实验 3 图像映射实验

编写代码完成以下实验内容，请将代码和运行效果截图插入到问题下方。

本次实验主要练习图像之间的变换，以及计算变换的一些方法。这些变化可以用于图像扭曲变形和图像配准。

第一部分 单应性变换

单应性变换是将一个平面内的点映射到另一个平面内的二维投影变换。这里的平面是指图像或者三维中的平面表面。单应性变换具有很强的实用性，比如图像配准、图像纠正和纹理扭曲。

图像预处理

1、 编写归一化函数 normalize(points)和齐次坐标转换函数 make_homog(points)，实现对点进行归一化和转换齐次坐标的功能。

归一化函数要求：在齐次坐标意义下，对点集进行归一化，使最后一行为 1。

In [1]:
```
from PIL import Image
from pylab import *
from numpy import *
import os
from scipy import ndimage


def normalize(points):
    for row in points:
        row /= points[-1]
    return points


def make_homog(points):
    return vstack((points, ones((1, points.shape[1]))))
```

齐次坐标转换函数要求：将点集转化为齐次坐标表示。

直接线性变化算法

2、 单应性矩阵可以由两幅图像中对应点对计算出来。一个完全射影变换具有 8 个自由度。根据对应点约束，每个对应点对可以写出两个方程，分别对应于 x 和 y 坐标。因此，计算单应性矩阵 H 需要 4 个对应点对。

DLT（Direct Linear Transformation，直接线性变换）是给定 4 个或者更多对应点对矩阵，来计算单应性矩阵 H 的算法。可以通过 SVD（Singular Value Decomposition，奇异值分解）算法来找到 H 的最小二乘解。

编写函数 H_from_points(fp,tp)，使用线性 DLT 方法计算单应性矩阵 H，使 fp 映射到 tp，点自动进行归一化。

参考代码如下：

```
def H_from_points(fp,tp):
    """ Find homography H, such that fp is mapped to tp
        using the linear DLT method. Points are conditioned
        automatically. """

    if fp.shape != tp.shape:
        raise RuntimeError('number of points do not match')

    # condition points (important for numerical reasons)
    # --from points--
    m = mean(fp[:2], axis=1)
    maxstd = max(std(fp[:2], axis=1)) + 1e-9
    C1 = diag([1/maxstd, 1/maxstd, 1])
    C1[0][2] = -m[0]/maxstd
    C1[1][2] = -m[1]/maxstd
    fp = dot(C1,fp)

    # --to points--
    m = mean(tp[:2], axis=1)
    maxstd = max(std(tp[:2], axis=1)) + 1e-9
    C2 = diag([1/maxstd, 1/maxstd, 1])
    C2[0][2] = -m[0]/maxstd
    C2[1][2] = -m[1]/maxstd
    tp = dot(C2,tp)

    # create matrix for linear method, 2 rows for each correspondence pair
    nbr_correspondences = fp.shape[1]
    A = zeros((2*nbr_correspondences,9))
    for i in range(nbr_correspondences):
        A[2*i] = [-fp[0][i],-fp[1][i],-1,0,0,0,
                  tp[0][i]*fp[0][i],tp[0][i]*fp[1][i],tp[0][i]]
        A[2*i+1] = [0,0,0,-fp[0][i],-fp[1][i],-1,
                    tp[1][i]*fp[0][i],tp[1][i]*fp[1][i],tp[1][i]]

    U,S,V = linalg.svd(A)
    H = V[8].reshape((3,3))

    # decondition
    H = dot(linalg.inv(C2),dot(H,C1))

    # normalize and return
    return H / H[2,2]
```

In [2]:

```
def H_from_points(fp, tp):
    if fp.shape != tp.shape:
        raise RuntimeError('number of points do not match')
    m = mean(fp[:2], axis=1)
    maxstd = max(std(fp[:2], axis=1)) + 1e-9
    C1 = diag([1 / maxstd, 1 / maxstd, 1])
    C1[0][2] = -m[0] / maxstd
    C1[1][2] = -m[1] / maxstd
    fp = dot(C1, fp)
    m = mean(tp[:2], axis=1)
    maxstd = max(std(tp[:2], axis=1)) + 1e-9
    C2 = diag([1 / maxstd, 1 / maxstd, 1])
    C2[0][2] = -m[0] / maxstd
    C2[1][2] = -m[1] / maxstd
    tp = dot(C2, tp)
    nbr_correspondences = fp.shape[1]
    A = zeros((2 * nbr_correspondences, 9))
    for i in range(nbr_correspondences):
        A[2 * i] = [-fp[0][i], -fp[1][i], -1, 0, 0, 0, tp[0][i] * fp[0][i], tp[0][i] * fp[1][i], tp[0][i]]
        A[2 * i + 1] = [0, 0, 0, -fp[0][i], -fp[1][i], -1, tp[1][i] * fp[0][i], tp[1][i] * fp[1][i], tp[1][i]]
    U, S, V = linalg.svd(A)
    H = V[8].reshape((3, 3))
    H = dot(linalg.inv(C2), dot(H, C1))
    return H / H[2, 2]
```

仿射变换

3、 放射变化具有 6 个自由度，需要三个对应点对来估计矩阵 H。通过将最后两个元素设置为 0，仿射变换可以用前面的 DLT 算法来估计得出。编写函数使用对应点对来计算仿射变换矩阵。

参考代码如下：

```
def Haffine_from_points(fp,tp):
    """ Find H, affine transformation, such that
        tp is affine transf of fp. """

    if fp.shape != tp.shape:
        raise RuntimeError('number of points do not match')

    # condition points
    # --from points--
    m = mean(fp[:2], axis=1)
    maxstd = max(std(fp[:2], axis=1)) + 1e-9
    C1 = diag([1/maxstd, 1/maxstd, 1])
    C1[0][2] = -m[0]/maxstd
    C1[1][2] = -m[1]/maxstd
    fp_cond = dot(C1,fp)

    # --to points--
    m = mean(tp[:2], axis=1)
    C2 = C1.copy() #must use same scaling for both point sets
    C2[0][2] = -m[0]/maxstd
    C2[1][2] = -m[1]/maxstd
    tp_cond = dot(C2,tp)

    # conditioned points have mean zero, so translation is zero
    A = concatenate((fp_cond[:2],tp_cond[:2]), axis=0)
    U,S,V = linalg.svd(A.T)

    # create B and C matrices as Hartley-Zisserman (2:nd ed) p 130.
    tmp = V[:2].T
    B = tmp[:2]
    C = tmp[2:4]

    tmp2 = concatenate((dot(C,linalg.pinv(B)),zeros((2,1))), axis=1)
    H = vstack((tmp2,[0,0,1]))

    # decondition
    H = dot(linalg.inv(C2),dot(H,C1))

    return H / H[2,2]
```

In [3]:
```
def Haffine_from_points(fp, tp):
    if fp.shape != tp.shape:
        raise RuntimeError('number of points do not match')
    m = mean(fp[:2], axis=1)
    maxstd = max(std(fp[:2], axis=1)) + 1e-9
    C1 = diag([1 / maxstd, 1 / maxstd, 1])
    C1[0][2] = -m[0] / maxstd
    C1[1][2] = -m[1] / maxstd
    fp_cond = dot(C1, fp)
    m = mean(tp[:2], axis=1)
    C2 = C1.copy()
    C2[0][2] = -m[0] / maxstd
    C2[1][2] = -m[1] / maxstd
    tp_cond = dot(C2, tp)
    A = concatenate((fp_cond[:2], tp_cond[:2]), axis=0)
    U, S, V = linalg.svd(A.T)
    tmp = V[:2].T
    B = tmp[:2]
    C = tmp[2:4]
    tmp2 = concatenate((dot(C, linalg.pinv(B)), zeros((2, 1))), axis=1)
    H = vstack((tmp2, [0, 0, 1]))
    H = dot(linalg.inv(C2), dot(H, C1))
    return H / H[2, 2]
```

第二部分 图像扭曲

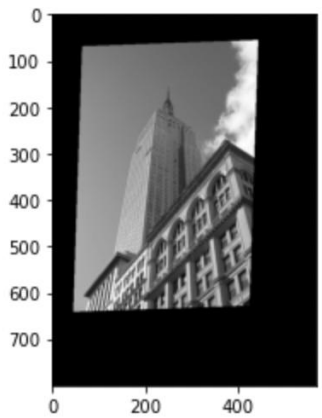
对图像块应用仿射变换即：图像扭曲。该操作经常应用在计算机图形学和计算机视觉的算法中。

1、 **扭曲操作**可以调用 SciPy 工具包中的 ndimage 包来完成：

transformed_im = ndimage.affine_transform(im,A,b,size)

如上所示，使用一个线性变化 A 和一个平移向量 b 来对图像块应用仿射变换。选项参数 size 可以用来指定输出图像的大小。默认输出图像设置为和原始图像同样大小。编写代码调用 SciPy 工具包中的函数用仿射变化来扭曲图像”empire.jpg”，并输出结果。

In [5]:
```
from scipy import ndimage
im = array(Image.open('../pics3/empire.jpg').convert('L'))
H = array([[1.4, 0.05, -100], [0.05, 1.5, -100], [0, 0, 1]])
im2 = ndimage.affine_transform(im, H[:2, :2], (H[0, 2], H[1, 2]))
figure()
gray()
imshow(im2)
show()
```



2、 图像中的图像：利用仿射扭曲将一幅图像放置在另一幅图像中。编写函数

image_in_image(im1,im2,tp)，输入为两幅图像和一个坐标，该坐标为将第一幅图像放置到第二幅图像中的角点位置。

In [11]:
```
def image_in_image(im1, im2, tp):
    m, n = im1.shape[:2]
    fp = array([[0, m, m, 0], [0, 0, n, n], [1, 1, 1, 1]])
    H = Haffine_from_points(tp, fp)
    im1_t = ndimage.affine_transform(im1, H[:2, :2],
                                     (H[0, 2], H[1, 2]), im2.shape[:2])
    alpha = (im1_t > 0)
    return (1 - alpha) * im2 + alpha * im1_t
```

3、 编写代码将一幅图像插入到另一幅图像中的特定位置，并将结果绘制出来。

In [73]:
```
im1 = array(Image.open('../pics3/Beatles.jpg').convert('L'))
im2 = array(Image.open('../pics3/turningtorso1.jpg').convert('L'))
# 选定一些目标点
tp = array([[162,313, 305, 143], [66, 86, 264, 229], [1, 1, 1, 1]])
im3 = image_in_image(im1, im2, tp)
figure()
gray()
imshow(im3)
axis('equal')
axis('off')
show()
```

4、 对于三个点，仿射变化可以将一幅图像进行扭曲，使这三对对应点对可以完美地匹

配上。可以将图像分成两个三角形，然后对它们分别进行扭曲图像操作。三角形的 alpha 图像可以简单地通过检查像素的坐标是否能够写成三角形顶点坐标的凸组合来计算。

编写函数 alpha_for_triangle(points,m,n)，实现对于带有由 points 定义角点的三角形，创建大小为(m,n)的 alpha 图。

参考代码如下：

```
def alpha_for_triangle(points,m,n):
    """ Creates alpha map of size (m,n)
        for a triangle with corners defined by points
        (given in normalized homogeneous coordinates). """

    alpha = zeros((m,n))
    for i in range(min(points[0]),max(points[0])):
        for j in range(min(points[1]),max(points[1])):
            x = linalg.solve(points,[i,j,1])
            if min(x) > 0: #all coefficients positive
                alpha[i,j] = 1
    return alpha
```

In [13]:
```
def alpha_for_triangle(points, m, n):
    alpha = zeros((m, n))
    for i in range(int(min(points[0])), int(max(points[0]))):
        for j in range(int(min(points[1])), int(max(points[1]))):
            x = linalg.solve(points, [i, j, 1])
            if min(x) > 0:
                alpha[i, j] = 1
    return alpha
```

5、 编写代码实现：使用包含两个三角形的仿射变换来实现将一个图像放置到另一个图像中。给出代码，并绘制出效果图像。

In [78]:
```
im1 = array(Image.open('../pics3/beatles.jpg').convert('L'))
im2 = array(Image.open('../pics3/turningtorso1.jpg').convert('L'))
m,n = im1.shape[:2]
fp = array([[0,m,m,0],[0,0,n,n],[1,1,1,1]])
tp2 = tp[:,:3]
fp2 = fp[:,:3]
H = Haffine_from_points(tp2,fp2)
im1_t = ndimage.affine_transform(im1,H[:2,:2],
                                 (H[0,2],H[1,2]),im2.shape[:2])
alpha = alpha_for_triangle(tp2,im2.shape[0],im2.shape[1])
im3 = (1-alpha)*im2 + alpha*im1_t
tp2 = tp[:,[0,2,3]]
fp2 = fp[:,[0,2,3]]
H = Haffine_from_points(tp2,fp2)
im1_t = ndimage.affine_transform(im1,H[:2,:2],
                                 (H[0,2],H[1,2]),im2.shape[:2])
alpha = alpha_for_triangle(tp2,im2.shape[0],im2.shape[1])
im4 = (1-alpha)*im3 + alpha*im1_t
figure()
gray()
imshow(im4)
axis('equal')
axis('off')
show()
```

分段仿射扭曲：给定任意图像的标记点，通过将这些点进行三角剖分，然后使用仿射扭曲来扭曲每个三角形，可以将图像和另一幅图像的对应标记点扭曲对应

6、 为了三角化这些点，通常使用狄洛克三角剖分法。狄洛克三角剖分选择一些三角形，使三角剖分中所有三角形的最小角度最大。可以调用 SciPy 包中的相关方法。编写三角剖分的函数。

参考代码如下：

```
from scipy.spatial import Delaunay
def triangulate_points(x,y):
    """ Delaunay triangulation of 2D points. """

    tri = Delaunay(np.c_[x,y]).simplices
    return tri
```

```
In [15]: from scipy.spatial import Delaunay
         def triangulate_points(x, y):
             tri = Delaunay(np.c_[x, y]).simplices
             return tri
```

7、 编写一个用于分段仿射图像扭曲的通用扭曲函数。

参考代码如下：

```
def pw_affine(fromim,toim,fp,tp,tri):
    """ Warp triangular patches from an image.
        fromim = image to warp
        toim = destination image
        fp = from points in hom. coordinates
        tp = to points in hom. coordinates
        tri = triangulation. """

    im = toim.copy()

    # check if image is grayscale or color
    is_color = len(fromim.shape) == 3

    # create image to warp to (needed if iterate colors)
    im_t = zeros(im.shape, 'uint8')

    for t in tri:
        # compute affine transformation
        H = Haffine_from_points(tp[:,t],fp[:,t])

        if is_color:
            for col in range(fromim.shape[2]):
                im_t[:,:,col] = ndimage.affine_transform(
                    fromim[:,:,col],H[:2,:2],(H[0,2],H[1,2]),im.shape[:2])
        else:
            im_t = ndimage.affine_transform(
                    fromim,H[:2,:2],(H[0,2],H[1,2]),im.shape[:2])

        # alpha for triangle
        alpha = alpha_for_triangle(tp[:,t],im.shape[0],im.shape[1])

        # add triangle to image
        im[alpha>0] = im_t[alpha>0]

    return im
```

In [19]:
```
def pw_affine(fromim, toim, fp, tp, tri):
    im = toim.copy()
    is_color = len(fromim.shape) == 3
    im_t = zeros(im.shape, 'uint8')
    for t in tri:
        H = Haffine_from_points(tp[:, t], fp[:, t])
        if is_color:
            for col in range(fromim.shape[2]):
                im_t[:, :, col] = ndimage.affine_transform(
                    fromim[:, :, col], H[:2, :2], (H[0, 2], H[1, 2]), im.shape[:2])
        else:
            im_t = ndimage.affine_transform(
                fromim, H[:2, :2], (H[0, 2], H[1, 2]), im.shape[:2])
        alpha = alpha_for_triangle(tp[:, t], im.shape[0], im.shape[1])
        im[alpha > 0] = im_t[alpha > 0]

    return im
```

8、 编写一个绘制出三角形的辅助函数。

参考代码如下:

```
def plot_mesh(x,y,tri):
    """ Plot triangles. """

    for t in tri:
        t_ext = [t[0], t[1], t[2], t[0]] # add first point to end
        plot(x[t_ext],y[t_ext],'r')
```

```
In [20]: def plot_mesh(x, y, tri):
             for t in tri:
                 t_ext = [t[0], t[1], t[2], t[0]]
                 plot(x[t_ext], y[t_ext], 'r')
```

9、 编写代码使用狄洛克三角剖分标记点对图像“sunset_tree.jpg”进行分段仿射扭曲，并将结果绘制出来。

```
In [21]: fromim = array(Image.open('../pics3/sunset_tree.jpg'))
x, y = meshgrid(range(5), range(6))
x = (fromim.shape[1] / 4) * x.flatten()
y = (fromim.shape[0] / 5) * y.flatten()
tri = triangulate_points(x, y)
im = array(Image.open('../pics3/turningtorso1.jpg'))
tp = loadtxt('../pics3/turningtorso1_points.txt')  # destination points
fp = vstack((y, x, ones((1, len(x)))))
tp = vstack((tp[:, 1], tp[:, 0], ones((1, len(tp)))))
im = pw_affine(fromim, im, fp, tp, tri)
figure()
imshow(im)
plot_mesh(tp[1], tp[0], tri)
axis('off')
show()
```

图像配准：是对图像进行变换，使变换后的图像能够在常见的坐标系中对齐。
编写代码使用狄洛克三角剖分标记点对图像“sunset_tree.jpg”进行分段仿射扭曲，并将结果绘制出来。

下面进行多个人脸图像的严格配准。图像中的人脸并不都有相同的大小、位置和方向。所以，这类配准中，实际是寻找一个相似变换，在对应点对之间建立映射。

任务描述：在 jkface.zip 文件中有 366 幅单人图像。这些图像都对眼睛和嘴的坐标进行了标记，结果保存在 jkface.xml 文件中。使用这些点，可以计算出一个相似变换，然后将可以使用该变换的图像扭曲到一个归一化的坐标系中。

10、 使用 Python 的内置 xml.dom 模块中的 minidom 类库，编写函数实现读取用于人脸对齐的控制点。这些标记点会在 Python 中以字典的形式返回，字典的键值为图像的文件名。格式为：图像中左眼的坐标为 xf 和 yf，右眼的坐标为 xs 和 ys，嘴的坐标为 xm 和 ym。

```
def read_points_from_xml(xmlFileName):
    """ Reads control points for face alignment. """

    xmldoc = minidom.parse(xmlFileName)
    facelist = xmldoc.getElementsByTagName('face')
    faces = {}
    for xmlFace in facelist:
        fileName = xmlFace.attributes['file'].value
        xf = int(xmlFace.attributes['xf'].value)
        yf = int(xmlFace.attributes['yf'].value)
        xs = int(xmlFace.attributes['xs'].value)
        ys = int(xmlFace.attributes['ys'].value)
        xm = int(xmlFace.attributes['xm'].value)
        ym = int(xmlFace.attributes['ym'].value)
        faces[fileName] = array([xf, yf, xs, ys, xm, ym])
    return faces
```

In [22]:
```
from xml.dom import minidom
def read_points_from_xml(xmlFileName):
    xmldoc = minidom.parse(xmlFileName)
    facelist = xmldoc.getElementsByTagName('face')
    faces = {}
    for xmlFace in facelist:
        fileName = xmlFace.attributes['file'].value
        xf = int(xmlFace.attributes['xf'].value)
        yf = int(xmlFace.attributes['yf'].value)
        xs = int(xmlFace.attributes['xs'].value)
        ys = int(xmlFace.attributes['ys'].value)
        xm = int(xmlFace.attributes['xm'].value)
        ym = int(xmlFace.attributes['ym'].value)
        faces[fileName] = array([xf, yf, xs, ys, xm, ym])
    return faces
```

11、 可以调用 linalg.lstsq()函数来计算最小二乘解。编写函数计算用于将点对齐到参考点的旋转、尺度和平移量。函数需返回一个具有尺度的旋转矩阵，以及在 x 和 y 方向上的平移量。

参考代码如下：

```
def compute_rigid_transform(refpoints,points):
    """ Computes rotation, scale and translation for
        aligning points to refpoints. """

    A = array([    [points[0], -points[1], 1, 0],
                [points[1],  points[0], 0, 1],
                [points[2], -points[3], 1, 0],
                [points[3],  points[2], 0, 1],
                [points[4], -points[5], 1, 0],
                [points[5],  points[4], 0, 1]])

    y = array([    refpoints[0],
                refpoints[1],
                refpoints[2],
                refpoints[3],
                refpoints[4],
                refpoints[5]])

    # least sq solution to mimimize ||Ax - y||
    a,b,tx,ty = linalg.lstsq(A,y)[0]
    R = array([[a, -b], [b, a]]) # rotation matrix incl scale

    return R,tx,ty
```

In [23]:
```
def compute_rigid_transform(refpoints, points):
    A = array([[points[0], -points[1], 1, 0],
               [points[1], points[0], 0, 1],
               [points[2], -points[3], 1, 0],
               [points[3], points[2], 0, 1],
               [points[4], -points[5], 1, 0],
               [points[5], points[4], 0, 1]])
    y = array([refpoints[0],
               refpoints[1],
               refpoints[2],
               refpoints[3],
               refpoints[4],
               refpoints[5]])
    a, b, tx, ty = linalg.lstsq(A, y)[0]
    R = array([[a, -b], [b, a]])
    return R, tx, ty
```

12、 编写函数实现严格地对齐图像，并将其保存为新的图像。这里要注意：我们使用 imageio 包中的 imwrite 方法来保存图像。保存图像的文件夹 aligned 需要提前创建好。

```
def rigid_alignment(faces,path,plotflag=False):
    """    Align images rigidly and save as new images.
        path determines where the aligned images are saved
        set plotflag=True to plot the images. """

    # take the points in the first image as reference points
    refpoints = list(faces.values())[0]

    # warp each image using affine transform
    for face in faces:
        points = faces[face]

        R,tx,ty = compute_rigid_transform(refpoints, points)
        T = array([[R[1][1], R[1][0]], [R[0][1], R[0][0]]])

        im = array(Image.open(os.path.join(path,face)))
        im2 = zeros(im.shape, 'uint8')

        # warp each color channel
        for i in range(len(im.shape)):
            im2[:,:,i] = ndimage.affine_transform(im[:,:,i],linalg.inv(T),offset=[-ty,-tx])

        if plotflag:
            imshow(im2)
            show()

        # crop away border and save aligned images
        h,w = im2.shape[:2]
        border = round((w+h)/20)

        # crop away border
        thePath = os.path.join(path, 'aligned/'+face)
        theImage = im2[border:h-border,border:w-border,:]
        imageio.imwrite(thePath,theImage)
```

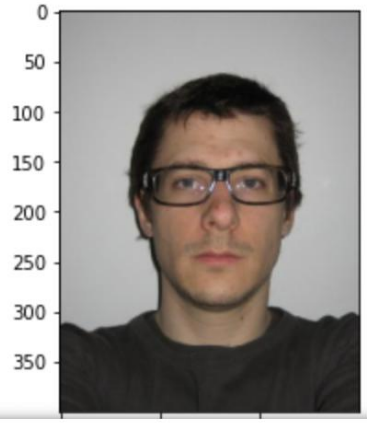
In [37]:

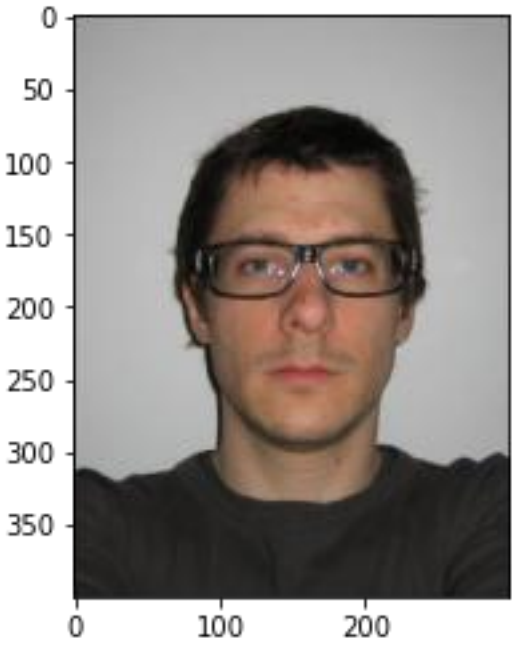
```
from imageio import imwrite
import os


def rigid_alignment(faces, path, plotflag=False):
    refpoints = list(faces.values())[0]
    for face in faces:
        points = faces[face]
        R, tx, ty = compute_rigid_transform(refpoints, points)
        T = array([[R[1][1], R[1][0]], [R[0][1], R[0][0]]])

        im = array(Image.open(os.path.join(path, face)))
        im2 = zeros(im.shape, 'uint8')

        for i in range(len(im.shape)):
            im2[:, :, i] = ndimage.affine_transform(im[:, :, i], linalg.inv(T), offset=[-ty, -tx])
        if plotflag:
            imshow(im2)
            show()

        h, w = im2.shape[:2]
        border = (w + h) / 20

        thePath = os.path.join(path, 'aligned/' + face)
        theImage = im2[int(border):int(h - border), int(border):int(w - border), :]
        imwrite(thePath, theImage)
```

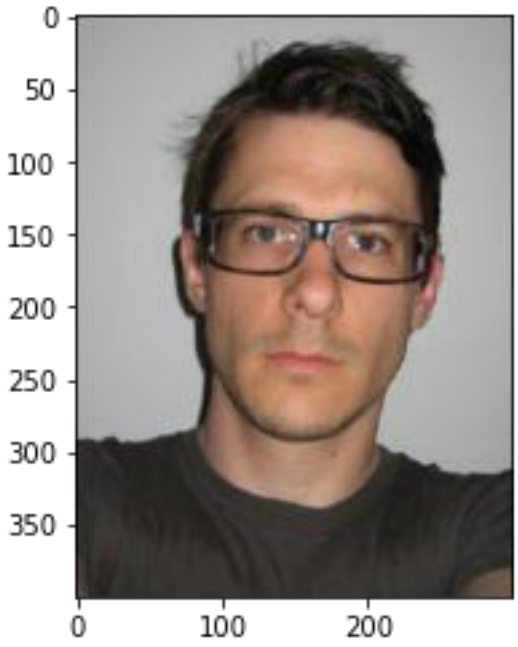
13、 编写代码读取 xml 文件“jkfaces.xml”，并根据文件中的标记点来配准所有的图像，将它们与第一幅图像对齐。运行结束后检查文件夹 aligned 中已经对齐的图像，并给出前十幅图像。

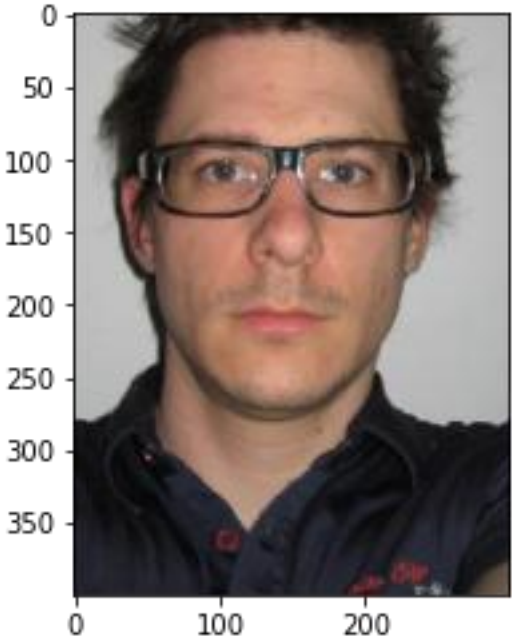
In [38]:
```
xmlFileName = '../pics3/jkfaces.xml'
points = read_points_from_xml(xmlFileName)
# 注册
rigid_alignment(points, '../pics3/jkfaces', plotflag=True)
```

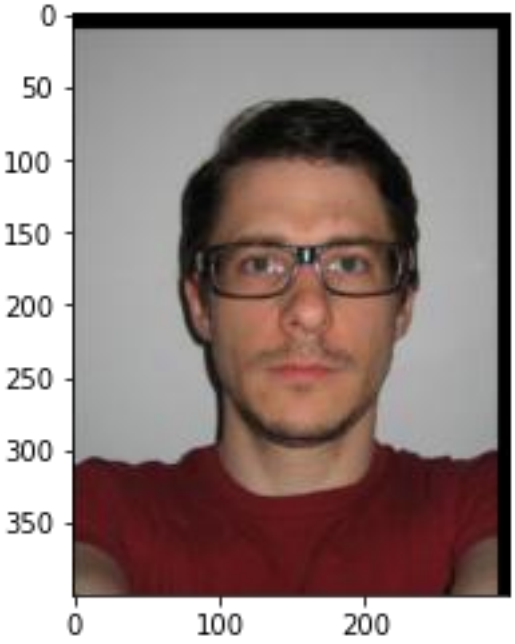
```
<ipython-input-23-8d3ac178f5e6>:14: FutureWarning: `rcond` pa
times ``max(M, N)`` where M and N are the input matrix dimens
To use the future default and silence this warning we advise
pass `rcond=-1`.
  a, b, tx, ty = linalg.lstsq(A, y)[0]
```

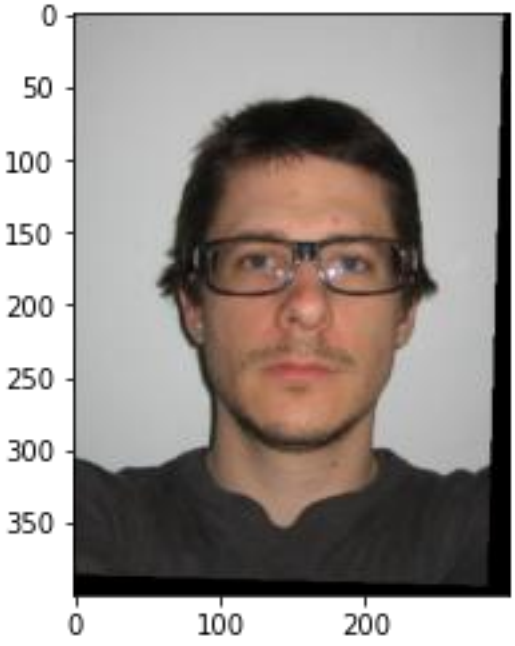


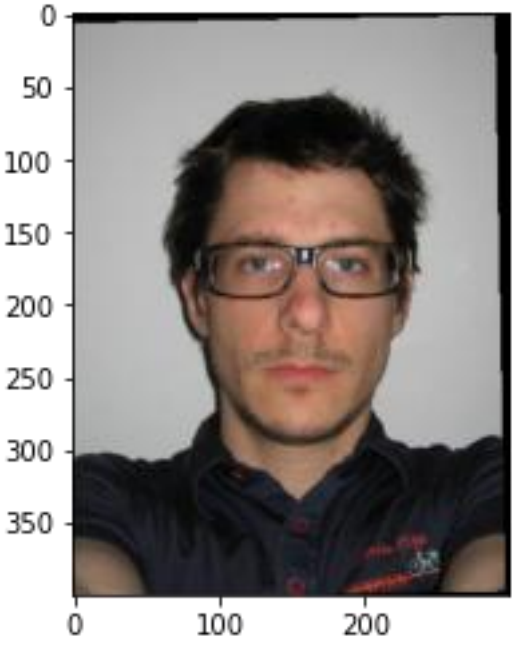


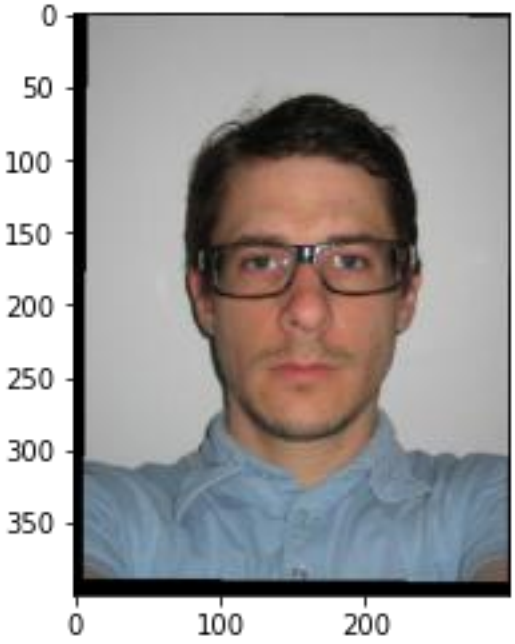


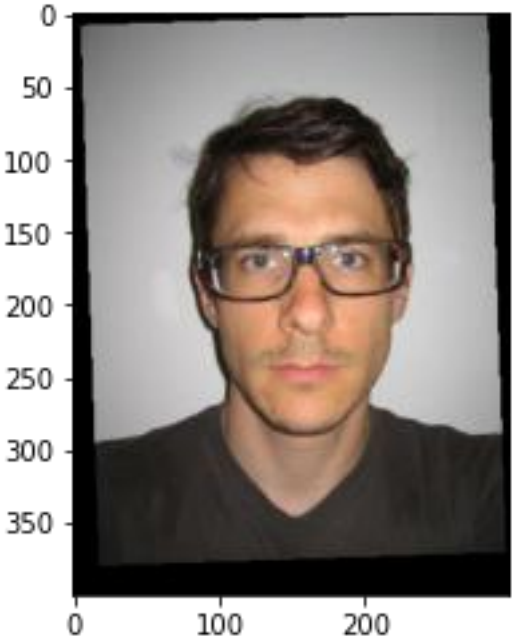

0
50
100
150
200
250
300
350
0
100
200

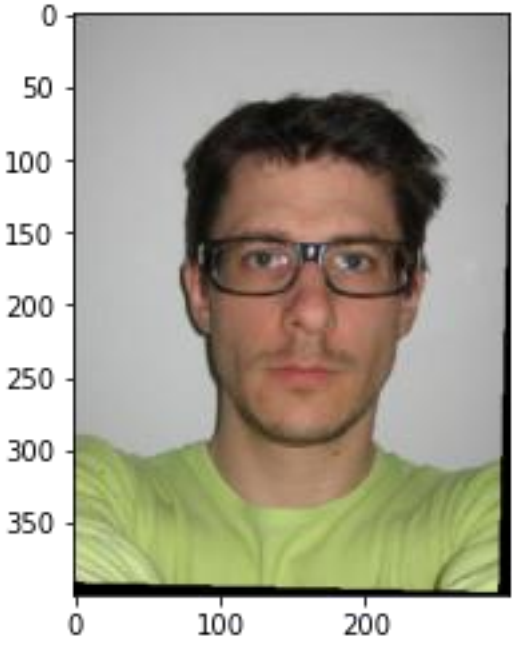

0
50
100
150
200
250
300
350
0
100
200

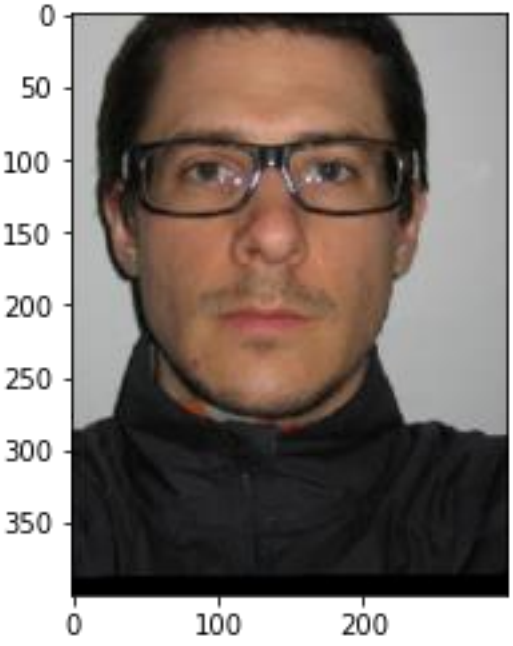

0
50
100
150
200
250
300
350
0
100
200


0
50
100
150
200
250
300
350
0
100
200


0
50
100
150
200
250
300
350
0
100
200


0
50
100
150
200
250
300
350
0
100
200


0
50
100
150
200
250
300
350
0
100
200


0
50
100
150
200
250
300
350
0
100
200