Trabalho 4

Processamento Digital de Imagem

Ieremies Vieira da Fonseca Romero

Introdução

O processamento digital de imagens é uma área de estudo fundamental em Ciência da Computação, que visa o tratamento e manipulação de imagens digitais por meio de técnicas e algoritmos computacionais. A análise e processamento de imagens digitais é uma tarefa complexa e multidisciplinar, que envolve conhecimentos em matemática, estatística, computação gráfica, processamento de sinais e outras áreas correlatas.

O objetivo deste trabalho é apresentar uma série de processamentos básicos em imagens digitais, abordando desde a leitura e escrita de imagens até operações de filtragem. Para o desenvolvimento das operações, será empregada a vetorização de comandos, que permite processar as imagens de forma mais eficiente e rápida, através da aplicação de operações vetoriais em vez de operações matriciais.

O programa

Neste trabalho, utilizamos as bibliotecas `numpy` 1.24.1 e OpenCV 4.7.0.72 utilizado via `cv2`.

```
import cv2
import numpy as np
```

Cada uma das preposições do enúnciado é resolvido por uma função contida no arquivo `funcs.py`. Todas as funções possuem como primeiro parâmetro a imagem ou janela sob a qual ela irá atuar.

Para leitura e escrita das imagens, utilizaremos as seguintes funções do `cv2`.

```
# read the pgm image
cv2.imread('in.pgm')
cv2.imwrite('out.pgm', img)
```

Perspectiva

Para essa primeira parte, o código está presente no arquivo `perspec.py`. O primeiro passo é definir os pontos de origem e destino da projeção perspectiva:

```
source = np.float32([[37, 51], [342, 42], [485, 467], [73, 380]])
dest = np.float32([[0, 0], [511, 0], [511, 511], [0, 511]])
```

Depois, podemos calcular a matriz de transformação usando a função `cv2.getPerspectiveTransform` pasando os pontos desejados como parâmetro.

```
M = cv2.getPerspectiveTransform(src, dest)
```

Por fim, aplicamos a projeção perspectiva em na imagem desejada.

```
image = cv2.imread("img/baboon_perspectiva.png")
output_image = cv2.warpPerspective(image, M, (512, 512))
cv2.imwrite("./out.png", output_image)
```

Transformações geométricas

A primeira parte do programa é determinar os parâmetros que utilizamos. Para isso, fazemos uso da bilbioteca `argparse` disponível no python, a fim de declarar quais as "flags" e argumentos possíveis. Dessa forma, no arquivo `arg_parser.py`, definimos um `parser` e utilizamos o método `add_argument` para declara cada um dos argumentos do programa. São eles:

Ângulo `-a`, ângulo de rotação medido em graus no sentido anti-horário.

Escala `-e`, fator de escala.

Dimensão `-d`, largura e altura da imagem resultante desejada.

Método de interpolação `-m`, pode ser `proximo` (vizinho mais próximo), `bilinear`, `bicubica` ou `lagrange`.

Entrada `-i`, caminho para a imagem a ser processada

Saída `-o`, caminho para a imagem a ser gerada.

Todas essas informações estão disponíveis utilizando `python prog.py --help`.

Nosso programa principal, no arquivo `prog.py` conciste em pegar os parâmetros na variável `args` e chamar a função `scale_and_rotate_image` com eles. Esta, por sua vez, é quem realiza as duas transformações.

Primeiro passo é determinar algumas informações sobre a imagem que estamos trabalhando. A utilizadade delas ficará mais clara logo após.

```
# converter o ângulo de rotação para radianos
angle = np.deg2rad(angle)

# Criar uma imagem com a dimensão dim
res = np.zeros((dim[1], dim[0], 3), np.uint8)

# O centro da imagem de saída
center = (dim[0] // 2, dim[1] // 2)

# O centro da imagem de entrada
center_in = (image.shape[1] // 2, image.shape[0] // 2)
```

Para operar as transformações, faremos o processo inverso: ao invés de determinar onde um pixel da entrada irá estar na saída, determinaremos de onde o pixel da saída irá tirar o seu valor. Isso faz com que o processo de interpolação usado para preencher os espaços vazios fique mais fácil. Mas para isso, precisaremos aplicar o inverso das operações.

Dessa forma, para cada ponto `(y,x)` da saída, realizaremos o seguinte processo: transladamos de forma que o centro da imagem fique no `(0,0)` e multiplicamos pelo inverso do fator de escala.

```
coord = (
    (y - center[0]) / scale_factor,
    (x - center[1]) / scale_factor,
)
```

Depois, utilizamos a matriz inversa da rotação para obter as seguintes equações: $x' = cos(a)x + sin(a)y$ e $y' = -sin(a)x + cos(a)y$.

```
coord = (
    coord[0] * np.cos(angle) + coord[1] * np.sin(angle),
    coord[1] * np.cos(angle) - coord[0] * np.sin(angle),
)
```

Enfim, desfazemos a translação realizada no começo, agora com o centro da imagem de entrada.

```
coord = (
    coord[0] + center_in[0],
    coord[1] + center_in[1],
)
```

Por fim, as coordenadas obtidas podem não ser inteiras, efetivamente se encontrando "entre pixels" da entrada. Para isso, utilizamos o método de interpolação desejado para obter o valor do pixel da saída.

```
res[y, x] = interp(image, *coord)
```

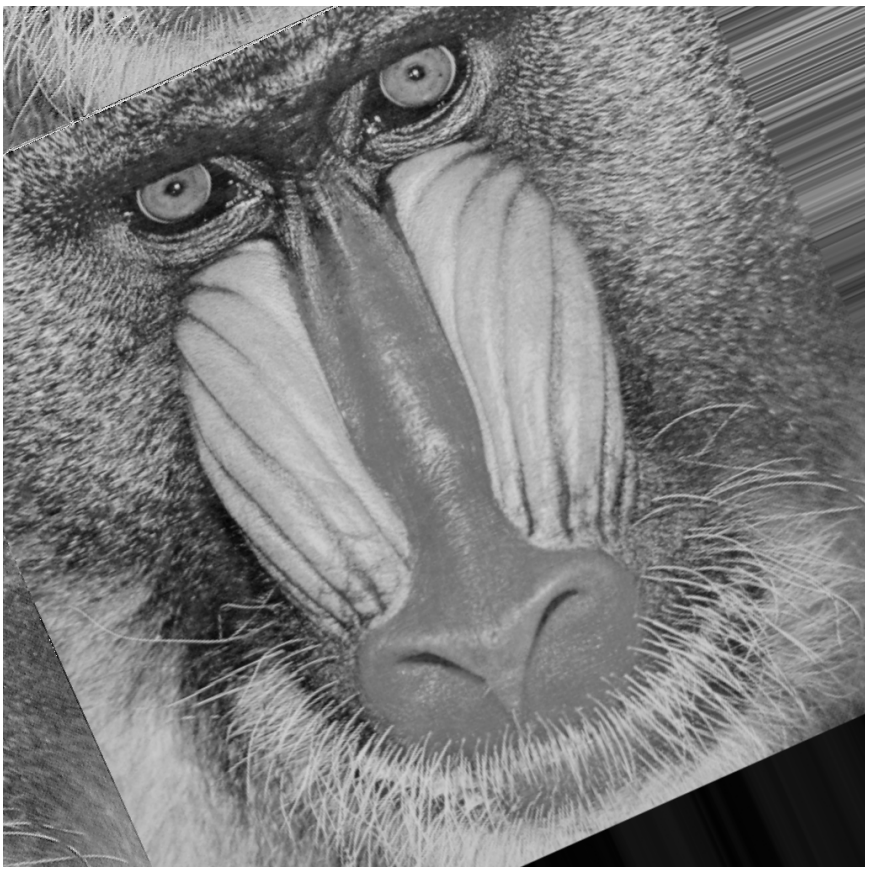

Figure 1: Resultado de uma escala por 2.5 e rotação de 24 graus.

Métodos de interpolação

Cada método de interpolação é implementado por uma função que recebe a imagem original e as coordenadas `(x,y)` e retorna o valor do pixel para a saída.

O primeiro método de interpolação é utilizar o pixel mais próximo. Em python, podemos fazer isso com o uso da função `round` que arredonda ao inteiro mais próximo.

```
def proximo(img, x, y):
    x = round(x)
    y = round(y)
    # caso o round passe dos limites da imagem original
    if x >= img.shape[0] or x < 0:
        return 0
    if y >= img.shape[1] or x < 0:
        return 0
    return img[x, y]
```

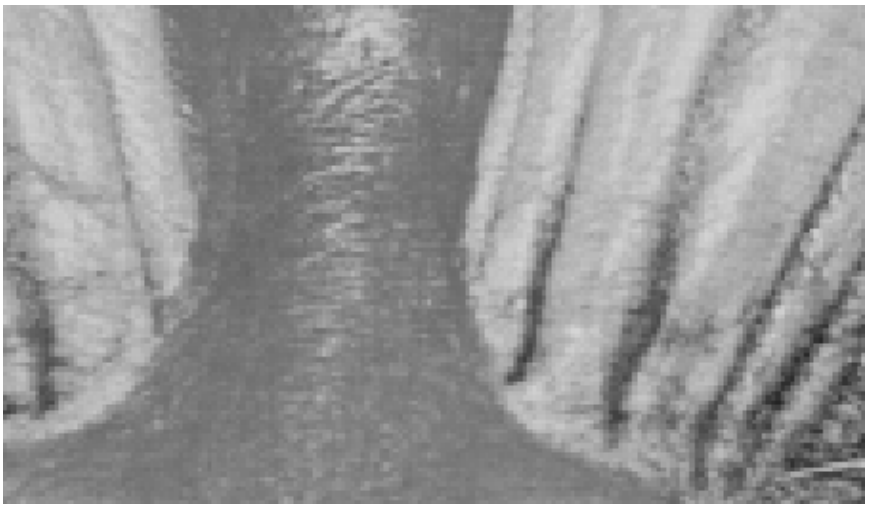

Figure 2: Zoom no resultado utilizando o método da interpolação do vizinho mais próximo. Observamos um efeito pixelado forte.

O segundo método é o `bilinear` que utiliza a média ponderada no inverso da distância dos quatro pontos mais próximos. Este método tende a dar resultados mais "suaves" tendo em vista a natureza de média

```python
def bilinear(img, x, y):
    dx = x - int(x)
    dy = y - int(y)
    x = int(x)  # arredondar para baixo
    y = int(y)
    res = (1 - dx) * (1 - dy) * img[x, y]
    res += dx * (1 - dy) * img[x + 1, y]
    res += (1 - dx) * dy * img[x, y + 1]
    res += dx * dy * img[x + 1, y + 1]
    return res
```

Figure 3: Zoom no resultado utilizando o método da interpolação bilinear. Observamos que o resultado fica mais suave.

Não foi possível obter resultados satisfatórios com nossas implementações dos métodos bicúbicas nem do polinômio de lagrange.

Conclusão

Neste trabalho, exploramos as transformações geométricas e a perspectiva como técnicas fundamentais no processamento digital de imagens. Utilizando as bibliotecas numpy e OpenCV, abordamos a aplicação de transformações como rotação, escala e projeção perspectiva em imagens digitais. Na seção de

perspectiva, aprendemos como definir os pontos de origem e destino da projeção e aplicar a transformação utilizando a função cv2.getPerspectiveTransform. Isso nos permitiu realizar projeções perspectivas em imagens, obtendo resultados interessantes. Já nas transformações geométricas, exploramos a aplicação de interpolação para preencher os espaços vazios resultantes das transformações, utilizando métodos como o vizinho mais próximo e o bilinear. Essas técnicas são amplamente utilizadas em áreas como visão computacional, realidade virtual e processamento de imagens médicas, proporcionando uma compreensão essencial para a manipulação de imagens digitais e abrindo portas para futuras aplicações e estudos mais avançados.