Leitura de Códigos de Barras usando o MatLab

Sandro Augusto Costa Magalhães, up201304932@fe.up.pt
Tiago José Ferreira Mendonça, up201305394@fe.up.pt

Resumo **—** Pretende-se desenvolver um *script* que seja capaz de desencriptar códigos de barras, recorrendo ao software MatLab, nomeadamente, à *toolbox* de processamento de imagem.

Index Terms **—** *processamento de imagem, análise de imagem, códigos de barras, EAN 13, imagens digitais, descodificação de imagem.*

I. Introdução

Como projeto final da unidade curricular de Sistemas Baseados em Visão, pretende-se implementar um *script* em MatLab que seja capaz de analisar imagens reais de códigos de barras e descodificar o seu conteúdo.

Para cumprir este objetivo, procurou-se aplicar os diferentes conceitos estudados na unidade curricular.

Como se verificará no capítulo seguinte, existem vários tipos de códigos de barras. No entanto, neste caso, apenas se propõe descodificar códigos da norma EAN13, que são os códigos tradicionalmente utilizados nos produtos comerciais.

II. Fundamentos Teóricos

Os códigos de barras são sistemas binários de encriptação de informação, nomeadamente números.

Existem várias normas de codificação desta informação [1], como a norma EAN13 que se vai estudar.

Esta norma [2] codifica 13 algarismos em 95 barras pretas (número binário 1) e brancas (número binário 0).

Figura 1 Código de Barras da norma EAN13

Os códigos iniciam com um código verificador de início de código do tipo 101.

De seguida sucedem-se seis dígitos com sete barras cada um que têm uma codificação par e impar que varia de acordo com o dígito verificador (todas as formas de codificar os dígitos são diferentes, isto é, embora possa seguir uma codificação par ou impar nenhum se repete).

No final destes seis algarismos, surge mais um código verificador do código de barra composto por 01010.

Os seguintes seis algarismos seguem todos um mesmo padrão de codificação que corresponde ao complementar da codificação par dos primeiros seis algarismos.

TABELA I
Tabela de descodificação do código de barras

Dígito	Codificação de Dígitos			Codificação em relação ao 1º dígito
	Esquerda		Direita	
	Ímpar	Par	Par	
0	0001101	0100111	1110010	OOOOOO EEEEEE
1	0011001	0110011	1100110	OOEOEE EEEEEE
2	0010011	0011011	1101100	OOEEOE EEEEEE
3	0111101	0100001	1000010	OOEEEO EEEEEE
4	0100011	0011101	1011100	OEOOEE EEEEEE
5	0110001	0111001	1001110	OEEOOE EEEEEE
6	0101111	0000101	1010000	OEEEOO EEEEEE
7	0111011	0010001	1000100	OEOEOE EEEEEE
8	0111011	0010001	1001000	OEOEEO EEEEEE
9	0001011	0010111	1110100	OEEOEO EEEEEE

O primeiro dígito, também denominado de *check digit*, pode ser calculado pela soma dos algarismos de índice impar (do terceiro até ao último) com a soma de três vezes o produto da soma dos números índice par do segundo até ao último, equação (1), o que corresponde a multiplicar por uma matriz de 3 e 1 e por fim somar todos os elementos. O *check digit*

corresponde à diferença do múltiplo de 10 seguinte a esta soma com ela própria.

$$\sum_{i=2}^{13} v_i \ (com\ i\ ímpar) + 3 \cdot \sum_{i=2}^{13} v_i \ (com\ i\ par) \qquad (1)$$

III. METODOLOGIA

A. Pré-processamento

No sentido de se cumprir o objetivo inicialmente proposto, é necessário, numa primeira fase, proceder a operações de pré-processamento, de forma a melhorar a qualidade da imagem e permitir a deteção e posterior extração das barras pretendidas.

Assim, após a leitura de uma imagem multicromática, realiza-se a sua conversão para a escala de cinza, utilizando a função do MatLab *rgb2gray*.

No sentido de melhorar a qualidade da imagem, aplica-se previamente um filtro gaussiano, com um desvio padrão de 0,5, eliminando as altas frequências e minimizando a influência de ruído no processamento. Para melhorar o contraste da imagem, procede-se à equalização de histograma, através do método CLAHE, que possui a vantagem de obter um melhor contraste da imagem, mas limitando o mesmo de forma a não amplificar o ruído nas regiões homogéneas [3].

B. Rotação da Imagem

Uma vez melhorada a imagem inicialmente captada, coloca-se a possibilidade de esta se encontrar com um ângulo de inclinação não nulo em relação à horizontal, sendo imperativo a aplicação de uma rotação prévia à sua segmentação.

Primeiramente, detetam-se todas as fronteiras entre as regiões existentes. Por teste, conclui-se que o filtro de *Sobel* constitui a abordagem que produz melhores resultados para este caso. Ao nível do código de barras, o resultado gerado compreende todos os pontos fronteira entre as barras constituintes, que unidos entre si constituem um segmento de reta. Prossegue-se com a deteção da inclinação desses segmentos de retas. Após a exploração dos diferentes algoritmos adequados para o fenómeno pretendido, retira-se que a transformada de *Hough* representa aquele em que a taxa de sucesso, embora não sendo de 100%, é a mais elevada [4].

Tendo por base as orlas obtida da imagem, por filtro de *Sobel*, convertem-se todos os pontos para o plano de *Hough*. De seguida, determinam-se os pontos mais intensos e traçam-se retas que corresponderão a algumas orlas dos objetos, nomeadamente, das barras da imagem. Uma vez que não se consegue precisar com elevada exatidão as orlas que serão detetadas, assumiu-se, e com um bom resultado, que a aquisição de 500 pontos seria suficiente.

```
 [H, T, R] = hough (imgEdge);
 P = houghpeaks (H, 500,'Threshold',
0.4*max(H(:)));
 lines = houghlines (imgEdge, T, R, P);
```

Calcula-se a reta de comprimento maior, que deverá corresponder a uma das arestas da caixa que delimita exteriormente o código de barras e reorienta-se a imagem segundo esse segmento, rodando-a de acordo com o valor de θ dado pela transformada de Hough.

C. Segmentação

Estando a imagem reorientada, avança-se para a sua segmentação. Para aumentar a eficácia desta fase, aplica-se o operador morfológico *top-hat*, eliminando elementos da região que envolve o código que poderiam dificultar a sua deteção.

Tendo em conta a representação gráfica de um código de barras, barras pretas sobre um fundo branco, é pertinente proceder, em primeiro lugar, à binarização da imagem. Com recurso ao método de *Otsu*, através da função *im2bw*, determina-se uma intensidade limiar adequada para comparar com a intensidade dos *pixels* na escala de cinza. Este algoritmo de limiarização fornece o valor ideal de um *threshold* que separe os elementos do fundo e da frente da imagem em dois *clusters*, atribuindo a cor branca ou preta para cada um deles.

Em segundo lugar, aplicam-se projeções horizontais e verticais. Procede-se à normalização dos resultados obtidos, dividindo-se todos os valores pela soma máxima da intensidade dos pixels na respetiva direção. Assim, estes ficam concentrados entre 0 e 1. Estando as cores da imagem invertidas face à configuração usual (barras pretas em fundo branco) e sendo a intensidade do preto indicado por 0 e o branco por 1, sabe-se que, quando o somatório numa determinada direção for não nulo, se intersetou uma região de cor branca que, hipoteticamente, poderá constituir o fundo do código. No sentido de eliminar falsos positivos e dada a normalização implementada, define-se um limiar de 0,5 para assinalar a deteção da fronteira do fundo do código de barras com a região externa. Realiza-se esta operação nas duas

direções e de forma detetar o limite inicial e final em cada uma. Obtidas as coordenadas dos quatro vértices pretendidos, efetua-se o corte da imagem.

D. Descodificação

Uma vez identificada a caixa exterior que delimita o código, avança-se para a sua descodificação. O algoritmo consiste, em primeiro lugar, em identificar a primeira e a última barra. Devido à diferença de intensidade dos *pixels* entre as barras e o fundo, realiza-se um processo iterativo ao longo da linha central da imagem de forma a determinar os referidos limites.

```
[x,y] = size (imgMorph);
midx = round (x/2);

 i = 1;
 yinit = i+1;
 while (imgMorph (midx, i) == 0)
    i=i+1;
    yinit = i;
 end

 i = y;
 yend = i-1;
 while (imgMorph (midx, i) == 0)
     i=i-1;
     yend = i;
 end
```

A codificação das barras implica 95 bits. Logo, o objetivo seguinte consiste em associar um *pixel* a um bit. Assim, calcula-se o número de *pixels* por bit, tendo em conta o tamanho de cada linha da imagem. Para cada linha, verificam-se todos os *pixels* para um determinado bit e atribui-se a este a moda desses valores, 0 ou 1. Itera-se a ação até perfazer os 95 bits. Por último, sabendo que cada número é identificado por 7 bits, recolhem-se, consecutivamente, 7 *pixels* de cada vez e comparam-se com a tabela de codificações (TABELA I), calculando o respetivo *check digit*. Efetua-se este procedimento para todas as linhas da imagem. Por comparação entre o *check digit* e a forma como cada algarismo estava codificado, tendo em conta a última coluna da TABELA TABELA I, verifica-se qual a primeira relação válida e imprime-se a mesma.

IV. Discussão de Resultados

Após a sua implementação, procedeu-se ao teste da robustez do algoritmo, recorrendo a várias imagens, retiradas de diferentes cenários e com caraterísticas geométricas distintas. A título de exemplo, indica-se um desses casos.

Figura 2 Imagem original do código de barras

A Figura 2 representa um dos casos de sucesso na descodificação de códigos de barras. Após a respetiva reorientação geométrica referida em III.B, concluísse que se obtém a Figura 3 (a), que, após a respetiva binarização por método de *Otsu* e um *top-hat*, obtém-se a Figura 3 (b).

Figura 3 (a) Imagem à escala de cinza reorientada
(b) Imagem binária para a descodificação

Tal como era esperado, neste momento o MatLab retorna como resultado de descodificação o código 5601313061557.

Por inspeção dos resultados, concluiu-se que na maioria dos casos analisados consegue-se descodificar com sucesso o código de barras presente. As operações de pré-processamento revelaram-se eficientes, sobressaindo apenas algumas vulnerabilidades em situações em que o contexto envolvente é dotado de formas geométricas variáveis, de área considerável e de cores distintas ou quando a distância ao objeto de deteção era assinalável. Nestas situações, a descodificação não era total, havendo, no entanto, uma correspondência parcial do código.

Outro fator limitante está relacionado com obtenção do ângulo de inclinação da imagem, constatando-se que o método utilizado, transformada de Hough, apenas permite assegurar a determinação desse valor, com fiabilidade, para amplitudes situadas entre os -90^o e $+90^o$.

Inicialmente, na lógica de descodificação, implementou-se a extração do código apenas ao longo da linha média da imagem. Contudo, após os primeiros testes, rapidamente se concluiu que esta linha nem sempre apresentava um código válido. Posto isto, avançou-se para a descodificação de todas as linhas da matriz, tal como indicado na metodologia, efetuando-se uma varredura completa a todo o código e imprimindo apenas o resultado que, à partida, será válido, culminando numa maior taxa de sucesso.

V. Conclusão

O objetivo do trabalho prendia-se com a leitura de um código de barras com recurso aos fundamentos teóricos subjacentes às técnicas de processamento de imagem e fazendo uso das ferramentas disponibilizadas *pela Image Processing Toolbox* do MatLab.

O processamento da imagem contemplou várias fases. Realizaram-se operações de pré-processamento no sentido de a suavizar, procedeu-se à sua reorientação, fez-se uso de técnicas de segmentação, isolando as regiões mais relevantes, e, por fim, descodificaram-se as barras extraídas.

Finalmente, os resultados obtidos permitiram validar o algoritmo implementado, constatando-se que a descodificação era total para grande parte das imagens utilizadas com ângulo de inclinação situado entre -90^o e $+90^o$.

VI. Referências

[1] Wikipedia, “Barcode,” 2 Dezembro 2016. [Online]. Available: https://en.wikipedia.org/wiki/Barcode.

[2] Wikipedia, “International Article Number,” Wikipedia - The free encyclopedia, 1 Dezembro 2016. [Online]. Available: https://en.wikipedia.org/wiki/International_Article_Number.

[3] A. M. Mendonça, Slides Aulas Teóricas, 2016.

[4] R. C. Gonzalez, Digita Image Processing, 2008.

[5] GS1, “How to calculate a check digit manually,” GS1, [Online]. Available: http://www.gs1.org/how-calculate-check-digit-manually. [Acedido em Novembro 2016].

[6] R. C. Gonzalez, Digital image processing using Matlab., 2008.