UNIVERSIDADE D
COIMBRA

Visão por Computador

2021/2022

Image Filtering and Hough Transform

Duarte Cruz
uc2017264087@student.uc.pt

16 de Março de 2022

Conteúdo

1 Introdução

Neste trabalho foi-nos proposto no âmbito da disciplina de Visão por Computador com o objetivo de implementar alguns algoritmos de processamento de imagem. Estes vão ser reunidos para construir um detetor de linha baseado na Transformada de *Hough*. Os processos desenvolvidos serão:

1. *Image Filter*
2. *Edge Filter*
3. *Hough Transform*
4. *Hough Lines*

2 Image Filter

A *Image Filter* vai fazer a convolução de uma dada imagem com qualquer filtro de convolução. Como parâmetros de entrada, vai ter uma imagem em tons de cinza e um filtro de convolução armazenado na matriz h.

Primeiro vou verificar se o kernel é separável ou não. Em caso de ser separável, vou fazer a convolução separadamente entre vertical e horizontal. Para poder convolver os pixeis da borda da imagem foi necessário recorrer à função *padarray* para aumentar metade do tamanho do kernel em todas as direções da imagem.

A convolução entre o kernel vertical e a imagem aumentada, vai resultar numa imagem com o número de linhas da imagem original e o número de colunas da imagem aumentada. Terminámos o filtro, com a convolução horizontal que vai originar o parâmetro de saída desta função com o tamanho da imagem original.

Para tornar a nossa função mais eficiente nos filtros de convolução vertical e horizontal foi aplicada a função *repmat*, que vai replicar a matriz na direção e com o tamanho que desejarmos. Isto permitiu reduzir dois ciclos na função, o que fez com que se tornasse muito mais rápida.

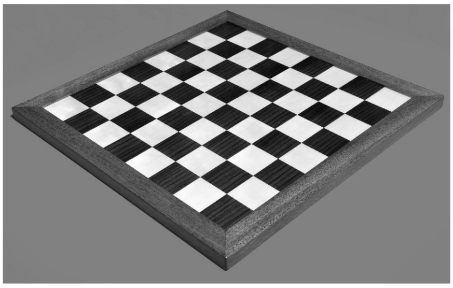

Figura 1: Imagem Original

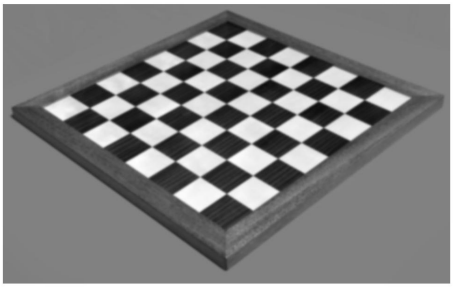

Figura 2: Imagem filtrada

3 Edge Filter

Esta função tem como objetivo encontrar a intensidade e orientação dos cantos numa imagem. Como parâmetros de entrada, vai ter a imagem original em tons de cinza e um sigma que é a variação do filtro de Gauss.

3.1 Filtro de Gauss

Um filtro gaussiano é um filtro passa-baixo usado para reduzir o ruído (componentes de alta frequência) e desfocar regiões de uma imagem.

Em primeiro lugar, tenho que usar a função de convolução para suavizar a imagem com o gaussiano especificado núcleo. Isso ajuda a reduzir o ruído e as bordas finas espúrias na imagem. Quanto maior o tamanho deste kernel, menor a sua sensibilidade a ruído. Portanto, para o cálculo do tamanho do kernel usamos: hsize $= 2 \cdot \text{ceil}(3 \cdot \text{sigma}) + 1$;

Outro parâmetro importante para que a remoção de ruído tenha sucesso é o sigma. Quanto maior for este valor, mais intensa será a distorção do filtro fazendo com que a imagem fique mais distorcida. Se este parâmetro for demasiado baixo, o filtro de Gauss terá pouco efeito por isso a imagem irá continuar com bastante ruído. Por outro lado, se o sigma for demasiado elevado, vai acabar por remover não só o ruído mas também alguns elementos essenciais da imagem.

Tal pode ser visto nas figuras abaixo, posso ver que na figura em que foi usado um sigma muito baixo, a imagem aparece com muito ruído. Ao contrário da imagem da direita em que isso já não acontece, pois, já foi usado um sigma maior e então mais sensível a ruído.

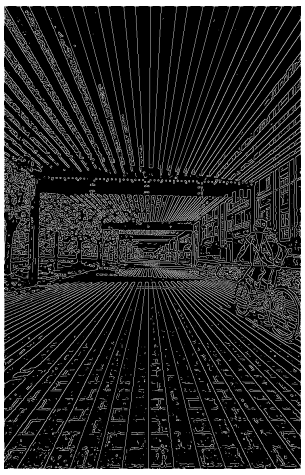

Figura 3: Deteção de edges com sigma=1.5 no filtro de gauss

Figura 4: Deteção de edges com sigma=3 no filtro de gauss

3.2 Gradientes da imagem

Para o cálculo dos gradientes usei duas máscaras de *Sobel*, uma máscara para o gradiente em x e outra para o gradiente em y. No fim de convolvermos a imagem que sai do filtro de Gauss com estas máscaras, obtemos uma imagem com as variações em x, Fig.5 e outra imagem com as variações em y, Fig.6.

$$x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}, \quad y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \tag{1}$$

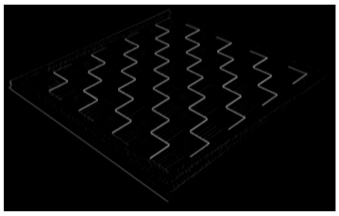

Figura 5: Gradiente X da imagem

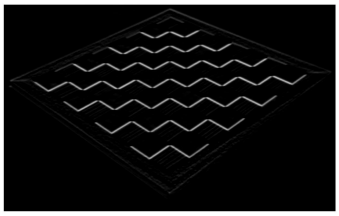

Figura 6: Gradiente Y da imagem

Após termos o gradiente em x e o gradiente em y, vamos obter a magnitude e orientação do gradiente para poder ter apenas uma imagem com a informação de ambos os gradientes, Fig.7. Vou também calcular a orientação deste gradiente, Fig.8, que será perpendicular à orientação da edge. Esta orientação vai ser útil para poder efetuar o próximo passo, a supressão não máxima das edges.

$$G_{mag} = \sqrt{G_x^2 + G_y^2} \tag{2}$$

$$G_{dir} = atan2(G_y, G_x) \tag{3}$$

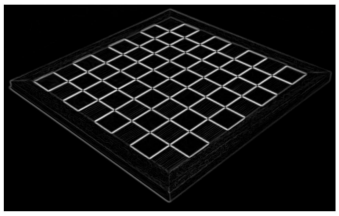

Figura 7: Magnitude do gradiente

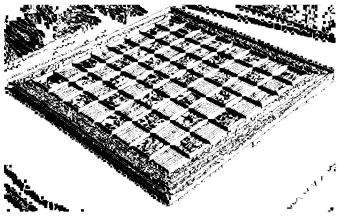

Figura 8: Orientação do gradiente

3.3 Supressão não máxima

Como posso observar na Fig.7, as edges foram detetadas com sucesso, no entanto, todas as edges têm mais que um píxel de espessura e isso não é desejado. Para fazer com que as edges tenham apenas um píxel de espessura teremos de efetuar supressão não máxima na imagem.

A supressão não máxima consiste em comparar o valor da magnitude do gradiente do píxel atual com os pixeis que estão na direção positiva e negativa do gradiente. Se a magnitude do gradiente do píxel atual for maior que os dois pixeis nas direções referidas então significa que o píxel atual é o píxel mais intenso e deve ser preservado. Se não o valor dele deve ser suprimido.

Para a implementação deste algoritmo, dividimos as orientações do gradiente em 4 grupos principais:

1. Se o valor arredondado for 0 ou π então significa que temos de comparar com os pixeis na horizontal.
2. Se o valor arredondado for $\frac{\pi}{4}$ ou $-\frac{3\pi}{4}$ então significa que temos de comparar com os pixeis na diagonal direita.
3. Se o valor arredondado for $\frac{\pi}{2}$ ou $-\frac{\pi}{2}$ então significa que temos de comparar com os pixeis na vertical.
4. Se o valor arredondado for $-\frac{3\pi}{4}$ ou $-\frac{\pi}{4}$ então significa que temos de comparar com os pixeis na diagonal esquerda.

As aproximações são feitas da seguinte forma: se o ângulo estiver contigo no intervalo [0, 22, 5] ou [0, 22, 5].

Posso observar os resultados da nossa implementação da supressão não máxima nas imagens abaixo:

Figura 9: Sem supressão não máxima

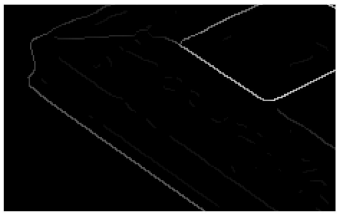

Figura 10: Depois da supressão não máxima

3.4 Resultado final

No fim destas etapas obtemos então uma imagem de edges. Alguns resultados de imagens dadas podem ser observados abaixo:

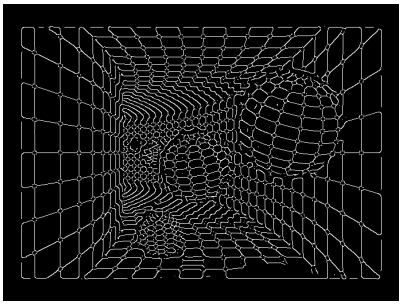

Figura 11:

Figura 12:

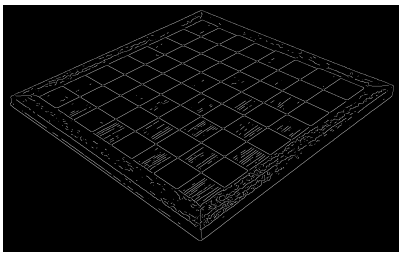

Figura 13:

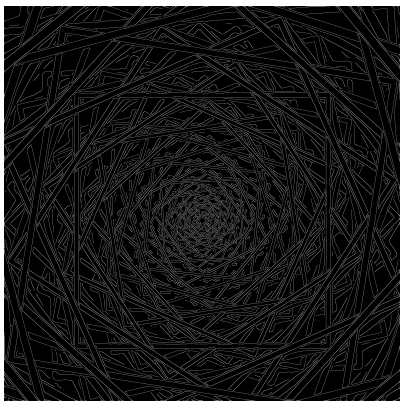

Figura 14:

Posso ver que os algoritmos de deteção de edges funcionam bem em qualquer uma das imagens.

4 Hough Transform

O próximo passo para a deteção de linhas é aplicar à imagem de edges a transformada de Hough. A transformada de Hough consiste num procedimento de votação para uma determinada feature, neste caso, linhas retas. Para detetar linhas usei a parametrização $\rho = xcos\theta + ysin\theta$, em que ρ é a distância da origem até ao ponto mais perto na linha e θ é o ângulo entre o eixo x e a linha que liga a origem ao ponto mais perto da reta, referido anteriormente.

Esta função tem como parâmetros de entrada a imagem de edges binarizada e dois vetores, *rhoRes* e *thetaRes*, que são, respetivamente o vetor de resolução de ρ e o vetor de resolução de θ. Estes vetores são os valores que contam para as votações do acumulador da transformada de Hough, portanto é muito importante que sejam escolhidos criteriosamente. Se forem escolhidas resoluções muito baixas, é possível que os parâmetros das retas não sejam estimados corretamente.

No que diz respeito ao algoritmo usado para a geração da transformada de Hough, comecei por calcular todos os valores de ρ para os pixeis de edges, fazendo variar $\theta\epsilon\ [-\pi, \pi]$. A seguir usando a função de Matab, *binValues=discritize(rho,rhoScale)*, conseguimos associar a cada valor de ρ calculado um bin, que corresponderá ao intervalo de ρ a que pertence no eixo da transformada de Hough. Depois para cada valor que não seja negativo, incrementei o número de votos na célula da transformada de Hough correspondente ao par (ρ, θ) atual. No fim ficamos com uma matriz H, de tamanho dos vetores *rhoScale,thetaScale*, com o número de voto correspondente a cada par.

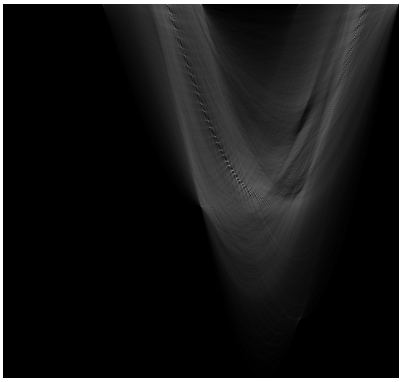

Figura 15: Transformada de Hough da Fig.12

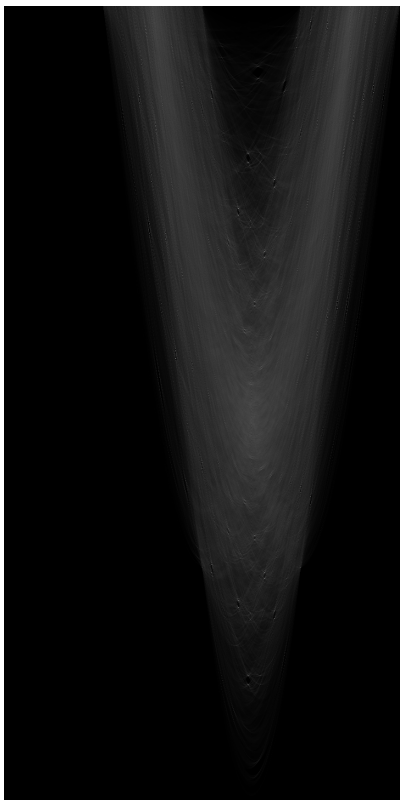

Figura 16: Transformada de Hough da Fig.14

5 Hough Lines

Nesta fase do processo tinha de retirar, dada a transformada de Hough, os valores de ρ e θ das linhas mais votadas. Esta função recebia como parâmetros a transformada de Hough da imagem e um inteiro, *nLines*, que diz respeito ao número de linhas que queremos identificar na imagem. Portanto, o algoritmo implementado em Hough Lines consiste em retirar da transformada de Hough os *nLines* pares mais votados de ρ, θ.

Acontece que, uma reta pode ter votações em várias células da transformada de Hough, portanto é necessário ser feita uma supressão não máxima nas votações da transformada de Hough. Isto é, na vizinhança de uma célula com uma votação alta, existem outras células que provavelmente também tiveram votação alta, mas que não devem ser contabilizadas. O tamanho da vizinhança que será suprimida pode ser escolhido arbitrariamente, tem é de ser um número ímpar. Para a supressão implementamos o seguinte código, sendo N o tamanho da vizinhança:

```
1     lowLimitX = max([rho-N 1]); \\
2     highLimitX = min([rho+N size(H,1)]);\\
3     lowLimitY = max([theta-N 1]);\\
4     highLimitY = min([theta+N size(H,2)]);\\
5     H(lowLimitX:highLimitX,lowLimitY:highLimitY) = 0;
```

A variação do tamanho da vizinhança N pode ser observado nas imagens seguintes:

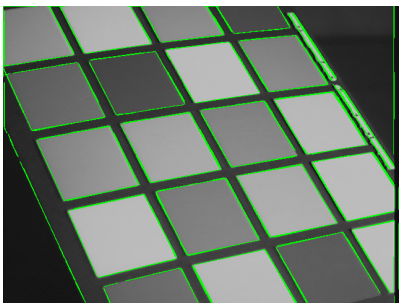

Figura 17: Supressão com N=1

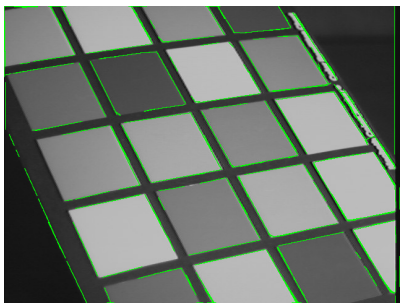

Figura 18: Supressão com N=7

Posso observar que na Fig.17, em que foi usado um N=1, são detetadas muitas linhas sobrepostas, ao contrário da Fig.18 em que já aparecem menos linhas no mesmo lugar. Portanto, penso que conseguimos efetuar a supressão não máxima das linhas com sucesso.

Noutras imagens pode também ser observado o efeito do tamanho da vizinhança. Posso ver que na Fig.19 em que foi usado um N=1, algumas linhas não foram detetadas, pois, foram detetadas várias linhas no mesmo sitio. Na Fig.20 algumas linhas já foram detetadas, pois, não foram detetadas tantas linhas no mesmo sitio.

Figura 19: Supressão com N=1

Figura 20: Supressão com N=7

6 Conclusão

Como tínhamos imagens de ambientes e objetos diferentes, os parâmetros não funcionam de igual forma para todas essas imagens. Por exemplo, o sigma da transformação de Hough tem diferentes valores para as diferentes imagens, com o valor 2 usado reparei que na imagem dos quadrados funciona bem, mas na imagem da estrada não conseguia remover ruído suficiente. Os parâmetros do *threshold* foram implementados fixamente para todas as imagens, no entanto, acho que seria mais eficiente e eficaz se esses parâmetros fossem calculados individualmente para cada imagem.

No decorrer deste trabalho senti várias dificuldades em implementar os algoritmos devido também ao desfasamento entre a teórica e prática da cadeira e por estar sozinho no grupo. Dito isto, a parte que senti mais dificuldade foi na transformada de Hough por ser mais complexa e abstrato de se aprender.

Após implementar o *edge filter* reparei que o resultado poderia ser melhor se implementasse o *double threshold*, porém devido a alguma falta de organização do tempo não o conseguirei implementar.

Para concluir, penso que consegui atingir o objetivo pedido no trabalho que consistia na deteção de linhas retas numa imagem. Em algumas imagens são mais difíceis efetuar essa deteção devido a conterem letras ou demasiada densidade de linhas em pouco espaço.