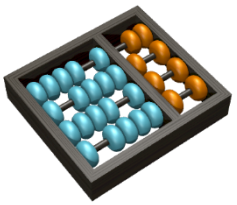
Universidade Estadual de Campinas
Instituto de Computação



Disciplina do 1º Semestre de 2024

IC - UNICAMP

Curso: Bacharelado em Ciência da Computação

MC920 - Introdução ao processamento de imagem digital

Trabalho 4

Alunos: Pedro Barros Bastos **RA**: 204481

Professor: Hélio Pedrini

Campinas – SP
2024

Introdução

Este relatório tem como objetivo detalhar o desenvolvimento da parte 1.1 do trabalho 4, no qual foram explorados conceitos de transformações geométricas. Foi implementada a escala, a rotação e o redimensionamento de uma imagem dada como entrada sem o uso de bibliotecas tal como opencv.

Também foram desenvolvidos 4 diferentes métodos de interpolação baseados nas fórmulas dadas no enunciado deste trabalho: interpolação por vizinho mais próximo, interpolação bilinear, interpolação bicúbica e interpolação por polinômios de Lagrange.

Arquivos executáveis

- **geometricTransformer.py** - script python que irá efetuar as devidas transformações geométricas
    - Parâmetros de execução:
        - ***--i***: Path da imagem a ser transformada
        - ***--o***: Nome da imagem de saída
        - ***--r***: Fator de rotação em graus
        - ***--e***: Fator de escala
        - ***--he***: Altura da imagem de saída
        - ***--w***: Largura da imagem de saída
        - ***--m***: Interpolação a ser utilizada durante as transformações:
            - NEAREST
            - BILINEAR
            - BICUBIC
            - LAGRANGE

Obs: Foi deixado um arquivo texto chamado **EXEMPLOS-EXECUCAO.txt** com exemplos de possíveis execuções dos scripts python.

Pacotes utilizados

- **OpenCV** - v4.6.0
- **Numpy** - v1.26.4
- **argparse** - v1.4.0

Obs: Foi utilizada a ferramenta **conda** para construção do ambiente de desenvolvimento para este trabalho, bem como utilização de SO **Ubuntu 22.04.4 LTS** inicializado em uma virtualbox.

Considerações para implementação

Para efetuar as transformações, teve-se que atentar para o fato de que a realização da transformação espacial pelo mapeamento direto do pixel de saída na forma $P' = TP$ pode resultar em falhas na imagem transformada. Devido ao problema de discretização da imagem, podem-se ter diferentes pontos da imagem original sendo mapeados para um mesmo ponto na imagem de saída. No mapeamento indireto, na forma $P = T^{-1}P'$, temos garantido que todos os pontos da imagem transformada terão um pixel associado à imagem original.

Além disso, ao realizar o cômputo do pixel original correspondente ao pixel da imagem de saída podemos ter a situação em que as coordenadas estejam em valores fracionados. Como exemplificado no enunciado, ao se efetuar um redimensionamento com um fator de escala de 2.25, então a posição de saída de um pixel Po = (10, 23) seria Pi = Po/s = (10/2.25, 23/2.25) = (4.444, 10.222). Para tanto, foram implementadas 4 interpolações com o intuito de se definir qual a coordenada inteira da imagem original a ser considerada na produção de um pixel da imagem de saída:

- Interpolação pelo vizinho mais próximo

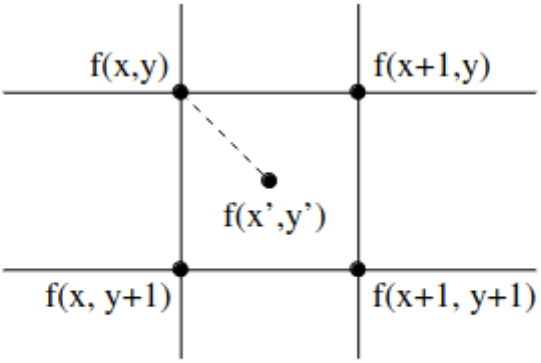


Figura 1: Interpolação pelo vizinho mais próximo.

$$f(x', y') = f(\text{round}(x), \text{round}(y))$$

- Interpolação Bilinear

$$f(x', y') = (1 - dx)(1 - dy)\ f(x, y) + dx(1 - dy)\ f(x + 1, y) + (1 - dx)dy\ f(x, y + 1) + dxdy\ f(x + 1, y + 1)$$

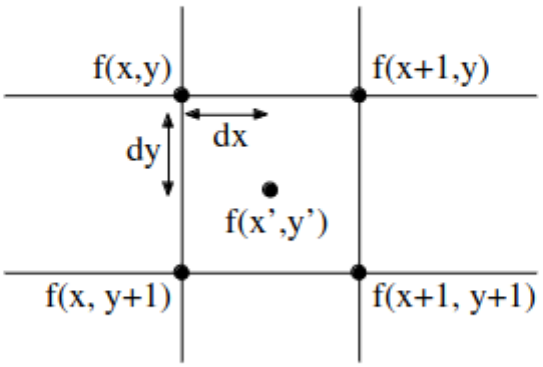


Figura 2: Interpolação bilinear.

- Interpolação Cúbica

$$f(x', y') = \sum_{m=-1}^{2} \sum_{n=-1}^{2} f(x+m, y+n) R(m - dx) R(dy - n)$$

sendo

$$R(s) = \frac{1}{6}[P(s+2)^3 - 4P(s+1)^3 + 6P(s)^3 - 4P(s-1)^3]$$

$$P(t) = \begin{cases} t, & t > 0 \\ 0, & t \leq 0 \end{cases}$$

- Interpolação por Polinômios de Lagrange

$$f(x', y') = \frac{-dy(dy-1)(dy-2)L(1)}{6} + \frac{(dy+1)(dy-1)(dy-2)L(2)}{2} + \frac{-dy(dy+1)(dy-2)L(3)}{2} + \frac{dy(dy+1)(dy-1)L(4)}{6}$$

As transformações geométricas realizadas foram a escala da imagem a partir de um determinado fator fornecido de entrada, a rotação em sentido horário a partir de um ângulo em graus fornecido de entrada e o redimensionamento da imagem em dadas largura e altura também fornecidas de entrada para execução do script python. Como salientado no enunciado do trabalho, a cada execução do script realiza-se apenas uma transformação geométrica (escala ou rotação), seguido do redimensionamento a partir da largura e altura fornecidos.

Tais transformações foram implementadas utilizando coordenadas homogêneas para que assim fosse possível a combinação de várias transformações para o resultado esperado, principalmente no caso da rotação.

Implementação Escala

Para realizar o mapeamento indireto, primeiro obteve-se, a partir do fator de escala fornecido de entrada, a matriz em coordenada homogênea da transformação,

Escala

$$S = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

onde Sx e Sy correspondem ao fator de escala passado.

Com essa matriz, obtém-se sua inversa a partir do comando *np.linalg.inv()* e assim pode-se iterar em todos os pixels da imagem de saída (inicialmente criada com zeros no tamanho correspondente ao fator de escala) e atribuir o respectivo valor da sua imagem de entrada.

```
for row in range(outputRows):
    for column in range(outptColumns):
        # Px
        # Py
        # 1
        currentOutputPixel = np.array([
                                        [column],
                                        [row],
                                        [1]
                                      ])
        # | Sx 0 0 |-1  *  | Px |
        # | 0 Sy 0 |       | Py |
        # | 0 0  1 |       |  1 |
        inputImgPixel = inverseScaleMatrix @ currentOutputPixel
```

Interessante verificar aqui a “troca da ordem” da intuição da nomenclatura dos componentes: Px = coluna, Py = linha. Tal aspecto foi de suma importância para o entendimento do trabalho e padronização para evitar confusão durante o desenvolvimento.

Como mencionado anteriormente, a maior parte dos pixels resultantes de tal operação possui um valor fracionado nas componentes de linha e coluna. Para corrigir este problema, executa-se a interpolação pelo método escolhido no chamada do script. Detalhamento da interpolação mais a adiante neste relatório.

```
inputImgPixel = inverseScaleMatrix @ currentOutputPixel
outputImage[row, column] = interpolation.interpolate(inputImgPixel[1,0], inputImgPixel[0,0], inputImg)
```

Implementação Rotação

Já para a implementação da rotação, usou-se a matriz em coordenadas homogêneas composta pelos senos e cossenos do ângulo passado por parâmetro.

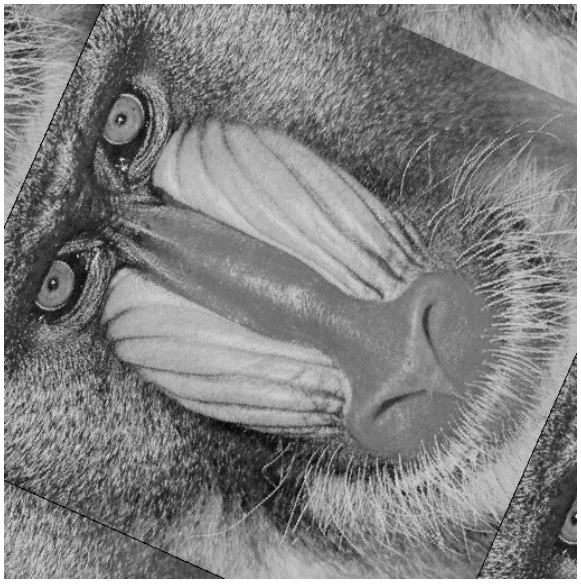
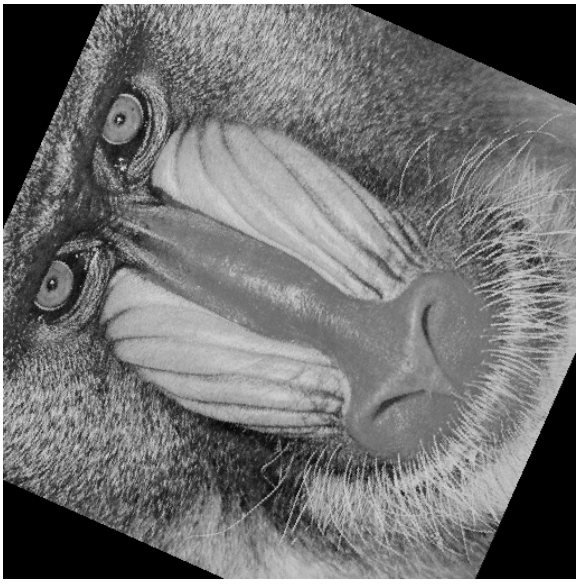
Rotação

$$R = \begin{bmatrix} \cos\alpha & -\text{sen}\,\alpha & 0 \\ \text{sen}\,\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

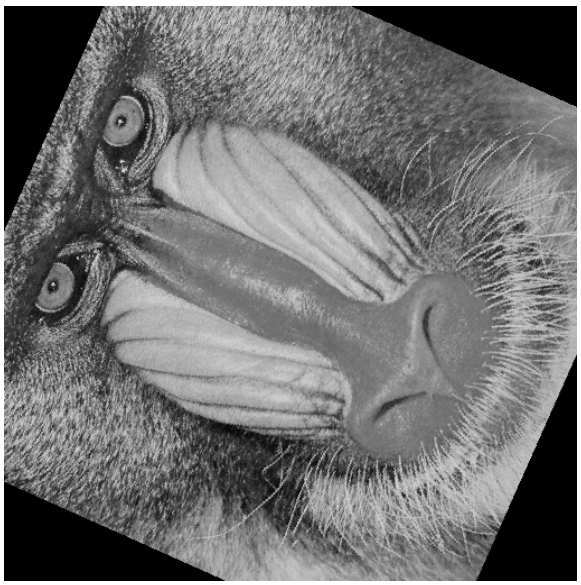
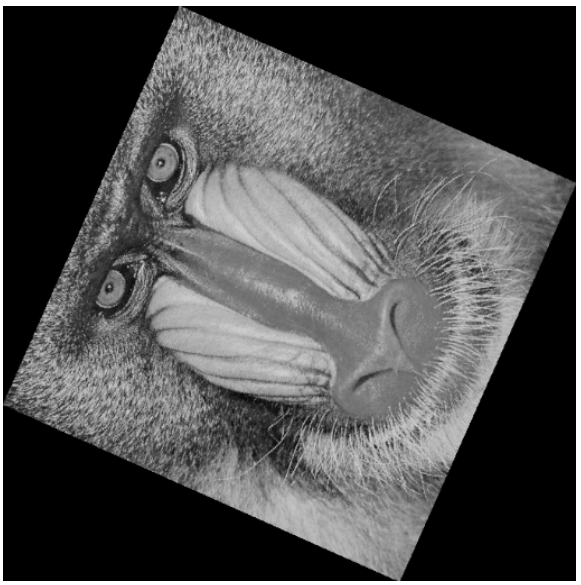
Mas no caso da rotação, apenas esta matriz não é capaz de realizar a rotação que se é esperada. Para tanto, deve-se trazer o centro da imagem para a origem do plano cartesiano, aplicar a rotação e voltar o centro da imagem para o ponto de origem. Assim, foram ainda introduzidas mais duas transformações geométricas: translação para a origem do plano cartesiano e volta da imagem para o ponto inicial.

```
rotationMatrix = np.array([
                    [math.cos(rotationFactorRadians), -math.sin(rotationFactorRadians), 0],
                    [math.sin(rotationFactorRadians), math.cos(rotationFactorRadians), 0],
                    [0, 0, 1]
                ])
translationToOriginMatrix = np.array([
                    [1, 0, -(image_width // 2)],
                    [0, 1, -(image_height // 2)],
                    [0, 0, 1]
                ])
translationBackMatrix = np.array([
                    [1, 0, (image_width // 2)],
                    [0, 1, (image_height // 2)],
                    [0, 0, 1]
                ])

resultingTransformation = translationBackMatrix @ rotationMatrix @ translationToOriginMatrix
inverse_matrix = np.linalg.inv(resultingTransformation)
```

Tal como na transformação de escala, o resultado da transformação composta acima também retorna valores fracionados, sendo também necessário realizar a devida interpolação. A diferença para a escala neste caso é que foi adicionado um tratamento para evitar na rotação que sejam mapeados pixels que não deveriam constar na imagem. Devido a rotação, a parte da imagem que não deve ter pixel nenhum (lado “de fora” da imagem) recebe o valor de 0, criando assim um fundo preto para se diferenciar da imagem rotacionada. Se isso não fosse realizado, pedaços da imagem ficariam duplicados no que seria esse fundo da imagem.

	
Rotação 66º sem tratamento de pixels do fundo	*Rotação 66º com tratamento de pixels do fundo*

Outro tratamento relacionado a rotação foi a adição de um padding antes de se efetuar a transformação para que nenhum pedaço da imagem fosse perdido. Para tanto, foi utilizada a função *cv.copyMakeBorder* após se realizar os devidos cálculos para a geração correta do padding necessário a partir do ângulo da rotação (função *calculate_padding* do módulo *utils*). Apesar de detalhes não serem perdidos, para manter as dimensões originais (se passadas assim como parâmetros de altura e largura do script) o conteúdo da imagem em si acaba tendo um scaling para sua redução assim cabendo no quadro de altura e largura passado.

	
Rotação 66° sem padding	*Rotação 66° com padding*

Implementação Redimensionamento

O redimensionamento pode ser considerado como duas escalas sendo realizadas ao mesmo tempo: redimensionamento de altura e de largura, considerando que podem ser passadas altura e largura diferentes, formando assim uma imagem que não seja quadrada. Desta forma foi apenas necessário computar o pixel da imagem original correspondente ao pixel de saída corrente e passar o resultado para a interpolação.

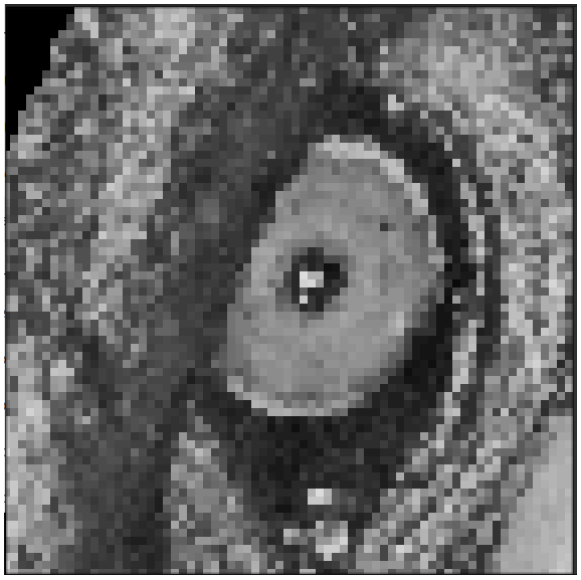
```python
def resize(image, resized_image, interpolation: interpolation.Interpolation):
    inputRows, inputColumns = image.shape[:2]
    outputRows, outptColumns = resized_image.shape[:2]

    rowsScaleFactor = outputRows / inputRows
    columnsScaleFactor = outptColumns / inputColumns

    for row in range(outputRows):
        for column in range(outptColumns):
            resized_image[row, column] = interpolation.interpolate(row / rowsScaleFactor, column / columnsScaleFactor, image)
```

Interpolações

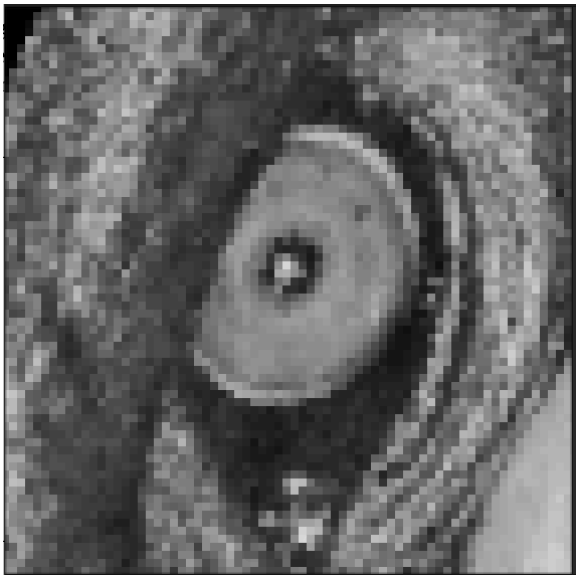
As interpolações utilizadas apresentaram diferentes resultados quando aplicadas nas transformações geométricas. Destaco aqui que não foi possível realizar a vetorização da interpolação a tempo da entrega. Assim, durante a execução do script, dependendo do método de interpolação escolhido, a saída pode levar alguns bons segundos até ser completada.

A interpolação pelo vizinho mais próximo foi a que teve a maior perda de detalhes mas também foi a que executou de forma mais rápida. A interpolação bilinear obteve menor perda de detalhes que a do vizinho mais próximo, mas já leva um tempo um pouco maior para ser executada. O mesmo acontece de forma análoga com a interpolação por polinômios de Lagrange e interpolação bicúbica. Esta última é a que apresentou menor perda de detalhes mas também é a interpolação que mais leva tempo para ser completada. Além disso, a bicúbica apresentou um leve borramento na imagem de saída comparado com a imagem original mas que não afetou drasticamente a percepção dos detalhes da imagem.

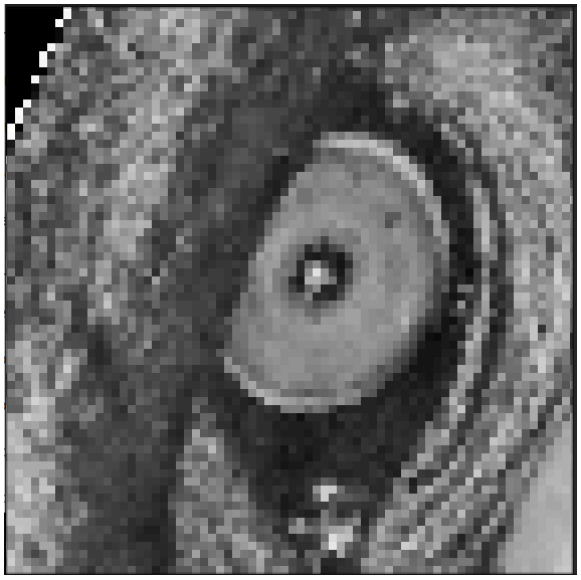
Para poder visualizar tais diferenças dos métodos de interpolação quanto aos detalhes, abaixo são mostradas imagens aproximadas do olho direito do baboon. Na primeira tabela a partir de uma rotação de 66º, na segunda tabela a partir de uma escala de 2x.



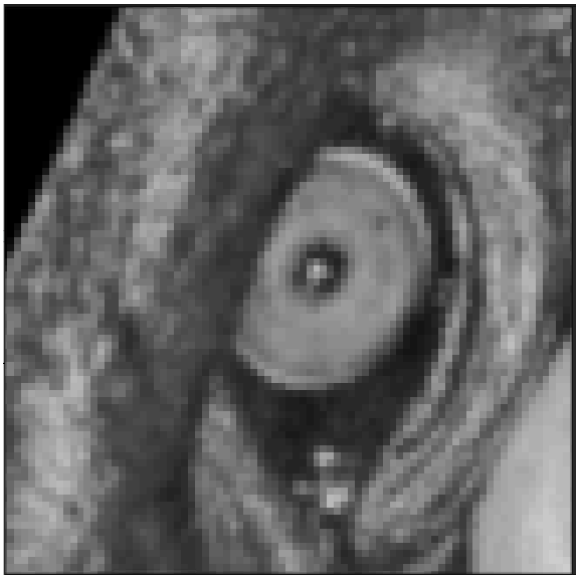
Rotação 66º com interpolação NEAREST



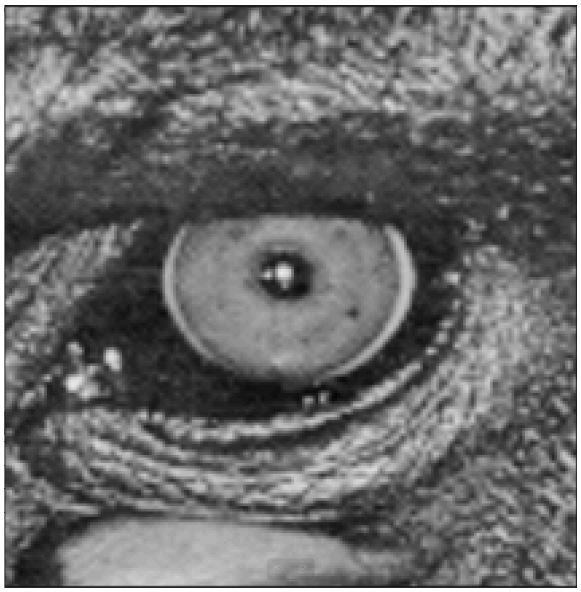
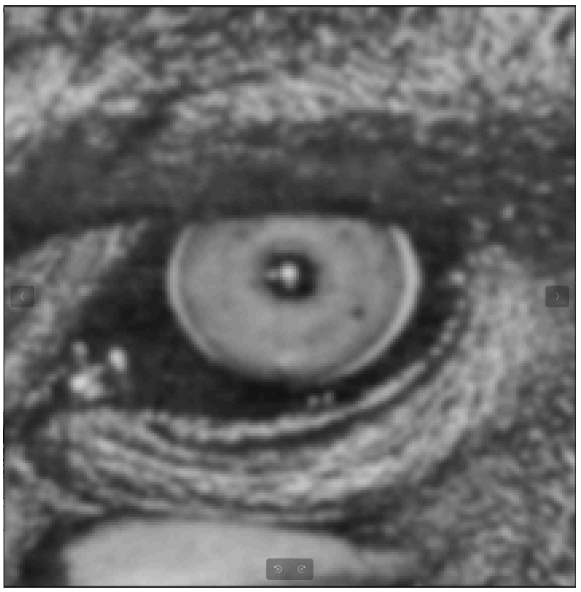
Rotação 66º com interpolação BILINEAR



Rotação 66º com interpolação LAGRANGE



Rotação 66º com interpolação BICUBIC

	
Scale 2x com interpolação NEAREST	*Scale 2x com interpolação BILINEAR*
	
Scale 2x com interpolação LAGRANGE	*Scale 2x com interpolação BICUBIC*