PSI-3432 — Processamento de Áudio e Imagem
Exercício Computacional 4
Filtragem de Imagem no Domínio da Frequência

Vítor H. Nascimento Thiago Y. Aoyagi

Como na filtragem de um sinal unidimensional, filtragem de imagens pode ser feita tanto no domínio espacial quanto no domínio da frequência. Podemos fazer um mesmo procedimento em qualquer um dos dois domínios, pois, pela propriedade da multiplicação na frequência (ou da convolução no espaço) da TDF, há uma operação equivalente no outro domínio. Porém, dependendo do objetivo, é mais fácil trabalhar em um domínio específico.

Quando se deseja uma filtragem com uma máscara de convolução simples, como uma suavização ou um detector de bordas convencional, é mais conveniente aplicar o filtro diretamente nos pixels da imagem. Essa abordagem também é desejada quando se faz um processamento em tempo real, pois permite utilizar máscaras pequenas e assim economizar processamento.

Existem ocasiões, porém, em que as características da imagem ficam mais nítidas no domínio da frequência, e fica mais prático filtrar diretamente no domínio transformado. Neste exercício, abordamos duas experiências que retratam esses casos: uma filtragem de interferência e um detector de bordas mais flexível.

1 Roteiro Geral para Filtragem em Frequência

Filtrar no domínio da frequência é multiplicar o espectro do imagem pela função de transferência desejada[1]. Considere a imagem $x[n_1,n_2]$ para $0 \leq n_1 \leq M_1 - 1$ e $0 \leq n_2 \leq M_2 - 1$. Seja $X[k_1,k_2]$ para $0 \leq k_1 \leq N_1 - 1$ e $0 \leq n_2 \leq N_2 - 1$ sua TDF-2D ($N_1 \geq M_1$ e $N_2 \geq M_2$, pois permite o uso de zero-padding). Seja $H[k_1,k_2]$ a função de transferência que se deseja aplicar na imagem.

Imagens naturais costumam ter espectro concentrado na origem ($k_1 = 0, k_2 = 0$). Por isso, é mais conveniente exibir o espectro com a origem no centro da imagem. Esse deslocamento pode ser feito de maneira simples pela propriedade do deslocamento em frequência: para obtermos $X[k_1 - N_1/2, k_2 - N_2/2]$, basta multiplicarmos a imagem $x[n_1,n_2]$ por:

$$\mathrm{e}^{j\frac{2\pi}{N_1}n_1\frac{N_1}{2}}\mathrm{e}^{j\frac{2\pi}{N_2}n_2\frac{N_2}{2}}x[n_1,n_2] = (\mathrm{e}^{j\pi})^{n_1+n_2}x[n_1,n_2] = (-1)^{n_1+n_2}x[n_1,n_2]$$

Assim, o roteiro geral da filtragem em frequência é:

[1]Este procedimento é equivalente a fazer a convolução circular da imagem pelo filtro no domínio espacial. Como no caso 1D, a convolução circular causa efeitos indesejados, principalmente nas bordas do sinal (repare!). Você pode melhorar a qualidade do resultado filtrando o sinal no tempo, calculando a TDF inversa de $H[k_1, k_2]$, e usando `filt2` (no Matlab) ou `imfilt` (em Julia) para aplicar o filtro — os melhores resultados são obtidos extendendo a imagem como uma reflexão nas bordas (em Julia isso é obtido usando o comando `imfilt(X, h, "reflect")`.

1. multiplicar a imagem por $(-1)^{n_1+n_2}$;
2. calcular a TDF-2D da imagem;
3. multiplicar a TDF-2D da imagem pela função de transferência desejada;
4. calcular a TDF-2D inversa do espectro filtrado;
5. tomar a parte real da imagem resultante e multiplicar por $(-1)^{n_1+n_2}$ (para desfazer o deslocamento em frequência).

Algumas dicas práticas de processamento de imagens, e sugestões de funções do Matlab e de Julia, são dadas no Apêndice. Em Julia, o procedimento acima pode ser feito também usando o pacote `FFTViews`, como visto em aula.

2 Filtragem de Interferência

A imagem `cassini-interference.tif`, disponibilizada no Moodle, mostra uma parte dos anéis de Saturno. Ela foi capturada por *Cassini*, o primeiro satélite humano a entrar em órbita no planeta. O padrão horizontal na imagem, aproximadamente senoidal, foi causada por uma interferência eletromagnética, que foi sobreposta ao sinal de vídeo e que corrompeu muitas imagens da missão espacial. Felizmente, este tipo de interferência pode ser facilmente corrigida com processamento de imagem.

1. Calcule a TDF-2D (com o mesmo tamanho da imagem), centralizada na origem, e visualize o módulo do espectro em escala log (sendo `X` o espectro obtido, visualize, por exemplo, com `imagesc(log(abs(X)+1))`). Em Julia, é necessário normalizar os valores entre 0 e 1: use `Gray.(log.(abs.(X) .+ 1) ./ maximum(log.(abs.(X) .+ 1)))`.
2. Qual é a região espectral que corresponde à interferência? Justifique.

   (Dica: qual é a interpretação espacial de um ponto da TDF-2D, junto com seu conjugado simétrico?)
3. Para remover a interferência, você irá projetar um filtro notch simples: $H[k_1,k_2] = 0$ para a região que se deseja eliminar, e $H[k_1,k_2] = 1$ para os demais pontos. Projete um filtro notch com rejeição de 11 pixels de largura ao longo da faixa de interferência, mas mantendo uma faixa de passagem de largura de 11 pixels em torno da frequência DC.

   (lembre que para um filtro ser de fase nula, $H[k_1,k_2]$ deve ser simétrico em relação à raia correspondente à frequência DC)
4. Aplique o filtro no espectro da imagem obtido no item (a) e obtenha a imagem no domínio espacial. Ainda é possível observar a interferência?

3 Apêndice: dicas práticas de processamento de imagem

Os valores que codificam um nível de cinza de um pixel em uma imagem são discretos e restritos a um intervalo. Esse intervalo depende do tipo da variável que usamos para

representar a intensidade do pixel. No Matlab (e também vale de maneira semelhante para várias outras linguagens):

- quando a imagem é do tipo `uint8` (unsigned integer de 8 bits), os níveis de cinza ocupam todo o intervalo de valores, ou seja, inteiros de 0 a 255;
- quando a imagem é do tipo `double`, os níveis de preto a branco são codificados por valores flutuantes de 0 a 1.

As funções do Matlab para ler e salvar arquivos de imagens (`imread` e `imwrite`) e a função `imshow`, que exibe a imagem no próprio ambiente do Matlab, seguem esse padrão de codificação de nível de cinza.

Quando processamos uma imagem, porém, tratamos os valores dos pixels como um número real e realizamos operações aritméticas dentro desse conjunto. Por isso, recomenda-se trabalhar com o tipo `double`, já que com `uint8` os valores ficam sujeitos a arrodondamentos e saturação. Use as funções `double` e `uint8` para converter de um tipo para o outro.

As funções `fft2` e `ifft2` podem ser usadas para calcular a TDF-2D e sua inversa, respectivamente. Note que os valores da transformada normalmente passam para fora do intervalo de codificação de níveis de cinza, mesmo que a imagem esteja dentro desses limites. Uma forma prática para visualizar o espectro, sem precisar alterar sua escala, é usar a função `imagesc`, que exibe uma imagem com escala de cor ajustada automaticamente aos valores extremos da imagem.

Em Julia o funcionamento é diferente: imagens são matrizes de pixels, que podem ser de vários tipos (`RGB`, `Gray`, etc). Um pixel do tipo `Gray`, por exemplo, contém um número entre 0 e 1, que pode ser um número em ponto fixo (normalizado para 0x00 representar 0.0 e 0xff representar 1.0), então os problemas de conversão são bem menores (repare que um número em ponto fixo é um inteiro interpretado como um número no intervalo $[0, 1]$).

A função `fft` do pacote `FFTW` de Julia já computa a FFT multidimensional (não há uma função separada). Além disso, Julia também tem o pacote `FFTViews`, que permite que se troque o centro de uma imagem sem a necessidade de multiplicações ou transposições.

A posição dos pixels na imagem em Julia pode ser inferida a partir do código a seguir:

```
using Images, Plots
exemplo = [RGB(0,0,0) for i=1:5, j=1:5]
exemplo[3,3] = RGB(0,1,0) # verde
exemplo[1,1] = RGB(1,0,0) # vermelho
exemplo[5,1] = RGB(0,0,1) # azul
exemplo[1,5] = RGB(1,1,0) # amarelo
exemplo[5,5] = RGB(1,0,1) # rosa
plot(exemplo, grid=false, axis=false)
```

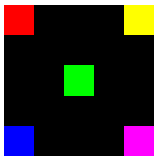

Figure 1: Posição (1,1): vermelho; posição (5,1): azul; posição (1,5): amarelo; posição (5,5): rosa; posição (3,3): verde.