Universidade Federal de Pelotas
Curso Ciência da computação de Computação
Disciplina: Processamento Digital de Imagens
Professor: Bruno Zatt

Relatório 1: Interpolação

Aluno:Yago Martins Pintos . Data:29/07/2024

1. Introdução

Para manipular imagens, é crucial entender as características capturadas pelos sensores, garantindo um registro conciso e sem perda de informações. Para desenvolver uma imagem digital, precisamos converter dados contínuos para o formato digital usando dois processos: amostragem e quantização.

As imagens são apresentadas através de matrizes, onde cada pixel representa um ponto da imagem original. Para isso, usamos cálculos específicos para encontrar o valor esperado em cada ponto, como a interpolação, essencial para redimensionar a imagem, como em ampliações.

Neste trabalho, vamos discutir amostragem e quantização, apresentar o algoritmo de interpolação por vizinhos mais próximos e bilinear, usando MATLAB®, e mostrar os resultados gerados. Vamos concluir com uma discussão sobre o tema.

2. Discussões

A amostragem e a quantização envolvem converter uma imagem contínua para o formato digital. Na amostragem, capturamos valores em coordenadas e amplitude; na quantização, digitalizamos esses valores. Assim, podemos tratar e manipular as imagens, aplicando ampliações, reduções, rotações e correções geométricas.

A interpolação ajuda a reamostrar imagens. Estimamos novos pontos com base em valores conhecidos na matriz da imagem original. Discutiremos a interpolação por vizinho mais próximo e bilinear, mostrando partes do código em MATLAB® e os resultados obtidos.

2.1 Interpolação por Vizinho Mais Próximo

Este método atribui a cada nova posição a intensidade do vizinho mais próximo na imagem original, mas pode causar distorções nas bordas. No trabalho, desenvolvemos um código (Figura 1) que lê a imagem original (Figura 2), particiona as matrizes e amplia ou reduz a imagem com um fator de correção, criando vizinhos semelhantes ao valor original.

Testamos com a função `redimensionar()`, usando exemplos como `redimensionar('barbara_gray.bmp',1,2, 2)` e `redimensionar('barbara_gray.bmp', 1, 0.3, 0.3)`. A função recebe o nome do arquivo, o tipo de interpolação e os fatores de correção para x e y. As Figuras 3 e 4 mostram a expansão e compressão da imagem.

2.2 Interpolação Bilinear

Este método, similar ao anterior, estima pontos na imagem original com o fator de correção na função `redimensionar()`. Usamos equações para encontrar o valor do ponto correspondente na matriz destino, como P(1, 3), e funções R, S, T, U.

O código (Figura 5) mostra como calculamos a distância entre os pontos e usamos a parte fracionária para definir o valor final do ponto destino. Testamos com `redimensionar('barbara_gray.bmp', 2, 2, 3)` e `redimensionar('barbara_gray.bmp', 2, 0.3, 0.3)`, e os resultados são

mostrados nas Figuras 6 e 7.

3. Conclusões

Através desses processos, entendemos a descrição da imagem em coordenadas e amplitude usando amostragem e quantização para gerar processamento digital. Desenvolvemos métodos de redimensionamento, capazes de ampliar ou reduzir a imagem de entrada na interpolação, com resultados conforme esperado.

Figura 1: Código interpolação por vizinhos mais próximos

```
if tipo_interpolacao == 1
    % Interpolação por vizinho mais próximo
    for i = 1:novas_linhas
        for j = 1:novas_colunas
            orig_x = round((i - 1) / fator_x) + 1;
            orig_y = round((j - 1) / fator_y) + 1;
            orig_x = min(max(orig_x, 1), linhas);
            orig_y = min(max(orig_y, 1), colunas);
            nova_img(i, j) = img(orig_x, orig_y);
        end
    end
```

Figura 2: Barbara

Figura 3:Barbara após aplicar interpolação por vizinhos mais próximo para entrada (2,2)

Figura 4: Barbara após aplicar interpolação por vizinhos mais próximo para entrada (0.3,0.3)

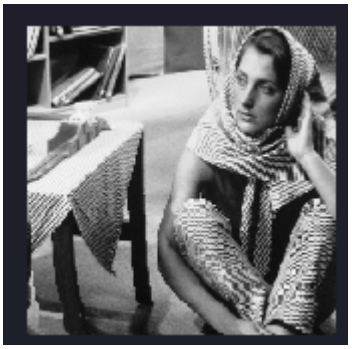
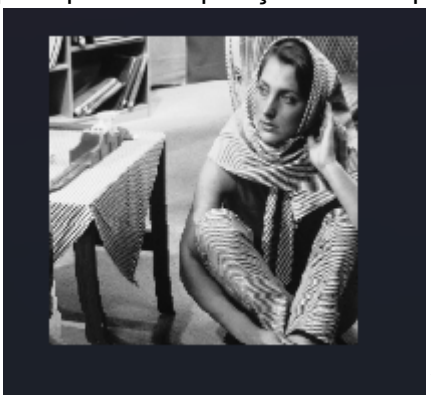


Figura 5: Código Interpolação Bilinear

```
I_reshape = zeros((size(I)+2));
I_reshape = cast(I_reshape, 'uint16');
I_reshape(1:size(I, 1), 1:size(I, 2)) = I(:, :);
I_reshape(:, size(I, 1)+1) = I_reshape(:, size(I, 1));
I_reshape(:, size(I, 1)+2) = I_reshape(:, size(I, 1));
I_reshape(size(I, 2)+1, :) = I_reshape(size(I, 2), :);
I_reshape(size(I, 2)+2, :) = I_reshape(size(I, 2), :);
fx=num1; fy = num2;
remodular_bilinear = zeros(ceil(size(I, 1)*fy), ceil(size(I, 2)*fx));
for y = 1:size(I, 1)*fy
   for x = 1:size(I, 2)*fx
      tx = uint16(abs((x-1)/fx+1 - fix((x-1)/fx+1)));
      ty = uint16(abs((y-1)/fy+1 - fix((y-1)/fy+1)));
      R = I_reshape(uint16((y-1)/fy+1), uint16((x-1)/fx+1));
      S = I_reshape(uint16((y-1)/fy+1), uint16((x-1)/fx+2));
      T = I_reshape(uint16((y-1)/fy+2), uint16((x-1)/fx+1));
      U = I_reshape(uint16((y-1)/fy+2), uint16((x-1)/fx+2));
      Q1 = tx*S + (1-tx)*R;
      Q2 = tx*T + (1-tx)*U;
      remodular_bilinear(y,x) = ty*Q1 + (1-ty)*Q2;
   end
end
```

Figura 6: Barbara após aplicar interpolação bilinear para entrada (2,3)

Figura 7: Barbara após aplicar interpolação bilinear para entrada (0.3,0.3)


)