

# Trabalho 1 - Processamento de Imagens - MC920

Luiz Eduardo Cartolano - 183012

Abril de 2019

## Resumo

O objetivo deste trabalho é a implementação dos conceitos de filtragem vistos em aula. A filtragem de uma imagem digital é uma operação bastante comum que altera os valores de intensidade dos pixels de uma imagem.

No trabalho foi possível obter resultados significativos para os dois tipos de filtros implementados, o espacial e o de frequência, fazendo uso das bibliotecas *opencv*, *numpy* e *scipy* do *Python*.

## 1 Introdução

O objetivo deste trabalho é a implementação dos conceitos de filtragem vistos em aula. A filtragem de uma imagem digital é uma operação bastante comum na qual se altera os valores de intensidade dos *pixels* de uma imagem. Para realizar tal operação realiza-se uma operação de convolução da matriz da imagem com uma máscara.

O uso de filtros na área de processamento de imagens objetiva aumentar sua qualidade, através da remoção ruídos ou imperfeições que podem ser gerados durante os diversos processos pelo qual a imagem pode passar, como a aquisição, a transmissão, além dos processos de compressão.

Este trabalho encontra-se organizado da seguinte forma: a seção 2 apresenta a maneira como o trabalho foi implementado a fim de solucionar o problema. A seção 3 descreve os resultados que foram obtidos a partir da solução implementada. As discussões encontram-se na seção 5, na seção 6 temos as conclusões. As imagens de saída obtidas através das operações de processamento estão presentes na seção de Anexos, no final deste relatório.

## 2 Implementação

### 2.1 Uso do Código

O arquivo *LuizCartolano\_183012.zip* submetido no *Google Classroom*, possui a seguinte estrutura:

```
proj1-2019
├── basicImage
├── constants
├── inputs
│   └── images
├── logger_lib
└── main
└── outputs
    ├── espacial
    │   └── filter_h1 to filter_h5
    ├── frequencia
    │   ├── gaussiana
    │   └── sigma-1 to sigma-9
    └── frequencia-2
        ├── gaussiana-2
        └── sigma-1 to sigma-9
```

```
└── spectro
```

Os arquivos *constants.py* e *logger.lib.py* são códigos auxiliares que ajudam no gerenciamento do projeto, o primeiro guarda os *paths* importantes para as pastas de *inputs* e *outputs*, enquanto o segundo armazena em um arquivo *.log* todas as ações executadas pelo programa (e eventuais erros).

Os arquivos *basicImage.py* e *main.py* são os principais e podem ser considerados o "coração" do projeto. O primeiro, foi montado com a intenção de ser uma biblioteca que implemente funções para manipulação de imagens, portanto, ele contém as funções solicitadas para esse projeto (aplicação dos filtros espaciais e de frequência) e também funções solicitadas para o projeto 0, além de manipulações básicas que podem ser feitas com imagens (como leituras ou armazenamentos). O segundo arquivo citado, por sua vez, contém o fluxo de execução, sendo inclusive o responsável por fazer o projeto funcionar, como será apresentado adiante.

Para executar o projeto basta executar a linha de comando *python3.7 main.py*, e então todos as imagens de saída serão salvas na pasta de *outputs*, como explicaremos na próxima subseção. Para garantir que o projeto execute como esperado é necessário possuir a versão 3 do *python* e também as seguintes bibliotecas: *datetime*, *cv2*, *numpy*, *scipy* e *matplotlib*, as demais bibliotecas usadas não necessitam de *download*.

## 2.2 Detalhes da Implementação

Nesta seção será explicada com mais detalhes as implementações feitas nos arquivos *main.py* e *basicImage.py*.

Como explicado anteriormente, o arquivo *main* contém a lógica de execução do programa. Para este projeto, uma vez que não foi solicitado pelo enunciado a maneira como o projeto deveria ser executado optou-se por uma abordagem na qual não ocorre interação com o usuário após dar início ao *script*. Então, a primeira ação executada é armazenar em uma lista as imagens lidas da pasta *inputs*, e então, executar, para cada uma das imagens as ações de filtro solicitadas. Após realizar a filtragem, as imagens de saída são armazenadas nas respectivas pastas dentro da pasta de *outputs*, isto é, as imagens sobre as quais aplicou-se o filtro  $h_1$  estarão salvas no caminho *outputs/espacial/h1/* e assim por diante. A pasta *outputs/frequencia* armazena as imagens obtidas pelo filtro do *scipy* e a *outputs/frequencia-2* do filtro implementado manualmente, como será visto mais adiante.

No arquivo *basicImage.py* realizou-se a implementação de todas as funções que de fato manipulam as imagens. Na maioria delas fez-se uso das funções disponibilizadas pela biblioteca *opencv* e também da biblioteca *scipy*. Cada uma das funções será explicada com mais detalhes nas subseções seguitens:

### 2.2.1 Filtro Espacial

Assim como os sinais unidimensionais, as imagens também podem ser filtradas com vários filtros passa-baixo (LPF), filtros passa-alto (HPF), etc. O processo de filtragem consistem em realizar uma convolução entre a matriz do filtro e matriz da imagem.

Para realizar a operação neste projeto contou-se com auxílio das bibliotecas *numpy* e *opencv*. O *numpy* foi utilizado para criar e manipular as matrizes dos filtros que foram criadas como a vista na Equação 2 e representam matrizes como a vista Matriz 3. Já o *OpenCv* foi usado para realizar a convolução propriamente dita.

A convolução em si, foi feita através de uma função disponibilizada pela biblioteca do *opencv*, sua documentação pode ser vista com mais detalhes em 1. Mas de modo geral, seu funcionamento se da pela chamada da função *cv2.filter2D(src, ddepth, kernel)*. O argumento *src* refere-se a imagem sobre a qual irá se aplicar o filtro. Já o *ddepth* refere-se a profundidade na qual o filtro será aplicado. Por fim, o parâmetro *kernel* representa a matriz que realizará a convolução.

### 2.2.2 Filtro Frequênci

A transformada de Fourier é usada para analisar as características de frequência de vários filtros. Para imagens, a Transformada Discreta Fourier 2D (DFT) é usada para encontrar o domínio da frequência. Um algoritmo rápido chamado Fast Fourier Transform (FFT) é usado para o cálculo de DFT.

Para realizar a operação neste projeto contou-se com auxílio das bibliotecas *numpy* e *scipy*. O *numpy*

foi utilizado para realizar manipulações necessárias com as matrizes das imagens. Já o *scipy* foi usado para realizar a aplicação do filtro.

O filtro de frequência funciona como o explicado (em alto nível) no Algoritmo 1. A lógica para aplicarmos os filtros de frequência é, em primeiro lugar, aplicar tanto a matriz da imagem quanto a matriz do filtro, no caso um gaussiano, a transformada de fourier. Após aplicar a transformada, é operada a convolução entre ambas as matrizes e, por fim, aplica-se uma transformada inversa para poder obter a imagem de saída que será, no nosso caso, armazenada.

O projeto adotou duas implementações diferentes para os filtros de frequência. A primeira delas foi realizando os passos descritos um a um, usando as funções da biblioteca *numpy*. Para isso, primeiro aplicou-se as funções *np.fft.fft2(img)* e *np.fft.fftshift fft* para realizar a *transformada de Fourier* e deslocar o centro da imagem. Depois, usou-se a função *cv2.getGaussianKernel(ksize, sigma)* para gerar a matriz do kernel gaussiano e então, aplicou-se as mesmas funções mencionadas anteriormente para obter a transformada da matriz. Após obtida as duas transformadas realizou-se a multiplicação de ambas as matrizes, aqui temos um detalhe interessante das vantagens do uso da transformada, antes era preciso usar uma convolução para aplicar o filtro a matriz, contudo, no domínio da frequência basta operar o que seria equivalente a uma multiplicação no domínio espacial. Por fim, é preciso retornar as imagens para o domínio espacial, para tal operação foram usadas as funções *np.fft.ifftshift(img-borrada)* e *np.fft.ifft2(f\_ishift)*.

Apesar de complexo, todos esses passos podem ser simplificados com aplicação de uma função da biblioteca *scipy*, que realiza todas essas operações e nos retorna a imagem com o filtro aplicado. Ela é usada fazendo a chamada da função *signal.convolve(img, kernel, mode='same')*. O argumento *src* refere-se a imagem sobre a qual irá se aplicar o filtro. Por fim, o parâmetro *kernel* representa a matriz que realizará a convolução. Esta opção também foi adotada na implementação do projeto, a fim de se comparar o desempenho e a qualidade das imagens de saída obtidas. O *kernel* usado para a segunda implementação foi obtido a partir da combinação das funções *np.outer(in1,in2)* e *cv2.getGaussianKernel(ksize, sigma)*, a primeira agrupa duas listas como uma matriz, enquanto a segunda gera uma matriz gaussiana com a dimensão e desvio informados pelo usuário. Para este trabalho duas abordagens foram usadas, primeiro foi usado um valor fixo de *sigma* e variou-se o *ksize*, na segunda tentativa fez-se exatamente o contrário. Os filtros gaussianos encontram-se na Figura 4.

Um detalhe importante que deve ser observado para ambos os filtros é no *range* e no tipo dos valores presentes na matriz, no caso deste projeto foi preciso realizar uma normalização dos valores para que a imagem pudesse ser armazenada de maneira correta ao fim do processamento.

### 3 Resultados

O projeto após executado gera dois tipos de *outputs*, as imagens obtidas após aplicação do filtro espacial e as obtidas após aplicação do filtro de frequência. Cada um deles, por sua vez, será dividido em cinco subtipos, o primeiro é subdividido para as máscaras  $h_1$ ,  $h_2$ ,  $h_3$ ,  $h_4$  e pela junção das imagens  $h_3$  e  $h_4$ . Os filtros de frequência também serão subdivididos em cinco tipos, sendo que cada um deles representa um tamanho diferente nas dimensões do *kernel*.

As imagens de saída foram agrupadas a fim de facilitar a visualização das mesmas. Para as imagens obtidas pela aplicação do filtro espacial, podemos observar seu resultado na Figura 1. Já as obtidas para o filtro de frequência estão na Figura 2.

#### 3.1 Resultados Filtros Espaciais

Os resultados desta seção encontram-se na Figura 1. Para o filtro  $h_1$ , representado pela Matriz 3, é possível perceber uma melhora nos contornos das imagens pós aplicação do filtro. Já para  $h_2$  (Matriz 4, há pouca mudança quando comparado as imagens de entrada. Após aplicação dos filtros  $h_3$  e  $h_4$  (Matrizes 5 e 6) percebemos um realce das linhas verticais e horizontais, respectivamente. Por fim, a junção das imagens resultantes após os últimos filtros comentados nos fornece um realce de ambas as bordas da imagem.

### 3.2 Resultados Filtros de Frequência

Os resultados desta seção encontram-se nas Figura 2 e 3. A primeira, diz respeito a implementação do filtro usando a biblioteca do *scipy* e a segunda da implementação manual, é possível perceber uma leve diferença nas imagens mas nada muito significativo. Vale notar que em ambos os casos a alteração do fator sigma que origina o filtro gaussiano acarreta diferenças na imagem de saída, e também, que em ambos há uma pequena deformação nas bordas da imagem.

## 4 Discussão

A fim de testar as soluções implementadas visando a resolução dos problemas propostos usou-se o conjunto de imagens disponibilizado pelo professor e que pode ser encontrado em [3]. Para as quais se obteve as imagens de saída apresentadas neste documento.

### 4.1 Filtros Espaciais

Analizando as imagens de saída para o filtro  $h_1$  é possível perceber uma leve melhoria na obtenção dos contornos dos objetos. Sendo assim, no global, pode-se concluir que o uso deste filtro funciona como uma boa ferramenta para a obtenção dos contornos. Portanto, analisando a imagem e a estrutura da matriz podemos dizer que esse é um *filtro laplaciano*.

Analizando as imagens de saída para o filtro  $h_2$  é possível perceber uma leve diferença nas imagens de saída. Pela análise da matriz correspondente podemos dizer que é um *filtro gaussiano*, este tipo de filtro tem um comportamento similar ao filtro passa-baixo, a aplicação destes, por sua vez, resulta numa suavização da imagem original.

Os filtros  $h_3$  e  $h_4$  são ambas do tipo *filtro passa-alta*. Eles são utilizados para detectar bordas, uma vez que, quanto mais alta a frequência selecionada, maior é o número de variações de cor que são filtradas, sendo que as únicas variações que não são filtradas são as mais bruscas. No nosso caso, o primeiro realiza uma detecção de bordas verticais e o segundo de horizontais.

Já o último filtro, resultante da aplicação da Equação 1 as imagens resultantes após aplicação dos filtros três e quatro tem comportamento parecido ao de um filtro de Sobel. Ele funciona detectando bordas verticais e horizontais na imagem.

### 4.2 Filtro de Frequência

As imagens da Figura 2, também trazem detalhes interessantes sobre os filtros de frequência. É possível perceber que a medida que aumentamos o tamanho do desvio (sigma) do *kernel* da gaussiana - que é o nome dado para a matriz da máscara que será aplicada - aumentamos o nível de intensidade da borra dos *outputs*. O que nos leva a uma conclusão interessante sobre os *filtro gaussianos* que diz respeito ao fato de que a suavização da imagem é tanto mais visível quanto maior for o desvio padrão sigma considerado, não dependendo muito do tamanho da matriz utilizada. A fim de ter certeza do que está sendo dito foram feitos dois testes diferentes, o primeiro deles foi fixar um valor de sigma e alterar apenas o tamanho do *kernel*, nesta não foram percebidas qualquer mudança das imagens. O segundo teste foi manter um tamanho fixo de *kernel* e variar o sigma, os resultados foram os mostrados nas Figuras 2 e 3.

Outra análise interessante que podemos tirar está na comparação entre as imagens de saída obtidas na Figura 2 e na Figura 3, visto que ambas são originadas através de processos diferentes. É possível perceber que a qualidade das imagens obtidas através da execução passo a passo das transformadas é maior quando comparada a implementação feita via *scipy*. Essa qualidade, contudo, tem o preço, o custo computacional dessas operações é consideravelmente maior como pode ser visto na Tabela 1.

Também é interessante analisarmos os resultados da Figura 5, nele temos o representação da magnitude do espectro antes e após a aplicação do *filtro gaussiano*, e é bastante visível a alteração imposta pelo filtro ao espectro de cada uma das imagens.

### 4.3 Limitações

Ainda que as soluções apresentadas tenham apresentados resultados significativos e sejam executadas de maneira eficiente, o programa apresenta limitações. Uma delas é o fato de que o algoritmo não pode

ser aplicado para imagens que não sejam as pré definidas na pasta de *inputs*.

Outra limitação está na função gaussiana, como podemos perceber as bordas da imagem estão danificadas em ambas as interações, desse modo, é algo que precisa ser corrigido em iterações futuras.

A maneira como é feita a geração do *kernel* da gaussiana não é das melhores, uma vez que não se tem controle sobre os valores que compõem a matriz, já que esta é gerada a partir da biblioteca *opencv*.

## 5 Conclusão

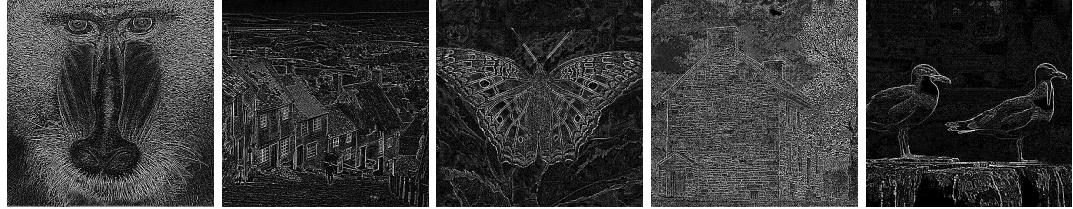
O uso de filtros na área de processamento de imagens objetiva na maioria dos casos aumentar sua qualidade, através da remoção de ruídos e imperfeições que podem estar presentes na imagem. Com base nos filtros vistos na implementação deste trabalho é possível concluir que os filtros espaciais e de frequência são ferramentas potentes para lidar com imperfeições das imagens, além de serem de fácil implementação.

Para um melhor resultado das imagens de saída, pode-se, em uma próxima iteração avaliar os filtros de frequência para valores menores de dimensão do *kernel* e caso não se tenha resultados significativos talvez até mesmo realizar uma própria implementação da função, sem uso das funções disponibilizadas no *scipy*.

## Referências

- [1] Funcao filtro espacial opencv. Disponível em: <https://docs.opencv.org/3.0-beta/modules/imgproc/doc/filtering.html#filter2d>, Acesso em: 09-04-2019.
- [2] Funcao gaussiana scipy. Disponível em: [https://scipy-lectures.org/intro/scipy/auto\\_examples/solutions/plot\\_image\\_blur.html](https://scipy-lectures.org/intro/scipy/auto_examples/solutions/plot_image_blur.html), Acesso em: 09-04-2019.
- [3] PEDRINI Helio. Projeto 1. Disponível em: <http://www.ic.unicamp.br/~helio/disciplinas/MC920/trabalho1.pdf>, Acesso em: 09-04-2019.

## ANEXOS



(a) Figuras após aplicação do filtro  $h_1$



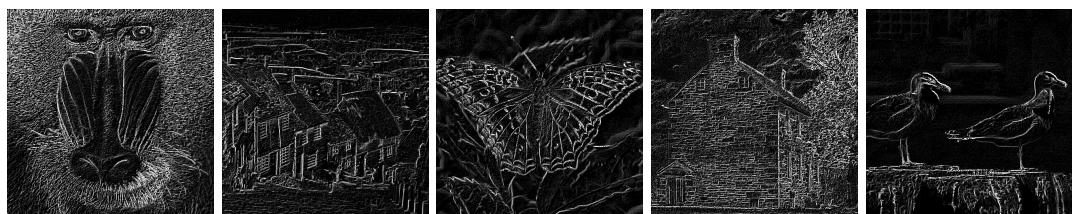
(b) Figuras após aplicação do filtro  $h_2$



(c) Figuras após aplicação do filtro  $h_3$



(d) Figuras após aplicação do filtro  $h_4$



(e) Figuras após aplicação de uma junção dos filtros  $h_3$  e  $h_4$  de acordo com a Equação 1

Figura (1) Figuras após aplicação dos filtros espaciais.

$$\sqrt{h_3^2 + h_4^2} \quad (1)$$

(1) Fórmula para o cálculo da junção dos filtros  $h_3$  e  $h_4$ .



(a) Figuras após aplicação do filtro com sigma 1.0.



(b) Figuras após aplicação do filtro com sigma 3.0.



(c) Figuras após aplicação do filtro com sigma 5.0.



(d) Figuras após aplicação do filtro com sigma 7.0.



(e) Figuras após aplicação do filtro com sigma 9.0.

Figura (2) Figuras após aplicação dos filtros de frequência usando a função da biblioteca *scipy*

$$\text{array}([[0., 0., -1., 0., 0.], [0., -1., -2., -1., 0.], [-1., -2., 16., -2., -1.], [0., -1., -2., -1., 0.], [0., 0., -1., 0., 0.]]) \quad (2)$$

(2) Exemplo de *numpy array* usado no trabalho.



(a) Figuras apóis aplicação do filtro com sigma 1.0.



(b) Figuras apóis aplicação do filtro com sigma 3.0.



(c) Figuras apóis aplicação do filtro com sigma 5.0.



(d) Figuras apóis aplicação do filtro com sigma 7.0.



(e) Figuras apóis aplicação do filtro com sigma 9.0.

Figura (3) Figuras apóis aplicação dos filtros de frequêcia usando a função implementada manualmente.

$$\begin{bmatrix} 0 & 0 & -1 & 0 & 0 \\ 0 & -1 & -2 & -1 & 0 \\ -1 & -2 & 16 & -2 & -1 \\ 0 & -1 & -2 & -1 & 0 \\ 0 & 0 & -1 & 0 & 0 \end{bmatrix} \quad (3)$$

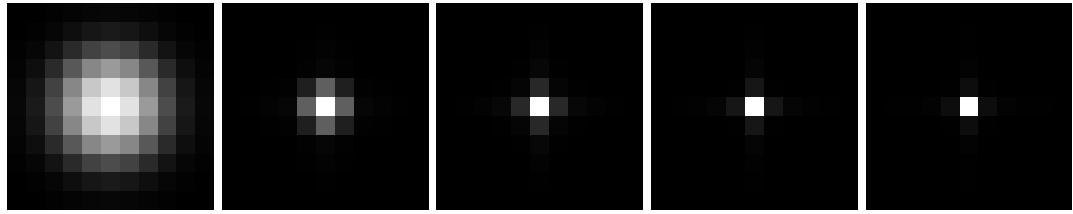
(3) Máscara  $h_1$ .

**Algoritmo 1:** Algoritmo para aplicação do filtro em frequêcia

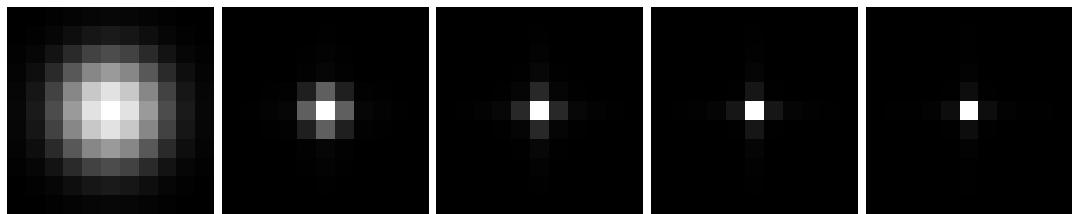
**Data:** *imagem\_sem\_filtro.png*

**Result:** *imagem\_com\_filtro.png*

- 1 leitura da imagem ;
- 2 cria um filtro gaussiano ;
- 3 aplica fourier na imagem ;
- 4 aplica fourier no filtro ;
- 5 convolucao da imagem com o filtro ;
- 6 aplica o inverso de fourier na imagem pos convolucao ;

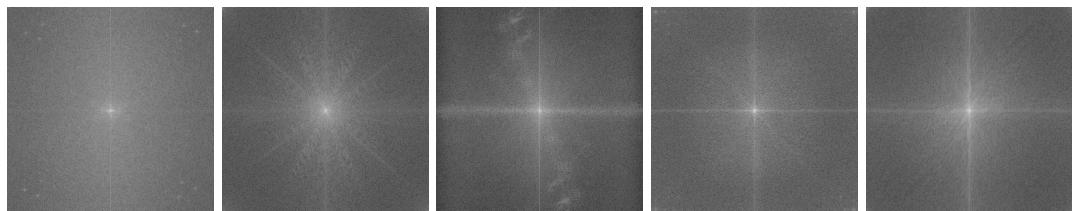


(a) Filtros gaussianos aplicados na segunda implementação.

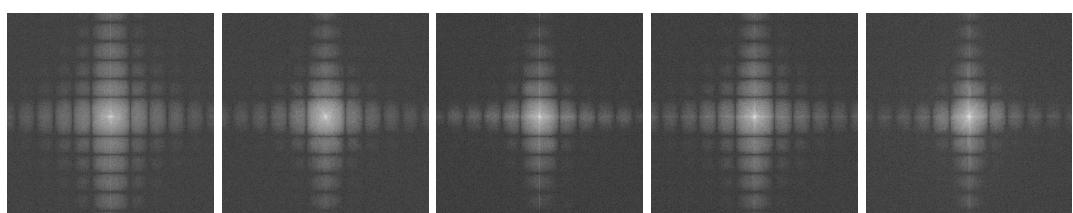


(b) Filtros gaussianos aplicados na primeira implementação.

Figura (4) Filtros gaussianos aplicados.



(a) Espectro das imagens antes da aplicação do filtro gaussiano.



(b) Espectro das imagens antes da aplicação do filtro gaussiano.

Figura (5) Espectro de *Fourier* das imagens.

Tabela (1) Tabela com o desempenho para os cinco sigmas em cada uma das funções implementadas.

<b>Função</b>	<b>Média</b>	<b>Desvio</b>	<b>Intervalo de Confiança</b>
Implementação passo a passo	0.294	0.034	(0.289,0.299)
Implementação Scipy	0.180	0.034	(0.177,0.187)

$$\frac{1}{256} \cdot \begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix} \quad (4)$$

(4) Máscara  $h_2$ .

$$\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \quad (5)$$

(5) Máscara  $h_3$ .

$$\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \quad (6)$$

(6) Máscara  $h_4$ .