作业六报告

陈乐偲·刘原冶·加兴华

1 题目叙述

1 任务：编程（1）实现基于FFD**或**局部仿射的形变算法，（2）实现**基于反向的**图像变换算法，从而实现图像甲到图像乙（如人脸到狒狒脸）的图像变换。作业的基本算法内容参考课堂上讲解和课件。**可以**参考其他有关的学术资料改进效果（optional），**但不能**只使用其他的算法而没有实现题目要求的两个基本方法。

2 可以独立完成，也可以组成小组（不超过3个成员）一起完成。

3 提交内容包括：（1）报告：在报告中清晰描述问题和数据，数据处理的各个步骤及中间结果，代码结构，开发环境，可执行文件使用手册等细节问题。如以小组为单位的请由一个同学提交，请**不要**多人重复提交；**请在报告里面说明成员的贡献。**（2）代码，代码要有非常清晰的注释。（3）数据（如果有用到）。（4）如果有可执行文件请顺便提交（optional）。

2 本报告介绍

首先最关键的，是本次我们小组的开发环境为python3.0+，需要安装第三方包matplotlib、numpy、opencv-pyhton、easygui、scikit-image。

在这次作业里，我们小组实现了基于局部仿射的形变算法，包含基于前向和基于后向的变换算法；而区域控制和点控制两种方式中，由于python处理图像速度较慢，我们选择了全部采用点控制的方式来防止运行时间过久。

为了更方便选点，我们在代码中添加了少量的图形用户界面（GUI）方便进行交互，并通过选取不同的控制点产生了许多有意思的结果。

如需运行我们小组的代码，请保证抬头的第三方包已经全部安装。

3 仿射变换

首先我们回顾了课堂知识，并查阅了网上相关的文章，最后比较认同这一网址文章的梳理：图像坐标空间变换：仿射变换（Affine Transformation）*吹吹自然风-CSDN博客*tensorflow 仿射变换

下面是我们从理论到代码的过程：

开始前我们需要进行**符号约定**：`(u,v)`用来表示原始图像中的坐标，`(x,y)`用来表示变换后图像的坐标。

仿射变换通式

我们可以将各种基础变换融合起来用一个式子表达，也就是仿射变换的通式，其线性方程组形式为：

$$\begin{cases} x = t_{11} * u + t_{21} * v + t_{31} \\ y = t_{12} * u + t_{22} * v + t_{32} \end{cases}$$

也可以写成矩阵形式：

$$\begin{bmatrix} x & y & 1 \end{bmatrix} = \begin{bmatrix} u & v & 1 \end{bmatrix} \begin{bmatrix} t_{11} & t_{12} & 0 \\ t_{21} & t_{22} & 0 \\ t_{31} & t_{32} & 1 \end{bmatrix}$$

在python中，我们通过定义以下函数实现生成变换矩阵

```python
# 得到控制点的变换矩阵T
def get_matrix_T(x_src, y_src, x_dst, y_dst):
    # 获取控制点个数
    num_of_point = len(x_src)
    # 初始化T
    T = np.zeros((num_of_point, 3, 3))
    # 得到各个控制点的变换矩阵，实质是一个平移矩阵
    for i in range(num_of_point):
        T[i] = np.array([[1, 0, x_dst[i] - x_src[i]], [0, 1, y_dst[i] - y_src[i]], [0, 0, 1]])
    return T
```

一个需要小心的坑：图像索引与坐标的关系

在推导公式的时候，我们通常不需要引入 图像索引 这个概念，也就是我们常用的(i, j)，但是写代码的时候就必然得用到了，这个时候需要特别小心图像索引与坐标的对应关系。图像索引中我们常常使用 `i` 来表示行，用 `j` 来表示列，此时需注意以下对应关系：

```
i --> 行 --> y
j --> 列 --> x
```

所以像素索引为(i, j)的点的坐标为(y, x)。这个对应关系如果搞错了的话，可能会发生莫名其妙的错误，比如明明我们想要顺时针旋转图片，结果却是逆时针旋转。

对于这部分，我们的代码在选取图像像素作为控制点时做了转置处理：

```python
for i in range(0, n):
    x_src.append(round(pos[i][1])) # 行数存的是纵坐标
    y_src.append(round(pos[i][0])) # 列数存的是横坐标
```

在这次作业中，我们并非对整张图做全局仿射，而是除了控制区域和点以外都做一个加权处理：

局部仿射：区域控制+全局影响

- 假设有n个局部区域（坐标点）U_i，每个区域对应的局部仿射变换为G_i，则局部仿射空间变换公式由下式计算

$$T(X)=\begin{cases} G_i(X), & X\in U_i,\ i=1\ldots n \\ \sum_{i=1}^{n} w_i(X)G_i(X_i), & X\notin \bigcup_{i=1}^{n} U_i \end{cases} \qquad w_i(X)=\left(1/d_i(X)^e\right)\Big/\left(\sum_{i=1}^{n}1/d_i(X)^e\right)$$

- 其中$d_i(X)$表示X到U_i的距离。

为计算控制点权重，我们写了一个函数"get_weight_by_dist"，具体如下：

```python
# 获得距离
def get_dist(h, w, x, y):
    dist = np.sqrt((h - x) ** 2 + (w - y) ** 2)
    # 防止距离为0影响后续计算
    eps = 1e-8
    if dist <= eps:
        return eps
    else:
        return dist

# 得到各点到各个控制点的权重值
def get_weight_by_dist(h, w, X, Y, e):
    num_of_point = X.shape[0]
    # 权重矩阵
    W = np.zeros((num_of_point, 1))
    for i in range(num_of_point):
        x, y = X[i], Y[i]
        W[i] = 1 / (get_dist(h, w, x, y)**e)
    W /= np.sum(W)
    return W
```

前向映射

前向映射

将五组对应的点代入`仿射变换通式`可得：

$$\begin{cases} x_1 = t_{11} * u_1 + t_{21} * v_1 + t_{31} * 1 \\ y_1 = t_{12} * u_1 + t_{22} * v_1 + t_{32} * 1 \\ x_2 = t_{11} * u_2 + t_{21} * v_2 + t_{31} * 1 \\ y_2 = t_{12} * u_2 + t_{22} * v_2 + t_{32} * 1 \\ ..... \\ x_5 = t_{11} * u_5 + t_{21} * v_5 + t_{31} * 1 \\ y_5 = t_{12} * u_5 + t_{22} * v_5 + t_{32} * 1 \end{cases}$$

简而言之，前向映射是遍历原图像素点，算出其变换后的位置再做整数化处理，我们实现的代码如下：

```python
# 前向变换
def forward_transform(img, x_src, y_src, x_dst, y_dst, e):

    new_img = np.zeros_like(img)
    H, W, C = new_img.shape

    # 对每个像素点标志是否已经填过
    is_paint = np.zeros((H,W),dtype='uint8')

    T = get_matrix_T(x_src, y_src, x_dst, y_dst)

    # 逐点变换
    for h in range(H):
        for w in range(W):
            nh, nw = get_mean_transform_point(h, w, T, x_src, y_src, e)
            nh, nw = int(np.round(nh)), int(np.round(nw))
            if nh < 0 or nw < 0 or nh >= H or nw >= W:
                continue
            new_img[nh, nw] = img[h, w]
            # 已经填过的点标记一下
            is_paint[nh, nw] = 1

    # 对没有填的区域进行修复
    inpaint_mask = 1 - is_paint
    new_img = cv2.inpaint(new_img, inpaint_mask, 13, cv2.INPAINT_NS)

    # blur
    new_img = cv2.blur(new_img, (5, 5))

    # median blur for denoise
    new_img = cv2.medianBlur(new_img, 7)
    return new_img
```

代码中涉及如何获取仿射后位置的函数"get_mean_transform_point"，具体如下：

```python
# 得到各点仿射后的位置
def get_mean_transform_point(h, w, T, X, Y, e):
    num = T.shape[0]
    W = get_weight_by_dist(h, w, X, Y, e)

```

```
    D = np.zeros((num, 2))
    # 计算出受各个控制点的变换矩阵T所应该映射到的位置
    for i in range(num):
        scr_point = np.array([h, w, 1]).transpose()
        dst_point = np.matmul(T[i], scr_point)
        dst_point = dst_point.transpose()
        # get new_h and new_w
        nh, nw, _ = dst_point
        D[i] = np.array([nh, nw])

    mean_d = np.zeros(2)
    # 根据权重计算在所有控制点作用下的位置
    for i in range(num):
        D[i] = W[i] * D[i]
        mean_d += D[i]

    return mean_d
```

后向映射

后向映射

后向映射 从变换后图像的坐标(x,y)出发，经矩阵变换，得到其在原始图像中对应的坐标(u,v)，后向映射中(x,y)是整数，(u,v)一般是浮点数。

后向映射与前向映射的推导基本雷同，只需将(x,y)与(u,v)的位置调换一下就可以，下面罗列一下推导流程。
将五组对应的点代入通式：

$$\begin{cases} u = t_{11} * x + t_{21} * y + t_{31} \\ v = t_{12} * x + t_{22} * y + t_{32} \end{cases}$$

简而言之，后向映射就是遍历变换后图像的坐标，做变换到原始图像空间中的某一位置，通过插值获得具体的像素值。

之前作业已经实现过插值，这里再次给出插值函数"biInterpolate"，代码如下：

```
# 双线性插值
def biInterpolate(x, i, j):
    rows, cols, C = x.shape
    up = int(np.floor(i))
    down = int(np.ceil(i))
    left = int(np.floor(j))
    right = int(np.ceil(j))
    if up < 0 or left < 0 or down >= rows or right >= cols:
        return 0
    u, v = i - up, j - left
    y = u * v * x[up, left] + u * (1 - v) * x[up, right] + (1 - u) * v *
x[down, left] + (1 - u) * (1 - v) * x[
        down, right]
    return y
```

而后，我们的后向映射函数如下：

```
# 后向变换
def backward_transform(img, x_src, y_src, x_dst, y_dst, e):
    new_img = np.zeros_like(img)
```

```
    H, W, C = new_img.shape
    T = get_matrix_T(x_src, y_src, x_dst, y_dst)

    for h in range(H):
        for w in range(W):
            nh, nw = get_mean_transform_point(h, w, T, x_src, y_src, e)

            img_gray_by_interpotate = biInterpolate(img, nh, nw)

            new_img[h, w] = img_gray_by_interpotate

    # 模糊化
    new_img = cv2.blur(new_img, (3, 3))
    return new_img
```

4 人脸图像变换

1 图像预处理

因为原图和模板图往往会尺寸不一致，因此我们在加载图片后对模板图进行了resize操作，可能会使得模板图比例失调，但没有大碍：

```
# 图片加载
img_monkey = io.imread('imgs/monkey.png')
img_woman = io.imread('imgs/woman2.jpg')
# 尺寸相同化
img_monkey =cv2.resize(img_monkey, img_woman.shape[:2])
```

2 控制点选取

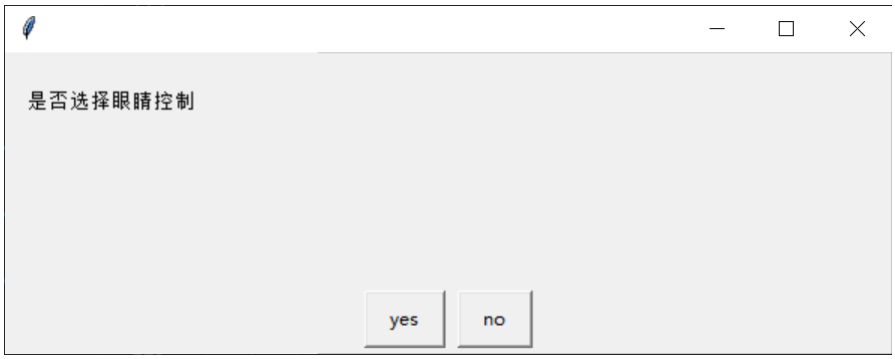
为了灵活取点，我们的代码包含了少量GUI方便与用户进行交互，并将选点行为分解为对眼睛、鼻子、嘴巴分别询问，减少用户在原图和模板图上去点顺序不一致发生的概率，另外，为保证变形后图像不会有太多黑边，已默认在四个角创建控制点，代码如下：

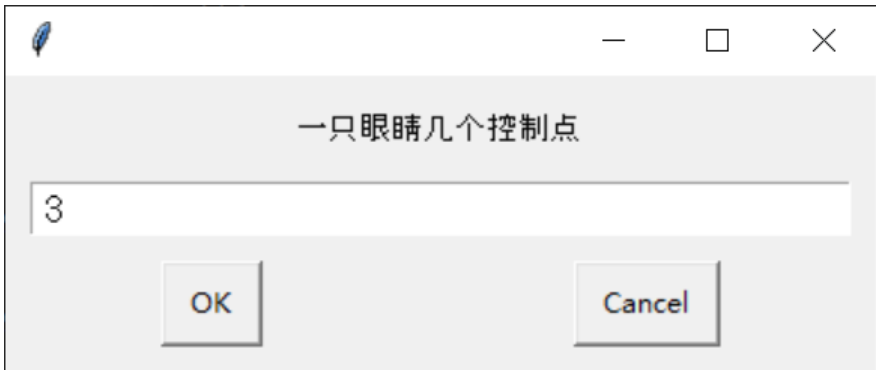
```
# 变换控制点的坐标列表
    x_src = []
    y_src = []
    x_dst = []
    y_dst = []
    # ---------------选择控制点
    # 眼睛
    con = ccbox(msg='是否选择眼睛控制', title=' ', choices=(' yes ', ' no '), image=None)
    if con == True:
        n = enterbox(msg=' 一只眼睛几个控制点 ', title=' ', default=' ', strip=True, image=None, root=None)
        n = int(n)
        plt.imshow(img_woman)
        plt.title('Choose Left Eye Control points')
        pos = plt.ginput(n)
        for i in range(0, n):
            x_src.append(round(pos[i][1]))
            y_src.append(round(pos[i][0]))
        plt.close()
```

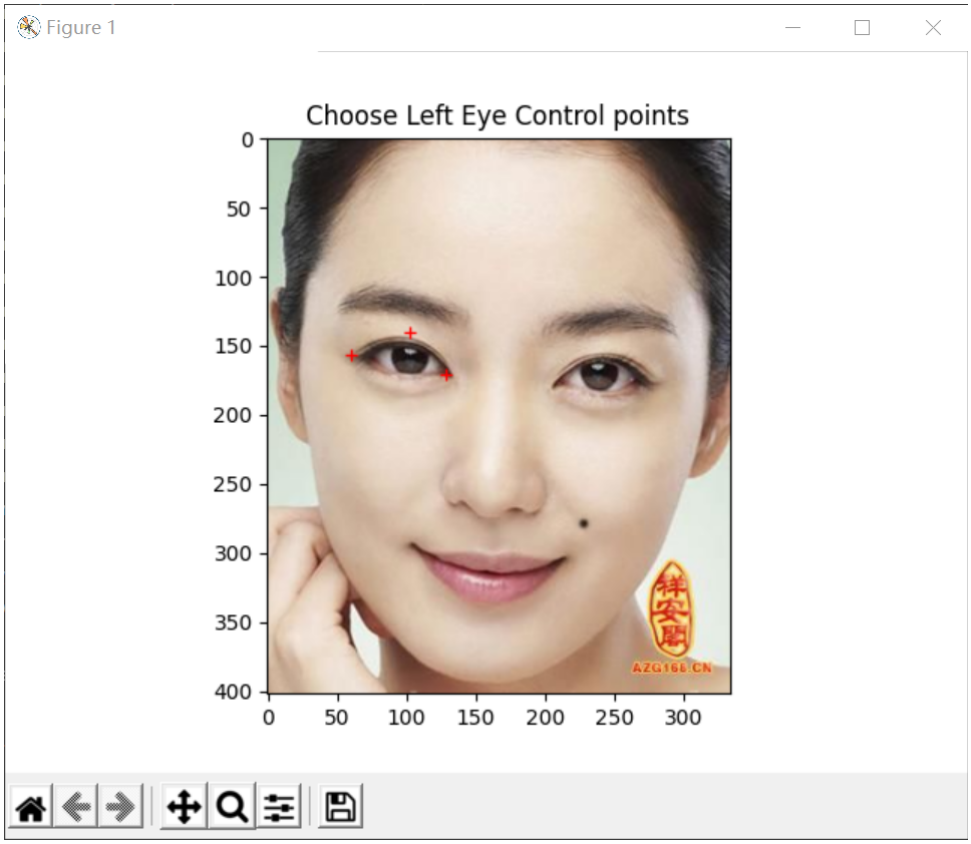
```
        plt.imshow(img_monkey)
        plt.title('Choose Left Eye Control points')
        pos = plt.ginput(n)
        for i in range(0, n):
            x_dst.append(round(pos[i][1]))
            y_dst.append(round(pos[i][0]))
        plt.close()
        plt.imshow(img_woman)
        plt.title('Choose Right Eye Control points')
        pos = plt.ginput(n)
        for i in range(0, n):
            x_src.append(round(pos[i][1]))
            y_src.append(round(pos[i][0]))
        plt.close()
        plt.imshow(img_monkey)
        plt.title('Choose Right Eye Control points')
        pos = plt.ginput(n)
        for i in range(0, n):
            x_dst.append(round(pos[i][1]))
            y_dst.append(round(pos[i][0]))
        plt.close()
        # 鼻子
    con2 = ccbox(msg='是否选择鼻子控制', title=' ', choices=(' yes ', ' no '), image=None)
    if con2 == True:
        n = enterbox(msg=' 鼻子几个控制点 ', title=' ', default=' ', strip=True, image=None, root=None)
        n = int(n)
        plt.imshow(img_woman)
        plt.title('Choose Nose Control points')
        pos = plt.ginput(n)
        for i in range(0, n):
            x_src.append(round(pos[i][1]))
            y_src.append(round(pos[i][0]))
        plt.close()
        plt.imshow(img_monkey)
        plt.title('Choose Nose Control points')
        pos = plt.ginput(n)
        for i in range(0, n):
            x_dst.append(round(pos[i][1]))
            y_dst.append(round(pos[i][0]))
        plt.close()
        # 嘴巴
    con3 = ccbox(msg='是否选择嘴巴控制', title=' ', choices=(' yes ', ' no '), image=None)
    if con3 == True:
        n = enterbox(msg=' 嘴巴几个控制点 ', title=' ', default=' ', strip=True, image=None, root=None)
        n = int(n)
        plt.imshow(img_woman)
        plt.title('Choose Mouth Control points')
        pos = plt.ginput(n)
        for i in range(0, n):
            x_src.append(round(pos[i][1]))
            y_src.append(round(pos[i][0]))
        plt.close()
        plt.imshow(img_monkey)
        plt.title('Choose Mouth Control points')
```

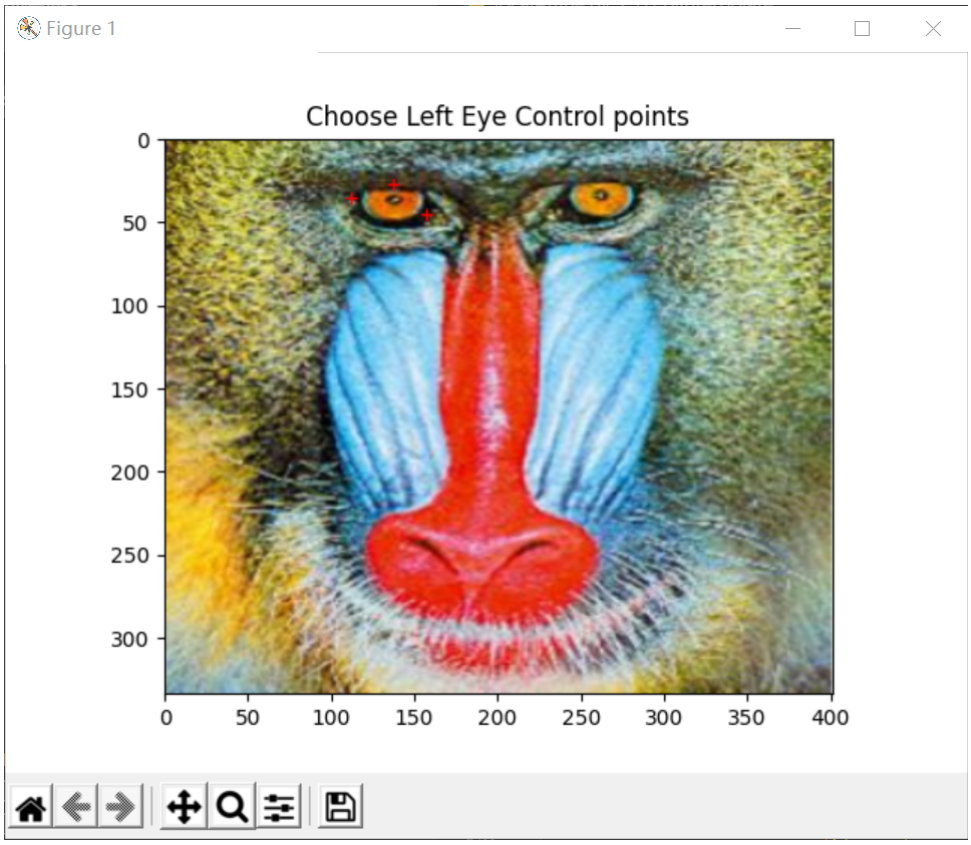
```
        pos = plt.ginput(n)
        for i in range(0, n):
            x_dst.append(round(pos[i][1]))
            y_dst.append(round(pos[i][0]))
        plt.close()
    # 额外设置四个角的控制点
    [a1,b1]=img_woman.shape[:2]
    x_src.extend([0,0,a1,a1])
    y_src.extend([0,b1,0,b1])
    x_dst.extend([0,0,a1,a1])
    y_dst.extend([0,b1,0,b1]
    # 得到控制点坐标
    x_src2 = np.array(x_src)
    x_dst2 = np.array(x_dst)
    y_src2 = np.array(y_src)
    y_dst2 = np.array(y_dst)
```

运行时实际反馈如下：（只展示选取左眼流程）









3 调用函数处理

在选取好控制点后，调用第三部分所述代码即可将图片变形，随后保存到本地：

```
    # 参数e(权重公式中的)
    e = 2
    # 前向
    new_img_forward = forward_transform(img_woman, x_src2, y_src2, x_dst2, y_dst2, e)
    # 后向
    new_img_back = backward_transform(img_woman, x_dst2, y_dst2, x_src2, y_src2, e)
    savepath_forward = 'results/forward_result-6112032.png'
    savepath_back = 'results/back_result-6112032.png'
    io.imsave(savepath_forward, new_img_forward)
    io.imsave(savepath_back, new_img_back)
```

结果如下：

前向图

后向图

结果分析：

首先需要说明，眼睛和鼻子均为四点控制，嘴巴为五点控制，dist中的权重e设置为2；其次两种方式中都有对边缘进行一定修复。

结果上来看，后向变换视觉上更加清晰与顺滑，但是边缘的损坏也比前向要大，产生了波纹效应，总体上是瑕不掩瑜。