Primeiro Trabalho de Implementação Visão Computacional

|  |  |  |
| --- | --- | --- |
| Ícaro Goulart Faria Motta França | Semestre 01/17 |  |

# Problema

Implementar sistemas LIT que modelem o borramento Gaussiano, o borramento por movimento, e o desfocamento:

## Introdução

Neste trabalho usamos 2 funções nativas do Matlab para realizarmos as transformadas de Fourier e a Transformada Inversa de Fourier.

A função fft2 (2-D Fast Fourier Transform) do Matlab foi usada para fazer a Transformada de Fourier (F) das imagens e para gerar a Função de Transferência de Modulação (H), fazendo a Transformada de Fourier da Função de Espalhamento do Ponto (h).

A função ifft2 (2-D Inverse Fast Fourier Transform) do Matlab foi usada para fazer a Transformada de Fourier Inversa (f) da frequência da imagem.

Cada filtro produzido neste trabalho possui seu próprio código fonte. Com isso, grande parte do código foi reutizado. A diferença de cada filtro se resume na aplicação da transformada de Fourier em uma matriz construída pela Função de Espalhamento do Ponto. Desta forma, obtemos a Função de Transferência de Modulação (H).

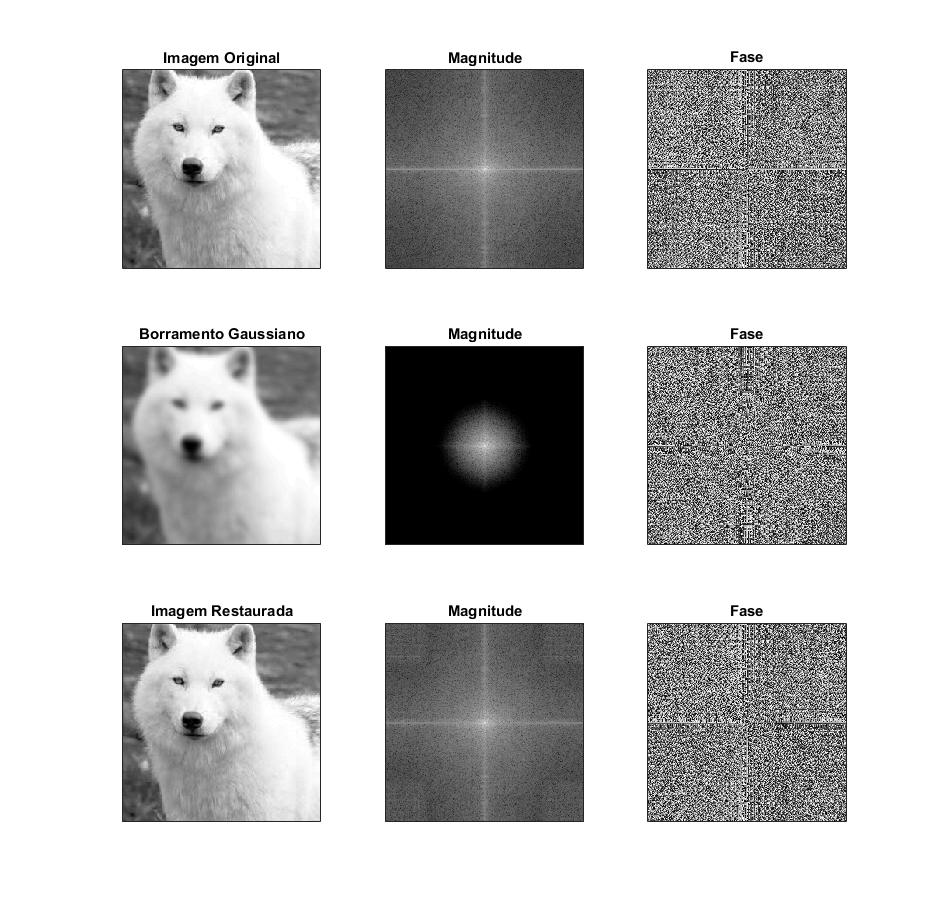
A imagem resultante do sistema LIT é obtida por meio da multiplicação da transformada da imagem original pela Função de Transferência de Modulação.

A recuperação da imagem resultante do sistema LIT é recuperada multiplicando a transformada de Fourier da imagem resultante pelo inverso da função de Transferência de Modulação (1/H) e depois fazendo a transformada Inversa de Fourier no resultado obtido no primeiro passo.

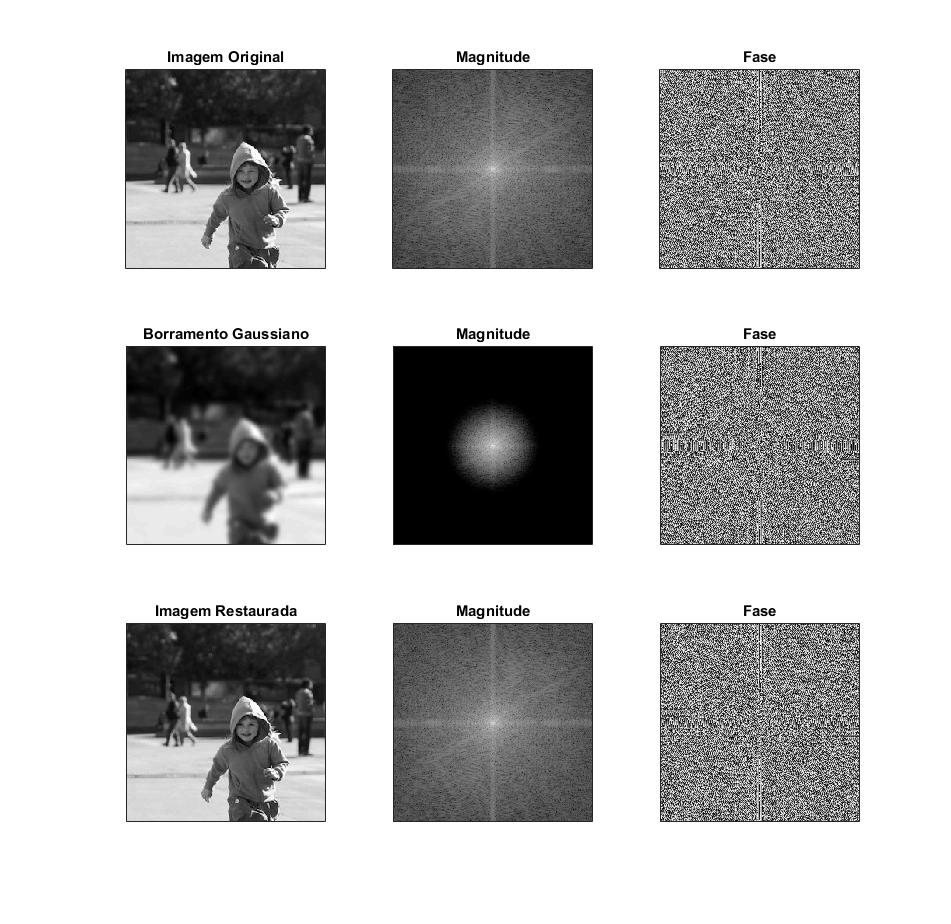
As imagens de entrada e saída foram colocadas na sessão correspondente a cada filtro. Também foram agrupadas numa tabela de 3 linhas e 3 colunas, em que a primeira coluna refere-se ao domínio espacial, a segunda à magnitude da frequência, e a terceira à fase da frequência.

## Borramento Gaussiano

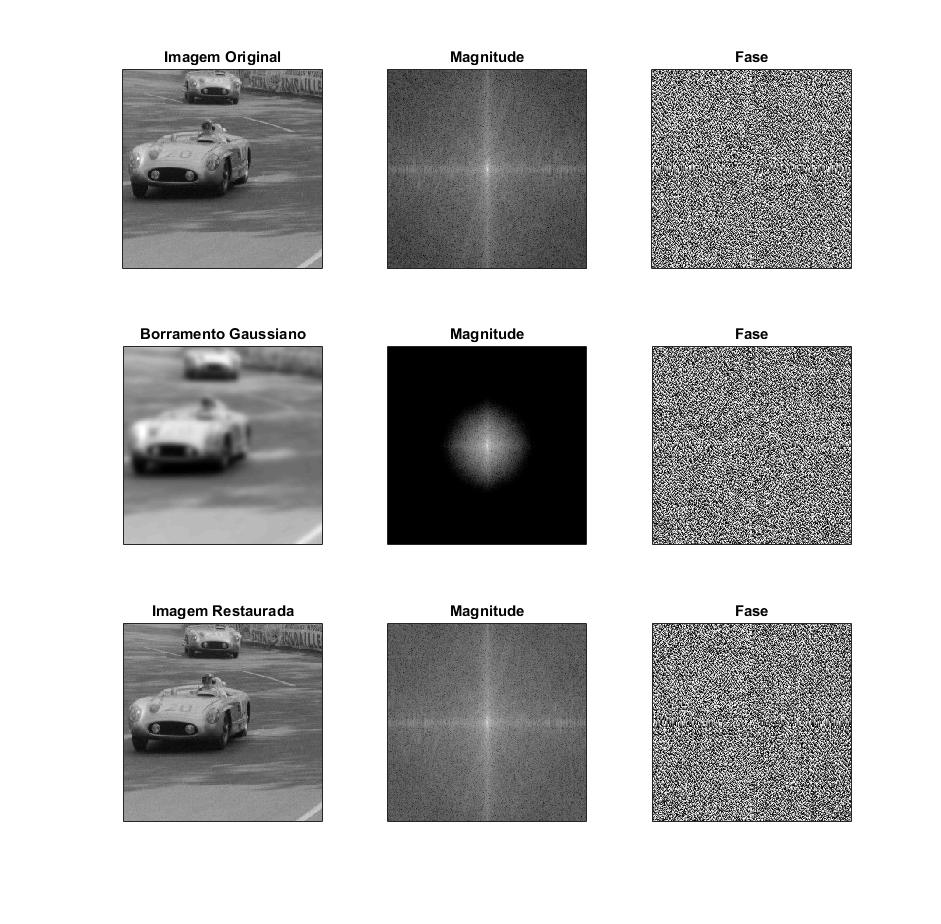
### Primeira Imagem: cachorro



### Segunda Imagem: criança



### Terceira Imagem: Ferrari



### Código:

%Lê a imagem original do arquivo

originalImage = imread('cachorro.jpg');

%Recupera só um canal da imagem caso a imagem não esteja em tons de cinza

%originalImage = originalImage(:, :, 1);

%configurando plot

figure;

subplot(3,3,1);

imagesc(originalImage);

axis square;

colormap gray;

title('Imagem Original');

set(gca, 'XTick', [], 'YTick', []);

kernelPSFSize = 31;

kernelPSF = zeros(kernelPSFSize);

sigma = 3;

m = kernelPSFSize/2;

[X, Y] = meshgrid(1:kernelPSFSize);

%Calculamos a função de espalhamento do ponto aqui

kernelPSF = (1/(2\*pi\*sigma^2)) \* exp(-((X-m).^2 + (Y-m).^2)/(2\*sigma^2));

%Adicionamos um pad na imagem para evitar problemas nas bordas

originalImagePad = padimage(originalImage, kernelPSFSize);

%recuperamos a altura e largura da imagem original

[originalHeight, originalWidth] = size(originalImage);

%recuperamos a altura e largura da imagem com pad

[height, width] = size(originalImagePad);

%criamos a imagem do kernel com mesma altura e largura da imagem original

kernelImage = zeros(height, width);

%adicionamos o kernelPSF no início do kernelImagem. Poderia ser colocado no

%meio também ou no final

kernelImage(1:kernelPSFSize, 1:kernelPSFSize) = kernelPSF;

%convertemos a imagem para o domínio da frequência

fftImage = fft2(originalImagePad);

%convertemos o kernel para o domínio da frequência

fftKernel = fft2(kernelImage);

%recurso técnico para poder recuperar a imagem. Senão pode ocorrer divisão

%por zero

fftKernel(find(fftKernel == 0)) = 1e-6;

%aplicamos a multiplicação no domínio da frequência para obtermos a imagem

%borrada no domínio da frequência

fftBlurImage = fftImage.\*fftKernel;

%desenhamos as imagens no domínio da frequência

subplot(3,3,2);

fftImageToShow = fftshift(fftImage);

fftImageToShow = abs(fftImageToShow);

fftImageToShow = log(fftImageToShow+1);

imagesc(fftImageToShow);

axis square;

colormap gray;

title('Magnitude');

set(gca, 'XTick', [], 'YTick', []);

subplot(3,3,3);

imagesc(angle(fftshift(fftImage)),[-pi pi]);

axis square;

colormap gray

title('Fase')

set(gca, 'XTick', [], 'YTick', []);

subplot(3,3,5);

fftBlurImageToShow = fftshift(fftBlurImage);

fftBlurImageToShow = abs(fftBlurImageToShow);

fftBlurImageToShow = log(fftBlurImageToShow+1);

imagesc(fftBlurImageToShow);

axis square;

colormap gray;

title('Magnitude');

set(gca, 'XTick', [], 'YTick', []);

subplot(3,3,6);

imagesc(angle(fftshift(fftBlurImage)),[-pi pi]);

axis square;

colormap gray

title('Fase')

set(gca, 'XTick', [], 'YTick', []);

%fazemos a Inverse Fast Fourier Transform para obter a imagem no domínio

%espacial

blurImage = ifft2(fftBlurImage());

%Removemos o pad adicionado no início

blurImageUnpad = blurImage(kernelPSFSize+16:kernelPSFSize+originalHeight+15.5,kernelPSFSize+16:kernelPSFSize+originalWidth+15.5);

%figure;

subplot(3,3,4);

imagesc(blurImageUnpad);

axis square;

title('Borramento Gaussiano');

colormap gray;

set(gca, 'XTick', [], 'YTick', []);

%Tentando recuperar a imagem degradada com a formula H' = 1/H

fftRecoverKernel = 1./fftKernel;

fftBlurImageToRecover = fft2(blurImage);

fftRecoverBlurImage = fftBlurImageToRecover.\*fftRecoverKernel;

recoverImage = ifft2(fftRecoverBlurImage);

recoverImageUnpad = recoverImage(kernelPSFSize+1:kernelPSFSize+originalHeight,kernelPSFSize+1:kernelPSFSize+originalWidth);

subplot(3,3,7);

imagesc(recoverImageUnpad);

axis square;

title('Imagem Restaurada');

colormap gray;

set(gca, 'XTick', [], 'YTick', []);

subplot(3,3,8);

fftRecoverBlurImageToShow = fftshift(fftRecoverBlurImage);

fftRecoverBlurImageToShow = abs(fftRecoverBlurImageToShow);

fftRecoverBlurImageToShow = log(fftRecoverBlurImageToShow+1);

imagesc(fftRecoverBlurImageToShow);

axis square;

colormap gray;

title('Magnitude');

set(gca, 'XTick', [], 'YTick', []);

subplot(3,3,9);

imagesc(angle(fftshift(fftRecoverBlurImage)),[-pi pi]);

axis square;

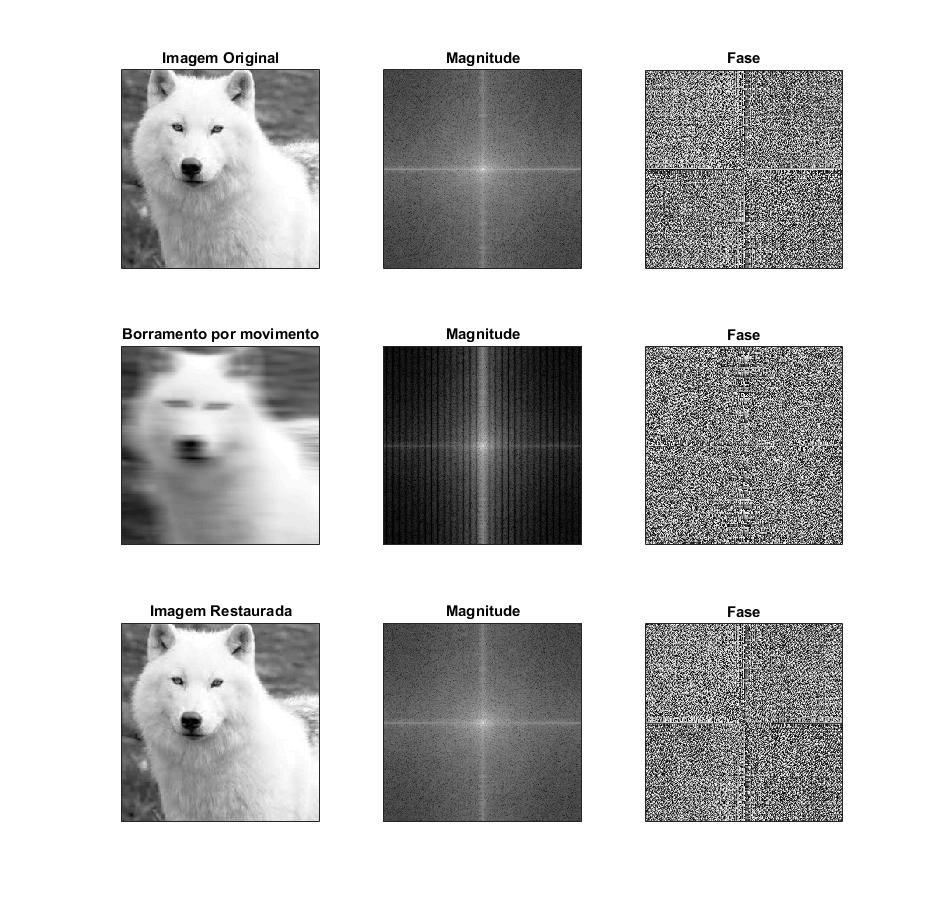
colormap gray

title('Fase')

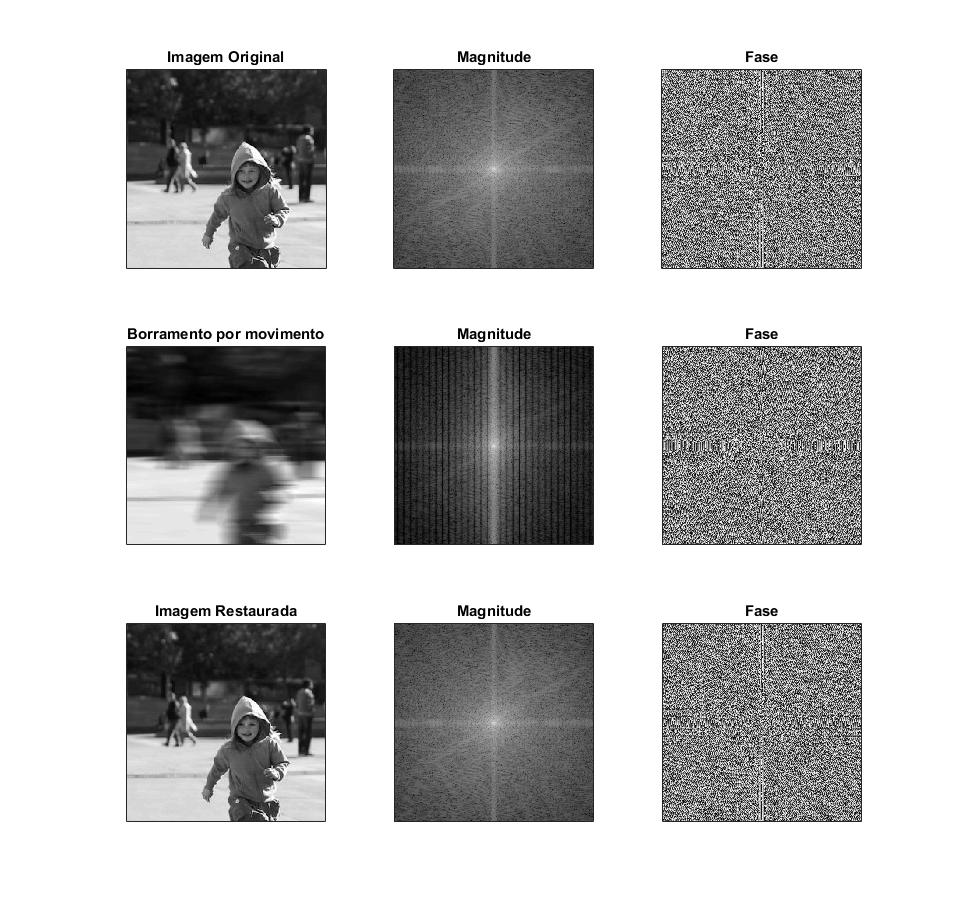
set(gca, 'XTick', [], 'YTick', []);

## Borramento por movimento

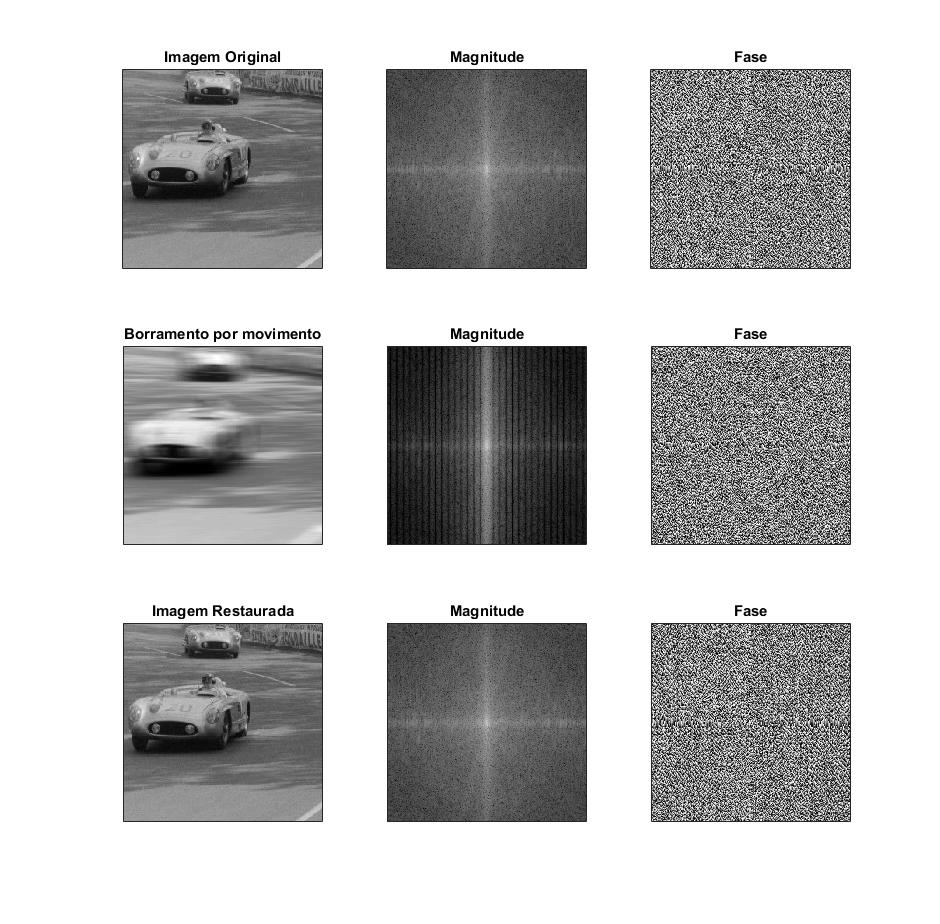
### Primeira Imagem: cachorro



### Segunda Imagem: criança



### Terceira Imagem: Ferrari



### Código:

%Lê a imagem original do arquivo

originalImage = imread('cachorro.jpg');

%Recupera só um canal da imagem caso a imagem não esteja em tons de cinza

%originalImage = originalImage(:, :, 1);

%configurando plot

figure;

subplot(3,3,1);

imagesc(originalImage);

axis square;

colormap gray;

title('Imagem Original');

set(gca, 'XTick', [], 'YTick', []);

kernelPSFSize = 31;

kernelPSF = zeros(kernelPSFSize);

lineLength = 30;

m = kernelPSFSize/2;

[X, Y] = meshgrid(1:kernelPSFSize);

%Calculamos a função de espalhamento do ponto aqui

deltaY = delta(Y, ceil(m));

kernelPSF = (1/(2\*lineLength)) \* (heaviside(X-m+lineLength) - heaviside(X-m-lineLength)).\*deltaY;

%Adicionamos um pad na imagem para evitar problemas nas bordas

originalImagePad = padimage(originalImage, kernelPSFSize);

%recuperamos a altura e largura da imagem original

[originalHeight, originalWidth] = size(originalImage);

%recuperamos a altura e largura da imagem com pad

[height, width] = size(originalImagePad);

%criamos a imagem do kernel com mesma altura e largura da imagem original

kernelImage = zeros(height, width);

%adicionamos o kernelPSF no início do kernelImagem. Poderia ser colocado no

%meio também ou no final

kernelImage(1:kernelPSFSize, 1:kernelPSFSize) = kernelPSF;

%convertemos a imagem para o domínio da frequência

fftImage = fft2(originalImagePad);

%convertemos o kernel para o domínio da frequência

fftKernel = fft2(kernelImage);

%recurso técnico para poder recuperar a imagem. Senão pode ocorrer divisão

%por zero

fftKernel(find(fftKernel == 0)) = 1e-6;

%aplicamos a multiplicação no domínio da frequência para obtermos a imagem

%borrada no domínio da frequência

fftBlurImage = fftImage.\*fftKernel;

%desenhamos as imagens no domínio da frequência

subplot(3,3,2);

fftImageTemp = fftshift(fftImage);

fftImageTemp = abs(fftImageTemp);

fftImageTemp = log(fftImageTemp+1);

imagesc(fftImageTemp);

axis square;

colormap gray;

title('Magnitude');

set(gca, 'XTick', [], 'YTick', []);

subplot(3,3,3);

imagesc(angle(fftshift(fftImage)),[-pi pi]);

axis square;

colormap gray

title('Fase')

set(gca, 'XTick', [], 'YTick', []);

subplot(3,3,5);

fftBlurImageToShow = fftshift(fftBlurImage);

fftBlurImageToShow = abs(fftBlurImageToShow);

fftBlurImageToShow = log(fftBlurImageToShow+1);

imagesc(fftBlurImageToShow);

axis square;

colormap gray;

title('Magnitude');

set(gca, 'XTick', [], 'YTick', []);

subplot(3,3,6);

imagesc(angle(fftshift(fftBlurImage)),[-pi pi]);

axis square;

colormap gray

title('Fase')

set(gca, 'XTick', [], 'YTick', []);

%fazemos a Inverse Fast Fourier Transform para obter a imagem no domínio

%espacial

blurImage = ifft2(fftBlurImage);

%Removemos o pad adicionado no início

blurImageUnpad = blurImage(kernelPSFSize+16:kernelPSFSize+originalHeight+15.5,kernelPSFSize+16:kernelPSFSize+originalWidth+15.5);

%figure;

subplot(3,3,4);

imagesc(blurImageUnpad);

axis square;

title('Borramento por movimento');

colormap gray;

set(gca, 'XTick', [], 'YTick', []);

%Tentando recuperar a imagem degradada

fftRecoverKernel = 1./fftKernel;

fftBlurImageToRecover = fft2(blurImage);

fftRecoverBlurImage = fftBlurImageToRecover.\*fftRecoverKernel;

recoverImage = ifft2(fftRecoverBlurImage);

recoverImageUnpad = recoverImage(kernelPSFSize+1:kernelPSFSize+originalHeight,kernelPSFSize+1:kernelPSFSize+originalWidth);

subplot(3,3,7);

imagesc(recoverImageUnpad);

axis square;

title('Imagem Restaurada');

colormap gray;

set(gca, 'XTick', [], 'YTick', []);

subplot(3,3,8);

fftRecoverBlurImageToShow = fftshift(fftRecoverBlurImage);

fftRecoverBlurImageToShow = abs(fftRecoverBlurImageToShow);

fftRecoverBlurImageToShow = log(fftRecoverBlurImageToShow+1);

imagesc(fftRecoverBlurImageToShow);

axis square;

colormap gray;

title('Magnitude');

set(gca, 'XTick', [], 'YTick', []);

subplot(3,3,9);

imagesc(angle(fftshift(fftRecoverBlurImage)),[-pi pi]);

axis square;

colormap gray

title('Fase')

set(gca, 'XTick', [], 'YTick', []);

### Funções extras

function [value] = delta(val, m)

if (nargin ~= 1 && nargin ~= 2)

disp ("Faltando parametro");

end

value = zeros (size (val), class (val));

value(find(val == m)) = 1;

value(isnan (val)) = NaN;

end

function y = heaviside (x, zero\_value)

if ~exist('arg2', 'var')

zero\_value = 0.5;

end

if (nargin < 1)

disp("Faltando parametro");

return;

end

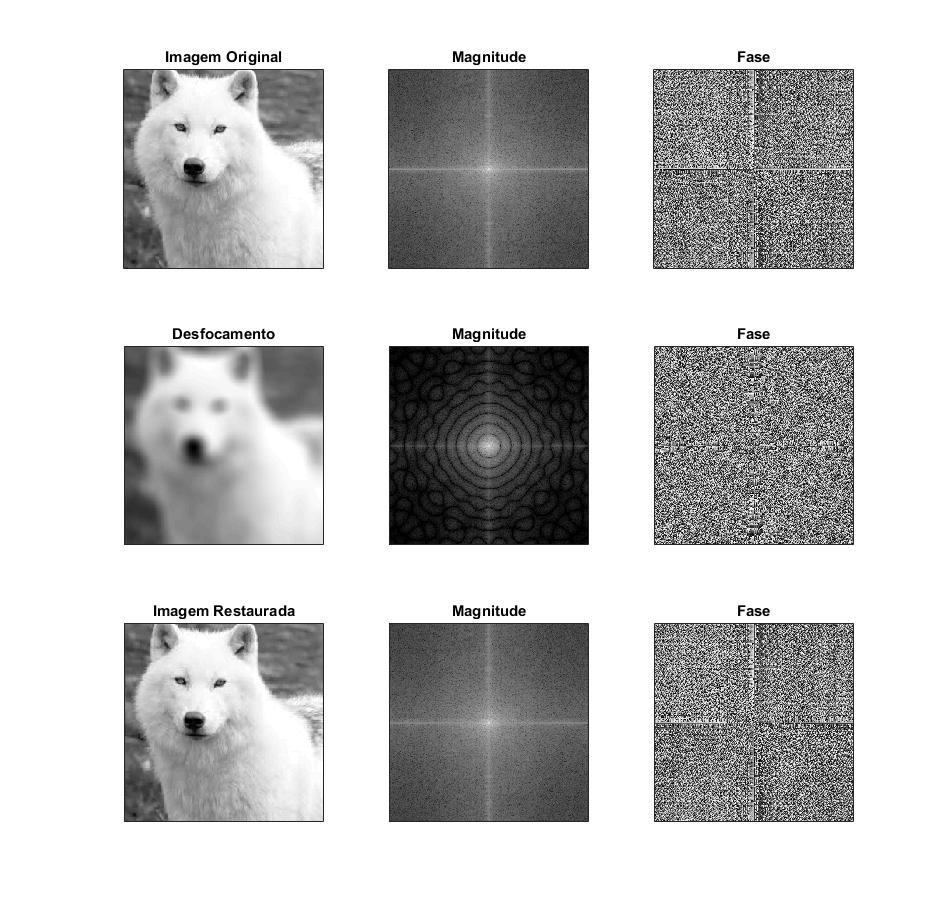
y = cast (x > 0, class (x));

y (x == 0) = zero\_value;

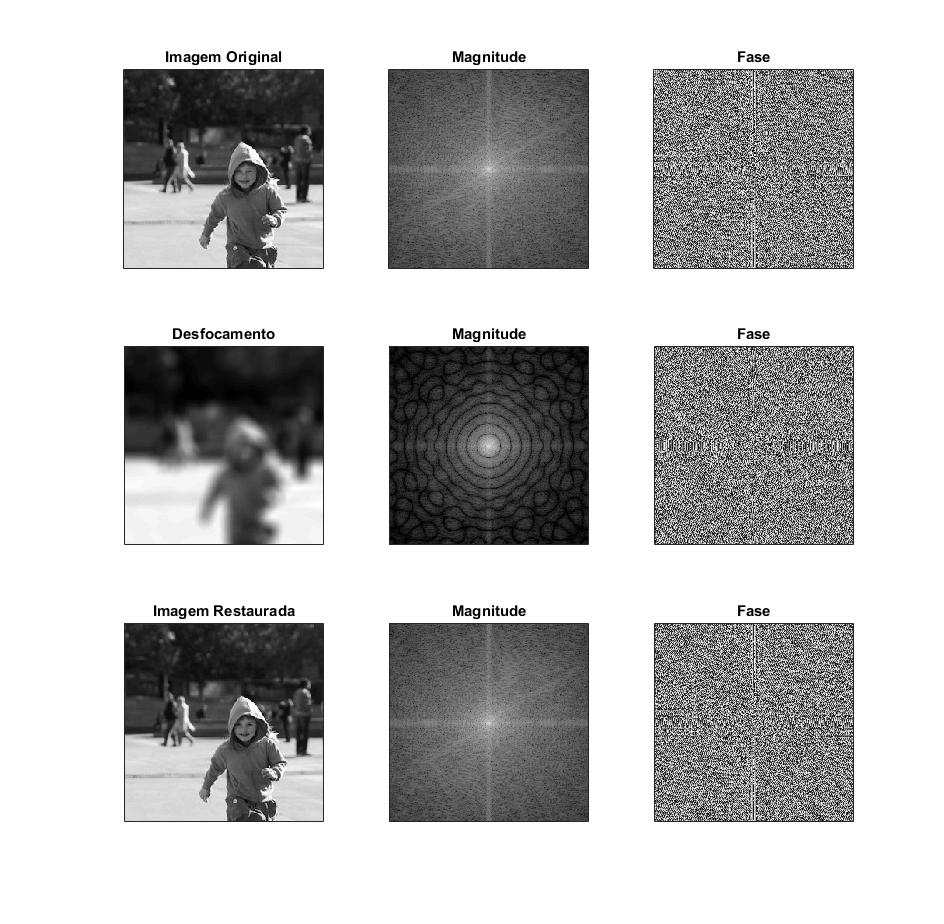
end

## Desfocamento

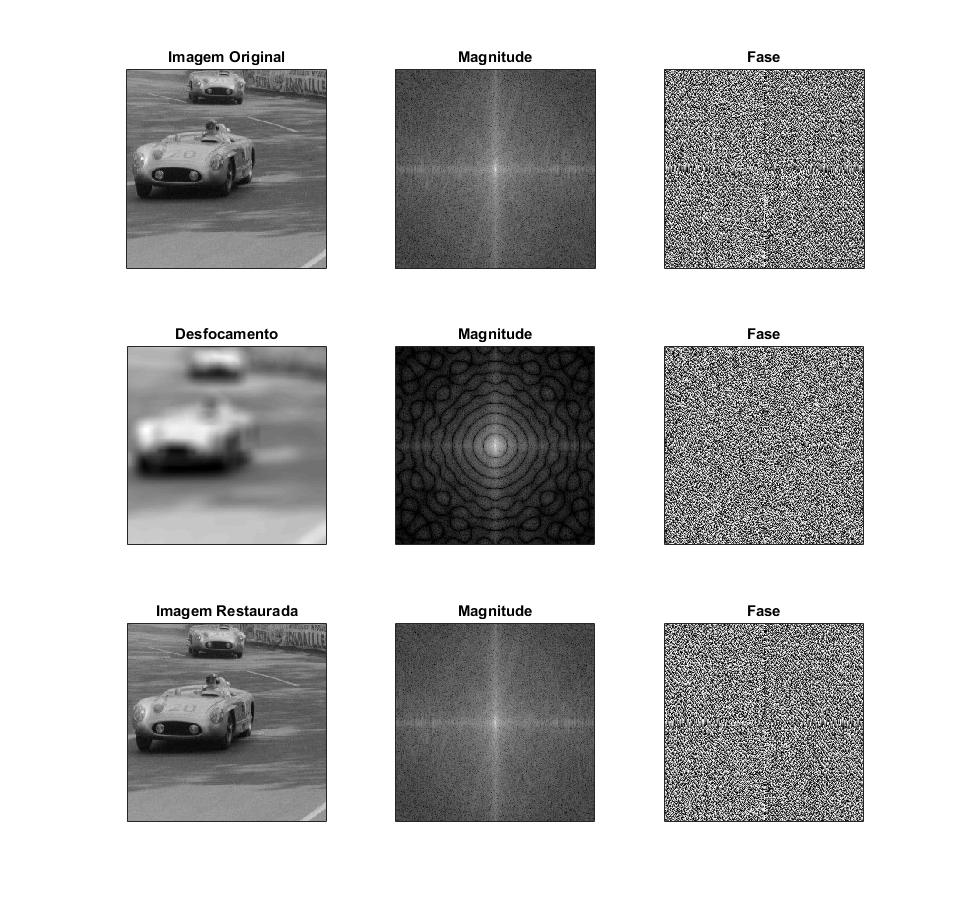
### Primeira imagem: cachorro



### Segunda Imagem: criança



### Terceira Imagem: Ferrari



### Código:

%Lê a imagem original do arquivo

originalImage = imread('cachorro.jpg');

%Recupera só um canal da imagem caso a imagem não esteja em tons de cinza

%originalImage = originalImage(:, :, 1);

%configurando plot

figure;

subplot(3,3,1);

imagesc(originalImage);

axis square;

colormap gray;

title('Imagem Original');

set(gca, 'XTick', [], 'YTick', []);

kernelPSFSize = 31;

kernelPSF = zeros(kernelPSFSize);

%lensDiameter = 50;

%focalDistance = 100;

%focalPlaneDisplacement = 200;

%R = (lensDiameter \* focalPlaneDisplacement) / (2 \* focalDistance);

R = 10;

m = kernelPSFSize/2;

[X, Y] = meshgrid(1:kernelPSFSize);

%Calculamos a função de espalhamento do ponto aqui

kernelPSF = psfDesfocamento(X, Y, R);

%Adicionamos um pad na imagem para evitar problemas nas bordas

originalImagePad = padimage(originalImage, kernelPSFSize);

%recuperamos a altura e largura da imagem original

[originalHeight, originalWidth] = size(originalImage);

%recuperamos a altura e largura da imagem com pad

[height, width] = size(originalImagePad);

%criamos a imagem do kernel com mesma altura e largura da imagem original

kernelImage = zeros(height, width);

%adicionamos o kernelPSF no início do kernelImagem. Poderia ser colocado no

%meio também ou no final

kernelImage(1:kernelPSFSize, 1:kernelPSFSize) = kernelPSF;

%convertemos a imagem para o domínio da frequência

fftImage = fft2(originalImagePad);

%convertemos o kernel para o domínio da frequência

fftKernel = fft2(kernelImage);

%recurso técnico para poder recuperar a imagem. Senão pode ocorrer divisão

%por zero

fftKernel(find(fftKernel == 0)) = 1e-6;

%aplicamos a multiplicação no domínio da frequência para obtermos a imagem

%borrada no domínio da frequência

fftBlurImage = fftImage.\*fftKernel;

%desenhamos as imagens no domínio da frequência

subplot(3,3,2);

fftImageTemp = fftshift(fftImage);

fftImageTemp = abs(fftImageTemp);

fftImageTemp = log(fftImageTemp+1);

imagesc(fftImageTemp);

axis square;

colormap gray;

title('Magnitude');

set(gca, 'XTick', [], 'YTick', []);

subplot(3,3,3);

imagesc(angle(fftshift(fftImage)),[-pi pi]);

axis square;

colormap gray

title('Fase')

set(gca, 'XTick', [], 'YTick', []);

subplot(3,3,5);

fftBlurImageToShow = fftshift(fftBlurImage);

fftBlurImageToShow = abs(fftBlurImageToShow);

fftBlurImageToShow = log(fftBlurImageToShow+1);

imagesc(fftBlurImageToShow);

axis square;

colormap gray;

title('Magnitude');

set(gca, 'XTick', [], 'YTick', []);

subplot(3,3,6);

imagesc(angle(fftshift(fftBlurImage)),[-pi pi]);

axis square;

colormap gray

title('Fase')

set(gca, 'XTick', [], 'YTick', []);

%fazemos a Inverse Fast Fourier Transform para obter a imagem no domínio

%espacial

blurImage = ifft2(fftBlurImage);

%Removemos o pad adicionado no início

blurImageUnpad = blurImage(kernelPSFSize+16:kernelPSFSize+originalHeight+15.5,kernelPSFSize+16:kernelPSFSize+originalWidth+15.5);

%figure;

subplot(3,3,4);

imagesc(blurImageUnpad);

axis square;

title('Desfocamento');

colormap gray;

set(gca, 'XTick', [], 'YTick', []);

%Tentando recuperar a imagem degradada

fftRecoverKernel = 1./fftKernel;

fftBlurImageToRecover = fft2(blurImage);

fftRecoverBlurImage = fftBlurImageToRecover.\*fftRecoverKernel;

recoverImage = ifft2(fftRecoverBlurImage);

recoverImageUnpad = recoverImage(kernelPSFSize+1:kernelPSFSize+originalHeight,kernelPSFSize+1:kernelPSFSize+originalWidth);

subplot(3,3,7);

imagesc(recoverImageUnpad);

axis square;

title('Imagem Restaurada');

colormap gray;

set(gca, 'XTick', [], 'YTick', []);

subplot(3,3,8);

fftRecoverBlurImageToShow = fftshift(fftRecoverBlurImage);

fftRecoverBlurImageToShow = abs(fftRecoverBlurImageToShow);

fftRecoverBlurImageToShow = log(fftRecoverBlurImageToShow+1);

imagesc(fftRecoverBlurImageToShow);

axis square;

colormap gray;

title('Magnitude');

set(gca, 'XTick', [], 'YTick', []);

subplot(3,3,9);

imagesc(angle(fftshift(fftRecoverBlurImage)),[-pi pi]);

axis square;

colormap gray

title('Fase')

set(gca, 'XTick', [], 'YTick', []);

### Funções extras

function [value] = psfDesfocamento(X, Y, R)

if (nargin ~= 1 && nargin ~= 2 && nargin ~= 3)

disp ("Faltando parametro");

end

[sizeX, sizeY] = size(X);

offset = floor(sizeX / 2) + 1;

disp(offset)

result = (X-offset).^2 + (Y-offset).^2;

value = zeros (size (X), class (X));

value(find(result <= R^2)) = (1 / (pi \* (R^2)));

end

## Funções Auxialiares

function Ipad = padimage( I, p )

[h, w] = size(I);

%Pad edges

Ipad = zeros(h+2\*p, w+2\*p);

%para fazer pad com cor preta

%sz = size(I);

%Ipad = zeros([sz(1:2)+2\*p size(I,3)], class(I));

%Ipad((1:sz(1))+p, (1:sz(2))+p, :) = I;

%----------------------------------------------------

%Meio

Ipad(p+1:p+h, p+1:p+w) = I;

%Superior e Inferior

Ipad(1:p, p+1:p+w) = repmat(I(1,1:end), p, 1);

Ipad(p+h+1:end, p+1:p+w) = repmat(I(end,1:end), p, 1);

%Esquerda e Direita

Ipad(p+1:p+h, 1:p) = repmat(I(1:end,1), 1, p);

Ipad(p+1:p+h, p+w+1:end) = repmat(I(1:end,end), 1, p);

%Cantos

Ipad(1:p, 1:p) = I(1,1); %Superior-esquerdo

Ipad(1:p, p+w+1:end) = I(1,end); %Superior-direito

Ipad(p+h+1:end, 1:p) = I(end,1); %Inferior-esquerdo

Ipad(p+h+1:end,p+w+1:end) = I(end,end); %Inferior-direito

end