Universidade Estadual de Campinas

MO443 – Introdução ao Processamento de Imagem Digital

Prof. Dr. Hélio Pedrini

Trabalho 1

Felipe Bezerra

043372

Campinas, setembro de 2022

Sumário

Introdução

O presente relatório é parte do processo de avaliação da disciplina MO443 – Introdução ao Processamento da Imagem Digital, ministrada pelo Prof. Dr. Hélio Pedrini, do Instituto de Computação da Unicamp, no segundo semestre de 2022.

Este relatório acompanha o código comentádo ao qual faz referência durante o texto, ambos constantes do arquivo .zip que conforma a entrega desta avaliação, bem como imagens resultantes da aplicação dos procedimentos aqui descritos.

Como preâmbulo, gostaria de destacar que este trabalho foi bastante relevante na aceleração do meu aprendizado em programação e processamento de imagem digital. De formação, sou fotógrafo, portanto meu conhecimento sobre o assunto da disciplina diz respeito a noções distantes. Ao realizar este trabalho e estudar os conteúdos das aulas, tive meu primeiro contato com Python, Numpy e outras ferramentas e referências basilares para a matéria. Diante disso, peço, antecipadamente, desculpas pela muito provável ausência de boas práticas na construção tanto do código quanto deste relatório, pelas ênfases excessivas em pontos óbvios para iniciados, pela ausência de destaque para pontos mais relevantes, bem como pelo emprego de soluções ineficientes e provavelmente ingênuas nas questões. O aprendizado até este breve momento já tem sido considerável para mim, cuja experiência com programação, até então, não passava de pontuais curiosidades.

Na medida do possível, busquei relacionar os pontos deste trabalho com a minha área de exploração, a fotografia, bem como utilizei algumas imagens minhas para alguns testes.

Os nomes dos arquivos produzidos pelo código são descritivos das questões que pretendem atender e respeitam os diretórios e as extensões originais dos arquivos.

1.1. Transformação de intensidade

Negativo da imagem

O exercício 1.1 propõe uma série de transformações para serem executadas em uma imagem, um arquivo digital de uma fotografia com dimensões de 512 por 512 pixels em formato PNG em escala de tons de cinza. A profundidade de cor[1] da imagem é de 8 bits, portanto o arquivo comporta valores de intensidade dos pixels entre 0 e 255. Segue uma prévia da imagem proposta para as transformações solicitadas:

Figura 1.1.1 - Imagem do arquivo city.png

Para a transformação desejada, a intensidade de cada pixel deve assumir valores os opostos em relação a escala no intervalo [0, 255]. Por exemplo, o valor zero (correspondente ao preto puro nessa escala) passará a 255 (branco puro). O primeiro tom de cinza após o preto (representado numericamente pelo valor 1), passará a 254, e assim para cada nível de intensidade. Matematicamente, se f for um pixel de entrada e g o pixel correspondente na matriz de saída, podemos escrever a transformação almejada segundo a seguinte equação:

[1] Por profundidade de cor, pretendo expressar a intensidade máxima para cada pixel. É um jargão comum na área da fotografia, porém me escapa se neste contexto é adequado.

$$f(x,y) = 255 - g(x,y).$$

Em outros termos, o valor de cada pixel na imagem de saída será igual a 255 menos o valor do pixel de entrada correspondente. De fato,

$$f(x,y) + g(x,y) = 255$$

para cada pixel numa imagem com intensidade máxima de 8 bits.

O resultado obtido para a imagem city.png segue abaixo, ao lado da imagem original.

Figura 1.1.2 - Imagens city.png (esquerda) e city_negative.png (direita)

A imagem resultante da transformação descrita é o "negativo" da imagem original, de aspecto semelhante ao que teríamos em uma aplicação de fotografia analógica em preto e branco.

Observações sobre o código

O algoritmo utilizado para executar essa operação realiza o comando por vetorização, por meio de uma operação envolvendo matrizes implementadas pela biblioteca Numpy.

Redefinição da escala de níveis de cinza

Para este item, foi solicitada a alteração da escala dos níveis de cinza da imagem apresentada. O enunciado pediu, literalmente, para se "converter o intervalo de intensidades para [100, 200]". Após consulta ao monitor Leandro e explicações em aula do professor Hélio, optei por interpretar o enunciado da seguinte forma:

$$g(x,y) = \left(\frac{200-100}{255-0}\right) f(x,y) - \left(\frac{200-100}{255-0}\right) 255 + 100.$$

Simplificando, a imagem de saída seria calculada por

$$g(x,y) = \frac{100}{255} f(x,y) + 100.$$

Esse procedimento promove a readequação de cada um dos níveis de intensidade na imagem original para que fiquem restritos à faixa [100, 200]. Uma vez que – segundo a interpretação adotada para ao enunciado – a faixa de intensidades que cada pixel da imagem city.png pode ocupar é consideravelmente restringida (de 0 a 255 para 100 a 200), o resultado esperado seria uma imagem com notável redução de contraste, como podemos ver no resultado obtido abaixo.


Figura 1.1.3 - Imagens city.png (esquerda) e city_new_gray_scale_from_100_to_200.png (direita)

Observações sobre o código

Independentemente da interpretação adotada, o enunciado foi explícito ao definir a nova faixa de intensidades a ser utilizada na transformação, [100, 200]. Todavia, a função implementada para esta solução permite a definição de outros valores para a nova escala tonal[2], caso assim se deseje.

Para definirmos um procedimento matemático parametrizado, podemos generalizar a última equação apresentada da seguinte forma:

$$g(x,y) = \left(\frac{q_{máx} - q_{mín}}{p_{máx} - p_{mín}}\right) f(x,y) - \left(\frac{q_{máx} - q_{mín}}{p_{máx} - p_{mín}}\right) p_{máx} + q_{máx},$$

em que

p_{mix} é o valor mínimo possível na escala original,

$p_{máx}$ é o valor máximo possível na escala original,

$q_{mín}$ é o valor mínimo possível na nova escala e

$q_{máx}$ é o valor máximo possível na nova escala.

[2] Por escala tonal, me refiro ao total de níveis de cinza em questão.

No caso dessa aplicação, $p_{mín}$ e $p_{máx}$ assumem sempre 0 e 255 respectivamente, portanto podemos simplificar a equação acima para

$$g(x,y) = \left(\frac{q_{máx} - q_{mín}}{255}\right) f(x,y) + q_{mín},$$

que, com $q_{mín} = 100$ e $q_{máx} = 200$ resolve o exercício pela interpretação adotada.

O código implementado restringe os valores dos limites da nova escala para que não excedam o intervalo [0, 255]. Todavia, a implementação abre possibilidades para explorações. Por exemplo, se $q_{mín} > q_{máx}$ a escala é invertida, produzindo uma imagem de fato no intervalo solicitado, porém com aspecto de negativo. Abaixo, um exemplo para $q_{mín} = 200$ e $q_{máx} = 100$.

Figura 1.1.4 - Imagens city.png (esquerda) e city_new_gray_scale_from_200_to_100.png (direita)

Note que a imagem resultante possui menos contraste do que a imagem city_negative.png, obtida para o exercício anterior. Isso se deve ao fato de que a imagem city_new_grayscale_from_200_to_100.png não apenas inverte a imagem, mas também restringe a faixa de intensidades para o intervalo de 100 a 200.

A partir da mesma lógica, poderíamos resolver o exercício do negativo com o código da função `new_gray_scale`, bastando, para tanto, colocar como

parâmetros 255 para o valor mínimo e 0 para o valor máximo da nova escala[3]. Mais claramente, fazendo $q_{máx} = 0$ e $q_{mín} = 255$, vem

$$f(x, y) = 255 - g(x, y),$$

que recupera exatamente equação que calcula o negativo de um pixel de 8 bits de intensidade.

Outra possibilidade seria $q_{máx} = q_{mín}$, o que, trivialmente, produziria uma imagem totalmente uniforme no tom de cinza correspondente.

O reajuste de escalas tonais, em fotografia, geralmente se faz necessário quando se deseja fazer impressões. Por exemplo, para se evitar uma imagem com pretos excessivamente densos em um sistema[4] com *gamut* mais restrito, pode-se promover a elevação do nível mínimo dos canais de cores da imagem. De forma semelhante, para se evitar altas luzes excessivamente claras, pode-se rebaixar o nível máximo de brilho da foto em questão.

Linhas pares invertidas

O enunciado solicita que se inverta a orientação das linhas pares da imagem city.png. O aspecto esperado para a imagem resultante seria ao de um tipo de embaralhamento horizontal, lembrando um pouco a aparência de efeitos de vídeos entrelaçados[5]. O resultado da aplicação da função `flip_even_rows` para a imagem city.png aparece a seguir.

[3] Em verdade, a implementação dos dois códigos difere na medida em que a função `new_gray_scale` produz valores decimais em seu cálculo, enquanto a função `negative`, por não fazer uso de divisão, não incorre em erro de truncamento ou arredondamento.

[4] Neste ponto, considera-se o sistema que abarca a impressora, as tintas e o papel/suporte escolhido para impressão, com perfil de cores devidamente determinado.

[5] *Interlaced video*.

Figura 1.1.5 - Imagens city.png (esquerda) e city_flipped_even_rows.png (direita)

O aspecto de simetria é apenas aparente: em princípio, cada linha da imagem, apesar de geralmente semelhante, não é exatamente igual às adjacentes.

Observações sobre o código

Para a realização desse exercício, utilizou-se a técnica de *slicing* para selecionar as linhas pares por meio da indexação `[1::2,:]`, ou seja, selecionar, a partir da segunda linha (de índice 1), cada duas linhas e todas as colunas da matriz com os valores da imagem. Depois, a indexação `[1::2,::-1]` procede com a reversão horizontal das (colunas das) linhas pares.

Reflexão de linhas

O enunciado, aqui, pede que se realize a reflexão da imagem na direção vertical, espelhando-se a metade horizontal superior na metade inferior da imagem.

Para tanto, do ponto de vista de um procedimento que execute a tarefa, é necessário colher a metade horizontal superior da imagem e repeti-la sobre –

isto é, substituindo – a metade inferior, produzindo uma imagem com a reflexão desejada. O resultado obtido pela aplicação da função `reflex_row` na imagem city.png segue.

Figura 1.1.6 - Imagens city.png (esquerda) e city_row_reflexion.png (direita)

A imagem resultante exibe simetria vertical em relação ao eixo horizontal imaginário que divide a imagem em duas metades de mesma dimensão.

Observações sobre o código

Para implementar o procedimento, utilizou-se as funções `numpy.split`, para se fatiar a matriz original ao meio e `numpy.vstack`, para se empilhar as duas metades que compõem a imagem resultante. Para a inversão da metade espelhada, utilizou-se *slicing*, com intervalo de seleção das linhas igual a -1.

Espelhamento vertical

O exercício pede, neste tópico, algo semelhante ao cobrado no anterior. Em vez de, porém, executar o espelhamento de metade da imagem, produzindo uma simetria no quadro final, aqui é pedido que se inverta a orientação vertical de toda a imagem. O resultado esperado seria uma imagem com aspecto semelhante ao que observaríamos se pudéssemos colocar um espelho logo abaixo da imagem em city.png. O resultado, gerado pela função `v_mirror`, corrobora a predição e aparece na sequência.

Figura 1.1.7 - Imagens city.png (esquerda) e city_vertical_mirror.png (direita)

Repare que o resultado não equivale a uma rotação em 180° da imagem original, uma vez que a orientação vertical é de fato alterada, ou seja, não é possível obter a imagem city_vertical_mirror.png por meio de rotações no plano da imagem, apenas por reflexão[6].

Observações sobre o código

[6] Todavia, se considerarmos uma rotação espacial da imagem em relação a um eixo horizontal paralelo ao plano que contém a informação, é possível de se obter a imagem city_vertical_mirror.png.

O procedimento para este item é simples de ser executado por *slicing* e pela inversão da ordem das linhas (portanto inversão vertical), utilizando-se a indexação `[::-1]` da matriz que contém os valores da imagem.

1.2. Ajuste de brilho

Este exercício propõe que se execute a correção, ou o ajuste, do brilho da imagem por meio do uso do expoente gama, γ. A equação que descreve esta operação é a seguinte:

$$g(x, y) = f(x, y)^{\frac{1}{\gamma}},$$

em que γ é o expoente a ser ajustado para a imagem a ser corrigida.

Para que a equação acima seja devidamente aplicada, padroniza-se os valores de f para que as intensidades dos pixels, em vez de estarem contidas em um intervalo de 0 a 255, cubram o intervalo de 0 a 1, havendo a conversão de volta para o intervalo original após a correção citada. Dessa forma, valores de γ superiores a 1 produzirão um expoente menor do que 1 sobre as intensidades de f, o que tenderá a aumentar a intensidade de cada pixel, portanto a imagem se tornará mais clara. Alternativamente, se $\gamma < 1$, f tenderá a ter suas intensidades resultantes reduzidas, o que escurece a imagem em g. Naturalmente, se $\gamma = 1$, a imagem permanece inalterada. Os resultados foram produzidos pela função `gamma_correction` aplicada na imagem baboon.png. Abaixo, alguns exemplos de imagens resultantes para essa solução. Para facilitar a visualização, as imagens foram reduzidas em tamanho.

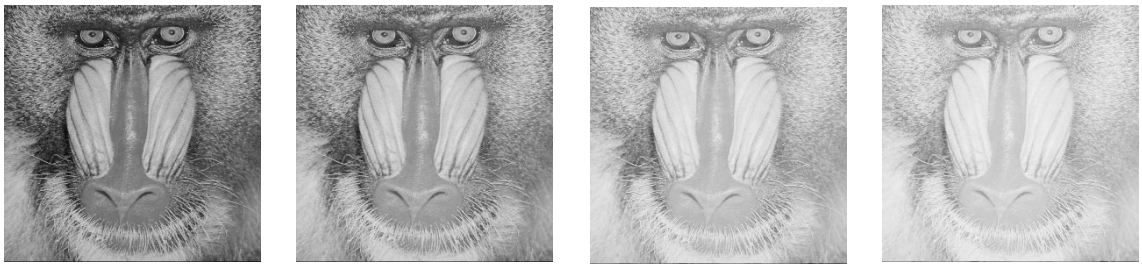
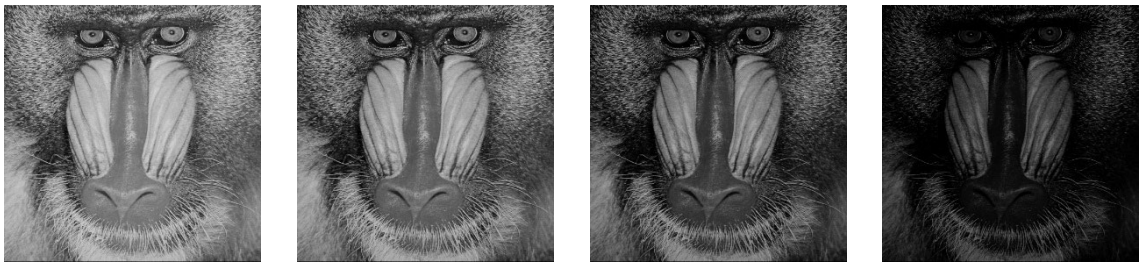


Imagem original | $\gamma = 1.5$ | $\gamma = 2.5$ | $\gamma = 3.5$



$\gamma = 1.0$ | $\gamma = 0.75$ | $\gamma = 0.5$ | $\gamma = 0.25$

Figura 1.2.1 – A começar pela primeira linha, da esquerda para a direita: baboon.png, baboon_gamma_1.5.png, baboon_gamma_2.5.png, baboon_gamma_3.5.png, baboon_gamma_1.png, baboon_gamma_0.75.png, baboon_gamma_0.5.png e baboon_gamma_0.25.png.

É interessante notar que a relação do valor de γ com o brilho da imagem não é, de fato, linear, como podemos ver na equação regente. Há um salto na intensidade dos níveis de preto percebidos ao se passar de 0.5 a 0.25, muito mais significativo do que a variação entre 1.0 e 0.75.

Comentários sobre o código

A implementação do algoritmo se vale, como os demais procedimentos até aqui, de vetorização para converter cada pixel da imagem de entrada para a escala unitária (ou seja, de 0 a 1), para aplicar o ajuste de gama e para converter de volta os valores obtidos para o intervalo de 8 bits por pixel. Ao final, realiza-se a conversão para tipos inteiros de 8 bits sem sinal.

1.3. Planos de bits

O enunciado pede que se extraia os planos de bits de uma imagem. Como as imagens que estamos utilizando possuem 8 bits por pixel, haverá 8 planos para resultantes da aplicação do procedimento.

O plano de bit é uma imagem com valores ou preto ou branco em cada pixel, em que os valores brancos indicam que a imagem original possuía informação naquele pixel para o dígito em questão, e valores pretos, alternativamente, indicam a ausência daquela informação.

A informação de um pixel de uma imagem de d bits pode ser expressa como um polinômio em base binária da seguinte maneira:

$$\sum_{k=0}^{d-1} b_k 2^k = b_0 2^0 + b_1 2^1 + \ \ldots \ + b_{d-2} 2^{d-2} + b_{d-1} 2^{d-1}.$$

Os coeficientes b_k determinam se o pixel em questão possui ou não informação no plano k. Para uma imagem de 8 bits,

$$\sum_{k=0}^{8-1} b_k 2^k = b_0 2^0 + b_1 2^1 + \ b_2 2^2 + b_3 2^3 + b_4 2^4 + b_5 2^5 + b_6 2^6 + b_7 2^7.$$

Portanto, para se extrair um plano de bits k de uma imagem, é necessário comparar, em base binária, cada pixel com o valor de 2^k. Por exemplo, se desejamos extrair o plano de bits para $k = 6$, temos $2^6 = 64$, que, em base binária, é representado por 1000000. Imaginemos, então, um pixel com valor 219. Em base binária, 219 é representado por 11011011. Procedemos então à comparação com 1000000 para verificarmos se esse pixel possui informação no plano de bit 6:

0**1**000000

1**1**011011

O dígito em negrito evidencia que esse pixel possui valor 1 no 7º dígito (que corresponde ao plano 6), portanto há informação no plano 6 para esse pixel. Caso o valor do pixel no sétimo dígito fosse 0, diríamos que não há informação naquele plano para aquele pixel.

O conjunto de imagens abaixo demonstra os 8 planos de bits, de 0 a 7, para a imagem seagull.png.

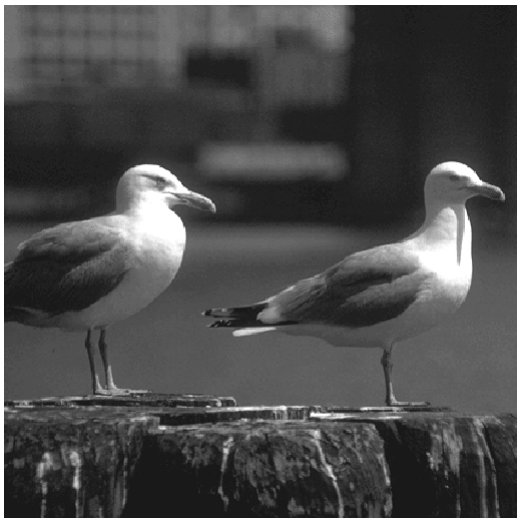
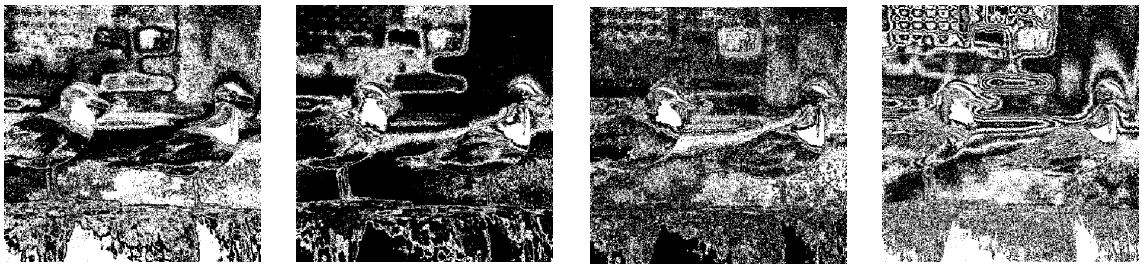


Figura 1.3.1 - Imagem do arquivo seagull.png.



Plano de bit 0 | Plano de bit 1 | Plano de bit 2 | Plano de bit 3

Plano de bit 4 | Plano de bit 5 | Plano de bit 6 | Plano de bit 7

Figura 1.3.2 – A começar pela primeira linha, da esquerda para a direita: seagull_bit_plane_0.png, seagull_bit_plane_1.png, seagull_bit_plane_2.png, seagull_bit_plane_3.png, seagull_bit_plane_4.png, seagull_bit_plane_5.png, seagull_bit_plane_6.png e seagull_bit_plane_7.png.

É possível observar que conforme evoluímos nos planos de bits da imagem, mais bem resumida se apresenta a informação visual. De fato, os planos de bits menores guardam considerável quantidade de informação de textura fina da imagem, que geralmente é composta por ruídos. Os planos de maior valor correspondem aos algarismos mais significativos da intensidade armazenada em cada pixel, portanto são os planos que contém as informações mais relevantes – do ponto de vista da representação – para a imagem.

Uma aplicação interessante para a separação e extração dos planos de bits diz respeito à utilização dos planos inferiores para se armazenar informações outras além da imagem por esteganografia. Por serem referentes aos dígitos menos significativos de intensidade, alterações nesses planos são quase sempre imperceptíveis. Pode-se utilizar esse espaço da imagem, então, para se codificar, por exemplo, mensagens cuja transmissão pode permanecer em segredo. Outra aplicação advém de técnicas de controle de transmissão de cópias de arquivos de vídeo, em que se pode gravar verificadores nesses planos em diferentes quadros[7].

[7] É importante salientar que, para que essas técnicas funcionem adequadamente, o arquivo não pode, em princípio, ser reprocessado, sob pena de se descartar as informações contidas nesses

Observações sobre código

Para a extração dos planos de bits, empregou-se a função `bit_plane`, que além de receber a imagem, recebe o valor do plano que se deseja extrair. De posse do valor do plano, compara-se, por vetorização, cada pixel com o valor de 2^k, em que k é o plano escolhido. A comparação é feita por operação *bitwise*, ou seja, compara-se esses valores bit por bit. Como desejamos saber os pixels em que há informação no plano k, utilizamos o operador &, que executa o teste lógico AND entre dois valores. Os valores iguais a 1 recebem 255 na imagem de saída, portanto são representados pela cor branca, e os valores iguais a 0 recebem 0, portanto, pretos na imagem final.

1.4. Mosaico

Este exercício solicita a reorganização da imagem, fatiada em 16 blocos quadrados, segundo uma configuração apresentada.

Para que se demonstre o funcionamento do código, utilizaremos outras imagens além da apresentada no enunciado, porém, todas elas estão presentes no arquivo .zip.

planos. Outra questão diz respeito ao fato de o receptor da mensagem precisar, naturalmente, possuir o devido algoritmo para decodificar a informação armazenada nesses planos inferiores.

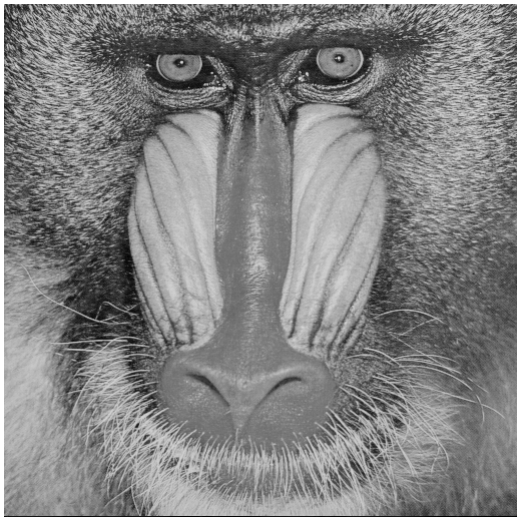
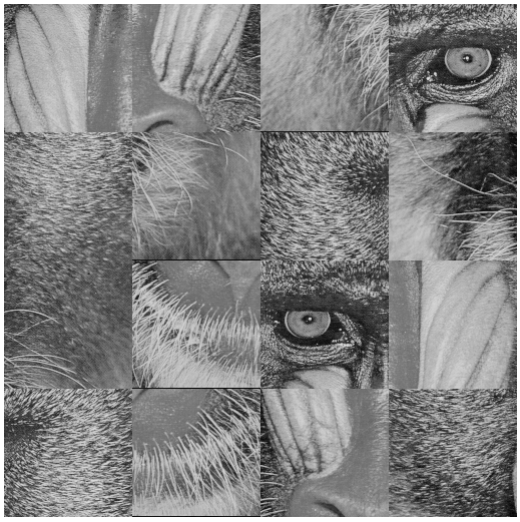



Figura 1.4.1 - Imagens baboon.png (esquerda) e baboon_mosaic.png (direita).

De fato, a imagem resultante corresponde à do enunciado.

Vejamos resultados para outras imagens:


Figura 1.4.2 - Imagens city.png (esquerda) e city_mosaic.png (direita).


Figura 1.4.3 - Imagens flower_bnw.png (esquerda) e flower_bnw mosaic.png (direita).

Embora não seja trivial a observação, a ordem de embaralhamento aplicada em cada exemplo é a mesma do enunciado.

Observações sobre o código

Para a execução desse algoritmo, utilizou-se diversas técnicas de *slicing*. Uma descrição mais detalhada encontra-se no próprio código comentado, de modo que aqui nos ateremos a comentar o algoritmo de forma mais geral. A ideia utilizada consistiu em se receber como input não apenas a imagem a ser reordenada, mas também a configuração a ser utilizada na reordenação dos blocos. A vantagem dessa abordagem, embora não fosse necessária a execução objetiva do exercício, é a possibilidade de se empregar essa mesma função, `mosaic`, em imagens de diferentes tamanhos para diferentes tamanhos de blocos, inclusive com blocos e imagens retangulares (em vez de quadradas). Caso a função não receba informação para a configuração, aplicará aquela do enunciado, com 16 blocos.

Provavelmente, há maneira mais eficientes e elegantes de se resolver esse problema, porém o modo presente no código funciona e possui a flexibilidade apontada.

Caso a quantidade de blocos informada à função não se encaixe perfeitamente às dimensões da imagem, haverá um *padding* preto nas bordas direita e inferior.

Para testar a função, escrevi uma pequena função auxiliar, `random_grid`, que recebe as dimensões da malha de blocos que se deseja aplicar em `mosaic` e retorna uma malha segundo essas dimensões, porém em ordem aleatória.

A seguir, alguns resultados bem interessantes para diversas imagens com diversos parâmetros em `random_grid`, que informa `mosaic`, para a imagem base clouds_2.png.

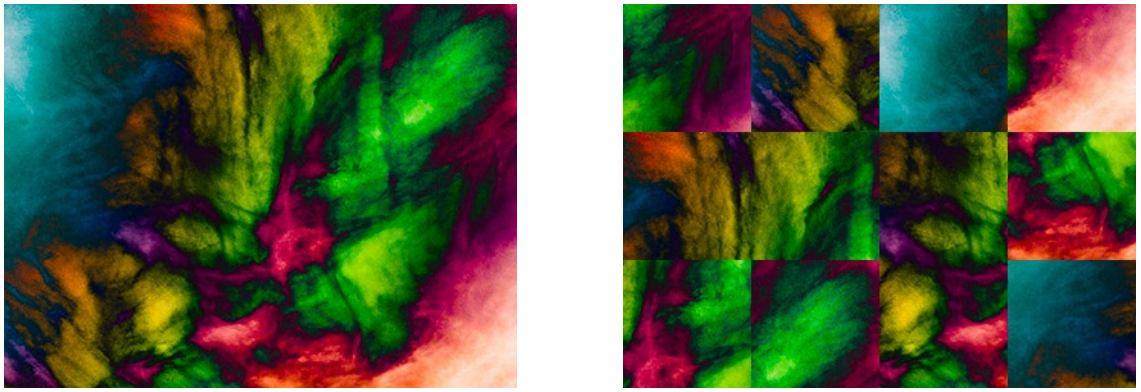


Figura 1.4.4 - Imagens clouds.png (esquerda) e clouds_2_mosaic.png (direita) (malha 3x4).

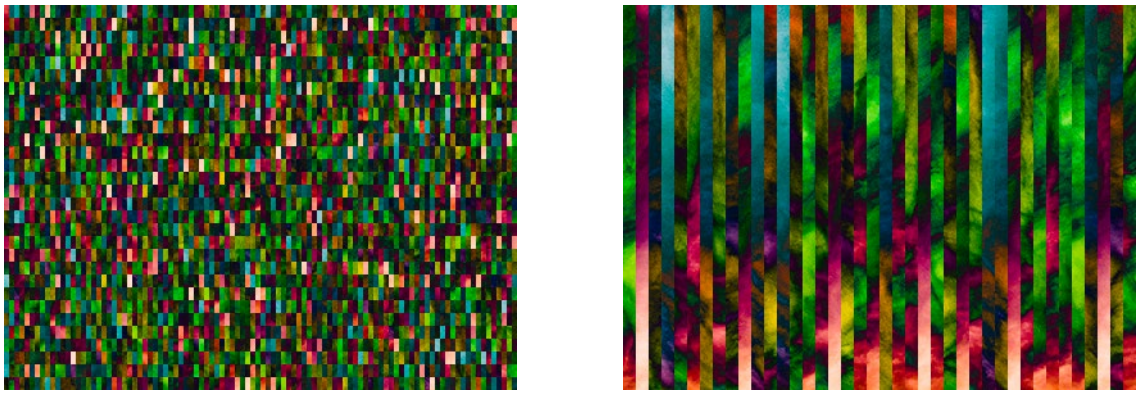


Figura 1.4.5 – Mais resultados para clouds_2_mosaic.png com diferentes configurações.

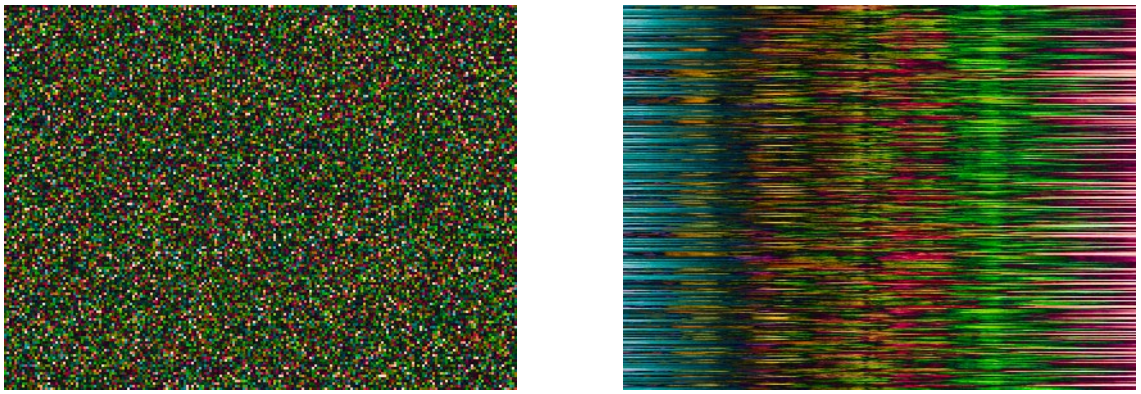


Figura 1.4.6 – Mais resultados para clouds_2_mosaic.png com diferentes configurações.

Observo que o fato de a imagem possuir cores é irrelevante para esse procedimento, uma vez que a operação de embaralhamento não lida com os canais dentro de cada pixel na matriz que representa a imagem.

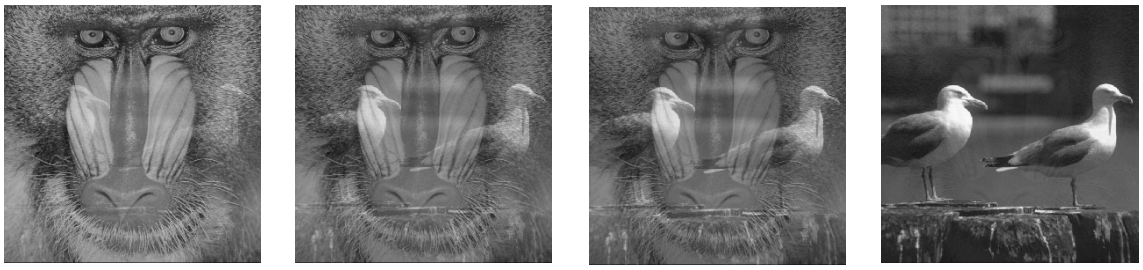
1.5. Combinação de imagens

O exercício pede que se opere uma média ponderada entre duas imagens. A função que realiza esse procedimento aqui é `blend_images`, que recebe duas imagens como parâmetro e o peso da primeira imagem em porcentagem. O peso da segunda imagem é o complemento do peso da primeira em relação a 100.

O resultado esperado é a sobreposição de imagens com relativa transparência, a depender do parâmetro peso escolhido. Em fotografia, a combinação de imagens dessa forma é comumente chamada de "dupla exposição"[8], remontando as técnicas de se expor mais de uma vez o negativo, quando de uso de fotografia analógica.

[8] No caso de mais de uma exposição sobre o mesmo fotograma, fala-se em "múltipla exposição". No paradigma digital, algumas câmeras suportam múltiplas exposições no próprio corpo do equipamento, gravando tanto os quadros individuais quanto a combinação. O efeito de múltipla exposição também pode ser facilmente obtido pelo uso de pacotes como GIMP e Photoshop.

Abaixo, seguem alguns resultados para diferentes pares de imagens. O primeiro valor é o peso da primeira imagem.



80 e 20 | 60 e 40 | 50 e 50 | 10 e 90

Figura 1.5.1 – Diferentes combinações entre baboon.png e seagull.png.

100 e 0 | 67 e 33 | 33 e 67 | 0 e 100

Figura 1.5.2 – Diferentes combinações entre flower_bnw.png e flower_color.png.

Como esperado, as imagens se mesclam de acordo com o nível de transparência (peso) determinado na chamada da função.

É interessante observar o resultado entre a combinação de flower_bnw.png e flower_color.png[9]. O modo como o procedimento foi implementado comporta imagens coloridas. Conforme o peso de imagem monocromática se reduz – e, consequentemente, o peso da colorida aumenta – o resultado é a incorporação da cor à imagem final, partindo de uma saturação nula a uma imagem totalmente colorida.

[9] Essas imagens possuem a mesma matriz fotográfica digital, diferindo apenas no tratamento das cores.

Observações sobre o código

O código da função `blend_images` é relativamente simples, consistindo na multiplicação da primeira imagem pelo peso informado e da segunda, pelo complemento em relação a 100. É possível passar valores decimais, como 37.5. O código pressupõe que as imagens escolhidas possuem as mesmas dimensões.

1.6 Filtragem de imagens

O exercício 1.6 apresenta diversos filtros e solicita a aplicação e a interpretação dos resultados obtidos. A aplicação dos filtros consiste em uma operação de convolução, um produto de Hadamard entre uma janela da imagem e uma máscara, com a subsequente soma dos elementos da matriz resultante e a atribuição do valor obtido ao pixel central.

A biblioteca OpenCV possui a função `filter2D`, que executa convoluções sobre uma imagem segundo uma máscara informada. As filtragens executadas para a resolução deste exercício foram feitas com essa função. Uma importante escolha para a convolução é o tratamento dos pixels de borda, visto que esses não possuem todos os vizinhos necessários para o cálculo da convolução. No contexto presente, optou-se por utilizar a reflexão da imagem em relação aos limites do arquivo. A escolha foi feita pensando-se aplicações em fotografia, que, embora não seja regra, geralmente fazem sentido ao se supor alguma continuidade homogênea da informação.

Vejamos a seguir, cada um dos filtros e seus resultados ao serem aplicados à imagem butterfly.png.

Filtro h_1 – Laplaciano do Gaussiano

O filtro h_1 possui a seguinte equação:

$$h_1 = \begin{pmatrix} 0 & 0 & -1 & 0 & 0 \\ 0 & -1 & -2 & -1 & 0 \\ -1 & -2 & 16 & -2 & -1 \\ 0 & -1 & -2 & -1 & 0 \\ 0 & 0 & -1 & 0 & 0 \end{pmatrix}$$

Os coeficientes da máscara possuem soma igual a zero. Disso podemos deduzir que se todos os pixels no cálculo forem iguais ou tiverem valores muito próximos, o resultado será igual ou próximo de zero, portanto, na imagem final, preto. A magnitude dos pesos diminui conforme os pixels envolvidos no cálculo se afastam do pixel central. Com exceção do pixel central, todos os demais coeficientes não nulos atribuem valores negativos aos pixels satélites. A análise sugere que se trata de um filtro de realce, uma vez que, ao adotar sinais diferentes para os pixels ao redor do centro da máscara, "força", exacerba, a diferença entre esses valores na imagem original.

A seguir, vemos o efeito de h_1 em butterfly.png.

Figura 1.6.1 - Imagens butterfly.png (esquerda) e butterfly_h1_convolution.png (direita).

A imagem resultante realça os contornos, ou seja, as transições tonais, presentes na imagem original. É possível observar também que o realce é mais forte quando as transições são mais abruptas, como nas linhas que desenham os padrões nas asas, bem como nas antenas da borboleta. Por se tratar de um filtro que destaca as transições tonais proporcionalmente à frequência espacial com que aparecem na imagem de entrada, trata-se de um exemplo de um filtro passa-alta, pois transmite as frequências mais altas e filtra as mais baixas. Esse tipo de

filtro é útil na detecção de limites e bordas dentro da imagem, podendo também ser utilizado para aumentar a nitidez quando devidamente combinado com a imagem de origem (por exemplo, com o uso de diferentes modos de mesclagem).

O filtro h_1 é um exemplo do Laplaciano do Gaussiano. Trata-se de uma combinação entre o filtro Laplaciano, um passa-alta, com o Gaussiano, um filtro passa-baixa. A ideia consiste em se atenuar o efeito do realce de ruídos aplicando-se um filtro Gaussiano e, depois, o Laplaciano, para se realçar as frequências altas que não fazem parte da textura dos ruídos.

Filtro h_2 – Gaussiano

O filtro h_2 tem a definição como segue:

$$h_2 = \begin{pmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{pmatrix} \frac{1}{256}.$$

Diferentemente do filtro anterior, o valor da soma dos coeficientes, ponderados pela fração $\frac{1}{256}$, é 1, o que significa que haverá preservação do nível médio de claridade na região de aplicação da máscara.

É possível ver que a operação realizada pela convolução de h_2 é uma média ponderada dos pixels que circundam o pixel central. Quanto maior a distância do centro da máscara, menor o peso do pixel no cálculo.

Vejamos o resultado em h_2 em butterfly.png:


Figura 1.6.2 - Imagens butterfly.png (esquerda) e butterfly_h2_convolution.png (direita).

De fato é possível notar que a imagem resultante é uma versão ligeiramente "borrada" da original. Esse efeito é consistente com a ideia de filtro passa-baixa – caso do Gaussiano – que filtra as altas frequências e transmite as baixas. Quanto maiores forem em coeficientes dos pixels ao redor do centro da máscara, espera-se que maior seja o efeito de desfoque aparente[10].

Filtro h_3 – Operador de Sobel, limites verticais

Eis a equação para o filtro h_3:

$$h_3 = \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix}.$$

Os coeficientes de h_3 somam zero, portanto podemos inferir que se trata de um filtro de realce. A orientação da matriz sugere que o filtro é indiferente a variações nas frequências na direção horizontal. Em contrapartida, na vertical

[10] Em termos de uma função Gaussiana, significa se empregar um valor σ mais elevado, portanto, uma superfície com decaimento mais suave.

há grande reforço das diferenças tonais. Por ser um filtro pequeno, seu custo computacional tende a ser reduzido quando comparado com filtros maiores.

Segue o resultado da convolução de h_3 com butterfly.png:

Figura 1.6.3 - Imagens butterfly.png (esquerda) e butterfly_h3_convolution.png (direita).

Ficam evidentes, após a operação, que h_3 reforça as linhas de orientação vertical. Esse filtro é um tipo de operador gradiente, especificamente, um operador de Sobel para a componente horizontal do gradiente da imagem (portanto para detecção de limites verticais).

Outro aspecto do filtro como mostrado é a orientação: o filtro rebaixa os valores à esquerda e realça os valores à direita, traduzindo uma orientação no sentido positivo da imagem[11].

Filtro h_4 – Operador de Sobel, horizontais

O filtro h_4 é apresentado como segue:

[11] A convenção adotada aqui é a de que os valores aumentam na matriz da imagem da esquerda para direita e de cima para baixo, como é comum na maioria dos textos aplicações em processamento de imagem.

$$h_4 = \begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{pmatrix}.$$

É semelhante a h_3, porém com orientação diversa. De fato, é claro que h_4 é h_3 após uma rotação de 90° em sentido horário na posição dos seus elementos. Isso significa que se trata, igualmente, de um filtro passa-alta, porém de detecção de bordas horizontais em vez de verticais.

Vejamos como fica o resultado de h_4 aplicado a butterfly.png.

Figura 1.6.4 - Imagens butterfly.png (esquerda) e butterfly_h4_convolution.png (direita).

A imagem resultante é o realce das bordas horizontais da imagem de entrada. Da mesma forma que o filtro anterior, h_4 possui orientação positiva em relação aos eixos da imagem.

O filtro em questão é um operador de Sobel, um operador gradiente para a direção vertical da imagem (portanto para a detecção de limites horizontais).

Filtro h_5 – Detecção de pontos isolados

O filtro h_5,

$$h_5 = \begin{pmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{pmatrix},$$

é um filtro de realce, um passa-alta. Isso vem do fato de a soma de seus coeficientes ser nula. Todos os pixels em torno do centro da máscara recebem o mesmo peso, independentemente da direção.

A seguir, o resultado de h_5 em butterfly.png:

Figura 1.6.5 - Imagens butterfly.png (esquerda) e butterfly_h5_convolution.png (direita).

O resultado se assemelha do de h_1, com realces nas regiões de transições tonais. Trata-se de um filtro de detecção de pontos isolados. O nome condiz com a análise da matriz, que atribui grande peso positivo ao elemento central e pesos negativos aos satélites sem privilegiar direções.

Filtro h_6 – Média aritmética

O filtro h_6 tem a seguinte equação:

$$h_6 = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix} \frac{1}{9}$$

A soma de seus coeficientes ponderados é 1. Trata-se de um passa-baixa. É assente que esse filtro realiza a média aritmética entre o pixel no centro da máscara e os seus vizinhos em vizinhança-8.

Eis o produto da aplicação de h_6 em butterfly.png:

Figura 1.6.6 - Imagens butterfly.png (esquerda) e butterfly_h6_convolution.png (direita).

É um resultado muito semelhante ao da aplicação de h_2, o filtro Gaussiano, porém ligeiramente menos borrado. Isso se deve ao fato de que aquele Gaussiano possuía 25 elementos em seu cômputo, enquanto este, média aritmética, utiliza apenas 9.

Filtro h_7 – Realce de diagonais decrescentes

O filtro h_7 é definido como

$$h_7 = \begin{pmatrix} -1 & -1 & 2 \\ -1 & 2 & -1 \\ 2 & -1 & -1 \end{pmatrix}$$

A soma de seus coeficientes é zero. Trata-se de um filtro de realce, um passa-alta. Os valores positivos encontram-se todos sobre a diagonal decrescente. Espera-se que o efeito produzido seja o de reforçar as linhas paralelas a essa direção.

O resultado da convolução de h_7 em butterfly.png está abaixo.

Figura 1.6.7 - Imagens butterfly.png (esquerda) e butterfly_h7_convolution.png (direita).

Vemos que o realce desta aplicação, diferentemente dos casos do operador de Sobel, não privilegia as direções verticais ou as horizontais, mas a diagonal em 135°. Pode ser aplicado como uma ferramenta de detecção de limites diagonais.

Filtro h_8 – Realce de diagonais crescentes

A máscara do filtro h_8

$$h_8 = \begin{pmatrix} 2 & -1 & -1 \\ -1 & 2 & -1 \\ -1 & -1 & 2 \end{pmatrix}$$

é semelhante a h_7, a menos de uma reflexão horizontal ou vertical. O efeito esperado, portanto, é também semelhante, a menos de uma diferença na orientação do realce.

Vejamos o desempenho de h_8 em butterfly.png:

Figura 1.6.8 - Imagens butterfly.png (esquerda) e butterfly_h8_convolution.png (direita).

A diferença é sutil, porém perceptível. A orientação de realce favorecida aqui é das linhas paralelas à direção da diagonal crescente no plano de pixels da imagem.

Filtro h_9 – Matriz identidade (motion blur)

Em h_9 temos

$$h_9 = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \frac{1}{9}$$

Trata-se de um filtro baseado na matriz identidade de dimensão nove, com coeficientes normalizados para que a sua soma seja igual a 1.

A convolução realizada pela aplicação de h_9 é uma espécie de média dos valores da imagem, porém privilegiando exclusivamente uma direção específica, a saber a da diagonal principal.

Vejamos o que sucede da aplicação de h_9 no nosso arquivo de testes butterfly.png.

Figura 1.6.9 - Imagens butterfly.png (esquerda) e butterfly_h9_convolution.png (direita).

Realmente, a imagem resultante apresenta um nível de "desfoque", porém sob a orientação da diagonal da máscara. É muito semelhante ao efeito que se obtém em fotografia quando se move a câmera em uma direção específica durante a exposição[12], causando um "motion blur".

Filtro h_{10} – Nitidez

O filtro h_{10} tem a seguinte apresentação:

[12] A despeito disso, o efeito não é idêntico, pois, na captura fotográfica, a intensidade do borrão nos elementos do quadro é inversamente proporcional à distância que estão da câmera.

$$h_{10} = \begin{pmatrix} -1 & -1 & -1 & -1 & -1 \\ -1 & 2 & 2 & 2 & -1 \\ -1 & 2 & 8 & 2 & -1 \\ -1 & 2 & 2 & 2 & -1 \\ -1 & -1 & -1 & -1 & -1 \end{pmatrix} \frac{1}{8}$$

É esperado algum tipo de realce, uma vez que a máscara possui diferentes sinais para os coeficientes. O filtro não privilegia direções, visto que nenhuma possui diferença em relação às demais na distribuição dos coeficientes. Fica evidente também que o filtro combina elementos de passa-baixa – pelos coeficientes positivos nos nove elementos centrais e pelo fator $\frac{1}{8}$ – e de passa-alta – pelos coeficientes negativos na orla da matriz.

O efeito de h_{10} em butterfly.png é como segue:

Figura 1.6.10 - Imagens butterfly.png (esquerda) e butterfly_h10_convolution.png (direita).

Claramente, houve um ganho de nitidez após a convolução. O efeito é, de alguma forma, o inverso do obtido com o filtro Gaussiano, que reduz a nitidez da imagem sobre a qual é aplicado. Junto com o ganho de nitidez, é possível observar um ganho de textura inclusive nas regiões de baixa frequência, proveniente do realce sobre ruído.

Filtro h_{11} – Bordas diagonais

A equação de h_{11},

$$h_{11} = \begin{pmatrix} -1 & -1 & 0 \\ -1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix},$$

se assemelha à forma dos operadores de Sobel, porém em relação à direção diagonal de 45° (e sem o peso maior para os valores de vizinhança-4 do elemento central). Trata-se de um filtro de realce.

Vemos o resultado da convolução entra h_{11} e butterfly.png a seguir:

Figura 1.6.11 - Imagens butterfly.png (esquerda) e butterfly_h11_convolution.png (direita).

Como esperado, o aspecto é bastante próximo das imagens dos operadores de Sobel. A alteração da direção de detecção das bordas fica clara se notarmos que a antena direita da borboleta – praticamente paralela à diagonal de 45° - está fortemente realçada, enquanto a antena da esquerda – praticamente perpendicular à diagonal de 45° – está quase oculta na imagem resultante.

Observações sobre o código

Todos os filtros aplicados no exercício 1.6 foram implementados pela função `convolution`. Esta recebe como argumento uma imagem e um parâmetro de 1 a 11 (inclusive), indicando o filtro desejado dentre os apresentados no enunciado.

Como mencionado anteriormente, a função `convolution` se baseia na função `filter2D` da biblioteca OpenCV e salva diretamente a imagem resultante no mesmo diretório da imagem original.

Combinação de h_3 e h_4 – a magnitude do gradiente

Como mencionamos anteriormente, os filtros h_3 e h_4 são conhecidos como operadores de gradiente. Suas definições se baseiam na do gradiente de uma função de mais de uma variável conforme as seguintes equações:

$$\nabla \mathbf{f}(x; y) = \begin{bmatrix} G_x \\ G_y \end{bmatrix} = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{y} \end{bmatrix}.$$

Em outros termos, os operadores de Sobel calculam o equivalente das derivadas parciais da intensidade em cada pixel do plano da imagem:

$$G_x = \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix} \text{ e } G_y = \begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{pmatrix}.$$

Do mesmo modo que podemos calcular a magnitude do gradiente de uma função contínua de duas variáveis, é possível calcular a magnitude do gradiente de uma imagem aplicando-se os operadores G_x e G_y acima, somando-se o quadrado das suas contribuições e extraindo a raiz quadrada do resultado.

Colocando sob a forma de equações, se ${G_x}^f$ é o resultado da aplicação de G_x sobre uma imagem f e ${G_y}^f$ a aplicação de G_y sobre f, a magnitude do gradiente será dada por

$$\nabla f(x,y) = \sqrt{\left({G_x}^f\right)^2 + \left({G_y}^f\right)^2}$$

A função `grad_mag` realiza o cálculo acima a partir das convoluções (calculadas internamente) de h_3 e h_4.

Abaixo, vemos os resultados das convoluções de h_3, h_4 e a magnitude do gradiente.

Figura 1.6.12 – Respectivamente, butterfly_h3_.png, butterfly_h4_png e butterfly_gradient_magnitude.png

É evidente o efeito que a combinação das convoluções entrega. As bordas realçadas na terceira imagem são a combinação das realçadas nas direções verticais e horizontais. Outro aspecto importante é o fato de as bordas em butterfly_gradient_magnitude.png serem todas positivas, pois surgem de uma operação da fórmula do Teorema de Pitágoras.

Bibliografia

H. Pedrini, W.R. Schwartz. *Análise de Imagens Digitais: Princípios, Algoritmos e Aplicações.* Editora Thomson Learning, 2007.

R.C. Gonzalez, R.E. Woods. *Digital Image Processing*. Prentice Hall, 2007.