图像处理与可视化 Homework-7 报告

林子开 21307110161

2023 年 12 月 9 日

目录

1 FFD 形变算法 **1**
1.1 FFD 原理简介 1
1.2 FFD 的 Python 实现 1

2 反向图像变换 **3**
2.1 反向变换原理简介 3
2.2 反向变换的 Python 实现 3
2.3 效果测试 6

1 FFD 形变算法

1.1 FFD 原理简介

FFD 算法的核心想法是：将控制点规则化，并且把控制点的位移产生的影响控制在局部。

FFD 的变换公式为：$Y = T(X) = X + Q_{local}(X)$，其中 $Q_{local}(X)$ 表示 X 发生的位移：

$$Q_{local}(X) = \sum_{i=-1}^{2}\sum_{j=-1}^{2} \Phi_{i+i_x,j+j_x}\beta_i(u)\beta_j(v)$$

$$u = \frac{x - P_{00}[0]}{l_x} - i_x,\ v = \frac{y - P_{00}[1]}{l_y} - j_y$$

$$i_x = \left\lfloor \frac{x - P_{00}[0]}{l_x} \right\rfloor,\ j_y = \left\lfloor \frac{y - P_{00}[1]}{l_y} \right\rfloor$$

其中 $\Phi_{i+i_x,j+j_x}$ 表示 $(i + i_x, j + j_x)$ 处的控制点的位移，β_i, β_j 是分段形式的 B-样条核函数。

1.2 FFD 的 Python 实现

Listing 1: FFD 形变算法

```python
"""实现了FFD形变算法"""

import numpy as np
from collections import defaultdict


def ffd(im_height: int,im_width: int, m: int, n: int, shift_dict: dict) -> np.array:
    """
```

```
    实现FFD形变算法，Y=X+Q(x)，其中Q(x)= \sum_i (shift_i * weight_i)
    input:
        im_height: 原图的高
        im_width: 原图的宽度
        m,n: 网格的高和宽，将会在原图上划分出(m+1)*(n+1)的均匀等分控制点
        shift_dict: 字典，用于传入所有发生位移的控制点，格式为{(i,j): delta_x, delta_y},
            其中 (i,j) 表示第i行，第j列的控制点，
            delta_x, delta_y 分别表示在第一个维度和第二个维度上发生的位移
    output:
        shifted_index: 一个三维张量，大小为 im_height * im_width * 2，记录原图的每一个像素在
            FFD形变后的新坐标
    """

    ## ---------------------第一步，定义用于计算权重的B样条核函数---------------
    def beta(a,index):
        if index == -1: return (1-a)**3 / 6
        if index == 0:  return (3*a**3 - 6 * a**2 + 4)/6
        if index == 1:  return (-3*a**3 + 3*a**2 + 3*a +1)/6
        if index == 2:  return a**3/6


    ## -------------第二步，构建用于存储所有控制点位移的defaultdict--------------
    def return_no_shift():
        return np.zeros(2)

    # 使用默认字典存储所有发生位移的控制点，可以节省存储空间
    # 而且可以巧妙避免后面计算位移加权和时可能发生的越界问题
    shift = defaultdict(return_no_shift)

    for item in shift_dict.items():
        i,j = item[0]
        delta_x,delta_y = item[1]
        shift[(i,j)] = np.array([delta_x,delta_y]) # 记录发生位移的控制点


    ## ---------------第三步，计算每个像素点在形变后的坐标----------------------

    lx = im_height/m # 网格单元大小
    ly = im_width/n

    # 用于记录每个【像素点】位移后的坐标

    # 初始化位移矩阵
    shifted_index = np.zeros((im_height,im_width,2))
    for x in  range(im_height):
        for y in  range(im_width):
            shifted_index[x,y,:] = np.array([x,y])

    for x in  range(im_height): # 遍历原图的每一个像素点
        dist_1 = (x-0)/lx
        ix =  int(dist_1)
        u = dist_1 - ix # 距离最近的左侧控制点的距离除以lx
```

```
60
61          for y in  range(im_width):
62              dist_2 = (y-0)/ly
63              iy =  int(dist_2)
64              v = dist_2 - iy # 距离最近的上方控制点的距离除以ly
65
66              for p in [-1,0,1,2]: # 对该像素点计算形变后的坐标
67                  for q in [-1,0,1,2]:
68                      shifted_index[x,y,:] += shift[(ix+p,iy+q)] * beta(u,p) * beta(v,q)
69                      # 由于使用defaultdict，因此不会发生越界问题；越界的部分都统一返回[0,0]
70
71      return shifted_index
```

2 反向图像变换

2.1 反向变换原理简介

设原始图的像素为 X，目标图的像素为 Y，满足 $Y = T(X)$。反向变换算法遍历目标图中的每一个像素坐标 $Y(i,j)$，并基于**反变换**求出在原始图对应的浮点型坐标 $X(p,q) = T^{-1}[Y(i,j)]$，再利用插值得到 $X(p,q)$ 处的灰度值 $f(p,q)$，并把 $Y(i,j)$ 处的灰度值设置为 $f(p,q)$。

在本次作业中，反变换是基于 FFD 形变算法实现的。由用户在**原图**上选取若干个点，不妨记作 $\{X_1,\dots,X_n\}$，然后指定这些点经过变换后在**目标图**上所在位置，不妨记作 $\{Y_1,\dots,Y_n\}$，也即在原图上的点 X_1 的坐标 $X_1(p,q)$，经过变换后，它的坐标将变成 Y_1 点的坐标 $Y_1(i,j)$。

将 $\{Y_1,\dots,Y_n\}$ 映射到最近的规则化控制点 $\{Z_1,\dots,Z_n\}$，并设控制点 Z_i 对应的位移为 $X_i(p,q)-Y_i(i,j)$。然后遍历所有的像素点的坐标 $Y(i,j)$，基于 FFD，找到在原图上的浮点型坐标 $X(p,q)$，进行插值并回填。这样就实现了基于 FFD 的反变换。

2.2 反向变换的 Python 实现

Listing 2: FFD 形变算法

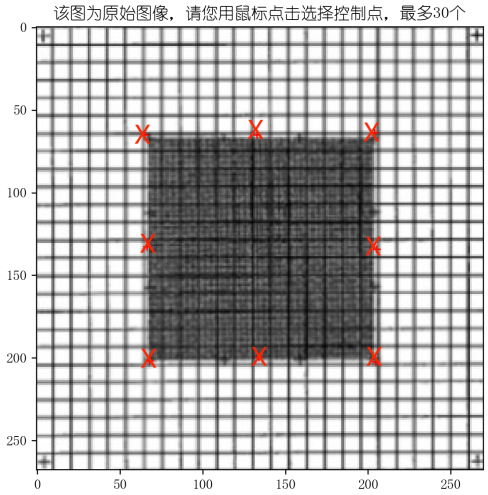
```
1  """实现基于FFD的图像反变换"""
2
3  from FFD import ffd # 自由形变算法
4  import numpy as np
5  import matplotlib
6  matplotlib.use('TkAgg') # 可交互
7  import matplotlib.pyplot as plt
8  matplotlib.rc("font",family='YouYuan') # 显示中文字体
9  import cv2 as cv
10 from collections import defaultdict
11
12 ## ----------------------------辅助函数---------------------------------
13 def interpolation(coord:np.array,ori_img:np.array):
14     """
15     线性插值函数
16     给定浮点型坐标和图像，找到最邻近的四个像素进行插值，对于越界的像素点直接截断
17     input:
```

```
        coord : the coordinates of the point
        ori_img: the original image
    output:
        value : the interpolation value
    """
    x , y = coord
    oh, ow = ori_img.shape

    # 对越界坐标进行截断
    if x < 0 : x = 0
    if x >= oh : x = oh-1
    if y < 0 : y = 0
    if y >= ow : y = ow-1

    # 找到距离最近的四个像素点位置
    min_x , min_y =  int(x) ,  int(y)
    max_x , max_y =  min(oh-1,min_x+1) ,  min(ow-1,min_y+1)

    # 线性插值
    u , v = x - min_x , y-min_y
    return (1-u)*(1-v)*ori_img[min_x,min_y] + u*(1-v)*ori_img[max_x,min_y] +\
          (1-u)*v*ori_img[min_x,max_y] + u*v*ori_img[max_x,max_y]


def backward_trans(ori_image:np.array, m: int, n: int, shift_dict: dict):
    """
    实现基于FFD的反变换
    input:
        ori_image:原始图
        m,n: 网格的高和宽，将会在【目标图】上划分出(m+1)*(n+1)的均匀等分控制点
        shift_dict: 字典，用于传入从【目标图】到【原始图】的控制点的位移，
            格式为{(i,j): delta_x, delta_y},
            其中 (i,j) 表示第i行，第j列的控制点，
            delta_x, delta_y 分别表示在第一个维度和第二个维度上发生的位移
    output:
        goal_image: 目标图
    """
    goal_image = np.zeros(ori_image.shape)
    shifted_index = ffd(goal_image.shape[0],goal_image.shape[1],m,n,shift_dict)
    for i in  range(goal_image.shape[0]):
        for j in  range(goal_image.shape[1]):
            goal_image[i,j] = interpolation(shifted_index[i,j,:],ori_image)

    return goal_image

## ----------------------读取图像，并由用户选取控制点----------------------------

fig = plt.figure(figsize=(16,12))
ori_img = cv.imread("grid.jpg",0)
goal_img = cv.imread("grid.jpg",0)
ori_img = cv.resize(ori_img,goal_img.shape)

```

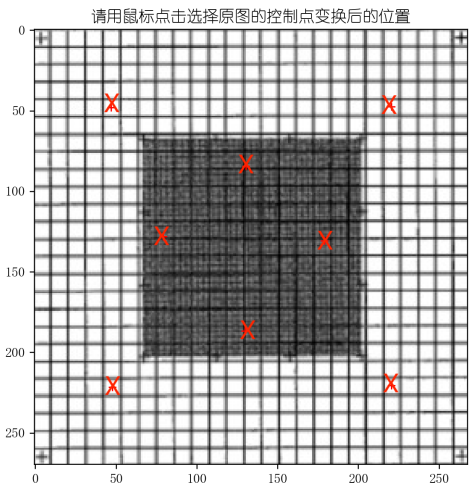
```
70
71  # 选取控制点
72  plt.subplot(121)
73  plt.imshow(ori_img,cmap=plt.get_cmap("gray"))
74  plt.title("该图为原始图像，请您用鼠标点击选择控制点，最多30个")
75  x = plt.ginput(n=30,timeout=0) # 回车结束选点
76  ori_list = np.float32([[ int(c[1]),
        int(c[0])] for c in x]) # 图像显示的坐标轴和numpy的矩阵坐标轴的位置正好相反
77
78  plt.subplot(122)
79  plt.imshow(goal_img,cmap=plt.get_cmap("gray"))
80  plt.title("请用鼠标点击选择原图的控制点变换后的位置")
81  y = plt.ginput(n=30,timeout=0) # 回车结束选点
82  goal_list = np.float32([[ int(c[1]),
        int(c[0])] for c in y]) # 图像显示的坐标轴和numpy的矩阵坐标轴的位置正好相反
83
84
85  ## -----------------将用户指定的控制点以及控制点位移转换为字典进行存储----------------------
86
87  def return_no_shift():
88      return np.zeros(2)
89  shift_dict = defaultdict(return_no_shift)  # 使用默认字典存储所有发生位移的控制点，可以节省存
        储空间
90  m = 10
91  n = 10
92  lx = goal_img.shape[1]/m
93  ly = goal_img.shape[0]/n
94  for k in  range( len(goal_list)):
95      p1 = goal_list[k] # 目标图上的控制点
96      p2 = ori_list[k]    # 原始图上的控制点
97      # 将任意选定的【目标图】上的控制点，移到最近的网格点上
98      ix =  round(p1[0]/lx)
99      iy =  round(p1[1]/ly)
100     shift_dict[(ix,iy)] = np.array([p2[0]-p1[0], p2[1]-p1[1]])
101
102
103 ## ----------------------基于FFD进行图像反变换--------------------------------
104
105 fig = plt.figure(figsize=(16,12))
106 plt.subplot(121)
107 plt.imshow(ori_img,cmap="gray")
108 # plt.axis("off")
109 plt.title("原始图像")
110
111 new_img = backward_trans(ori_img, m, n, shift_dict)  # 基于FFD进行图像反变换
112
113 plt.subplot(122)
114 plt.imshow(new_img,cmap="gray")
115 plt.title("形变图像")
116 # plt.axis("off")
117
118 plt.show() # 展示图像
```

2.3 效果测试

在原图上选取 8 个控制点，分别位于正方形的四个顶点，以及四条边的中点，如图1(a)所示。位于顶点的控制点向边缘移动，位于四条边中点的控制点向中心移动，如图1(b)所示。

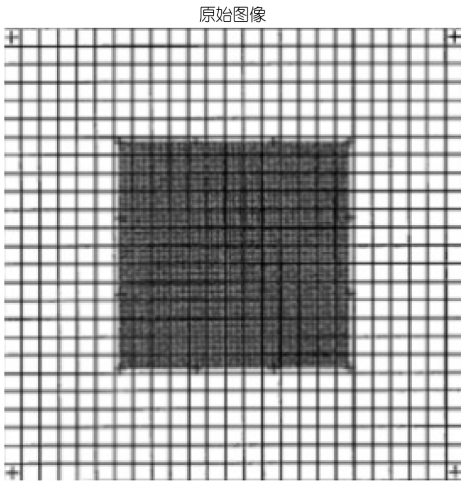


(a) 原图的控制点所处位置

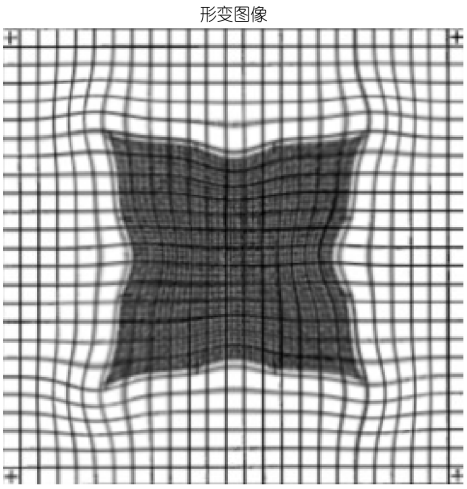


(b) 控制点变换后所处位置

以下是原图和形变图的对比。可以看出，形变的影响确实被控制在了局部范围内。



(c) 原图



(d) 形变图