Visão por Computador
03 - Low-level image processing

Diogo Corte, Diogo Silva
DETI, Universidade de Aveiro
Aveiro, Portugal
{diogo.corte, dbtds}@ua.pt

Resumo –

Este relatório foi realizado para a unidade curricular de Visão por Computadores com o objetivo de demonstrar a perceção da aula (VC Exercises 03). Sendo estes, threshold, espaço de cores, filtros e histogramas. Todos os temas referidos foram testados utilizado a biblioteca OpenCV.

I. THRESHOLDING

Este exercício consistiu em experimentar diferentes tipo de threshold, sendo que foram testados dois tipos que são fornecidos com o OpenCV.

A. *Threshold binário*

O mais simples, threshold, que consiste em fazer o threshold a um pixel de forma binária, se o pixel se encontra abaixo de um dado valor, fica a 0, se se encontrar cima desse mesmo valor, fica com o valor máximo nesse pixel, ou seja, 255. Isto para uma imagem de apenas 1 canal.

Se por acaso este método for aplicado a uma imagem com vários canais, o que o threshold irá fazer é aplicar a cada canal de cada vez e a cada pixel. Sendo que supondo que um ponto RGB tem as seguintes intensidades, 150; 255; 50, ou seja, uma cor verde. Apesar de ser ver, ao aplicar um threshold a cada canal (considerando metade, 127) o vermelho passará a ser 255 por estar acima dos 127, o verde continua a ser 255 e o azul passará a ser 0, sendo que este pixel irá ficar com um cor amarela.

B. *Threshold adaptativo*

O threshold adaptativo é melhor que o normal porque tem em consideração os vizinhos, sendo que se o blockSize for de 5 (por exemplo), o kernel será um quadrado de 5x5 sendo que o ponto central é o ponto que está a ser analizado, depois consoante o método utilizado (mean ou gaussian) será aplicado um formula com base na intensidade dos vizinhos e do ponto actual. Sendo que no método de média é aplicado uma média sobre o kernel inteiro e depois funciona com o threshold referido anteriormente mas com o valor da média em vez da intensidade do pixel. É possível visualizar que no threshold adaptativo consegue-se ter uma percepção muito superior do cenário devido ao facto de ter em consideração os vizinhos.

C. *Descrição do programa desenvolvido*

O programa desenvolvido encontra-se na pasta a1_adaptiveThreshold sendo que as teclas de '+' e '-' servem para trocar o threshold que é aplicando. Sendo que a sequência pela qual aparecem é: sem threshold, conversão para escala de cinzento, threshold binário, threshold adaptativo com kernel de média e threshold adaptativo com kernel gaussiano.

Figura 1: Esquerda: Original, Centro: Threshold simples, Direita: Threshold adaptativo (kernel com block-size de 3) com um método gaussiano

Pode-se reparar que na imagem central perde-se a percepção de quantos livros existiam na estante por exemplo, no entanto é mais simples de dizer onde estão as zonas com sombra. Por outro lado, o adaptativo consegue-se dizer a quantidade de livros na estante.

II. DETECÇÃO DE COR DA PELE

Na abordagem a este problema optou-se por uma pesquisa prévia de artigos sobre técnicas de deteção de cor de pele, é um tema extensamente abordado e com variadas abordagens, com algoritmos nos diversos sistemas de cor, de entre outros RGB, YCrCb, HSV e conjugação de algoritmos.

A. *Descrição do programa desenvolvido*

O programa desenvolvido considera o método [Peer et al. 2003]:

(R,G,B) é cor de pele se:

```
R > 95 & G > 40 & B > 20 & max{R,G,B} -
    min{R,G,B} > 15 & |R-G| > 15 & R > G & R > B
```

Tem implementação muito simples de construir e fornece um classificador rápido com resultados razoáveis, passíveis de melhoramento quando conjugado com outros algoritmos. A nossa implementação mostra os pixels não classificados a preto.

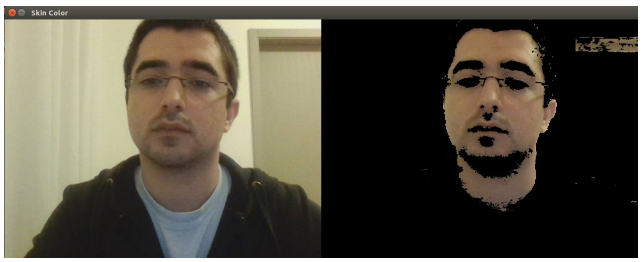

Figura 2: Esquerda: Imagem original, Direita: Imagem preto e cor de pele

III. FILTROS DO TIPO BLUR

A maior parte dos filtros de blur consistem num uso de kernel. Sendo que a ideia consiste em olhar para os pontos vizinhos ao qual estamos analisar. Para isso tem se ajuda de um kernel de tamanho K (sendo que será um quadrado K linhas por K colunas) em que cada elemento do kernel tem um dado peso que afecta o valor final da solução.

A função blur consiste em ter um kernel em que qualquer ponto do kernel contribuir de igual forma, ou seja, tem um peso de 1, fazendo a soma da intensidade de todos os pontos multiplicando pelo peso (neste caso é 1, logo é neutro) e dividindo o valor final pela totalidade de elementos do kernel, sendo esse o valor final para o pixel. Faz-se este processo pela imagem toda obtendo o efeito produzido pelo blur(). No medianBlur() o valor do meio é substituido pela mediana dos valores do kernel.

Figura 3: Efeitos de Bluring

Na imagem superior esquerda tem-se um efeito de blur em que os pesos do kernel são todos 1, acabando por se ver muito blur por toda a imagem, o que pode não ser o desejado por vezes. Se a ideia for remover um pequeno ruido pela imagem toda apenas, este filtro não vai resolver muito bem porque acaba perdendo qualidade na imagem (informação).

No filtro do canto inferior esquerdo tem-se o efeito de mediana em que se obtém-se um resultado melhor do que atribuir simplesmente uns ao kernel. No canto superior direito encontra-se o filtro gaussiano.

Para além desses ainda-se fez para o Bilateral Filter (canto inferior direito) que consegue retirar pequenos ruidos a imagem sem causar grande perda na qualidade da imagem, no entanto é um algoritmo bastante demorado.

Qualquer das imagens acima representadas encontram-se com um kernel de tamanho 11.

IV. HISTOGRAMAS E NORMALIZAÇÃO

Um histograma é a representação gráfica de uma distribuição de frequências. Numa imagem representa a quantidade de pixels ao longo das partições da imagem.

Para a representação de um histograma para cada componente do sistema de cor RGB a biblioteca OpenCV fornece a função split() que permite separar os 3 planos RBG e assim com o método calcHist(), em cada componente RGB e imagem em escala de cinzentos, obtivémos a concentração de pixels em cada bin da imagem. Foi benéfico para a sua representação gráfica aplicar a função de normalização normalize() pois permite modificar os valores obtidos de forma proporcional para um intervalo pré-definido, enquadrável na imagem de output.

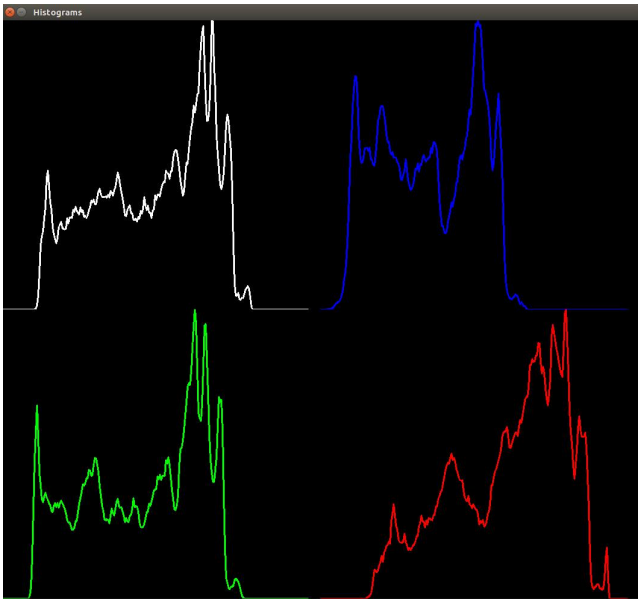

Figura 4: Histogramas normalizados, Esquerda Superior: escala de cinzentos, Direito Superior: plano RGB azul, Esquerda Inferior: plano RGB verde, Direito Inferior: plano RGB vermelho

V. SIMILARIDADE DE IMAGENS (COM HISTOGRAMAS)

Para resolver este problema optou-se por usar parte do código que já tinha sido desenvolvido anteriormente para calcular os histogramas de cada canal e da imagem em escala de cinza.

Para comparar os histogramas utilizou-se a função disponibilizada pelo OpenCV, compareHist, que basicamente permite aplicar diferentes métricas para calcular as distâncias entre os pontos dos histogramas, como por exemplo, Correlação, Chi-Square, entre outras. No entanto acabou por se optar pela correlação que ao passar dois histogramas devolve um valor entre 0 e 1, sendo que 0 significa que os histogramas não são nada similares e 1 são similares.

O programa desenvolvido considera um directório e lê todos os ficheiros desse direitório, vendo quais as imagens que são mais similares. O dataset utilizado para o teste do programa foi `http://www.imageprocessingplace.com/downloads_V3/root_downloads/image_databases/standard_test_images.zip`.

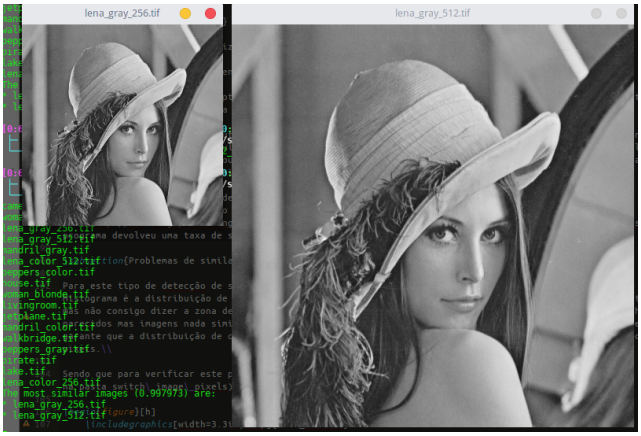

Figura 5: Imagens mais similares

Sendo que o programa devolveu uma taxa de similaridade de 99%, praticamente 100% para duas imagens que na verdade são iguais, mas uma é resize da outra. Ou seja, alguns pixeis tiveram de alterados/removidos de forma a comprimir a imagem num tamanho mais pequeno, mas a distribuição de cores manteve-se em 99% o que é praticamente igual, como seria de esperar.

A. *Problemas de similaridade com histograma*

Para este tipo de detecção de similaridade é preciso ter em conta um factor muito importante, o conceito de histograma. Histograma é a distribuição de cores de uma dada imagem, ou seja, eu consigo dizer se uma imagem tem muito vermelho ou não, mas não consigo dizer a zona desse vermelho com base só no histograma. Sendo que duas imagens podem ter histogramas muito parecidos mas imagens nada similares a nível de conteúdo, apenas a nível de cor. Similiaridade através de histogramas apenas garante que a distribuição de cores é a mesma, não a forma como essas cores são colocadas, ou seja, a localização dos pixeis.

Sendo que para verificar este problema foi desenvolvido um programa a parte que faz o shuffle dos pixeis de uma imagem (está na pasta switch_image_pixels).

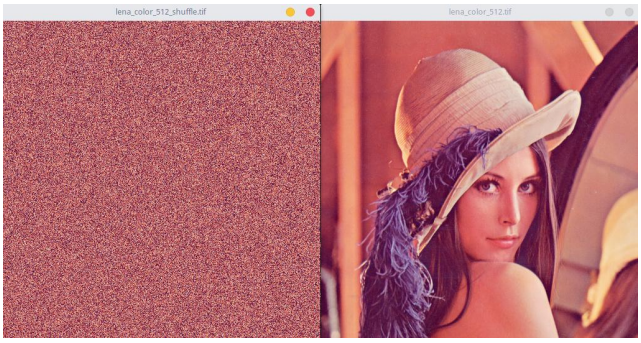

Figura 6: Esquerda: Imagem original com os pixeis Shuffled, Direita: Imagem original

Ao aplicar o compareHist aos histogramas de cada imagem devolve exactamente 1, ou seja, as imagens são 100% similares do ponto de vista do algoritmo, devido a terem exactamente a mesma distribuição de cores. No entanto, as posições dos pixeis não são as mesmas.

REFERÊNCIAS

[1] Gonzalez, Woods, Eddins, *Image Processing Place*

[2] *A Survey on Pixel-Based Skin Color Detection Techniques - by Vladimir Vezhnevets et al.*, published January 1, 2009