

# Visão por Computador - Guião 2

Miguel Azevedo  
lobaoazevedo@ua.pt  
38569

Gabriel Vieira  
gabriel.vieira@ua.pt  
68021

## *Resumo –*

Este relatório descreve a resolução do segundo guião prático da disciplina de Visão por Computador.

## *Palavras chave –*

Visão por computador, processamento de imagem, OpenCV.

## I. INTRODUÇÃO

Este guião prático consiste no processamento de imagem a baixo nível, usando as bibliotecas do OpenCV. Tem como objectivo aplicar transformações a imagens resultantes da camera digital ou carregadas localmente e tirar conclusões sobre as respectivas transformações que são mencionadas no guião prático.

### A. Exercícios propostos:

- **Exercício 1:** Implementar um programa que tenha a capacidade de mostrar a escala de cinza e a versão a preto e branco das imagens capturadas pela camera digital. Para este exercício recomenda-se o uso de funções tais como *cvtColor* e *adaptiveThreshold*.
- **Exercício 2:** Pegando no exercício 1, fazer uso da função *threshold* para obter uma imagem binária e comentar os respectivos resultados.
- **Exercício 3:** Implementar um programa que efectuasse uma deteção de pele de uma imagem baseada na cromaticidade ou outra propriedade de cor. O exercício, em primeiro lugar, passa por calcular os valores de cromaticidade e a correspondente distribuição de cada pixel, considerando vários espaços de cor. Uma maneira de implementar isto passa por pintar todos os pixéis da pele de uma determinada cor, como por exemplo a preto e branco.
- **Exercício 4:** Implementar um programa que capturasse imagens da camera digital e explorar filtros para modificar a imagem adquirida, tais como blur, gaussianblur e medianblur, e por fim comentar os resultados obtidos.
- **Exercício 5:** Implementar um programa que através da captura de imagens da camera digital construa o histograma de cada um dos canais (R,G e B), assim como o histograma da versão

escala de cinzas da respectiva imagem.

- **Exercício 6:** Incluir no exercício anterior a capacidade de aplicar a normalização do histograma e comentar os resultados.
- **Exercício 7:** Implementar um programa que carregue duas imagens localmente e compare ambos os histogramas. Explorar os resultados obtidos e comentar o seu uso na aplicação de identificar imagens semelhantes para o olho humano. Melhorar o programa de maneira a obter as imagens mais semelhantes existentes num directório.

## II. RESOLUÇÃO DOS EXERCÍCIOS PROPOSTOS:

### Exercício 1

Como foi introduzido, este exercício tinha como objectivo ver a transformação da imagem original, captada pela camera digital, na escala de cinzas e a preto e branco interagindo com o teclado para as várias transformações correspondentes. Começou-se então por, em primeiro lugar, obter a imagem natural captada pela camera. Com a imagem obtida definiu-se que clicando na tecla 'g', esta era transformada na escala de cinzas, 'b' era transformada a preto e branco e 'n' voltava à imagem natural por defeito. Para a escala de cinzas, usou-se a função *cvtColor(Mat src, Mat dest, MACRO, int)* para passar a imagem para a escala de cinzas, usando a macro CV RGB2GRAY e guardando-a numa matriz para poder ser mostrada na execução do exercício.

Para a preto e branco, foi necessário primeiro transformar a imagem em tons de cinza, pois a gama de cores é mais adequada para o efeito, e aproveitando a transformação efectuada aplicou-se então a transformação sobre esta, usando a função *adaptiveThreshold*. Esta função fazia uso da também de vários tipos de threshold a aplicar, sendo um deles o *THRESH\_BINARY* que faz uma binarização da imagem, ou seja, coloca-a a preto e branco, faz uso também de um valor máximo para as cores preto e branco, sendo o valor máximo de 255 em que estas estão muito destacadas como se pode verificar na figura abaixo.

Abaixo encontram-se as figuras dos resultados verificados das respectivas transformações:

### Exercício 2



Fig. 1 - Imagem original



Fig. 2 - Imagem original



Fig. 3 - Imagem original

Neste exercício era pedido que se pegasse no exercício anterior e se adicionasse uma nova opção: imagem binária. Para fazer se obter a binarização da imagem, teria de se converter a imagem nas escala de cinzas, como foi feito no exercício anterior e depois disso usar a função `threshold` atribuindo um valor de binarização para a imagem, ou seja, se o valor for 0 a imagem fica a branco, se for 255 (valor máximo) ela fica a preto (neste caso foi usado o valor central 127), definir o valor máximo de binarização (255) e por fim aplicar a macro `THRESH_BINARY` para especificar o tipo de operação.

Abaixo encontra-se a imagem binarizada do exercício anterior:

Depois de obtidos os resultados, podemos concluir que a função `threshold` e `adaptiveThreshold` são essenciais para transformar uma imagem a preto e branco ou binarizá-la. Podemos também afirmar



Fig. 4 - Imagem original binarizada

mar que fazendo variar o valor do tamanho dos blocos, aumentando, na função `adaptiveThreshold` nos ajuda a obter uma imagem mais definida a preto e branco no que toca mais aos seus contornos. Em relação à função `threshold` podemos afirmar que a binarização realça a parte mais iluminada da imagem original e coloca o ambiente à volta todo a preto (como se pode verificar na figura 4). Depois se alterarmos o valor de binarização verificamos que quanto menor é mais branca se torna a imagem e quanto maior é mais preta esta se torna. **Exercício 3**

Como foi introduzido acima, o objectivo deste exercício era carregar uma imagem e efectuar a deteção de pele, baseada na cromaticidade ou outra propriedade de cor. Para isso, era necessário em primeiro lugar encontrar uma gama de cores que ajudasse na deteção do rosto e trabalhar sobre a imagem resultante da aplicação desta mesma gama. Na aula, o docente aconselhou a converter a gama de cores da imagem de RGB para HSV (hue, saturation e value). Portanto, usou-se a função `cvtColor` para converter a imagem da gama RGB para HSV, usando a macro COLOR\_RGB2HSV, e separou-se a mesma (*função split*) em 3 canais (um canal com cada uma das gamas, hue, saturation e value). O canal a usar foi o canal 0 (hue color), pois este tinha a gama de valores que distinguia melhor o rosto do resto da imagem. Este estava a cinza e o resto da imagem a preto, como é possível ver abaixo na figura 2.

Depois disso, era necessário observar a imagem e com a ajuda do rato ver os valores máximo e mínimo correspondentes dos pixéis que pertenciam à zona da pele. Com estes dados, depois teria-se que usar a função `inRange` para meter todos os pixéis fora do intervalo de valores (máximo e mínimo) obtidos a zero, ou seja, todos esses pixéis eram colocados a preto.

No final deste processo, era feita operação de merge, ou seja, esta operação iria juntar os 3 canais numa só imagem, que seria guardada numa nova matriz. Por fim, para obtermos o resultado desejado (salientar a pele) fazia-se a operação "bitwise and" da mesma com a imagem original. Com esta operação, só os bits que

estavam a "1" numa imagem e na outra é que se salientavam. Os restantes, como era uma operação de AND, seriam colocados a 0, pois  $0 \text{ AND } 1 = 0$ . Desta forma obtivemos o resultado esperado na figura 3, que era a deteção de pele baseada na cromaticidade.



Fig. 5 - Imagem original



Fig. 6 - Imagem com a cabeça toda salientada a cinza



Fig. 7 - Rosto detectado através de cromaticidade

Este exercício permitiu concluir que, no caso desta imagem, a deteção de rosto é mais fácil, pois apenas o cabelo é que tem semelhança com a cor da pele e o meio envolvente nada tem haver com este, e portanto facilmente se conseguia excluí-lo. Também foram efectuados testes do mesmo exercício com a imagem da "lena", e o que podemos concluir é que esta é bastante mais difícil de fazer uma deteção do rosto, pois o meio envolvente tem cores muito semelhantes a este.

#### Exercício 4

Conforme explicito acima, o objectivo deste exercício seria aplicar filtros à imagem adquirida da camera digital, tais como **blur**, **gaussianBlur** e **medianBlur**. Na função de **Blur(Mat src, mat out, ksize, anchor, border type)**, temos os seguintes argumentos:

- **src**: imagem captada, neste caso, imagem obtida da camera digital
- **out**: matriz para a qual onde a imagem será guardada, do mesmo tipo e tamanho da imagem de entrada
- **ksize**: tamanho do kernel de blurring
- **anchor**: ponto de âncora, ou ponto de ancoragem. o seu valor por defeito é *Point(-1,-1)*, o que significa que a âncora é no centro do kernel
- **borderType**: modo de border usado para extrapolar os pixéis para fora da imagem

Todos estes filtros são usados para reduzir o ruído da imagem. Abaixo, é possível observar a imagem original adquirida da camera digital, e abaixo desta, a imagem resultante da aplicação do filtro de Blur:

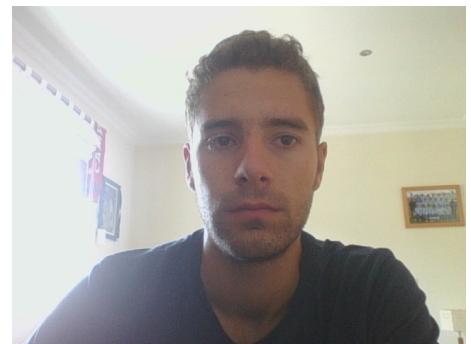


Fig. 8 - Imagem original

De seguida, foi também aplicado o filtro gaussianBlur, que serve para também reduzir o ruído da imagem, mas usa uma função Gaussiana para tal. Nesta função **GaussianBlur(Mat src, mat out, ksize, sigmaX, sigmaY, border type)**, temos os seguintes argumentos:

- **src**: imagem captada, neste caso, imagem obtida da camera digital
- **out**: matriz para a qual onde a imagem será guardada, do mesmo tipo e tamanho da imagem de

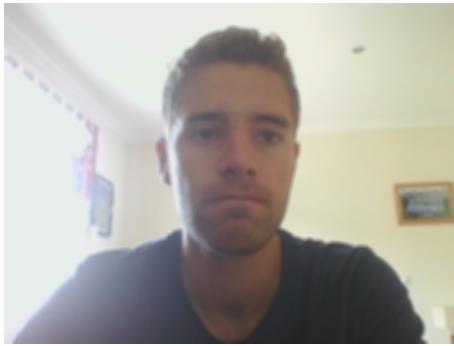


Fig. 9 - Imagem resultante do filtro Blur

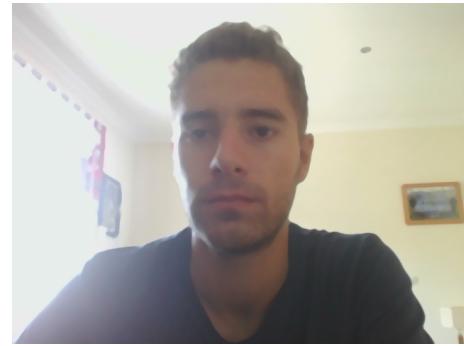


Fig. 11 - Imagem resultante do filtro Median Blur

entrada

- **ksize:** tamanho do kernel de blurring
- **sigmaX:** desvio padrão do kernel Gaussian na direção X.
- **sigmaY:** desvio padrão do kernel Gaussian na direção Y. Se sigmaY for zero, é ajustado para ser igual a sigmaX, se ambos os sigmas são zeros, que são calculados a partir ksize.width e ksize.height, respectivamente
- **borderType:** modo de border usado para extrapolar os pixéis para fora da imagem

Abaixo é mostrada a imagem resultante depois de aplicar o filtro gaussianBlur:

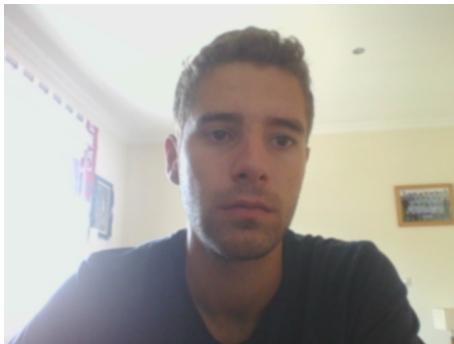


Fig. 10 - Imagem resultante do filtro Gaussian Blur

Por fim temos o filtro medianBlur, que apenas faz o uso do tamanho do kernel, que é de tamanho linear e dever ser ímpar com valores acima de 1 (foi usado o 7 na resolução). Na função **medianBlur(src, out, ksize)**, temos os seguintes argumentos:

- **src:** imagem captada, neste caso, imagem obtida da camera digital
- **out:** matriz para a qual onde a imagem será guardada, do mesmo tipo e tamanho da imagem de entrada
- **ksize:** tamanho do kernel de blurring. Como foi dito acima, neste caso é de tamanho linear e ímpar superior a 1

Abaixo é mostrada a imagem resultante depois de aplicar o filtro medianBlur:

### Exercício 5

Como foi dito na especificação dos exercícios, o objectivo é, através da imagem adquirida, construir o respectivo histograma RGB e também o histograma da escala de cinzas. Abaixo são mostrados os histogramas obtidos normalizados e não normalizados em RGB e na escala de cinzas:

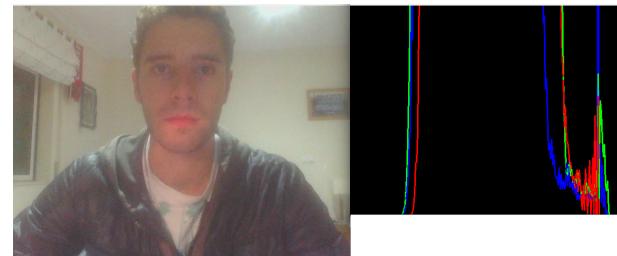


Fig. 12 - Histograma não normalizado RGB da imagem capturada



Fig. 13 - Histograma na escala de cinzas não normalizado da imagem capturada

Inicialmente começa-se por dividir a imagem em RGB, ou seja, um canal para cada cor, através do uso da função *split*. Depois são necessárias variáveis para definir o tamanho do histograma (neste caso 256 bins<sup>1</sup>), a gama dos histograma (neste caso como é RGB tem de ser 0 a 256), 2 variáveis booleanas: *acummulate*, que vai permitir que o histograma seja limpo no inicio da reprodução da imagem, e *uniform* porque queremos que os pixéis tenham o mesmo tamanho, e 3 matrizes onde iria cada histograma resultante em cada canal.

<sup>1</sup>nº de subdivisões de cada dimensão, ou seja, subdivisão de cada nº de parâmetros que queremos obter

De seguida, temos tudo o que necessitamos para calcular o histograma para cada canal, através da função *calcHist*. Com os histogramas calculados, podemos desenhar o histograma para R, G e B, definindo a altura, largura do histograma, bem como a largura de cada bin. De seguida, pode-se então desenhar o respetivo histograma em cada canal, usando a função *line* em cada canal, que vai ligar os diversos pontos calculados do histograma, usando também como um dos argumentos o construtor *Scalar*, que contém num triplô os valores para cada uma das componentes RGB. A primeira corresponde ao azul a segunda ao verde e a terceira ao vermelho. Ainda como últimos argumentos desta função *line* temos variantes que definem a espessura dos bins, tipo de linha e a posição do histograma na janela de amostras (*shift*). Depois de todo este processo, podemos observar o resultado nas figuras 8 e 9 (RGB e tons de cinza), em que a espessura = 2, tipo de linha = 8 e o shift = 0 (ou seja, histograma ocupa toda a janela).

Tudo o que foi efectuado para o histograma RGB também foi feito para o histograma em tons de cinza, sendo que neste existe apenas um canal em vez de 3 como no RGB. Também é possível verificar o histograma em tons de cinza não normalizado.

## Exercício 6

Neste exercício era pedido que pegando no histograma anterior, normalizá-lo para ser possível visualizá-lo por completo, conseguindo assim observar também o seu valor máximo. Para tal, antes do histograma ser desenhado é necessário primeiro normalizá-lo, usando a função *normalize*, para depois se poder proceder ao desenho sobre a sua normalização. Abaixo é possível verificar o histograma RGB e na escala de cinzas normalizado:

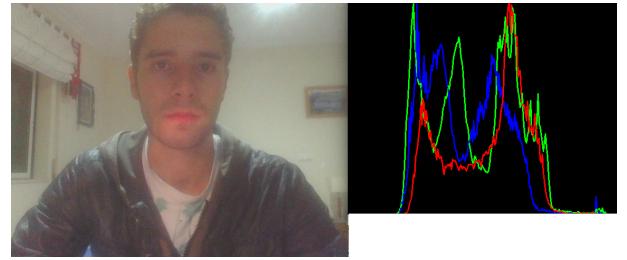


Fig. 14 - Histograma normalizado RGB da imagem capturada

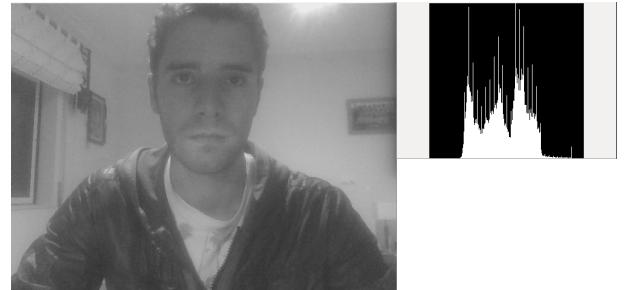


Fig. 15 - Histograma na escala de cinzas normalizado da imagem capturada

Com estes resultados, é possível verificar que não normalizando o histograma, não é possível vê-lo totalmente, sendo impossível ver o seu valor máximo. Se o normalizarmos isso já não acontece, uma vez que já é possível visualizá-lo por completo e verificar o seu valor máximo (principalmente).

### Exercício 7

A implementação deste exercício consistiu nas rotinas necessárias para a comparação de histogramas dos parâmetros de cromaticidade e saturação de uma lista de imagens usando quatro métricas diferentes oferecidas pelo OpenCV (Correlação, Qui-Quadrado, Interceção e Hellinger). Esta implementação foi feita com base no exemplo apresentado em 2. Esta implementação foi testada usando o seguinte conjunto de imagens:



Fig. 16 - Lena: Original, com HSV alterado, em escala de cinza, com contraste e brilho alterados, com *Blur Gaussiano* e imagem Underwater

Foram obtidos os seguintes resultados usando os três métodos supracitados pela mesma ordem:

```
Method [0]
lenna_hsv_changed.jpg : -0.043479
underwater.jpg : -0.024646
lenna_contrast_brightness.jpg : -0.007520
lenna_grayscale.jpg : -0.004011
lenna_blurred.jpg : 0.455038
lenna_original.jpg : 1.000000
Method [1]
lenna_original.jpg : 0.000000
lenna_grayscale.jpg : 41.439140
lenna_contrast_brightness.jpg : 41.548298
underwater.jpg : 41.908646
lenna_hsv_changed.jpg : 47.014584
lenna_blurred.jpg : 88.371613
Method [2]
lenna_grayscale.jpg : 0.000000
underwater.jpg : 0.000467
lenna_contrast_brightness.jpg : 0.022957
lenna_hsv_changed.jpg : 0.140449
lenna_blurred.jpg : 9.801786
lenna_original.jpg : 41.439140
Method [3]
lenna_original.jpg : 0.000000
lenna_blurred.jpg : 0.715321
lenna_contrast_brightness.jpg : 0.983631
lenna_hsv_changed.jpg : 0.991435
underwater.jpg : 0.999958
lenna_grayscale.jpg : 1.000000
```

Fig. 17 - Resultados da ordenação usando as quatro métricas de comparação: Correlação, Qui-Quadrado, Interceção e Hellinger

Embora existam diferenças entre as métricas de comparação pode-se reparar que em grande parte dos casos a imagem em escala de cinza é muitas vezes ordenada como uma das menos parecidas com a original, este é um exemplo de situação em que este algoritmo não tem uma boa precisão, comparativamente ao olho humano, pelo facto de os histogramas apenas conseguirem medir características de cor e não características morfológicas.