



UNIVERSIDADE FEDERAL DE SANTA CATARINA
DEPARTAMENTO DE ENG. DE CONTROLE, AUTOMAÇÃO E
COMPUTAÇÃO
CURSO DE GRADUAÇÃO EM ENGENHARIA DE CONTROLE E
AUTOMAÇÃO

Bruno Bueno Bronzeri

Trabalho 2: Reconhecimento de caracteres em placas de automóveis

Blumenau
2023

Bruno Bueno Bronzeri

Trabalho 2: Reconhecimento de caracteres em placas de automóveis

Relatório referente ao Trabalho número 2
(dois) da disciplina de Visão Computacional
em Robótica, no curso de graduação de En-
genharia de Controle e Automação na Uni-
versidade Federal de Santa Catarina (UFSC)
- Campus Blumenau.

Orientador:

Prof. Marcos Vinícius Matsuo, Dr.

Blumenau

2023

RESUMO

Este relatório visa abordar um algoritmo capaz de identificar em imagens de placas de carro (padrão Mercosul) os seus respectivos caracteres e retorná-los ao usuário. Ou seja, trata-se de um reconhecimento de caracteres.

Palavras-chave: Algoritmo; Identificar, Caracteres, Reconhecimento.

ABSTRACT

This report aims to approach an algorithm capable of identify characters in license car plate images (Mercosul standard) and return to the user. Therefore, the algorithm is capable of recognize characters.

Keywords: Algorithm; Identify, Characters, Recognize.

LISTA DE FIGURAS

Figura 1 – Exemplo de resultado esperado para o sistema de reconhecimento de caracteres de placas veiculares.	8
Figura 2 – (a) Imagem de entrada. (b) Histograma.	9
Figura 3 – Imagem <code>rice.png</code> limiarizada a partir do método de Otsu.	10
Figura 4 – (a) Não atribuição do valor 1 (erosão). (b) Atribuição do valor 1 (erosão).	11
Figura 5 – (a) Não atribuição do valor 1 (dilatação). (b) Atribuição do valor 1 (dilatação).	11
Figura 6 – (a) Imagem de entrada para preenchimento (binária). (b) Imagem de saída preenchida (binária).	12
Figura 7 – (a) Imagem de Máscara. (b) Imagem com Marcadores. (c) Imagem de Saída Invertida.	13
Figura 8 – (a) Imagem de entrada para detecção de borda. (b) Imagem de saída com bordas detectadas.	14
Figura 9 – Equação da reta e seu problema na Transformada Hough. . . .	15
Figura 10 – Equação da reta em coordenadas polares.	15
Figura 11 – Senoide para um ponto.	16
Figura 12 – (a) Retas traçadas para dois pontos. (b) Comparação das senoides. .	16
Figura 13 – (a) Grade fora de proporção para exemplo. (b) Identificação de picos.	17
Figura 14 – (a) Um ponto na imagem. (b) Curva da Tranformada Hough. . .	17
Figura 15 – (a) Dois pontos na imagem. (b) Curvas da Transformada Hough. .	18
Figura 16 – (a) Reta na imagem. (b) Diversas curvas da Transformada Hough. .	18
Figura 17 – (a) Diversas restas na imagem. (b) Conjunto de retas da Transformada Hough.	18
Figura 18 – (a) Saída da função. (b) saída função com plot da elipse equivalente.	25
Figura 19 – <code>struct</code> de saída da função para componente 1 (um).	25
Figura 20 – (a) Correção placa1. (b) Correção placa2. (c) Correção placa3. .	28
Figura 21 – (a) Caracteres individualmente separados - placa1. (b) Caracteres individualmente separados - placa2.	30
Figura 22 – Caracteres juntados para definição de seus valores.	31
Figura 23 – (a) Banco de letras. (b) Banco de números.	33
Figura 24 – (a) Banco de letras alterado. (b) Banco de números alterado. . .	34
Figura 25 – Apresentação dos caracteres reconhecidos.	37

Figura 26 – (a) N1 placa 1. (b) Resultado N1 placa 1.	38
Figura 27 – (a) N1 placa 2. (b) Resultado N1 placa 2.	38
Figura 28 – (a) N1 placa 3. (b) Resultado N1 placa 3.	39
Figura 29 – (a) N1 placa 4. (b) Resultado N1 placa 4.	39
Figura 30 – (a) N2 placa 2. (b) Resultado N2 placa 2.	39
Figura 31 – (a) N2 placa 3. (b) Resultado N2 placa 3.	40
Figura 32 – (a) N3 placa 1. (b) Resultado N3 placa 1.	40
Figura 33 – (a) N3 placa 2. (b) Resultado N3 placa 2.	40
Figura 34 – (a) N3 placa 3. (b) Resultado N3 placa 3.	41
Figura 35 – (a) N3 placa 4. (b) Resultado N3 placa 4.	41
Figura 36 – (a) N3 placa 5. (b) Resultado N3 placa 5.	41
Figura 37 – (a) N3 placa 6. (b) Resultado N3 placa 6.	42
Figura 38 – (a) N3 placa 7. (b) Resultado N3 placa 7.	42
Figura 39 – (a) N3 placa 8. (b) Resultado N3 placa 8.	42

LISTA DE CÓDIGOS

2.1	Algoritmo de Preenchimento de Buracos.	12
3.1	Obtenção dos pontos de <i>bounding-box</i>	20
3.2	Cálculo da distância Euclidiana.	23
3.3	Cálculo do ângulo a partir da Matriz de Inércia.	23
4.1	Operação de Homografia Planar.	26
4.2	Definição de cada caractere.	28
4.3	Separação dos caracteres em imagens individuais.	29
4.4	Junção dos caracteres em imagens individuais.	31
4.5	Alteração de caracteres no banco.	33
4.6	Comparação de Curvas de Distância.	35

SUMÁRIO

1	IDENTIFICAÇÃO DA PROPOSTA	8
1.1	TÍTULO	8
1.2	TEMA PRINCIPAL	8
2	REVISÃO BIBLIOGRÁFICA	9
2.1	OPERAÇÃO DE LIMIARIZAÇÃO	9
2.2	OPERAÇÃO DE EROSÃO	10
2.3	OPERAÇÃO DE DILATAÇÃO	10
2.4	OPERAÇÃO DE PREENCHIMENTO DE BURACOS	12
2.5	ALGORITMO DE DETECÇÃO DE BORDA	13
2.6	TRANSFORMADA HOUGH	14
2.7	HOMOGRAFIA PLANAR	17
3	FUNÇÃO ANALISA REGIÕES	20
3.1	OBTENÇÃO DA CURVA DE DISTÂNCIAS	21
3.1.1	Momentos de Ordem Zero e Um	21
3.1.2	Centróides	22
3.1.3	Curva de Distância	22
3.2	ÂNGULOS DOS ELEMENTOS	23
3.3	RESULTADOS DA FUNÇÃO	24
4	ALGORITMO PROPOSTO	26
4.1	ABORDAGEM TEÓRICA	26
4.2	HOMOGRAFIA PLANAR	26
4.3	DEFINIÇÃO DA LOCALIZAÇÃO DOS CARACTERES	28
4.4	CONCATENAÇÃO DE CARACTERES	30
4.5	BANCO DE DADOS DE CARACTERES	32
4.6	CURVAS DE DISTÂNCIA	34
4.6.1	Comparação da Curva de Distância	34
4.7	DEFINIÇÃO FINAL DOS CARACTERES	37
4.8	RESULTADOS	38
5	CONCLUSÃO	43
	REFERÊNCIAS	44

1 IDENTIFICAÇÃO DA PROPOSTA

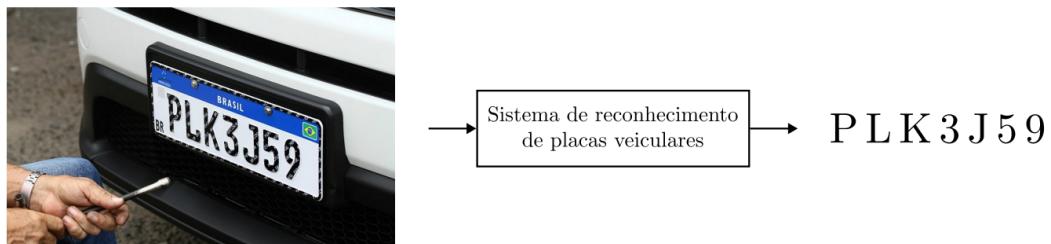
1.1 TÍTULO

Trabalho 2 - Identificação de caracteres em placas de automóveis.

1.2 TEMA PRINCIPAL

Este Trabalho consiste na implementação de um sistema de identificação e reconhecimento de caracteres presentes nas placas de automóveis no padrão Mercosul. O algoritmo em questão consiste em receber três níveis de imagens - nível 1, 2 ou 3 -, os quais tratam-se da dificuldade de localização dos caracteres e por fim retornar ao usuário os caracteres reconhecidos automaticamente nas placas.

Figura 1 – Exemplo de resultado esperado para o sistema de reconhecimento de caracteres de placas veiculares.



Fonte: Material do Professor.

2 REVISÃO BIBLIOGRÁFICA

2.1 OPERAÇÃO DE LIMIARIZAÇÃO

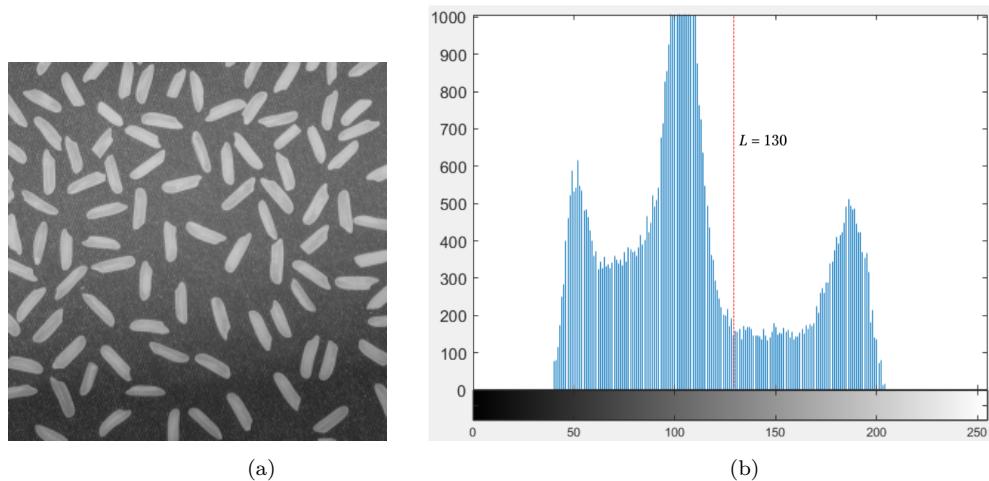
A limiarização é uma operação que consiste em atribuir valores lógicos a partir de uma imagem em escala de cinza. Ou seja, é possível transformar uma imagem em escala de cinza - seja no formato `uint8` ou `double` -, em uma imagem lógica (`logical`) com valores binários a partir de um valor de limiar definido pelo projetista. Em suma, imagens em escala de cinza, para formatos `uint8`, variam de 0 à 255, onde 0 representa preto e 255, o branco. Já para formatos `double`, variam de 0 à 1, onde 0 representa preto e 1, o branco. Esse limiar pode ser definido de forma arbitrária, ou através da função existente no MATLAB, `graythresh()` que já define o limite a partir do método de Otsu, em homenagem a Nobuyuki Otsu. Ou seja, é gerado um histograma a partir da imagem em escala de cinza, conforme exemplificado na Figura 2, e o método de Otsu determina o limiar L que minimiza a variância intraclasse, definido por:

$$\sigma_{\omega}^2(L) = \omega_0(L) \cdot \sigma_0^2(L) + \omega_1(L) \cdot \sigma_1^2(L),$$

onde ω_0 e ω_1 denotam as probabilidades das duas classes separadas por um limite L , e σ_0^2 e σ_1^2 são as variâncias das duas classes (OTSU, 1979).

Dessa forma, é possível visualizar na Figura 2 (b) o limiar de Ostu calculado ($L = 130$). Portanto no MATLAB, dada uma imagem de entrada **I** em escala de

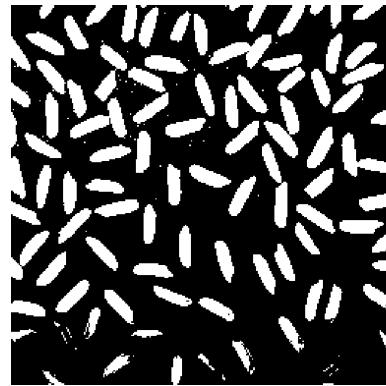
Figura 2 – (a) Imagem de entrada. (b) Histograma.



Fonte: Desenvolvido pelo Autor.

cinza, efetuando a operação lógica $I1 = I > L$, é possível obter $I1$, que é uma imagem lógica limiarizada, conforme mostra a Figura 3.

Figura 3 – Imagem `rice.png` limiarizada a partir do método de Otsu.



Fonte: Desenvolvido pelo Autor.

2.2 OPERAÇÃO DE EROSÃO

As operações morfológicas de erosão são bastante utilizadas para separar regiões ou para suprimir regiões indesejadas em imagens binárias. Nessas operações morfológicas é necessário definir um elemento estruturante (EL), que trata-se de uma imagem binária com dimensão menor que a imagem de entrada e ser processada, onde o *pixel* central representa o *pixel* de referência.

Na operação morfológica de erosão o pré-definido elemento estruturante passa por toda a imagem de entrada e atribui na posição de referência o valor 1 na imagem de saída somente quando todos os *pixels* do EL e a região em branco da imagem de entrada estão totalmente sobrepostos, conforme mostra Figura 4. E por fim sua notação matemática é dada:

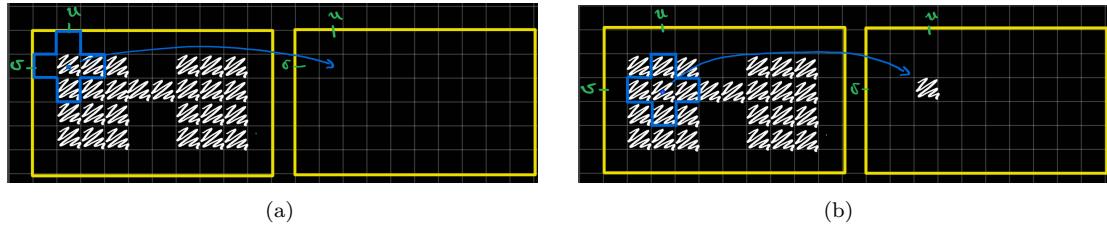
$$\mathbf{O} = \mathbf{I} \ominus \mathbf{S},$$

onde \mathbf{O} é a imagem de saída, \mathbf{I} é a imagem de entrada e \mathbf{S} é o elemento estruturante.

2.3 OPERAÇÃO DE DILATAÇÃO

As operações morfológicas de dilatação são amplamente utilizadas para corrigir "buracos" em imagens binárias, ou seja, preencher regiões que por algum

Figura 4 – (a) Não atribuição do valor 1 (erosão). (b) Atribuição do valor 1 (erosão).



Fonte: Material do Professor.

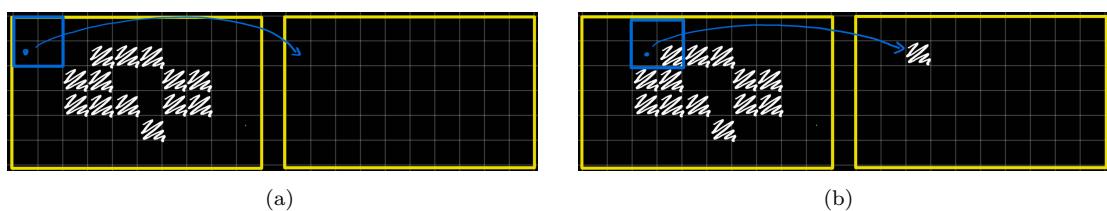
motivo não foram devidamente processadas ou apresentam alguma dificuldade na limiarização de uma região específica. Nessas operações morfológicas é necessário definir também um elemento estruturante (*EL*), com *pixel* de referência também definido.

Nessa operação de dilatação o elemento estruturante passa por toda a imagem de entrada, atribuindo *pixel* com valor 1 na imagem de saída toda vez que de alguma forma ocorrer ao menos uma sobreposição entre o *EL* e a região em branco na imagem de entrada, conforme explicita a Figura 5, valendo ressaltar que a operação de erosão seguida de dilatação é denominada de **abertura** e o contrário - dilatação seguida de erosão -, **fechamento**. E por fim sua notação matemática é dada:

$$\mathbf{O} = \mathbf{I} \oplus \mathbf{S},$$

onde **O** é a imagem de saída, **I** é a imagem de entrada e **S** é o elemento estruturante

Figura 5 – (a) Não atribuição do valor 1 (dilatação). (b) Atribuição do valor 1 (dilatação).

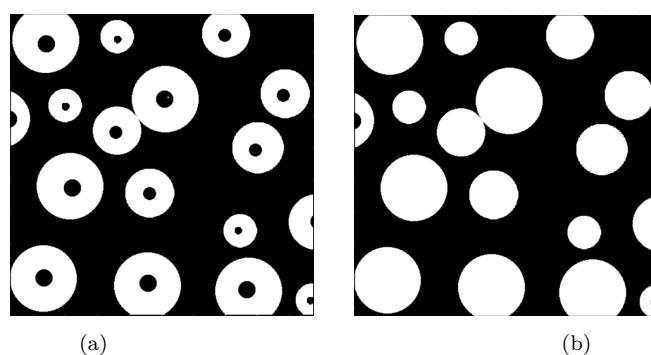


Fonte: Material do Professor.

2.4 OPERAÇÃO DE PREENCHIMENTO DE BURACOS

Este tipo de operação está relacionada com, a partir de uma imagem binária, preencher todos os buracos presentes. Ou seja, se há regiões com valor 0 (zero) ilhadas por valores um (1), - região preta cercada por *pixels* brancos -, como mostra a Figura 6 (a) a função ou algoritmo deve identificá-las e substituir os respectivos valores (zero) por (um), conforme explicitado na Figura 6 (b). A função pronta em MATLAB para isso é a **imfill()**, que recebe como parâmetros a imagem **I** (Figura 6 (a)) e o que deve ser preenchido, no caso, " 'holes' ".

Figura 6 – (a) Imagem de entrada para preenchimento (binária). (b) Imagem de saída preenchida (binária).



Fonte: Desenvolvido pelo Autor.

No entanto é possível implementar essa ação definindo um máscara **G** (Figura 7 (a)) como o inverso da imagem **I** e criar um marcador **F** (Figura 7 (b)), efetuar a reconstrução de **F** e **G** com a função **imreconstruct()** resultando em **H** (Figura 7 (c)). E por fim, inverter seus valores, resultando na imagem **H2** (Figura 6 (b)), conforme mostra o Código 2.1.

Listing 2.1 – Algoritmo de Preenchimento de Buracos.

```

1 % Mask
2 G = 1 - I;
3 figure; imshow(G);
4
5 % Img com Markers
6 F = zeros(size(G));
7 F(:,1) = 1 - I(:,1); % G(:,1)
8 F(:,end) = 1 - I(:,end);

```

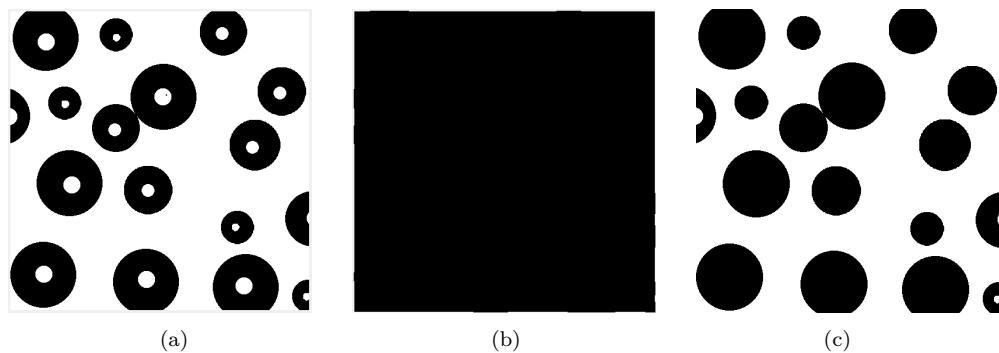
```

9 F(1,:) = 1 - I(1,:);
10 F(end,:) = 1 - I(end,:);
11
12 % Reconstrucao
13 H = imreconstruct(F, G);
14 figure; imshow(H); % resultado inverso
15
16 H2 = 1 - H;
17 figure; imshow(H2);

```

Fonte: Desenvolvido pelo Autor.

Figura 7 – (a) Imagem de Máscara. (b) Imagem com Marcadores. (c) Imagem de Saída Invertida.



Fonte: Desenvolvido pelo Autor.

2.5 ALGORITMO DE DETECÇÃO DE BORDA

O algoritmo de detecção de borda utilizado foi o Algoritmo de Detecção de Borda de Canny, implementado e explicado detalhadamente no Laboratório 2 dessa disciplina. Esse algoritmo consiste em a partir de uma imagem em escala de cinza, filtrá-la, calcular suas derivadas em u e v e os respectivos gradientes, identificando possíveis regiões de borda devido a variação de intensidade. Após é efetuada uma dupla limiarização, com um limiar baixo e outro alto, e através de uma operação diâdica, é possível identificar os pontos fortes e fracos de borda, passando por uma análise de conectividade para então obter uma imagem final de borda, na qual essa possui a espessura de 1 (um) *pixel*, conforme é possível ver da Figura 8.

Figura 8 – (a) Imagem de entrada para detecção de borda. (b) Imagem de saída com bordas detectadas.



Fonte: Desenvolvido pelo Autor.

Além disso, no MATLAB é possível utilizar a função `f_borda_canny`, desenvolvida pelo autor no Laboratório 2, onde como parâmetros recebe a imagem de entrada em escala de cinza, um valor de sigma (σ) para a filtragem e os respectivos limiares baixo e alto. E como outra opção em MATLAB, há a função pronta para detecção de borda de Canny, `edge()`, passados como parâmetros a imagem em escala de cinza e a palavra ”'Canny' ”, obtendo o mesmo resultado da Figura 8.

2.6 TRANSFORMADA HOUGH

A transformada Hough - elaborada por Paul Hough em 1962 -, é uma técnica matemática cujo intuito é, a priori, identificar grupos de *pixels* que pertencem à uma linha reta, mesmo que ruidosa. E portanto, sabe-se que retas são descritas no plano cartesiano pela equação:

$$v = \alpha \cdot u + \beta,$$

onde as características desta reta são a inclinação α e a intersecção β , que podem ser definidos como parâmetros (b, m) . No entanto, esses parâmetros se limitam à medida que as retas tendem a serem verticais, tendo as magnitudes de β e α tendendo ao infinito, conforme mostra a Figura 9.

Assim, para efeitos computacionais, é melhor parametrizar as retas usando dois outros parâmetros, no caso utilizando o sistema coordenadas polares, sendo eles o ρ e o θ , possuindo como equação:

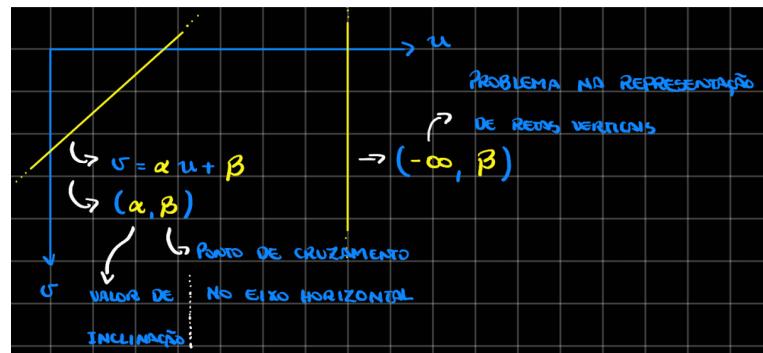
$$\rho = u \cdot \cos \theta + v \cdot \sin \theta,$$

onde θ possui um intervalo dado por:

$$-90^\circ \leq \theta \leq 90^\circ,$$

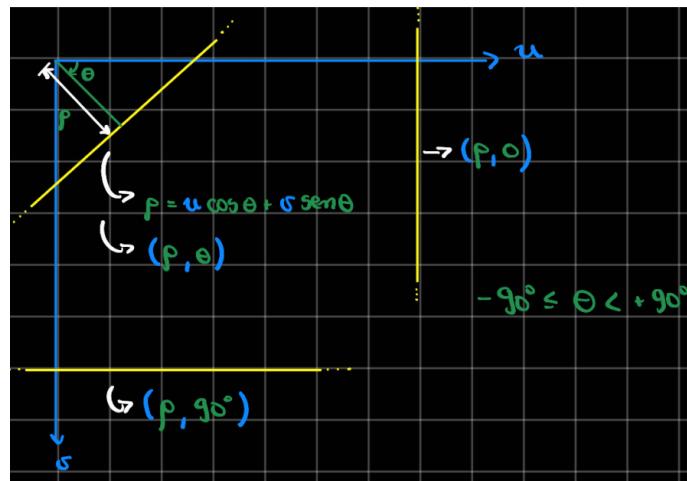
conforme ilustrado na Figura 10

Figura 9 – Equação da reta e seu problema na Transformada Hough.



Fonte: Material do Professor.

Figura 10 – Equação da reta em coordenadas polares.

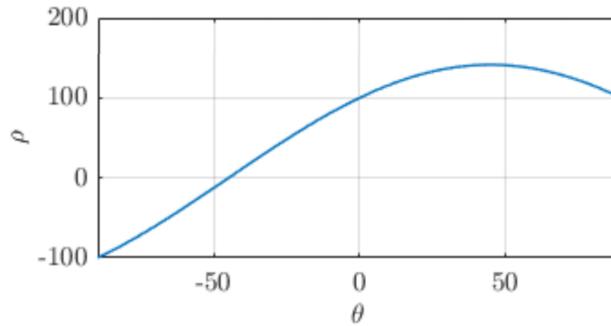


Fonte: Material do Professor.

Sabe-se que infinitas retas passam por um ponto. No caso, existem infinitos conjuntos de parâmetros (ρ_n, θ_n) . Então considerando um ponto (*pixel* branco) na

coordenada $(v, u) = (100, 100)$, a partir da equação da reta, $\rho = 100 \cdot \cos \theta + 100 \cdot \sin \theta$, é possível encontrar como resultado uma senoide, conforme ilustra a Figura 11.

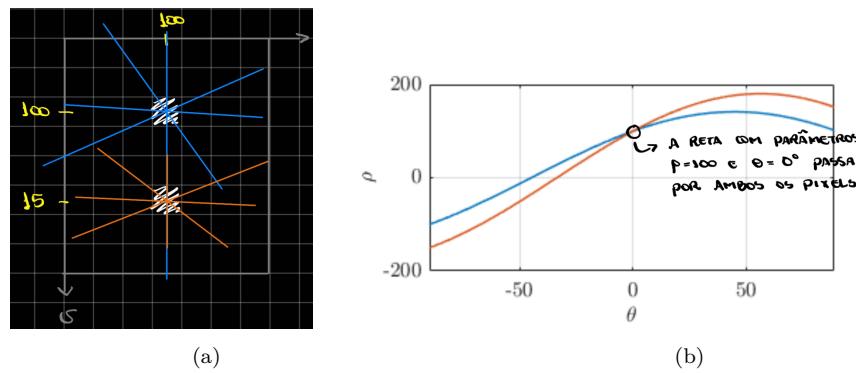
Figura 11 – Senoide para um ponto.



Fonte: Material do Professor.

No entanto, ao considerar duas coordenadas, $(v_1, u_1) = (100, 100)$ e $(v_2, u_2) = (150, 100)$, existem infinitas retas que passam por esses pontos. E para cada *pixel* branco, é possível obter as curvas (ρ_n, θ_n) , conforme mostra a Figura 12, sendo possível identificar os parâmetros da reta que passam pelos dois pontos, como a interseção das duas curvas geradas, no caso, $\rho = 100$ e $\theta = 0^\circ$.

Figura 12 – (a) Retas traçadas para dois pontos. (b) Comparaçāo das senoides.

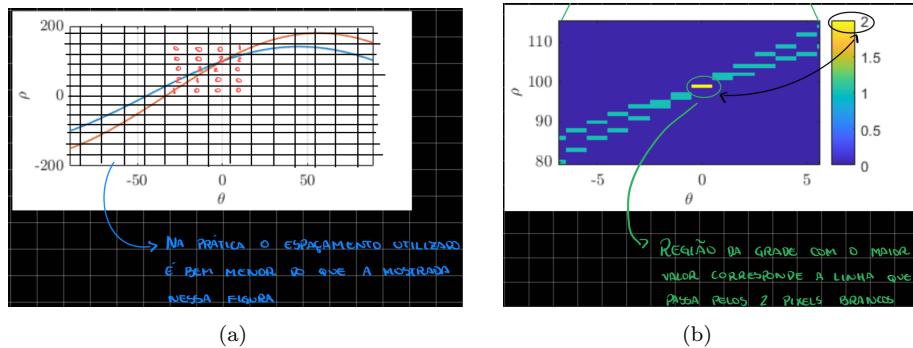


Fonte: Material do Professor.

Quando começa-se a trabalhar com retas na imagem de entrada, cada *pixel* possui sua senoide, portanto, devido ao maior número de *pixels*, essas curvas ficar mais aproximadas, possuindo mais de um ponto de interseção. E para resolver isso, é projetada uma grade com espaçamento muito pequeno e para cada um é calculado a quantidade de retas presentes, resultando aqueles com maior quantidade de retas,

pontos de pico, os quais representam os melhores parâmetros ρ e θ para a reta resultante, conforme mostra a Figura 13.

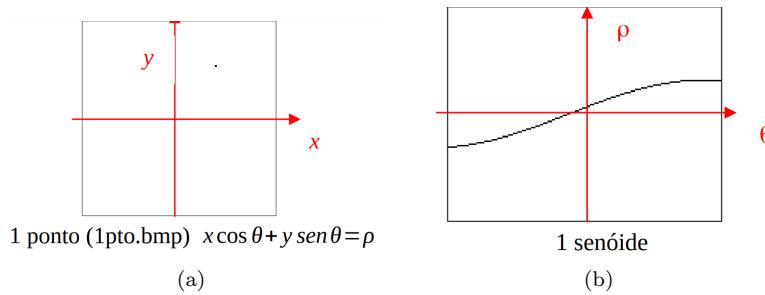
Figura 13 – (a) Grade fora de proporção para exemplo. (b) Identificação de picos.



Fonte: Material do Professor.

Em suma, foi possível definir o comportamento da análise para cada caso. Ou seja, como são identificados esses pontos para pontos sozinhos, dois ou mais pontos, uma reta ou duas ou mais retas. Então é possível visualizar em Figura 14, Figura 15, Figura 16, Figura 17.

Figura 14 – (a) Um ponto na imagem. (b) Curva da Transformada Hough.

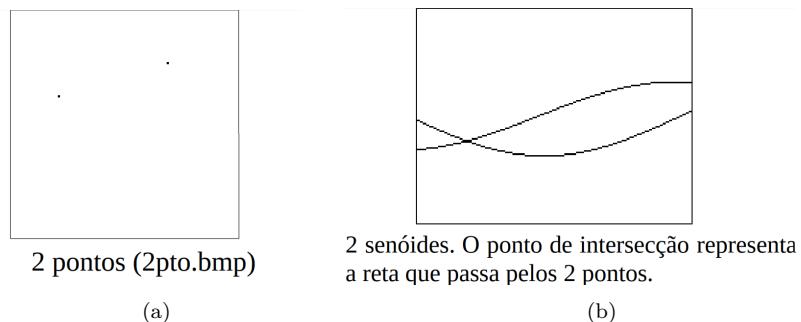


Fonte: Laboratório de Processamento de Sinais (LPS - USP).

2.7 HOMOGRAFIA PLANAR

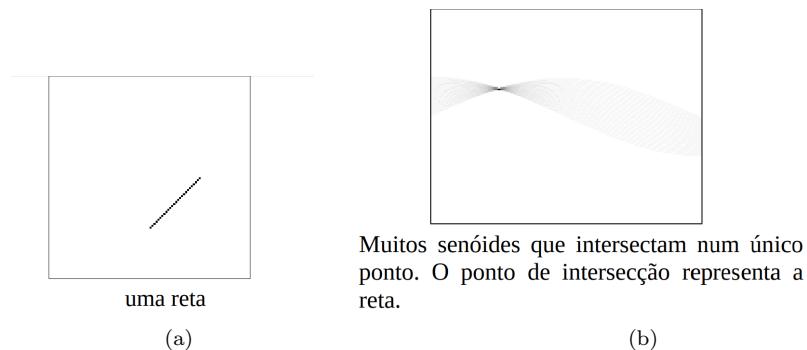
A homografia planar é uma técnica de transformação projetiva que consiste em corrigir a perspectiva de uma imagem, ou seja, muitas vezes em visão computacional, a imagem de entrada não pode ser obtida de modo a ter a câmera posicionada perpendicularmente ao objeto de interesse. Dessa forma, a homografia planar consiste em adquirir 4 (quatro) coordenadas na imagem de entrada que

Figura 15 – (a) Dois pontos na imagem. (b) Curvas da Transformada Hough.



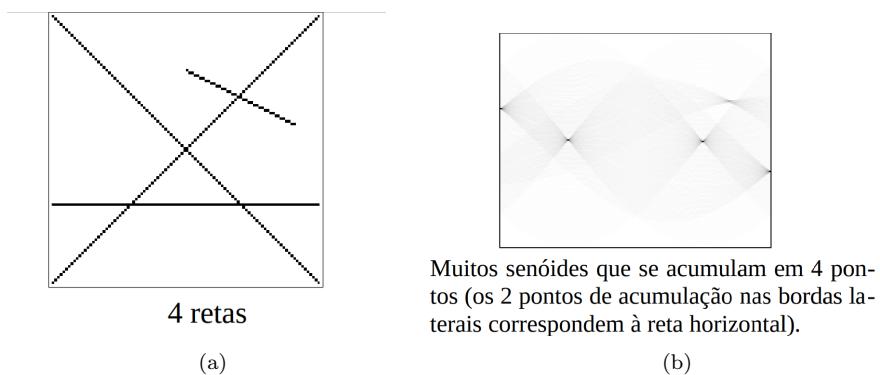
Fonte: Laboratório de Processamento de Sinais (LPS - USP).

Figura 16 – (a) Reta na imagem. (b) Diversas curvas da Transformada Hough.



Fonte: Laboratório de Processamento de Sinais (LPS - USP).

Figura 17 – (a) Diversas retas na imagem. (b) Conjunto de retas da Transformada Hough.



Fonte: Laboratório de Processamento de Sinais (LPS - USP).

servirão de referência para remapeá-la aos 4 (pontos) escolhidos pelo projetista - geralmente em forma de quadrado ou retângulo - para corrigir a perspectiva.

Essa transformação ocorre por intermédio de uma matriz de transformação homogênea com fator de escalamento 1 (um) dada pela Equação (1).

$$\begin{bmatrix} \tilde{u}_i \\ \tilde{v}_i \\ \tilde{w}_i \end{bmatrix} = \begin{bmatrix} h_{1,1} & h_{1,2} & h_{1,3} \\ h_{2,1} & h_{2,2} & h_{2,3} \\ h_{3,1} & h_{3,2} & 1 \end{bmatrix} \cdot \begin{bmatrix} u_i \\ v_i \\ 1 \end{bmatrix} \quad (1)$$

As relações para as coordenadas na imagem transformada é dada pela relação $u' = \frac{\tilde{u}_i}{\tilde{w}_i}$ e $v' = \frac{\tilde{v}_i}{\tilde{w}_i}$. Os coeficientes $h_{i,j}$ podem ser definidos pelas Equações (2) e (3).

$$u'_i = \frac{\tilde{u}_i}{\tilde{w}_i} = \frac{h_{1,1} \cdot u_i + h_{1,2} \cdot v_i + h_{1,3}}{h_{3,1} \cdot u_i + h_{3,2} \cdot v_i + 1}. \quad (2)$$

$$v'_i = \frac{\tilde{v}_i}{\tilde{w}_i} = \frac{h_{2,1} \cdot u_i + h_{2,2} \cdot v_i + h_{2,3}}{h_{3,1} \cdot u_i + h_{3,2} \cdot v_i + 1}. \quad (3)$$

Através dessas equações é possível operá-las e encontrar uma forma matricial de escrevê-las, facilitando a aplicação da mesma. Essa matriz por ser escrita conforme mostra a Equação (4).

$$\begin{bmatrix} u_1 & v_1 & 1 & 0 & 0 & 0 & -u'_1 \cdot u_1 & -u'_1 \cdot v_1 \\ 0 & 0 & 0 & u_1 & v_1 & 1 & -v'_1 \cdot u_1 & -v'_1 \cdot v_1 \\ u_2 & v_2 & 1 & 0 & 0 & 0 & -u'_2 \cdot u_2 & -u'_2 \cdot v_2 \\ 0 & 0 & 0 & u_2 & v_2 & 1 & -v'_2 \cdot u_2 & -v'_2 \cdot v_2 \\ u_3 & v_3 & 1 & 0 & 0 & 0 & -u'_3 \cdot u_3 & -u'_3 \cdot v_3 \\ 0 & 0 & 0 & u_3 & v_3 & 1 & -v'_3 \cdot u_3 & -v'_3 \cdot v_3 \\ u_4 & v_4 & 1 & 0 & 0 & 0 & -u'_4 \cdot u_4 & -u'_4 \cdot v_4 \\ 0 & 0 & 0 & u_4 & v_4 & 1 & -v'_4 \cdot u_4 & -v'_4 \cdot v_4 \end{bmatrix} \cdot \begin{bmatrix} h_{1,1} \\ h_{1,2} \\ h_{1,3} \\ h_{2,1} \\ h_{2,2} \\ h_{2,3} \\ h_{3,1} \\ h_{3,2} \end{bmatrix} = \begin{bmatrix} u'_1 \\ v'_1 \\ u'_2 \\ v'_2 \\ u'_3 \\ v'_3 \\ u'_4 \\ v'_4 \end{bmatrix} \quad (4)$$

A

h

b

Com isso é fácil determinar os coeficientes 'h', uma vez que o seguinte sistema pode ser resolvido, como mostra Equação (5). Com esses coeficientes já é possível obter a matriz de homografia planar pronta para aplicação.

$$h = A^{-1} \cdot b. \quad (5)$$

3 FUNÇÃO ANALISA REGIÕES

A função de *bounding box* e analisa regiões foi implementada em MATLAB pelo autor e recebe o nome de `f_bb_analisa_regioes`, recebendo quatro parâmetros, sendo respectivamente a imagem de entrada binária, valor "0" para não interpolação dos pontos de borda da imagem ou valor "1" para interpolação dos pontos (recomendado), o terceiro parâmetro é a quantidade de pontos limitados pós interpolação, e o último pode receber valor "0" para não apresentar os pontos de borda ou valor "1" para representá-los.

Essa função utiliza a ferramenta `bwlable()`, que faz a análise de componentes conectados, isto é, a partir de uma imagem binária, essa função consegue identificar todos os componentes presentes na imagem, atribuindo-lhes um rótulo numérico de acordo com a ordem que esses são processados. Então o componente 1 possui todos os seus *pixels* brancos substituídos pelo número 1, tal como o componente 2 possui todos os seus *pixels* brancos substituídos pelo número 2 e assim por diante, por exemplo. Dessa forma, utilizando `[Ilabel, N] = bwlable(I)` é possível saber a quantidade de componentes na imagem, onde N trata-se da quantidade de componentes presentes na imagem e I representa a imagem de entrada a ser identificados os componentes.

Dessa forma, com os componentes localizados, é possível calcular os respectivos *bounding-boxes*, uma vez que um laço de reição é percorrido o número de vezes referentes à quantidade de elementos conectados identificados. Esses por sua vez, consistem em, após identificados os *pixels* de menor e maior valor em u e em v , é possível obter dois pontos opostos pela diagonal do *box*, possibilitando a obtenção dos outros dois pontos, conforme aborda o Código 3.1 e como é possível observar na Figura 18.

Listing 3.1 – Obtenção dos pontos de *bounding-box*.

```

1 % Regiao de Interesse
2 I2 = (Ilabel == i);
3
4 % Bounding Box
5 [v,u] = find(I2);
6 vmin = min(v);
7 umin = min(u);
8 vmax = max(v);
9 umax = max(u);

```

```

10
11     bb = [umin, vmin, umax, vmax];
12
13     hold on, plot([ umin , umin ], [ vmin , vmax ] , 'y');
14     hold on, plot([ umin , umax ], [ vmin , vmin ], 'y');
15     hold on, plot([ umax , umax ], [ vmin , vmax ] , 'y');
16     hold on, plot([ umin , umax ], [ vmax , vmax ], 'y');

```

Fonte: Desenvolvido pelo Autor.

3.1 OBTENÇÃO DA CURVA DE DISTÂNCIAS

O objetivo desejado é a obtenção da curva de distância de cada componente da imagem, e para isso, é necessário algumas informações anteriores, como os momentos de ordem 0 (zero) e 1 (um), tal como os centróides de cada componente, abordados todos à frente. Com isso, a partir da ordenação das coordenadas dos *pixels* de borda, é possível encontrar o vetor referente aos valores da curva de distância, calculados como sendo a distância Euclidiana entre o ponto na borda e o centróide da imagem.

3.1.1 Momentos de Ordem Zero e Um

Os momentos são utilizados para descrever formato de regiões, tamanho e até mesmo localização, possuindo sua interpretação física na distribuição de massa. E sua interpretação matemática é dada por:

$$m_{pq} = \sum_{(u,v) \in \mathbf{I}} u^p \cdot v^q \cdot \mathbf{I}(u, v),$$

onde $p + q$ representa a ordem do momento.

Destarte existem dois momentos de ordem 1 (um) e um momento de ordem 0 (zero), definidos pelas Equações (6), (7) e (8), respectivamente.

$$m_{01} = \sum_{(u,v) \in \mathbf{I}} v^q \cdot \mathbf{I}(u, v). \quad (6)$$

Esse momento de ordem 1 (um) retorna a distribuição da região do componente em relação ao eixo horizontal, ou seja, um escalar.

$$m_{10} = \sum_{(u,v) \in \mathbf{I}} u^p \cdot \mathbf{I}(u, v). \quad (7)$$

Esse momento de ordem 1 (um) retorna distribuição da região do componente em relação ao eixo vertical, ou seja, um escalar.

$$m_{01} = \sum_{(u,v) \in \mathbf{I}} \mathbf{I}(u, v) \quad (8)$$

Esse momento de ordem 0 (zero) retorna o valor da área da região do componente. E com esses momentos calculados, é possível obter as coordenadas dos centróides.

3.1.2 Centróides

Para o cálculo dos centróides, basta apenas aplicar os momentos já calculados em operações de divisão da seguinte forma:

- Para v_c :

$$v_c = \frac{m_{01}}{m_{00}};$$

- Para u_c :

$$u_c = \frac{m_{10}}{m_{00}};$$

Logo foi possível obter a coordenada do centróide do componente. Isso permite o cálculo da curva de distância desejada.

3.1.3 Curva de Distância

A curva de distância pode ser obtida a partir do cálculo da distância Euclidiana entre o centróide e a coordenada do *pixel* da borda. Para isso é preciso mapear esses *pixels* e armazená-los em um vetor coluna de forma ordenada, possibilitando até extraír informações importantes sobre a forma da região.

Para isso, foi utilizada a função `bwtraceboundary`, onde ao ser aplicado em um *pixel* branco de borda, essa percorre toda ela e atribui os valores das coordenadas de forma ordenada. No caso esses dados foram salvos em uma matriz de duas colunas, uma para v e outra para u , chamada `boundaryLim`.

Os parâmetros utilizados na função `bwtraceboundary`, respectivamente, foram a imagem de entrada, a coordenada do ponto inicial, a string '*N*' - que representa norte, ou *north* do inglês -, e o parâmetro *default*, 8, que constitui uma matriz 8 por 2 (oito por dois), onde o número 8 é o número de *pixels* de limite para a região de análise.

Por fim, foi feito um laço **for** para percorrer todos os pontos de borda e calcular a distância Euclidiana de cada um em relação ao centróide. O algoritmo pode ser visualizado em Código 3.2 e a Equação (9) para cálculo da distância.

$$D = \sqrt{(v_{borda} - v_c)^2 + (u_{borda} - u_c)^2}. \quad (9)$$

Listing 3.2 – Cálculo da distância Euclidiana.

```

1 % Curva de Distancia
2 aux1 = aux(1); % tamanho da matriz de pontos de
                 borda
3 DC = zeros(1,aux1);
4 for k=1:aux1
5     DC(k) = sqrt((boundaryLim(k,1)-vc(1,i))^2+
                 ...
6                 (boundaryLim(k,2)-uc(1,i))^2);
7 end

```

Fonte: Desenvolvido pelo Autor.

3.2 ÂNGULOS DOS ELEMENTOS

Para o cálculo dos ângulos, é necessário em primeiro lugar a obtenção da matriz de inércia J_e equivalente, da qual é extraído o autovalor - direção do vetor - e autovetor - relacionado com o valor dos raios -, através da função $[V,D] = \text{eig}(J_e)$. E então ao aplicar o arco tangente do segundo valor do autovalor dividido pelo seu primeiro valor, é possível obter o ângulo do componente, conforme mostra o Código 3.3

Listing 3.3 – Cálculo do ângulo a partir da Matriz de Inércia.

```

1 % Matriz de Inercia da elipse equivalente
2 Je = 4/m00(1,i) * [u20, u11; u11, u02];
3
4 % Raios da elipse a partir dos autovalores/vetores
5 [V,D] = eig(Je); % V -> autovetor (direcao vetor !=
                  raios)
6 % D -> auto valores (rel. com valor dos raios)
7

```

```

8      lb1 = D(1,1);
9      lb2 = D(2,2);
10
11      a(1,i) = sqrt(lb1); % raios da elipse
12      b(1,i) = sqrt(lb2);
13      % razao de raios
14      r(1,i) = a(1,i)/b(1,i);
15
16      % calculo do angulo da elipse
17      v2 = V(:,2); % componentes horizontal e vertical
                      respec.
18      theta(1,i) = atand(v2(2)/v2(1));

```

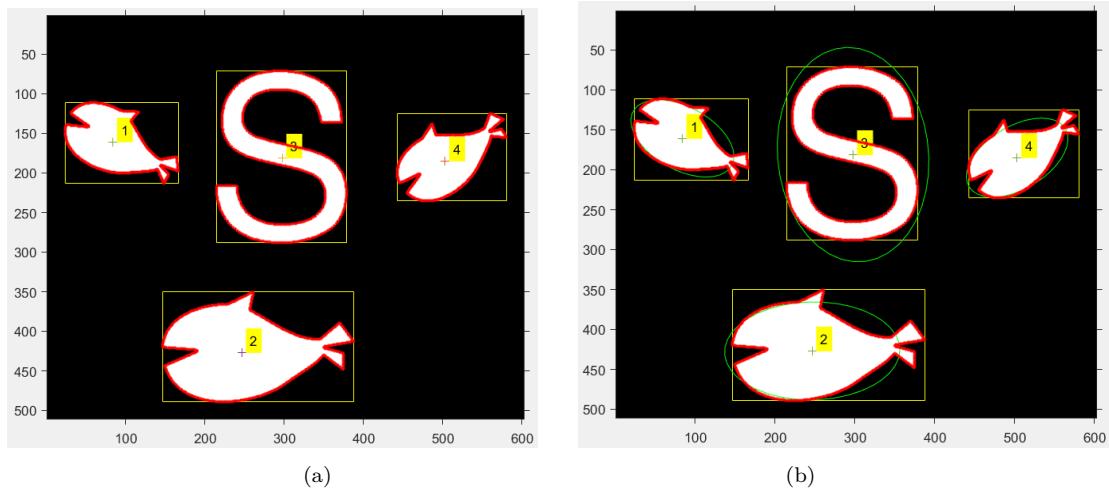
Fonte: Desenvolvido pelo Autor.

No final, há a criação de uma **structure** para poder armazenar várias informações de vários componentes de uma mesma imagem. Outras diversas operações foram implementadas, no entanto, para a realização desse Trabalho, muitas delas não serão utilizadas, portanto, para fim de não abordar assuntos não relevantes a implementação do sistema de localização de caracteres, o aprofundamento teórico nesses tópicos serão tratados em documentos subsequentes a esse.

3.3 RESULTADOS DA FUNÇÃO

Como já mencionado, a função realiza diversas operações, sendo bastante genérica no que diz respeito à análise de regiões de componentes conectados. Um exemplo de saída dessa função pode ser analisado na Figura 18, onde (a) representa a saída padrão da função, porém (b) possui a atribuição gráfica das elipses equivalentes de cada componente, por intermédio de função **plot_ellipse**, disponibilizada pelo Professor, na qual os parâmetro são as matrizes de inércia da elipse e as coordenadas dos centróides de cada componente. Outro exemplo de saída dessa função discutida nesse capítulo, é a **struct** gerada. No caso, na Figura 19, é possível visualizar as informações extraídas de cada componente, apesar de na figura estar representado somente as do componente classificado pelo número 1 (um).

Figura 18 – (a) Saída da função. (b) saída função com plot da elipse equivalente.



Fonte: Desenvolvido pelo Autor.

Figura 19 – **struct** de saída da função para componente 1 (um).

```

>> info{1}

ans =

  struct with fields:

    bb: [111 24 213 167]
    area: 7746
    centroide: [160.6634 84.2352]
    m_inercia: [2x2 double]
        razao: 0.5580
        angulo: 28.8709
        edge: [445x2 double]
    perimetro: 499.6288
    circularidade: 0.3899
    curva_dist: [1x400 double]
    curva_angulo: [1x400 double]

```

Fonte: Desenvolvido pelo Autor.

4 ALGORITMO PROPOSTO

Como continuação do Trabalho 1, o algoritmo efetua todas as operações referentes ao mesmo. No entanto, na intenção de reconhecer o valor de cada caractere presente na placa, algumas funcionalidades foram incluídas.

O algoritmo uma vez que possui através da transformada Hough os quatro pontos de quina (vértices) da parte azul da placa ordenados e a região de interesse com os caracteres, foi possível com esses pontos aplicar uma homografia planar corrigindo a perspectiva da placa, facilitando a leitura dos caracteres. E então algumas operações morfológicas foram impostas aos caracteres de modo a isolá-los e então ao analisar cada um particularmente com sua respectiva curva de distância, é possível compará-la ao do banco de dados e retornar qual caractere corresponde.

4.1 ABORDAGEM TEÓRICA

Anteriormente foi descrito de forma sucinta a estrutura e sequenciamento do algoritmo. No entanto, o que segue tem o objetivo de fazer uma abordagem teórica do passo a passo do algoritmo implementado do modo a apresentar os resultados intermediários, possíveis falhas e complicações que implicaram na alternância de método de solução.

4.2 HOMOGRAFIA PLANAR

No algoritmo do Trabalho 1, os quatro pontos já eram obtidos na transformada Hough e ordenados de modo a sempre obter os vértices ordenados para todas as placas. Então, através da imagem sem homografia com a máscara aplicada, ou seja, a imagem que contém somente a área dos caracteres sem correção, passa pela matriz de homografia com os quatro pontos da transformada Hough, os quais são mapeados para um retângulo iniciado no primeiro vértice da placa e seguindo proporções de dimensão da placa, de modo a essa ser mapeada para um retângulo com as dimensões mais aproximadas à realidade. Abaixo é possível visualizar o Código 4.1 que explicita a operação correção de perspectiva para as placas, onde **R2** é a imagem de entrada e **T** a imagem de saída, tal como **pi1**, **pi2**, **pf1** e **pf2** são os pontos da transformada Hough.

Listing 4.1 – Operação de Homografia Planar.

```
1 % Matriz de homografia planar
2 % Coordenadas
```

```

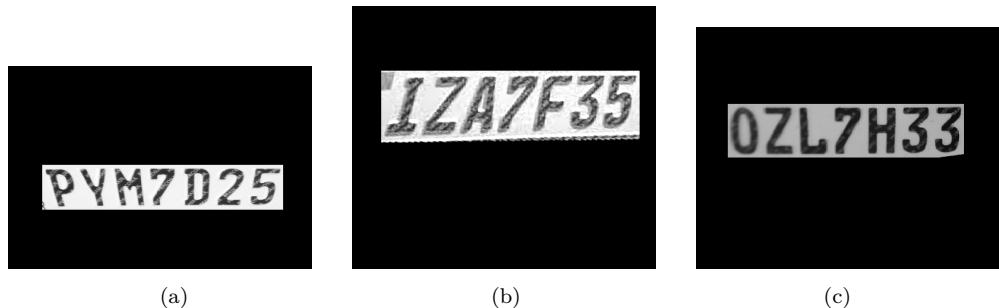
3 u1 = pi1(1); u2 = pi2(1); u3 = pf2(1); u4 = pf1(1);
4 v1 = pi1(2); v2 = pi2(2); v3 = pf2(2); v4 = pf1(2);
5 uc1 = pi1(1); uc2 = pi1(1); uc3 = pi1(1)+400; uc4 = pi1
   (1)+400;
6 vc1 = pi1(2); vc2 = pi1(2)+26; vc3 = pi1(2)+26; vc4 =
   pi1(2);
7
8 % Matriz de homografia (projecao)
9 A = [u1 v1 1 0 0 0 -uc1*u1 -uc1*v1 ;
10      0 0 0 u1 v1 1 -vc1*u1 -vc1*v1 ;
11      u2 v2 1 0 0 0 -uc2*u2 -uc2*v2 ;
12      0 0 0 u2 v2 1 -vc2*u2 -vc2*v2 ;
13      u3 v3 1 0 0 0 -uc3*u3 -uc3*v3 ;
14      0 0 0 u3 v3 1 -vc3*u3 -vc3*v3 ;
15      u4 v4 1 0 0 0 -uc4*u4 -uc4*v4 ;
16      0 0 0 u4 v4 1 -vc4*u4 -vc4*v4 ];
17
18 b = [uc1; vc1; uc2; vc2; uc3; vc3; uc4; vc4];
19 h = inv(A)*b;
20
21 % Matriz de homografia com coeficientes definidos
22 H = [h(1) h(2) h(3);
23       h(4) h(5) h(6);
24       h(7) h(8) 1];
25
26 tform = projective2d(H.');
27 T = imwarp(R2, tform);

```

Fonte: Desenvolvido pelo Autor.

É possível observar na Figura 20 três resultados da correção de perspectiva aplicados para as placas de nível 3 (três). Ademais, na Figura 20 (b) foi ainda necessário a aplicação de um filtro e um aumento de saturação para melhorar sua qualidade, visto que a imagem era muito pequena, portanto bastante ruidosa próximo aos caracteres

Figura 20 – (a) Correção placa1. (b) Correção placa2. (c) Correção placa3.



Fonte: Desenvolvido pelo Autor.

4.3 DEFINIÇÃO DA LOCALIZAÇÃO DOS CARACTERES

Após a correção, uma detecção de borda é aplicada para melhor seleção dos caracteres - método de Sobel para imagens pequenas por considerar menos ruídos e Canny para imagens maiores -. E então é feito um fechamento para identificar as características de cada caractere por uma função que, para fins de reduzir custo computacional, extrai somente os *bounding-boxes*, conforme o Código 4.2.

Listing 4.2 – Definição de cada caractere.

```

1 if(size(I,1)<170)
2     T = imadjust(T);
3     T = f_filter_gauss(T, 0.5, 1);
4     border = 'Sobel';
5 else
6     border = 'Canny';
7 end
8
9 % Obtencao de bordas para selecionar regiao desejada
10 I6 = edge(T, border);
11
12 % Dilatacao, preenchimento e erosao
13 if(size(I,1)<320)
14     R3 = imdilate(I6, strel('square', 3));
15     R3 = imerode(R3, strel('square', 2));
16 else
17     R3 = imdilate(I6, strel('square', 3));

```

```

18 R3 = imerode(R3, strel('square', 2));
19 end
20
21 %figure; imshow(R3);
22 Rinfo = f_bounding_box(R3);

```

Fonte: Desenvolvido pelo Autor.

Feito isso, algumas condicionais extraem os maiores e segundos maiores *bounding-boxes* para excluir ruídos e por fim através da comparação de *bounding-boxes* de dimensões semelhantes seleciona somente aqueles que são caracteres, criando uma delimitação de área mais estrita ainda aos mesmos.

Em seguida foram definidas imagens individuais para cada caractere para melhor operá-los de acordo com as necessidades de cada posição para cada placa. Então através dos *bounding-boxes* definidos como mencionado acima, conforme Código 4.3 que representa a separação dos dois primeiros caracteres para evitar a repetição de código, foi possível separar individualmente os caracteres. E com os esses separados, foi aplicado à eles uma detecção de borda e uma sucessão de operações morfológicas de erosão e dilatação de acordo com cada caractere e cada tamanho de placa. Além disso, também foi utilizada a função `imclearborder` e `imfill` no intuito de remover o que não fazia parte do caractere e preenchê-lo de modo a extrair suas informações, obtendo como resultado o que explica a Figura 21.

Listing 4.3 – Separação dos caracteres em imagens individuais.

```

1 % Definicao de imagens individuais para os caracteres
2 C1 = zeros(bb(1,4)-bb(1,2), bb(1,3)-bb(1,1));
3 C2 = zeros(bb(2,4)-bb(2,2), bb(2,3)-bb(2,1));
4 C3 = zeros(bb(3,4)-bb(3,2), bb(3,3)-bb(3,1));
5 C4 = zeros(bb(4,4)-bb(4,2), bb(4,3)-bb(4,1));
6 C5 = zeros(bb(5,4)-bb(5,2), bb(5,3)-bb(5,1));
7 C6 = zeros(bb(6,4)-bb(6,2), bb(6,3)-bb(6,1));
8 C7 = zeros(bb(7,4)-bb(7,2), bb(7,3)-bb(7,1));
9 vv = 0;
10
11 for i = bb(1,1):bb(1,3)
12     uu = 0; vv = vv+1;

```

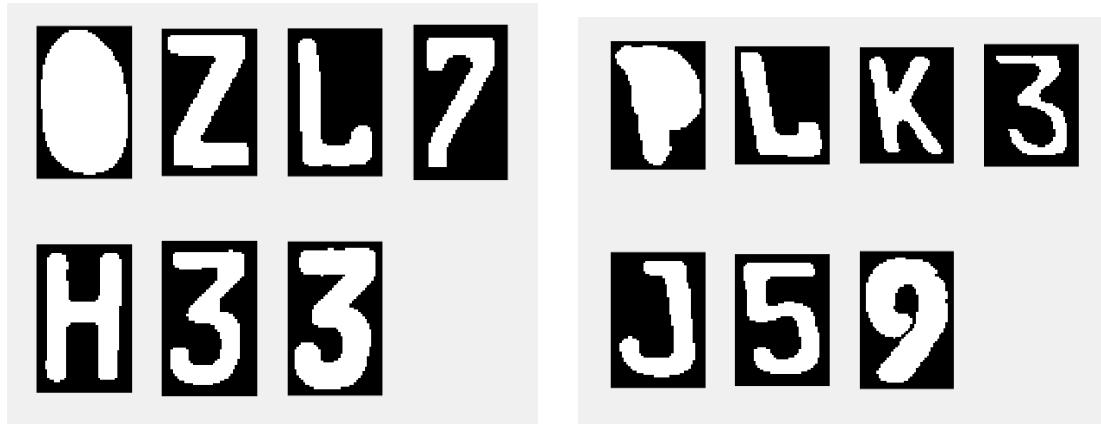
```

13   for j = bb(1,2):bb(1,4)
14     uu = uu+1;
15     C1(uu,vv) = Z(i,j);
16   end
17 end
18 vv = 0;
19 for i = bb(2,1):bb(2,3)
20   uu = 0; vv = vv+1;
21   for j = bb(2,2):bb(2,4)
22     uu = uu+1;
23     C2(uu,vv) = Z(i,j);
24   end
25 end

```

Fonte: Desenvolvido pelo Autor.

Figura 21 – (a) Caracteres individualmente separados - placa1. (b) Caracteres individualmente separados - placa2.



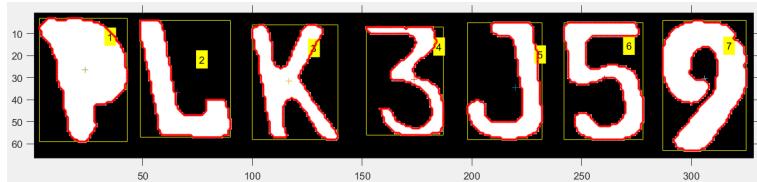
Fonte: Desenvolvido pelo Autor.

4.4 CONCATENAÇÃO DE CARACTERES

Após definição clara de cada caractere sem ruídos, foi decidida a junção de todos eles em uma única imagem para poder utilizar um laço **for** para percorrer todos eles de forma mais automática e eficiente. Então à essa imagem -exemplificada

pela Figura 22- foi aplicado a função `f_bb_analisa_regioes` desenvolvida pelo autor, como mostra o Código 4.4. Vale ressaltar que nesse código são encontradas as informações do banco de dados definidos como imagens **P1** e **P2** que serão tratados à frente.

Figura 22 – Caracteres juntados para definição de seus valores.



Fonte: Desenvolvido pelo Autor.

Listing 4.4 – Junção dos caracteres em imagens individuais.

```

1 % Definicao de uma imagem preta para juntar todos os
2 % caracteres operados
3 maiorH = [size(C1,2), size(C2,2), size(C3,2), size(C4,2)
4 , size(C5,2), ...
5 size(C6,2), size(C7,2)];
6 comprL = size(C1,1) + size(C2,1) + size(C3,1) + size(C4
7 ,1) + ...
8 size(C5,1) + size(C6,1) + size(C7,1);
9 maiorH = max(maiorH);
10
11 imC = zeros(maiorH, comprL);
12
13 Sc1 = size(C1); Sc2 = size(C2); Sc3 = size(C3); Sc4 =
14 size(C4);
15 Sc5 = size(C5); Sc6 = size(C6); Sc7 = size(C7);
16
17 % Adicionando cada caracter individual ordenado um ao
18 % lado do outro para
19 % analise unica
20 imC(1:Sc1(2), 1:Sc1(1)) = C1.';
21 imC(1:Sc2(2), Sc1(1)+1:Sc1(1)+Sc2(1)) = C2.';
```

```

17 imC(1: Sc3(2), Sc1(1)+Sc2(1)+1: Sc1(1)+Sc2(1)+Sc3(1)) = C3
18 .';
19 imC(1: Sc4(2), Sc1(1)+Sc2(1)+Sc3(1)+1: Sc1(1)+Sc2(1)+Sc3
20 (1)+Sc4(1)) = C4.';
21 imC(1: Sc5(2), Sc1(1)+Sc2(1)+Sc3(1)+Sc4(1)+1: Sc1(1)+Sc2
22 (1)+Sc3(1)+...
23 Sc4(1)+Sc5(1)) = C5.';
24 imC(1: Sc6(2), Sc1(1)+Sc2(1)+Sc3(1)+Sc4(1)+Sc5(1)+1: Sc1
25 (1)+Sc2(1)+...
26 Sc3(1)+Sc4(1)+Sc5(1)+Sc6(1)) = C6.';
27 imC(1: Sc7(2), Sc1(1)+Sc2(1)+Sc3(1)+Sc4(1)+Sc5(1)+Sc6(1)
28 +1: Sc1(1)+...
29 Sc2(1)+ Sc3(1)+Sc4(1)+Sc5(1)+Sc6(1)+Sc7(1)) = C7.';
30
31
32 % Somente caracteres de interesse operados e prontos
33 para comparacao
34
35 % Obtencao das infos dos caracteres analisados e os
36 carac. fonte
37 Pi = 400;
38 info_imC = f_bb_analisa_regioes(imC, 1, Pi, 1);
39 info_P = f_bb_analisa_regioes(P1, 1, Pi, 1);
40 info_Px = f_bb_analisa_regioes(P2, 1, Pi, 1);

```

Fonte: Desenvolvido pelo Autor.

4.5 BANCO DE DADOS DE CARACTERES

Era necessário um banco de dados de todas as letras e números das placas do modelo Mercosul para comparar com as imagens de entrada para definir de qual caractere cada um se trata. Então em primeiro lugar a imagem contendo todos os caracteres possíveis foi subdividida em letras e números para aumentar a acurácia do algoritmo. Por isso a imagem **P** diz respeito às letras e **Px** aos números, e após as operações **P1** e **P2** respectivamente, conforme já foram abordadas ao final do Código 4.4, e são representados pela imagem Figura 23.

No entanto foi percebido que as letras e números dessa forma estavam 'polidos' demais e o algoritmo apresentava alta taxa de erro, visto que as imagens

Figura 23 – (a) Banco de letras. (b) Banco de números.



(a)

(b)

Fonte: Desenvolvido pelo Autor.

após sofrerem tantas operações apresentavam algumas distorções que ao menos seguiam um padrão. Então para todas as letras e números foram aplicadas operação de fechamento e um preenchimento já assemelhando boa parte com os caracteres das placas, diminuindo drasticamente a taxa de erro do algoritmo.

O reconhecimento como ainda apresentava algumas confusões, ou seja, o mesmo reconhecia bem os caracteres mas invertia muito alguns específicos, como "M" com "H", "R" com "U", "Q" com "O" entre outros que apresentam semelhança de curva de distância. Então, esses caracteres individuais sofreram alterações e foram reinseridos na imagem do banco de letras e números. Essas alterações consistiam em operações morfológicas de dilatação ou fechamento, como é possível ver no Código 4.5 e como resultado é possível visualizar as letras "M", "N" e "R" na Figura 24 (a) e o número "0" na Figura 24 (b).

Listing 4.5 – Alteração de caracteres no banco.

```

1 % Laco para Dilatacao Letra 'M' e reinsercao na
   imagem
2 Paux = imdilate(P1(10:60, 405:440), strel('square', 5));
3 Paux = imerode(Paux, strel('disk', 5));
4 Psize = zeros(size(P,1), size(P,2));
5
6 conty = 0;
7 contx = 0;
8 for i = 10:60
9     conty = conty+1;
10    for j = 405:440
11        contx = contx+1;

```

```

12      Psize(i,j) = Paux(conty, contx);
13  end
14  contx=0;
15 end
16 P1 = P1 | Psize;

```

Fonte: Desenvolvido pelo Autor.

Figura 24 – (a) Banco de letras alterado. (b) Banco de números alterado.



(a)



(b)

Fonte: Desenvolvido pelo Autor.

4.6 CURVAS DE DISTÂNCIA

Para a identificação dos caracteres foram utilizadas as curvas de distância de cada caractere da placa, juntamente com a curva de distância de cada caractere do banco de caracteres. A esse ponto percebeu-se que os caracteres da placa e os do banco possuíam maior semelhança, dado a tentativa e erro para ajustá-los e diminuir os erros do algoritmo até o ponto de total acerto para as imagens do banco de placas. Como já mencionado, foram separadas as letras e números para aumentar a confiabilidade do algoritmo e diminuir seu custo computacional, uma vez que as placas do Mercosul apresentam na 4^a, 6^a e 7^a (quarta, sexta e sétima) posição, números. Portanto um laço **for** percorre caractere por caractere da placa e caso esteja em alguma dessas três posições, o banco de números é usado para comparação, caso contrário o banco de letras é usado.

4.6.1 Comparação da Curva de Distância

Como já abordado no Laboratório 3, para efetuar a comparação por curva de distância, é preciso remover o *off-set* entre elas e efetuar o cálculo da energia

(PDS) para limitar os valores obtendo correlações que variam de 0 (zero) à 1 (um), onde o primeiro não há correlação alguma e o segundo quando são iguais naquele ponto. Por isso é necessário através da função `circshift` do MATLAB percorrer todos os pontos para qual a curva de distância foi calculada e como uma 'defasagem na fase' comparar todos esses pontos, obtendo o maior como correlação final dos elementos. Como já discutido acima, o algoritmo que efetua todas essas operações é dado pelo Código 4.6. Esse algoritmo também permite que diversas máximas correlações sejam armazenadas, para ao final extrair a maior e definir de fato qual caractere em questão está sendo processado, retornando na variável `reconC`, um vetor, os números ordenados dos componentes reconhecidos no banco de dados.

Listing 4.6 – Comparaçao de Curvas de Distancia.

```

1 % Comparacao das curvas de distancia do caracter com o
2 banco de dados
3
4 for i = 1:7
5
6     tempCD = info_imC{i}.curva_dist;
7     tempMean = tempCD - mean(tempCD);
8     tempEnergy = tempMean/sqrt(sum(tempMean.^2));
9
10    cont = 0;
11
12    if(i==4 || i==6 || i==7)
13        for j = 1:10
14            fontCD = info_Px{j}.curva_dist;
15            fontMean = fontCD - mean(fontCD);
16            fontEnergy = fontMean/sqrt(sum(fontMean.^2));
17
18            for k = 1:Pi
19                fontEnergy = circshift(fontEnergy, 1);
20                correl(k) = sum(tempEnergy.*fontEnergy);
21
22            end
23
24            if(max(correl) > 0.8)
25                cont = cont+1;
26                hst_corr(cont,:)= [max(correl), j];
27            end
28        end
29    end
30
31 else

```

```
25     for j = 1:26
26         fontCD = info_P{j}.curva_dist;
27         fontMean = fontCD - mean(fontCD);
28         fontEnergy = fontMean/sqrt(sum(fontMean.^2));
29
30         for k = 1:Pi
31             fontEnergy = circshift(fontEnergy, 1);
32             correl(k) = sum(tempEnergy.*fontEnergy);
33
34         end
35         if(max(correl) > 0.85)
36             cont = cont+1;
37             hst_corr(cont,:)= [max(correl), j];
38         end
39     end
40
41
42     hst_corr
43     top_value = 0;
44     hst_ant = 0;
45     for q = 1:size(hst_corr, 1)
46         if(hst_corr(q) > hst_ant)
47             top_value = hst_corr(q+size(hst_corr,1));
48             hst_ant = hst_corr(q);
49         end
50     end
51
52     reconC(i) = top_value;
53     hst_corr(:,:)= 0;
54     top_value(:,:)= 0;
55
56 end
```

Fonte: Desenvolvido pelo Autor.

4.7 DEFINIÇÃO FINAL DOS CARACTERES

Ao final do algoritmo, o mesmo possui um vetor com o número dos elementos reconhecidos no banco em ordem que foram testados nas placas. Dessa forma um **switch case** com 36 (trinta e seis) possibilidade é utilizado para designar a cada valor sua respectiva letra, criando ao final da sétima iteração a formação de um vetor de *string* com os respectivos caracteres reconhecidos. E por fim a placa reconhecida é apresentada na Janela de Comando do MATLAB.

No entanto a fim de uma representação gráfica do reconhecimento foi utilizado um modelo em branco da placa do Mercosul e sobre essa imagem foi ajustado para aparecer no local correto o resultado do vetor *string*, conforme mostra a Figura 25. Vale ressaltar que para as imagens do banco de imagens de placas, houve um acerto de 100% dos caracteres.

Figura 25 – Apresentação dos caracteres reconhecidos.



Fonte: Desenvolvido pelo Autor.

4.8 RESULTADOS

Abaixo foram inseridos todos os resultados das placas disponíveis e suas respectivas saídas. Desse modo foi possível perceber o acerto de 100% do algoritmo no reconhecimento das placas disponíveis no banco de imagens.

Figura 26 – (a) N1 placa 1. (b) Resultado N1 placa 1.



(a)

(b)

Fonte: Desenvolvido pelo Autor.

Figura 27 – (a) N1 placa 2. (b) Resultado N1 placa 2.



(a)

(b)

Fonte: Desenvolvido pelo Autor.

Figura 28 – (a) N1 placa 3. (b) Resultado N1 placa 3.



Fonte: Desenvolvido pelo Autor.

Figura 29 – (a) N1 placa 4. (b) Resultado N1 placa 4.



Fonte: Desenvolvido pelo Autor.

Figura 30 – (a) N2 placa 2. (b) Resultado N2 placa 2.



Fonte: Desenvolvido pelo Autor.

Figura 31 – (a) N2 placa 3. (b) Resultado N2 placa 3.



Fonte: Desenvolvido pelo Autor.

Figura 32 – (a) N3 placa 1. (b) Resultado N3 placa 1.



Fonte: Desenvolvido pelo Autor.

Figura 33 – (a) N3 placa 2. (b) Resultado N3 placa 2.



Fonte: Desenvolvido pelo Autor.

Figura 34 – (a) N3 placa 3. (b) Resultado N3 placa 3.



(a)

(b)

Fonte: Desenvolvido pelo Autor.

Figura 35 – (a) N3 placa 4. (b) Resultado N3 placa 4.



(a)

(b)

Fonte: Desenvolvido pelo Autor.

Figura 36 – (a) N3 placa 5. (b) Resultado N3 placa 5.



(a)

(b)

Fonte: Desenvolvido pelo Autor.

Figura 37 – (a) N3 placa 6. (b) Resultado N3 placa 6.



(a)

(b)

Fonte: Desenvolvido pelo Autor.

Figura 38 – (a) N3 placa 7. (b) Resultado N3 placa 7.



(a)

(b)

Fonte: Desenvolvido pelo Autor.

Figura 39 – (a) N3 placa 8. (b) Resultado N3 placa 8.



(a)

(b)

Fonte: Desenvolvido pelo Autor.

5 CONCLUSÃO

Destarte esse relatório teve como objetivo demonstrar parte da teoria tratada ao longo da disciplina, que tem relação com o algoritmo implementado para resolução do problema apresentado como o Trabalho 2 da disciplina de Visão Computacional em Robótica. O sistema de identificação de caracteres exigiu diversos métodos e ferramentas desenvolvidas em sala de aula para tornar factível a implementação de um algoritmo autônomo e eficaz. Além disso, possibilitou o teste de diversos métodos de resolução do mesmo problema juntamente com o uso de artefatos gráficos, como imagens processadas, possibilitando maior absorção do conteúdo ministrado em sala de aula e uso prático do mesmo.

REFERÊNCIAS

CANNY., John F. **A Computational Approach to Edge Detection.** [S.l.]: IEEE, 1986.

OTSU, Nobuyuki. **A Threshold Selection Method from gray-Level Histograms.** [S.l.]: IEEE Transactions on Systems, Man, e Cybernetics, 1979.

PROCESSAMENTO DE SINAIS, USP Laboratório de. **Transformada de Hough para detectar retas.** -. [S.l.], 2023. Disponível em:
<http://www.lps.usp.br/hae/apostila/hough.pdf>.

*