação desse processo na Figura 8.

O fechamento pode ser implementado com a função `morphologyEx()` da biblioteca *OpenCV*, que recebe como parâmetros a imagem binária, o tipo de operação morfológica e o elemento estruturante. Neste caso, o elemento estruturante utilizado foi uma elipse de dimensões 6x6, definida por meio da função `getStructuringElement()` da mesma biblioteca.

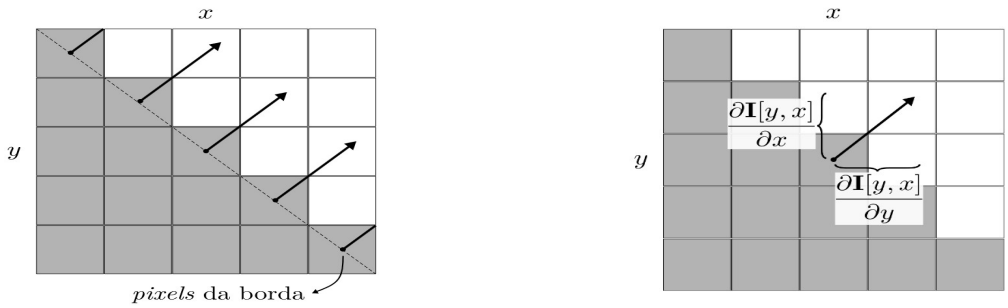
Figura 8 – Imagem após operação morfológica de fechamento.

Fonte: Autor, 2024.

Com o processo de fechamento finalizado, pode-se detectar as bordas, que serão utilizadas para identificar as características de linhas da placa. O processo de detecção de bordas é uma abordagem amplamente empregada para segmentar imagens com base em mudanças abruptas de intensidade.

O algoritmo básico para detecção de bordas é baseado no cálculo do gradiente da imagem, onde, para cada *pixel* da imagem, determina-se um vetor que indica a direção e a magnitude da maior variação de intensidade dos *pixels*, conforme apresentado na Figura 9. Assim, para cada *pixel* $[y, x]$ da imagem $\mathbf{I}$, o vetor gradiente pode ser dado pela Equação (6), onde $\frac{\partial I[y,x]}{\partial x}$ denota a taxa de variação da intensidade do *pixel* no eixo horizontal x, e $\frac{\partial I[y,x]}{\partial y}$ representa a taxa de variação em relação ao eixo vertical y.

$$\nabla I[y, x] = \begin{bmatrix} \frac{\partial I[y,x]}{\partial x} \\ \frac{\partial I[y,x]}{\partial y} \end{bmatrix} \tag{6}$$



(a) Detecção de bordas sem representação do gradiente.

(b) Detecção de bordas com a representação do gradiente.

Figura 9 – Processo de detecção de bordas.

Uma das possíveis formas de calcular $\frac{\partial I}{\partial x}$ e $\frac{\partial I}{\partial y}$ é apresentada na Equação (7) e Equação (8), respetivamente. Já o gradiente, para fins práticos, pode ser representado pela magnitude, conforme mostrado na Equação (9).

$$\frac{\partial I[y, x]}{\partial x} = I[y, x + 1] - I[y, x] \tag{7}$$

$$\frac{\partial I[y, x]}{\partial y} = I[y + 1, x] - I[y, x] \tag{8}$$

$$M[y, x] = \sqrt{\left(\frac{\partial I[y, x]}{\partial x}\right)^2 + \left(\frac{\partial I[y, x]}{\partial y}\right)^2} \tag{9}$$

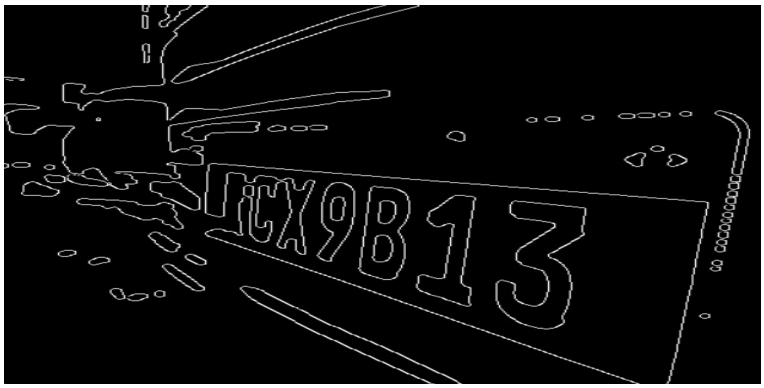
A representação do gradiente na equação anterior torna mais fácil a aplicação via código, uma vez que basta realizar o processo de limiarização, definindo um valor de τ. Nesse processo, se a magnitude do gradiente for maior ou igual a τ, o valor do pixel será

definido como branco; caso contrário, será definido como preto, conforme apresentado na Equação (10).

A abordagem de detecção de bordas pode ser facilmente implementada através da função `Canny()` do módulo *OpenCV*, que foi utilizada neste trabalho. Essa função requer uma imagem binária, juntamente com dois limiares de borda. Neste caso, o limite inferior foi definido como 50 e o superior como 120, ambos determinados por meio de simulações. Além disso, é importante destacar que a função requer mais um parâmetro, que é um *Kernel de Sobel*, utilizado para o cálculo das derivadas. Neste trabalho, foi adotado um *kernel* de dimensão 3×3. Na Figura 10, é possível visualizar a identificação das bordas resultante desse processo.

$$I_b[y,x] = \begin{cases} 1, & \text{se } M[y,x] \geq \tau \\ 0, & \text{caso contrário} \end{cases} \tag{10}$$

Figura 10 – Imagem com os contornos definidos.



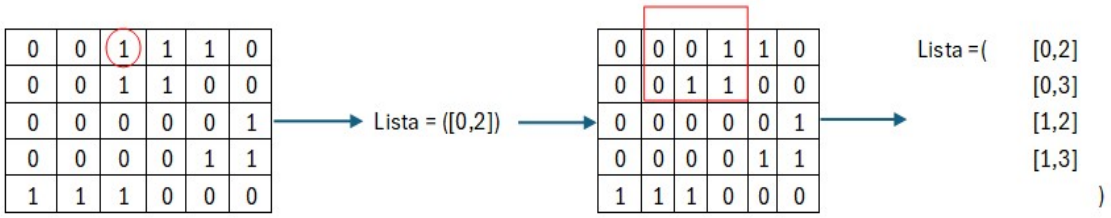
Fonte: Autor, 2024.

Dando continuidade e seguindo o fluxograma apresentado na Figura 6, após a detecção das bordas, realiza-se a análise ou rotulagem de componentes conectados. Esse processo é realizado através da função `connectedComponents()` da biblioteca *OpenCV*, que identifica e retorna regiões de interesse em uma imagem binária.

O algoritmo básico percorre a imagem binária até encontrar um *pixel* branco. Ao encontrá-lo, suas coordenadas são adicionadas a uma lista, e o *pixel* é removido da imagem binária. Para tornar o processo mais eficiente, o algoritmo analisa a vizinhança ao redor desse *pixel* branco. Caso haja outros *pixels* brancos nas proximidades, eles também são adicionados à lista e removidos em seguida, conforme ilustrado na Figura 11.

Após os componentes serem identificados na imagem, é possível extrair informações como a área e o *bounding box* (caixa delimitadora) de cada componente. A área de cada componente conectado pode ser calculada por meio do momento de ordem zero, que é

Figura 11 – Parte do algorítimo básico para rotulagem de componentes conectados.



Fonte: Autor, 2024.

obtido pela soma de todos os *pixels* da região, conforme apresentado na Equação (11). Já os pontos que definem a *bounding box* do componente são determinados pelas coordenadas mínimas e máximas dos *pixels* que pertencem a esse componente. Todos esses cálculos foram encapsulados em uma função chamada `analisaRegioes()`, contida no módulo `visaoComputacional`, que retorna um dicionário com os pontos desejados e a área.

$$M_{00} = \sum_{x} \sum_{y} I(x, y) \tag{11}$$

Em termos de aplicação, utiliza-se o dicionário retornado pela função `analisaRegioes()` para identificar a região de interesse com base na área e no *bounding box* de cada componente conectado. Para isso, é utilizado um laço de repetição que percorre o dicionário, avaliando cada componente de forma individual. Durante o processo, verifica-se se a área do componente está dentro de um intervalo previamente definido, como `600 < area < 2000`. Caso a área esteja dentro desse intervalo, passa-se para a análise de critérios suplementares, os quais são utilizados para resolver conflitos entre componentes com áreas próximas. Isso garante que a maior área ou a que mais se aproxima do valor esperado seja selecionada, conforme apresentado na Equação (12). Nesse caso, deve-se verificar se a área encontrada é maior que a anterior ou se a diferença entre elas é menor que 40. Essa lógica é necessária devido à presença de regiões próximas à área de interesse, que no caso corresponde à placa. Quando a área atende a esse critério, os pontos correspondentes ao *bounding box* são armazenados, permitindo identificar a região desejada de forma eficiente.

$$\text{Pontos selecionados} = \begin{cases} p_1, p_2 & (\text{se Área atual} > \text{Área anterior} \\ & \text{se não, se } |\text{Área atual} - \text{Área anterior}| < 40) \\ 0, & \text{caso contrário} \end{cases} \tag{12}$$

Com os pontos obtidos e armazenados, basta criar uma máscara com o mesmo tamanho da imagem original, destacando a região do *bounding box* definida pelos pontos selecionados na cor branca. Por fim, é necessário realizar uma operação E (AND) entre a máscara e a imagem original, para que sejam eliminados os *pixels* que não pertencem à região da placa. É possível utilizar a função `bitwise_and()` do módulo *OpenCV* para aplicar a máscara na imagem original. Através da Figura 12, é possível visualizar o recorte realizado, destacando apenas a região da placa.

Figura 12 – Imagem resultante após recorte da região de interesse.

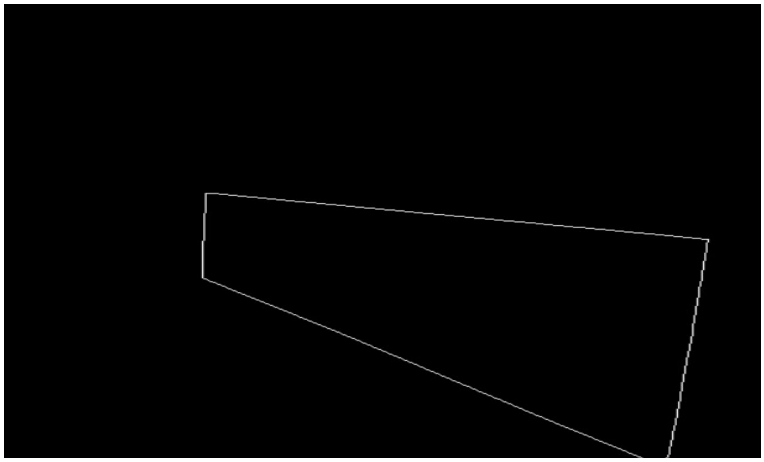
Fonte: Autor, 2024.

Através da imagem com a região da placa destacada, torna-se mais simples o processo de obtenção dos quatro pontos necessários para a homografia. Entretanto, devem ser realizados mais dois procedimentos, conforme apresentado no fluxograma da Figura 6, que são a obtenção do contorno e a aplicação do algoritmo *Ramer-Douglas-Peucker*.

Para realizar a obtenção dos contornos, deve-se converter novamente a imagem para o espaço de cor binário. Em seguida, pode-se utilizar a função `findContours()` do módulo *OpenCV*, passando o componente conectado do qual se deseja extrair o contorno, além do modo de recuperação de contorno, que é o parâmetro `RETR_EXTERNAL`. Esse parâmetro cria uma lista de pontos apenas para os *pixels* mais externos, ignorando os contornos internos. No entanto, os contornos podem conter muitos pontos redundantes. Sendo assim, para remover essas redundâncias e economizar espaço, usa-se em conjunto com a função o parâmetro `CHAIN_APPROX_NONE`.

Ao aplicar a função `findContours()`, ela retorna um parâmetro do tipo tupla contendo os pontos dos objetos que têm contornos interligados. Em vista disso, pode-se realizar um laço de repetição para percorrer essa tupla, filtrando os *pixels* que têm seu valor maior que 600. Após isso, é possível visualizar que o contorno destacado é somente o da placa, conforme apresentado na Figura 13.

Figura 13 – Imagem com o contorno da placa destacado.



Fonte: Autor, 2024.

Após a filtragem do contorno de interesse, neste caso a placa, tem-se que o mesmo possui diversos pontos. Sendo assim, é necessário utilizar o algoritmo *Ramer-Douglas-Peucker* para que seja possível realizar uma aproximação de todos os pontos encontrados no contorno para somente quatro.

A implementação via código do algoritmo de *Ramer-Douglas-Peucker* é dada através de um laço de repetição que varre desde o primeiro ponto do contorno, no qual foi definido pelo autor, até todos os demais pontos, verificando a maior distância entre eles através de um valor ϵ, definido pela multiplicação entre uma variável k, que seria o espaço percorrido em uma sequência de valores para ajustar dinamicamente a aproximação, e o retorno da função `arcLength()`, que é o tamanho do contorno inicial selecionado. Ressalta-se que a variável ϵ controla a precisão do algoritmo, sendo uma fração do comprimento total do contorno. Em sequência, utiliza-se a função `approxPolyDP()` para criar uma variável definida como `approx`, que é submetida a um condicional, que verifica se a mesma é igual a quatro. Essa abordagem garante que o contorno seja simplificado de forma eficiente, preservando as características essenciais do objeto analisado, como ilustrado na Figura 14.

Após obtidos os quatro pontos através da variável `approx`, é possível realizar a homografia, o qual o resultado é apresentado na Figura 15. Vale ressaltar que os *pixels* de interesse são armazenados em duas listas distintas: uma que contém os *pixels* antes do valor da coluna 250 e outra lista que contém os *pixels* acima desse valor. Isso ocorre pelo fato de a aplicação da homografia requerer essa separação, além de ser necessário para tornar o processo automático. Outro ponto importante é que todas as imagens sofrem um redimensionamento no início do código para 500x500, o que também serviu como um ajuste fino para realizar a homografia de forma prática.

Figura 14 – Região destacada após a aplicação algorítimo de *Ramer-Douglas-Peucker*.

Fonte: Autor, 2024.

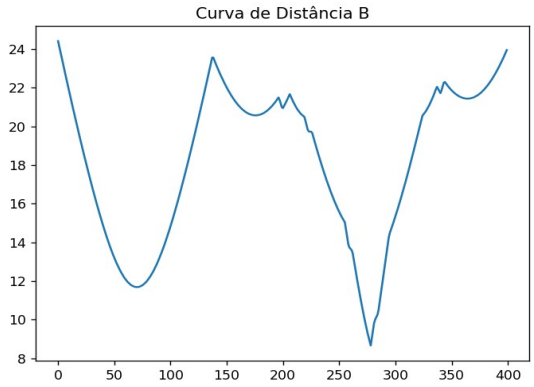
Figura 15 – Imagem após a aplicação da homografia planar.

Fonte: Autor, 2024.

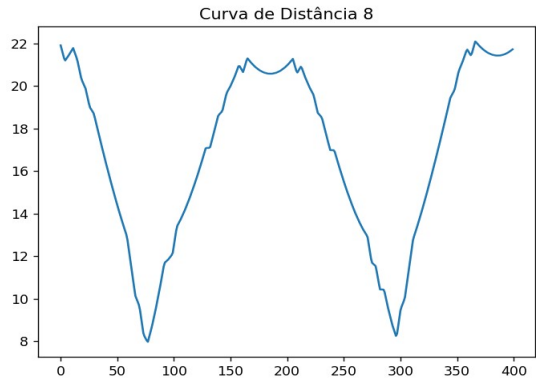
1.1.2 Filtragem 2D da Imagem

Após a realização da homografia, torna-se necessário extrair os componentes conectados da placa. Para isso, é indispensável aplicar o processo de filtragem 2D. Essa etapa é importante porque, ao analisar as curvas de distância (tema que será abordado adiante), surge um problema relacionado à semelhança entre as curvas de letras e números distintos. Um exemplo disso é a comparação entre as letras B e o número 8, conforme ilustrado na Figura 16, o qual, apesar de visualmente a letra e o número apresentarem diferenças em suas curvas, através da Figura 17, nota-se que a extração das letras e números de uma das placas do banco de imagens, evidencia que, devido ao ruído, a curva de distância da letra B se assemelha mais à do número 8 do que à da própria letra.

Esse fenômeno ocorre, pelo ruído existente na imagem, fazendo com que as bordas da letra sejam irregulares, tornando a letra B mais parecida com o número 8. Para corrigir



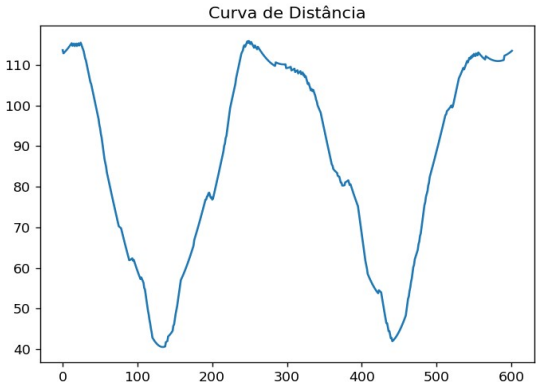
(a) Curva de distância letra B.



(b) Curva de distância letra 8.

Figura 16 – Comparação entre as curvas de distância letra B e número 8.

(a) Imagem pós-homografia limiarizada.



(b) Curva de distância da letra B com problema.

Figura 17 – Extração da curva de distância da letra B.

isso, o autor optou por utilizar a seguinte sequência de processos, que parte desde a remoção de elementos que estão próximos da borda que não são os caracteres da placa, e finaliza com a aplicação de uma máscara para o aumento da nitidez.

Figura 18 – Fluxograma de procedimentos para filtragem 2D.

Fonte: Autor, 2024.

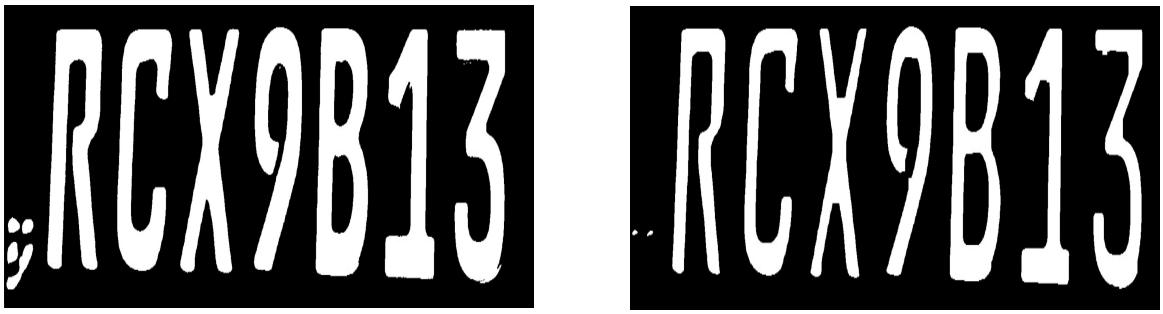
Conforme apresentado no fluxograma de procedimentos da imagem acima, inicia-se o processo de filtragem eliminando os elementos próximos das bordas. Esse procedimento utiliza a reconstrução morfológica, que envolve três imagens: a imagem pós-homografia, uma máscara (cópia da imagem pós-homografia) e uma imagem de marcadores.

Na imagem de marcadores, as bordas são destacadas e configuradas como Marker$[5 : -20, 5 : -20] = 0$. Em seguida, a mesma passa por uma dilatação com um kernel de 3×3 e depois, é aplicada uma interseção lógica E (AND) entre a máscara e os marcadores, utilizando a função `bitwise_and()`.

Esse ciclo de dilatação e interseção se repete até que não haja mais mudanças nos marcadores. Por fim, a imagem resultante desse processo é subtraída da imagem pós-homografia, removendo as bordas e gerando uma nova imagem, conforme ilustrado na Equação (13), onde $R_G(F)$ representa a operação de reconstrução morfológica por erosão e interseção. Já na Figura 19, é possível visualizar a diferença entre as bordas da placa antes e depois desse processo.

Vale ressaltar que não é possível retirar todos os elementos que não fazem parte dos caracteres que se deseja identificar. Isso ocorre devido às diferentes perspectivas das placas. Ou seja, caso a margem do marcador seja maior, pode acabar cortando alguns caracteres de outra placa, o que influenciaria na leitura.

$$I_{final} = I - R_G(F) \tag{13}$$



(a) Imagem com elementos na borda.

(b) Imagem com alguns elementos retirados da borda.

Figura 19 – Resultados após remoção dos elementos na borda

Além disso, é importante destacar que o processo de remoção dos elementos das bordas ocorre entre uma operação de dilatação e uma de erosão. Para a dilatação, utilizou-se um filtro *kernel* retangular com dimensões 9×8, enquanto para a erosão, foi utilizado o mesmo formato de filtro, mas com dimensões 17×17. A dilatação expande todos os elementos, aproximando aqueles que não são os caracteres desejados, das bordas, de modo que possam ser excluídos. Após a exclusão, apenas os caracteres relevantes permanecem na imagem. Em seguida, a erosão é aplicada para refinar esses caracteres, melhorando sua

definição e evitando confusões, uma vez que algumas letras podem se assemelhar devido as suas espessura.

A próxima etapa, após a remoção das bordas, é a aplicação de um filtro para suavizar a imagem, eliminando os ruídos. Nesse caso, utiliza-se um filtro gaussiano, no qual a suavização é feita por meio de uma janela com pesos definidos pela distribuição gaussiana. Isso significa que, ao aplicar o filtro, os *pixels* dentro da janela recebem diferentes pesos, sendo que os *pixels* mais próximos ao centro da janela têm maior peso, enquanto os mais distantes têm menor peso. Quanto maior o tamanho da janela, maior será o efeito de suavização, pois a média ponderada será calculada com base em uma área maior da imagem, suavizando os detalhes finos e os ruídos.

A equação Equação (14) mostra a fórmula do filtro gaussiano, na qual, quanto maior o valor de σ, menos concentrados estarão os coeficientes em torno do centro. Para definir o valor σ, pode-se utilizar a Equação (15), que definindo o valor de w (janela), é possível assim obter a concentração desejada. Ressalta-se que para esse trabalho, o autor definiu o valor da janela igual a 9, como também utilizou a função `GaussianBlur()` do módulo *OpenCV*. Através da Figura 20 é possível visualizar o antes e o depois da aplicação do filtro de suavização

$$K[j,i] = \frac{1}{2\pi\sigma^2} e^{-\frac{j^2+i^2}{2\sigma^2}} \tag{14}$$

$$\sigma = (w-1)/6 \tag{15}$$

Figura 20 – Imagem após aplicação da suavização.

Fonte: Autor, 2024.

Por fim, para finalizar e deixar a imagem de forma que os componentes tenham sua curva de distância calculada de maneira correta, é necessário realizar o aguçamento da imagem, ou seja, um aumento na nitidez, com o objetivo de salientar transições de intensidade. Esse processo ocorre após a aplicação do filtro gaussiano, seguido da subtração da imagem suavizada (I_g) da imagem após a erosão (I_e), resultando em uma nova imagem, que é uma máscara.

Com a máscara obtida na subtração, é possível utilizar a equação Equação (16), que representa a adição da máscara multiplicada por um escalar k (com $k \geq 0$), ajustando

a contribuição da máscara de nitidez no resultado final, na imagem após erosão. Através da figura 21, é possível visualizar a imagem com aumento na nitidez, na qual as bordas de cada componente possuem um contorno mais definido, sem tanto ruído.

$$I_{aguçado} = I_e + kI_g \tag{16}$$

Figura 21 – Imagem com aumento de nitidez.

Fonte: Autor, 2024.

1.1.3 Obtenção das Curvas de Distância

Após a filtragem da imagem, é possível partir para obtenção das curvas de distância tanto dos componentes das placas, como também, dos *templates* de todos os caracteres apresentados na Figura 4. Para isso, deve-se realizar a análise ou rotulagem de componentes conectados, que é dada através da função `connectedComponents()` da biblioteca *OpenCV*, conforme apresentado anteriormente neste trabalho.

Após os componentes identificados na imagem e no *template*, pode-se extrair informações como o centróide e as coordenadas do contorno de cada componente. Para obter o contorno, utiliza-se a função `findContours()` que também já foi previamente apresentada. Já para obter o centróide, utiliza-se os momentos de ordem zero e um. O momento de ordem zero é dado pela Equação (11), e os momentos de ordem um, representados pelas Equação (17) e Equação (18), calculam a distribuição da região nas coordenadas x e y em relação à origem.

$$M_{10} = \sum_x \sum_y x \cdot I(x, y) \tag{17}$$

$$M_{01} = \sum_x \sum_y y \cdot I(x, y) \tag{18}$$

Por fim, as coordenadas do centro de massa ou centróide podem ser fornecidas através da Equação (19) e Equação (20), que utilizam a razão entre os momentos de ordem um de x e y e a área definida pelo momento de ordem zero. Ressalta-se que esse cálculo

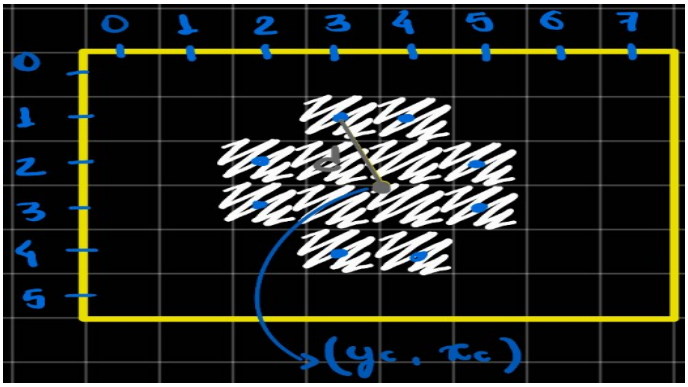
já está implementado no módulo de `visaoComputacional` e pode ser obtido através da função `infoRegiões()`.

$$y_c = \frac{m_{01}}{m_{00}} \tag{19}$$

$$x_c = \frac{m_{01}}{m_{00}} \tag{20}$$

Com os contornos e a forma de calcular os centroides definidos, calcula-se a distância, conforme apresentado na Figura 22. Para isso, foi implementada uma função denominada `calculaCurvaDistancia()`, que é chamada dentro da função `infoRegioes()`. Esta função recebe como parâmetros todas as coordenadas x e y dos *pixels* de contorno, além da localização do centróide. Dentro da função que calcula a curva de distância, pode-se utilizar um laço de repetição que itera sobre cada contorno, calculando a distância entre a sua localização e o centróide através da Equação (21). O resultado será um vetor de pontos em que, quando conectados, resultará em uma curva de distância.

Figura 22 – Cálculo da distância entre os *pixels* de contorno e o centóide.



Fonte: Autor, 2024.

$$d(k) = \sqrt{(y(k) - y_c)^2 + (x(k) - x_c)^2} \tag{21}$$

Ressalta-se que no caso dos *templates*, o processo se torna mais delicado. Primeiramente, é necessário realizar a conversão da imagem para escala de cinza, seguida pela aplicação de um *threshold* utilizando o limiar de *Otsu*, a fim de convertê-la para o espaço de cor binário. Dessa maneira, as letras assumem valores de *pixels* iguais a 255, enquanto o fundo possui valores iguais a 0, conforme mostrado na Figura 23. Após essa etapa, extrai-se os componentes conectados da imagem, possibilitando o cálculo das curvas de distância.

Figura 23 – Imagem contendo os caracteres utilizados na placa no espaço de cor binário.

Fonte: Autor, 2024.

Com o processo de extração finalizado, os componentes conectados são apresentados de forma desordenada, não respeitando a sequência mostrada na figura acima. Para corrigir essa situação, utilizou-se a função `sorted()` do *Python*, que organiza os componentes pela coordenada y do centróide de cada um.

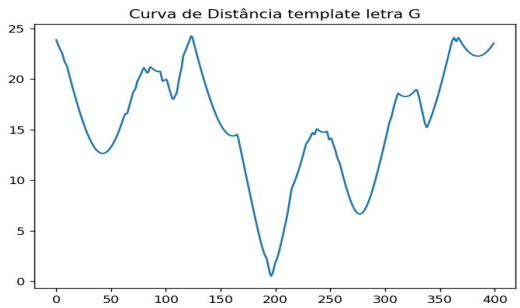
Posteriormente, foi aplicado um laço de repetição para agrupar os componentes já ordenados em y com base na coordenada x do centróide, utilizando uma tolerância de linha definida como 10. Esse procedimento garante a ordenação precisa das letras, respeitando a sequência alfabética dentro de suas respectivas linhas.

O processo de ordenação ocorre em duas etapas: inicialmente, os componentes são agrupados por linha, considerando a proximidade na coordenada y. Na etapa seguinte, dentro de cada linha, os componentes são ordenados pela coordenada x. Como resultado, obtém-se uma lista de componentes organizada de forma consistente, refletindo a posição correta de cada caractere na imagem.

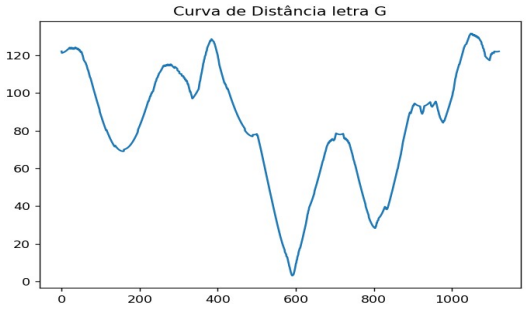
Com as letras devidamente ordenadas, foi criado um dicionário vazio, denominado `dic = {}`, e uma lista contendo todos os caracteres da imagem. Essas estruturas são utilizadas em um laço de repetição que percorre os índices da lista, de 0 até seu tamanho máximo, associando cada caractere a uma entrada no dicionário. Nesse dicionário, as chaves correspondem às letras, enquanto os valores armazenam as curvas de distância calculadas a partir dos *templates* organizados.

1.1.4 Comparação Entre Curvas de Distância

Ao obter as curvas de distância, é necessário compará-las. Esse processo pode ser dividido em etapas, visto que, dependendo do tamanho e da orientação do objeto, a curva pode ter o mesmo formato, mas estar deslocada vertical ou horizontalmente. Além disso, pode apresentar diferenças no comprimento e na escala. Conforme ilustrado na Figura 24, o *template* da letra G difere da mesma letra identificada na placa, tanto em largura quanto em tamanho.



(a) Curva de distância do *template* da letra *G*.



(b) Curva de distância da letra *G* na placa.

Figura 24 – Comparação entre curvas de distância em relação ao tamanho.

Em relação ao comprimento das curvas, uma solução é utilizar a interpolação para ajustar tanto a curva de referência quanto a que se deseja classificar, de modo que ambas tenham o mesmo comprimento. Ou seja, é necessário amostrar as curvas para que ao final, elas possuam o número de elementos definido pelo projetista, que no caso do autor, é de 600 pontos. Assim, ao final, tanto a curva de distância da imagem de referência quanto a da imagem a ser classificada estarão do mesmo tamanho.

A biblioteca *NumPy* possui uma função chamada `interp()` que permite realizar a interpolação, na qual é necessário passar como parâmetros um vetor com o número de pontos desejado para a interpolação. Além disso, deve-se passar um vetor correspondente ao tamanho das curvas e, por fim, os pontos obtidos da curva.

Para o deslocamento vertical e horizontal, como também o problema de escala diferem-se em relação a resolução do problema de comprimento. Para corrigir o problema de deslocamento basta subtrair o valor médio (*offset*) entre as curvas, de modo que fiquem centradas em torno de zero. Assim, realiza-se uma subtração entre a curva de distância, após a interpolação, e o seu valor médio. A implementação em código é feita da seguinte forma: `curva_distancia_interpolada - np.mean(curva_distancia_interpolada)`, cujo resultado é armazenado em uma nova variável que será utilizada adiante.

Já em relação ao problema de escala, utiliza-se o conceito de normalização, ou seja, a curva é dividida por um escalar, que corresponde à energia do sinal, conforme apresentado na Equação (22). Em outras palavras, eleva-se a curva ao quadrado, soma-se todos os pontos e, em seguida, extrai-se a raiz quadrada. Após esse processo, basta dividir todo o sinal pelo escalar obtido. Vale ressaltar que todos esses processos devem ser realizados tanto para a curva de referência quanto para a curva da imagem que se deseja classificar.

$$y_n(n) = \frac{y(n)}{\sqrt{\sum_{n=0}^{N-1} y^2(n)}} \tag{22}$$

Ao final das operações realizadas, pode haver um pequeno deslocamento entre as curvas devido à defasagem na orientação dos objetos. Para corrigir isso, calcula-se a correlação entre as curvas utilizando a equação Equação (23), na qual $d(y_2(n), k)$ representa um deslocamento circular de $y_2(n)$ para a direita por k amostras. Esse deslocamento, ao ser multiplicado por $y_1(n)$, gera valores diferentes de zero. O processo é repetido para diferentes valores de k, resultando em um gráfico cujo pico máximo indica a correlação entre as curvas.

$$C_{y1,y2}(k) = \sum_{n=0}^{N-1} y_1(n) \cdot d(y_2(n), k) \tag{23}$$

Para implementar o cálculo da correlação, o código utiliza laços aninhados, no qual o primeiro laço percorre todas as regiões detectadas na imagem e, para cada região, é aplicada uma filtragem baseada na área, descartando elementos irrelevantes com área inferior a 7000. Em seguida, a curva de distância da região é ajustada e normalizada, preparando-a para a comparação.

No segundo laço, a curva ajustada da região é comparada com as curvas armazenadas em um dicionário. Cada chave do dicionário corresponde ao nome de uma letra e seu valor, à curva de referência associada. Antes da comparação, as curvas de referência também são ajustadas e normalizadas para garantir compatibilidade. A correlação cruzada entre as duas curvas é calculada utilizando deslocamentos circulares, e a similaridade máxima é extraída para cada letra.

Após calcular as similaridades, o nome da letra com maior valor de similaridade é armazenado, juntamente com o centroide da região correspondente. Essa informação é utilizada posteriormente para ordenar as letras com base na posição horizontal (coordenada x) dos centroides. Por fim, as letras ordenadas são concatenadas para formar uma *string* que representa a sequência final (por exemplo, a placa de um veículo). Essa sequência é exibida como saída do programa, conforme apresentado na seção resultados obtidos.

1.2 RESULTADOS OBTIDOS

Nesta seção, serão apresentados os resultados obtidos na Atividade por meio de imagens. No entanto, vale ressaltar que não serão exibidas todas as imagens, mas apenas duas do nível um e duas do nível dois. Destaca-se que uma das placas do nível dois não estava disponível no banco de imagens, portanto, o autor realizou uma busca na internet e adicionou uma imagem adicional.

Através da Figura 25 é possível observar as duas placas que serão analisadas do nível um. Já na Figura 26 estão as duas imagens do nível dois, no qual a Figura 26a contém a imagem que não pertencia inicialmente no banco de dados.

(a) Primeira imagem de uma placa no nível 1.

(b) Segunda imagem de uma placa no nível 1.

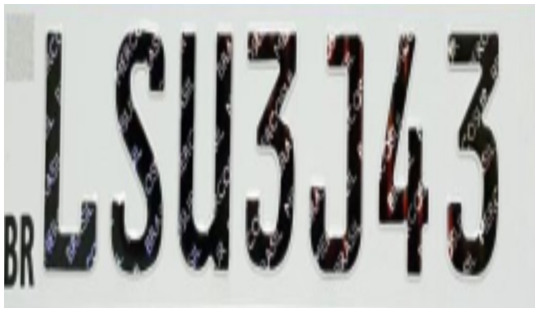
Figura 25 – Imagens de duas placas do nível 1.

(a) Primeira imagem de uma placa no nível 2.

(b) Segunda imagem de uma placa no nível 2.

Figura 26 – Imagens de duas placas do nível 2.

Ao realizar todos os procedimentos para a obtenção dos quatro pontos de forma automática, a homografia planar é realizada. O resultado da homografia planar é mostrado nas Figura 27 e Figura 28, o que demonstra que o código está, de certa forma, generalizado para diversas perspectivas da imagem.



(a) Primeira imagem de uma placa no nível 1 pós-homografia.

(b) Segunda imagem de uma placa no nível 1 pós-homografia.

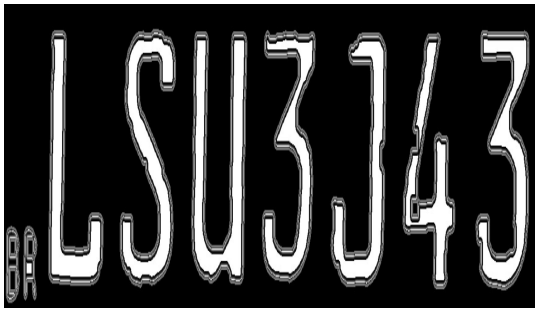
Figura 27 – Imagens de duas placas do nível 1 pós-homografia.

(a) Primeira imagem de uma placa no nível 2 pós-homografia.

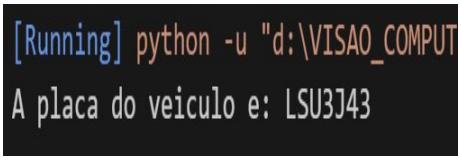
(b) Segunda imagem de uma placa no nível 2 pós-homografia..

Figura 28 – Imagens de duas placas do nível 2 pós-homografia.

Após a aplicação da homografia, realiza-se o processo de filtragem, bem como a obtenção das curvas de distâncias, cujas imagens não serão apresentadas aqui. No entanto, ao realizar esses processos e comparar as curvas de distâncias entre os *templates* e cada letra da placa, é possível obter a identificação completa e exibi-la no terminal *Python*, conforme mostrado nas imagens abaixo.



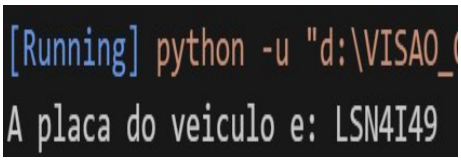
(a) Primeira imagem de uma placa no nível 1 aguçada.



(b) Print dos caracteres identificados da placa 4.

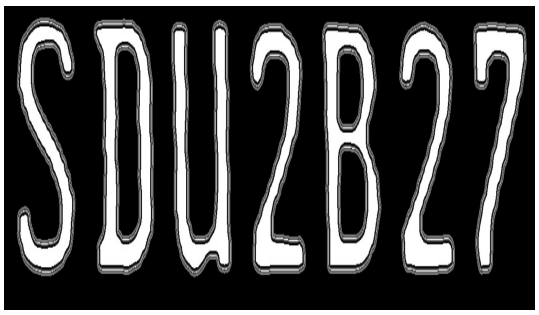
Figura 29 – Resultados obtidos após homografia e identificação da placa LSU3J43.

(a) Segunda imagem de uma placa no nível 1 aguçada.

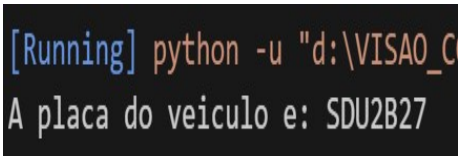


(b) Print dos caracteres identificados da placa 5.

Figura 30 – Resultados obtidos após homografia e identificação da placa LSN4I49.

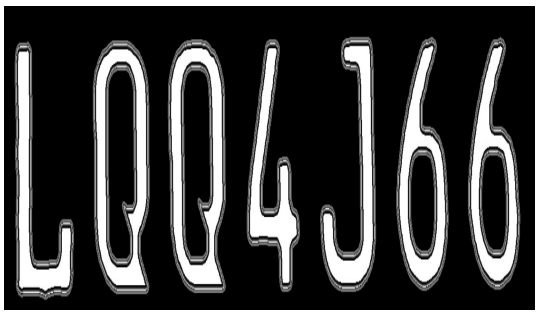


(a) Primeira imagem de uma placa no nível 1 aguçada.

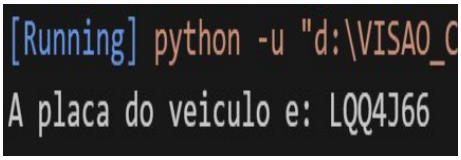


(b) Print dos caracteres identificados da placa 12.

Figura 31 – Resultados obtidos após homografia e identificação da placa SDU2B27.



(a) Segunda imagem de uma placa no nível 2 aguçada.



(b) Print dos caracteres identificados da placa 14.

Figura 32 – Resultados obtidos após homografia e identificação da placa LQQ4J66.

Portanto, conclui-se que o algoritmo está generalizado para aplicação de identificação de caracteres em placas. Não foi realizado um cálculo para a taxa de acerto, contudo, através da simulação que o autor realizou em todas as placas, pode-se dizer que tem 100% de taxa de acerto.