Sistemas de Visão Industrial: Biblioteca Genérica de Pré Processamento de Imagem em FPGA

Diogo Ferreira*, Filipe Moutinho*† e Pedro Deusdado‡
* NOVA School of Science and Technology, Monte de Caparica, Portugal
† UNINOVA, Center of Technology and Systems, Monte de Caparica, Portugal
‡ Introsys, Integração de Sistemas Robóticos, S.A, Quinta do Anjo, Portugal
Email: *dmr.ferreira@campus.fct.unl.pt, †fcm@fct.unl.pt, ‡pedro.deusdado@introsys.eu

***Resumo*—A necessidade de obter produtos de melhor qualidade e de forma mais rápida, provocou um aumento da complexidade dos sistemas de visão industrial, implicando a integração de mais *hardware* para aceleração e melhoria de eficiência. Neste artigo, é apresentado o trabalho em desenvolvimento que consiste na implementação de uma biblioteca genérica de métodos para execução em FPGA (*Field-Programmable Gate Array*), com o objetivo de diminuir o tempo de processamento de imagem em sistemas de visão. A sua implementação é baseada no projeto *Zybo Z7 Pcam 5C Demo*, ao qual, são acrescentados filtros implementados em VHDL (*VHSIC Hardware Description Language*).**

I. Introdução

Os avanços tecnológicos sentidos no setor industrial e a necessidade da sociedade atual possuir produtos de elevada qualidade, resultaram no aumento da utilização de sistemas de visão. Estes são caracterizados por uma sequência de operações, nomeadamente: captação de imagem, extração de características e tomada de decisão.

Na atualidade, existem sistemas de visão que contêm FPGA na sua constituição. O eventual menor consumo de energia e as capacidades de processamento de imagem em paralelo, são alguns dos argumentos a favor da utilização deste tipo de *hardware*, dedicado ao setor industrial.

Neste trabalho, pretende-se aumentar a eficiência de sistemas de visão através da diminuição do tempo de processamento de imagem. Tendo em conta este problema, irá ser desenvolvida uma biblioteca genérica de métodos em tempo real, para execução em FPGA.

A validação dos métodos desenvolvidos, será realizada através da integração da plataforma nos sistemas de visão da empresa Introsys. Serão comparados os tempos de processamento entre a solução atual (CPU - *Central Processing Unit*) e a solução proposta (CPU e FPGA). Como se trata de um projeto em desenvolvimento, a validação ainda tem como base a observação do vídeo resultante num monitor, depois de filtrado.

A biblioteca a desenvolver, em VHDL, terá na sua composição os seguintes métodos: conversão de RGB (*Red Green Blue*) para tons de cinzento, conversão de RGB para YCbCr, modificação de brilho, negativo, binarização, deteção de contornos, dilatação e erosão.

O documento tem a seguinte estrutura: na secção II, são apresentados alguns trabalhos relacionados no âmbito do processamento de imagem, de seguida, na secção III e IV são expostos os métodos desenvolvidos e os resultados obtidos. Por último, são apresentadas as conclusões (secção V).

II. Trabalhos Relacionados

Com a evolução tecnológica, os sistemas reconfiguráveis evoluíram dando origem aos *System-on-Chip* (SoC), caracterizados por conterem duas plataformas integradas (FPGA e CPU), que concedem ao projetista a hipótese de escolher em qual irá ser executado o código. Em [1], foi realizado um estudo acerca da sua utilização na identificação de linhas em estradas. Os algoritmos de pré processamento foram executados em FPGA (deteção de contornos, binarização e transformada de *Hough*), enquanto que, os métodos de extração de características foram executados em CPU. Em relação aos resultados obtidos, a solução desenvolvida apresentou melhor desempenho quando comparada a outros algoritmos de deteção de linhas [2], [3].

O processamento de imagem em FPGA pode ser realizado por vários métodos: *High-Level Synthesis* (HLS), *Open Computing Language* (OpenCL) ou linguagens de descrição de *hardware* (VHDL e Verilog).

Relativamente ao HLS, em [4], foi desenvolvido um algoritmo de deteção de faces em linguagem C++. Os resultados apresentados pelos autores, permitem concluir que a deteção é possível através deste método. Outro exemplo, são os algoritmos desenvolvidos através da ferramenta *Vivado HLS* (sobel e deteção de linhas) [5], onde foi concluído que o tempo de implementação é menor mas, a quantidade de recursos utilizados pela FPGA é elevada.

Em [6], foi estudada a utilização de OpenCL no processamento de imagem através de FPGA. Foram desenvolvidos três métodos: sobel, *canny* e extração de características. De seguida, foi efetuada uma comparação entre a implementação em VHDL e OpenCL, na qual, a segunda se destaca pela maior rapidez no desenvolvimento dos algoritmos, porém, através de VHDL, foi possível obter uma solução eficiente relativamente à quantidade de recursos utilizados pela FPGA.

Considerando as linguagens de descrição de *hardware*, em [7], foram implementados quatro filtros em Verilog (modificação de contraste, binarização, negativo e modificação de brilho). Os resultados obtidos demonstram que é possível desenvolver algoritmos de processamento de imagem através desta linguagem. Tendo em conta o VHDL, em [8], foi implementado um método de deteção de contornos (sobel), posteriormente executado em FPGA com sucesso. Em [9],

foram desenvolvidos filtros de erosão, dilatação, remoção de ruído e deteção de gradiente. Os resultados mostram que a utilização de VHDL aumenta o tempo de implementação do sistema mas, proporciona um maior controlo no desenvolvimento do circuito, assim como, maior flexibilidade nas alterações do mesmo.

Em suma, as linguagens de descrição de *hardware*, apesar de apresentarem um nível de complexidade superior em relação a métodos que utilizam linguagens de nível mais elevado, providenciam resultados mais eficazes em termos de velocidade e quantidade de recursos utilizados pela FPGA. Tendo em conta que o objetivo deste trabalho é a implementação de filtros de pré processamento, que apresentam uma complexidade de desenvolvimento inferior relativamente a métodos de extração de características, a linguagem escolhida para a implementação foi o VHDL.

III. Métodos Desenvolvidos

A implementação da biblioteca teve por base o *Zybo Z7 Pcam 5C Demo* [10]. Este recebe vídeo, sinal de entrada, captado por uma câmara *Pcam 5C*, conectada a uma plataforma *Zybo Z7-20*. Este sinal é convertido para RGB, e de seguida, para HDMI (*High-Definition Multimedia Interface*), possibilitando a sua visualização (sinal de saída).

Os módulos desenvolvidos em VHDL, referentes aos métodos de processamento de imagem, foram integrados no projeto indicado de modo a ser possível modificar cada *frame* do vídeo convertido, como se pode visualizar na figura 1. Cada módulo desenvolvido, recebe o valor das componentes RGB de cada pixel e apresenta na sua saída os valores filtrados correspondentes. Estes apresentam na sua estrutura uma entrada e uma saída do tipo *AXI4-Stream*, com intuito de fazer a interface com os outros blocos do circuito.

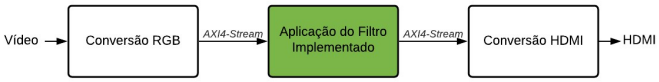

Fig. 1. Representação genérica do circuito utilizado no desenvolvimento da biblioteca com identificação do bloco de filtros implementados (Verde)

A. Conversão RGB/Cinza

Com o intuito de converter cada *frame* do vídeo captado para tons de cinza, foi calculada a média das três componentes (RGB) de cada pixel. De notar que, ao ser utilizada a linguagem VHDL só é possível dividir valores por múltiplos de 2. Tendo em conta essa especificidade, na equação 1 é apresentada a fórmula utilizada para realizar uma divisão por 3.

$$Cinzento = \frac{(R+G+B)*171}{512} \tag{1}$$

B. Conversão RGB/YCbCr

O módulo em questão tem como objetivo a conversão de RGB para YCbCr através das equações 2, 3 e 4.

$$Y = 0.257R + 0.504G + 0.098B + 16 \tag{2}$$

$$Cb = -0.148R - 0.291G + 0.439B + 128 \tag{3}$$

$$Cr = 0.439R - 0.368G - 0.071B + 128 \tag{4}$$

Devido às dificuldades referidas antes, foi feita uma aproximação das equações anteriores para ser possível o cálculo das componentes Y, Cb e Cr (equações 5, 6, 7).

$$Y' = \frac{4R + 8G + 2B + 256}{16} \tag{5}$$

$$Cb' = \frac{-2R - 5G + 7B + 2048}{16} \tag{6}$$

$$Cr' = \frac{7R - 6G - B + 2048}{16} \tag{7}$$

C. Modificação de Brilho

A implementação do filtro de modificação de brilho foi realizada através da adição de um valor de referência predefinido (positivo ou negativo), a todas as componentes (RGB) de cada pixel do vídeo original (equação 8). Caso o valor de referência seja positivo, o resultado filtrado terá maior luminosidade, caso contrário, a imagem resultante apresentará menos brilho. De notar que, foi feita uma condição com o intuito de garantir que o resultado da soma esteja compreendido entre 0 e 255, visto que, cada uma das componentes possui 8 *bits* de resolução.

$$Brilho(R,G,B)' = Componente(R,G,B) + Ref \tag{8}$$

D. Negativo

O método negativo tem como objetivo alterar o valor de cada pixel para o seu complemento, ou seja, os pixeis escuros da imagem original irão apresentar cores claras na imagem resultante, e vice versa. A sua implementação foi realizada através da modificação do valor das componentes (RGB) de cada pixel do vídeo captado. O efeito pretendido foi conseguido pela implementação da equação 9.

$$Negativo(R,G,B) = 255 - Componente(R,G,B) \tag{9}$$

E. Binarização

O algoritmo de binarização desenvolvido tem como objetivo modificar o valor de um pixel em tons de cinza para preto ou branco. A sua implementação depende de uma referência predefinida, à qual, é feita uma comparação com o pixel em questão. Caso a referência seja maior, o pixel resultante apresentará cor preta (zero), caso contrário, apresentará cor branca (255).

Ao contrário dos filtros anteriores, o módulo de binarização recebe como entrada o tom de cinza de um pixel e apresenta como saída, o seu valor correspondente depois de filtrado (preto ou branco).

F. Deteção de Contornos (Sobel)

O objetivo do método sobel é a deteção e realce de contornos. A sua implementação apresenta complexidade superior comparativamente aos algoritmos anteriores pois, o valor do pixel resultante depende do pixel original correspondente, da sua vizinhança e de uma máscara, contrariamente aos anteriores, onde depende apenas do pixel correspondente na *frame* original. Neste trabalho foi escolhida uma máscara 3x3 para implementar o método em questão. Tendo em conta a elevada resolução da imagem captada pela câmara (1920x1080 pixeis), optou-se por não processar as margens de modo a não dificultar a implementação do algoritmo e aumentar a sua velocidade.

O desenvolvimento do filtro sobel é realizado através do processamento do valor presente na posição central da máscara de convolução e dos seus vizinhos, logo, é necessário ter o conhecimento da sua localização na *frame* em questão. Neste projeto, o processamento é realizado pixel a pixel e cada *frame* é gerada da esquerda para a direita e de cima para baixo, o que não permite a aplicação imediata do filtro após receber o pixel central pois, o seu processamento depende de vizinhos que ainda não foram recebidos. Na figura 2, é visível que não é possível realizar o processamento do pixel com o valor 35 pois, a sua vizinhança não está completa.

1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	**24**	**25**	**26**	27	28	29	30
31	32	33	**34**	**35**	**36**	37	38	39	40
41	42	43	**44**	**45**	**46**	47	48	49	50
51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70

Fig. 2. Representação gráfica de uma *frame* do vídeo e da região utilizada no processamento de um pixel (a bege estão representados os pixeis recebidos e a azul os não recebidos)

O problema apresentado, motivou a alteração do procedimento anterior de modo a ser possível ter acesso a todos os pixeis contidos na vizinhança, e por conseguinte, possibilitar o processamento da posição central. Quando são conhecidas todas as posições da vizinhança do pixel, este é processado e colocado na imagem resultante com um desfasamento de uma linha e uma coluna em relação à posição original. Tendo isto em conta, a imagem resultante perderá as margens e será apresentada desfasada. Na figura 3, é possível observar a imagem original do lado esquerdo e a imagem resultante do lado direito.

1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	**24**	**25**	**26**	27	28	29	30
31	32	33	**34**	**35**	**36**	37	38	39	40
41	42	43	**44**	**45**	**46**	47	48	49	50
51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70

		12'	13'	14'	15'	16'	17'	18'	19'
		22'	23'	24'	25'	26'	27'	28'	29'
		32'	33'	34'	35'	36'	37'	38'	39'
		42'	43'	44'	45'	46'	47'	48'	49'
		52'	53'	54'	55'	56'	57'	58'	59'

Fig. 3. Representação gráfica da *frame* original (à esquerda) e da *frame* resultante (à direita).

De modo a poder ser utilizada a máscara de convolução, foram aplicadas as equações do método sobel (10, 11, 12) aos valores presentes na mesma. De notar que, o algoritmo desenvolvido deteta contornos através de duas componentes (horizontal e vertical). Na figura 4, estão representadas as posições da máscara utilizadas nas equações do método, onde "a" representa a intensidade do pixel presente no canto superior esquerdo.

a	b	c
d	e	f
g	h	i

Fig. 4. Representação gráfica das posições da máscara de convolução utilizadas nas equações do método sobel ("e" representa o pixel a processar e "i" a posição que o projetista tem acesso durante o processamento)

$$Sx(R,G,B) = (a+d+g) - (c+f+i) \tag{10}$$

$$Sy(R,G,B) = (a+b+c) - (g+h+i) \tag{11}$$

$$S(R,G,B) = |Sx| + |Sy| \tag{12}$$

G. Remoção de Ruído (Média)

Por último, para permitir a redução de ruído, foi implementado o filtro de média uniforme. Neste método, cada pixel foi substituído pela média do seu valor e dos seus vizinhos, tendo sido utilizada uma máscara unitária 3x3. Considerando a especificidade descrita antes, foi utilizada a equação 13, na qual, M(i,j) representa o valor contido em cada posição da máscara de convolução.

$$Média(R,G,B) = \frac{\sum M(i,j) * 57}{512} \tag{13}$$

IV. Validação e Testes

Na fase corrente do projeto, o procedimento usado para validar os algoritmos implementados é a visualização do vídeo resultante, depois de filtrado, num monitor através de HDMI (resolução: 1920x1080 pixeis e *frame rate*: 30 fps). Os resultados referentes aos métodos descritos no capítulo III são apresentados de seguida (figuras 5, 6, 7, 8, 9, 10, 11 e 12):

Fig. 5. Imagem de teste original

V. Conclusão

Este projeto tem como principal objetivo o desenvolvimento de uma biblioteca de pré processamento de imagem, destinada a reduzir o tempo de execução relativamente à solução mais convencional (CPU). Em relação aos resultados obtidos, apesar de apresentarem similaridades com o esperado, não permitem a confirmação de veracidade pois, o processo de

Fig. 6. Imagem resultante da aplicação do filtro de conversão RGB para tons de cinza

Fig. 7. Imagem resultante da aplicação do filtro de conversão para YCbCr

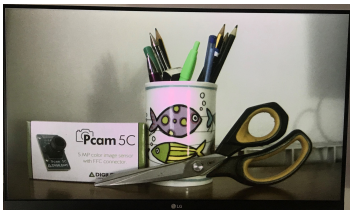

Fig. 8. Imagem resultante da aplicação do filtro de modificação de brilho (referência = -40)

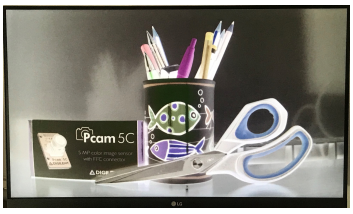

Fig. 9. Imagem resultante da aplicação do filtro negativo

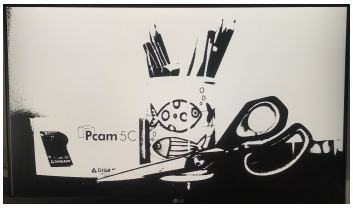

Fig. 10. Imagem resultante da aplicação do filtro de binarização (referência = 127)

Fig. 11. Imagem resultante da aplicação do filtro de deteção de contornos e binarização (referência = 127)

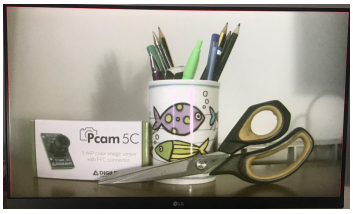

Fig. 12. Imagem resultante da aplicação do filtro de remoção de ruído

validação teve como base a visualização do vídeo resultante num monitor. Uma das observações a ter em conta, é o facto da implementação atual dos métodos de deteção de contornos e remoção de ruído, retirar informação das cinco primeiras e duas últimas colunas, assim como, das duas primeiras linhas de cada *frame* do vídeo resultante. Como trabalho futuro pretende-se concluir a implementação dos filtros propostos, bem como, iniciar a integração em aplicações reais e comparar tempos de execução em CPU e FPGA.

AGRADECIMENTOS

Este trabalho foi parcialmente financiado por Fundos Nacionais através da Fundação para a Ciência e a Tecnologia (FCT) no âmbito do projeto UIDB/00066/2020.

REFERÊNCIAS

[1] Y. Zhou. "Computer Vision System-On-Chip Designs for Intelligent Vehicles Computer Vision System-On-Chip Designs for Intelligent". Em: Worcester Polytechnic Institute April (2018).

[2] P.-Y. Hsiao, C.-W. Yeh, S.-S. Huang e L.-C. Fu. "A portable vision-based real-time lane departure warning system: day and night". Em: IEEE Transactions on Vehicular Technology 58.4 (2008), pp. 2089–2094.

[3] C. Lipski, B. Scholz, K. Berger, C. Linz, T. Stich e M. Magnor. "A fast and robust approach to lane marking detection and lane tracking". Em: 2008 IEEE Southwest Symposium on Image Analysis and Interpretation. IEEE. 2008, pp. 57–60.

[4] D. O'Loughlin, A. Coffey, F. Callaly, D. Lyons e F. Morgan. "Xilinx vivado high level synthesis: Case studies". Em: IET Conference Publications 2014.CP639 (2014), pp. 352–356. doi: 10.1049/cp.2014.0713.

[5] A. Cortes, I. Velez e A. Irizar. "High level synthesis using Vivado HLS for Zynq SoC: Image processing case studies". Em: 2016 Conference on Design of Circuits and Integrated Systems, DCIS 2016 - Proceedings (2017), pp. 183–188. doi: 10.1109/ DCIS.2016.7845376.

[6] K. Hill, S. Craciun, A. George e H. Lam. "Comparative analysis of OpenCL vs. HDL with image-processing kernels on Stratix-V FPGA". Em: Proceedings of the International Conference on Application-Specific Systems, Architectures and Processors 2015- Septe (2015), pp. 189–193. issn: 10636862. doi: 10.1109/ASAP.2015.7245733.

[7] N. Singla e M. S. Narula. "FPGA Implementation of Image Enhancement Using Verilog HDL". Em: International Research Journal of Engineering and Technology (IRJET) 05.05 (2018), pp. 1794–1794.

[8] D. J. Ayyed. "Image Steganography Based Sobel Edge Detection Using FPGA". Em: Technium 2.6 (2020), pp. 23–34.

[9] J Carmona-Suárez, I Santillan, S Martínez-Díaz, J. Gómez-Torres e J. Sandoval- Galarza. "Image Processing Implementation on FPGA Hardware with Mathematical Morphological Operators Centered in VHDL". Em: Memorias del Congreso Nacional de Control Automático ISSN: 2594-2492 (2018).

[10] Zybo Z7 Pcam 5C Demo. url: https://reference.digilentinc.com/learn/programmable-logic/tutorials/zybo-z7-pcam-5c-demo/start. (Acedido em: 05/05/2021)