**图像配准实验报告**

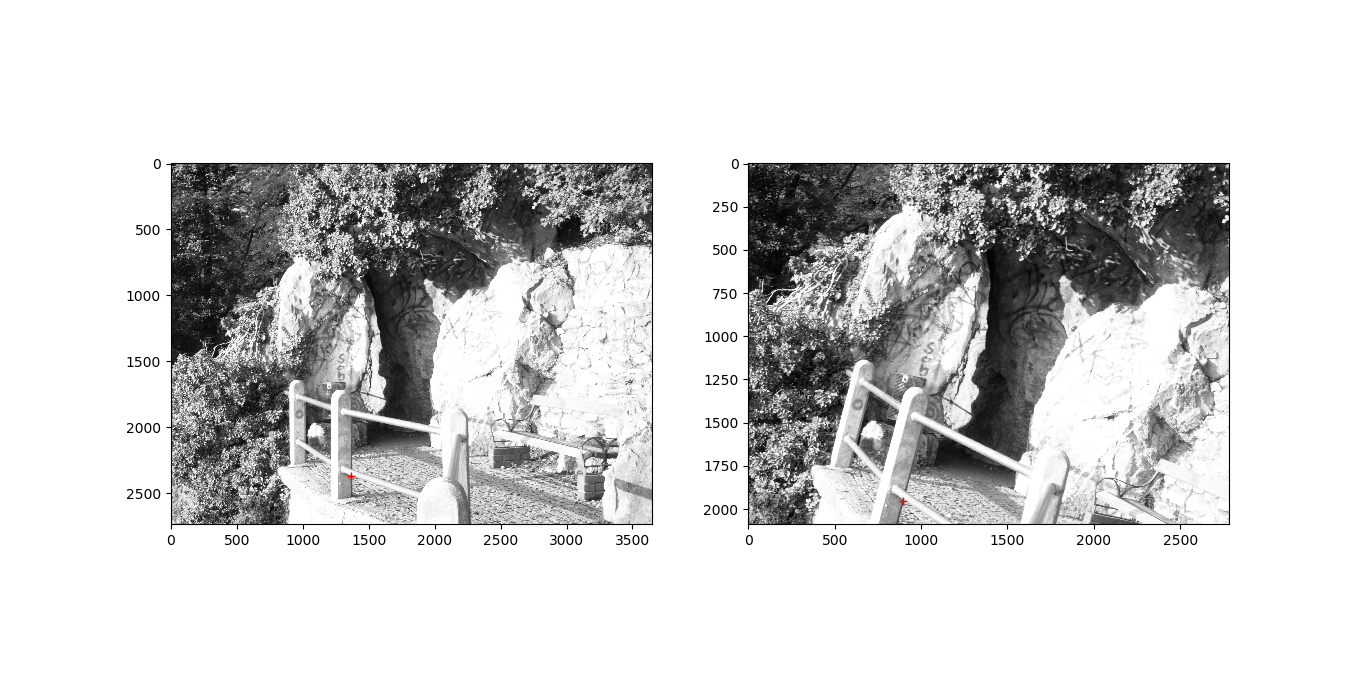
**裴森**

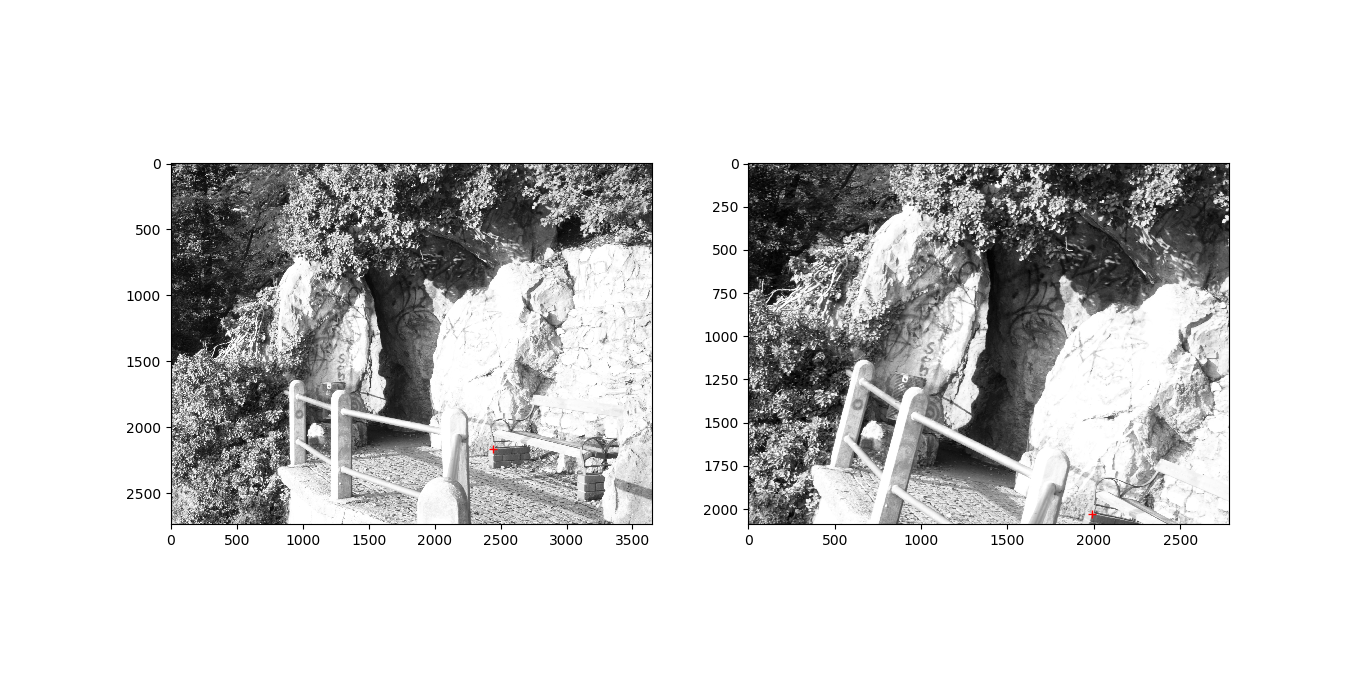
**自动化 65**

**2160504126**

**图像配准作业**

**（标注7个点）**

**一．手动标点：**

图1.一号点(柱子下侧)

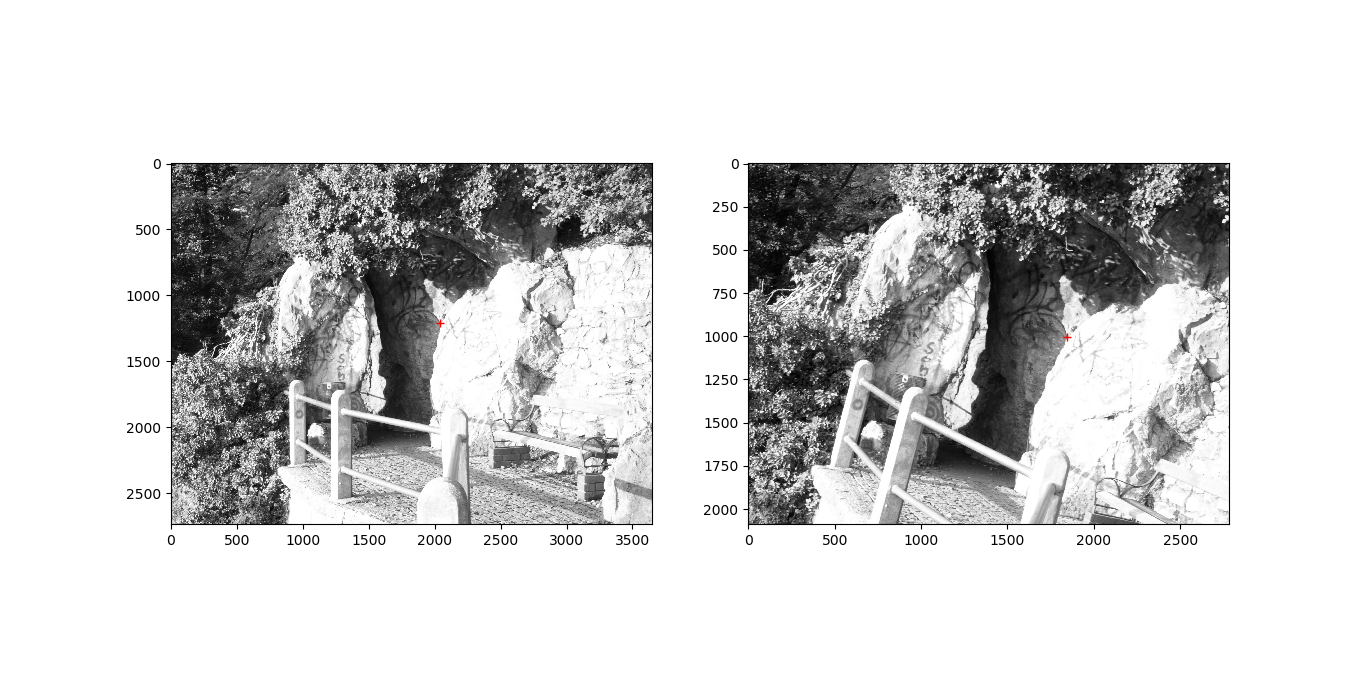
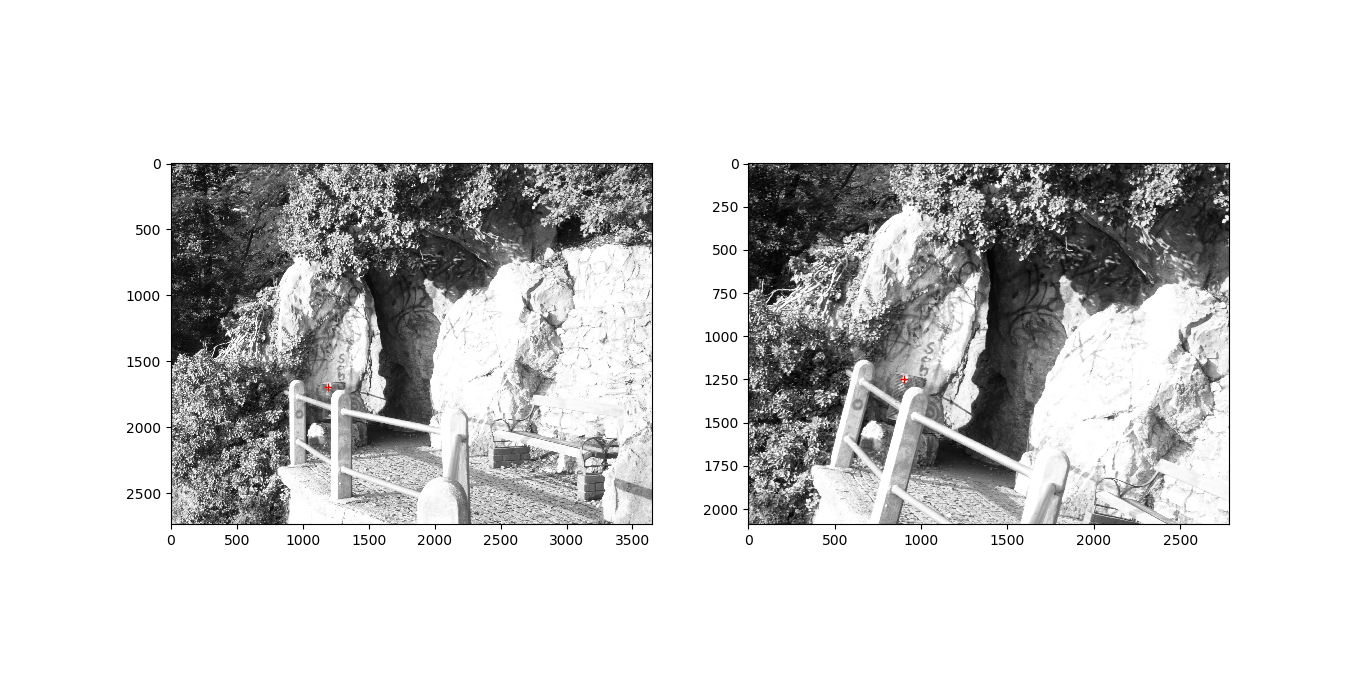
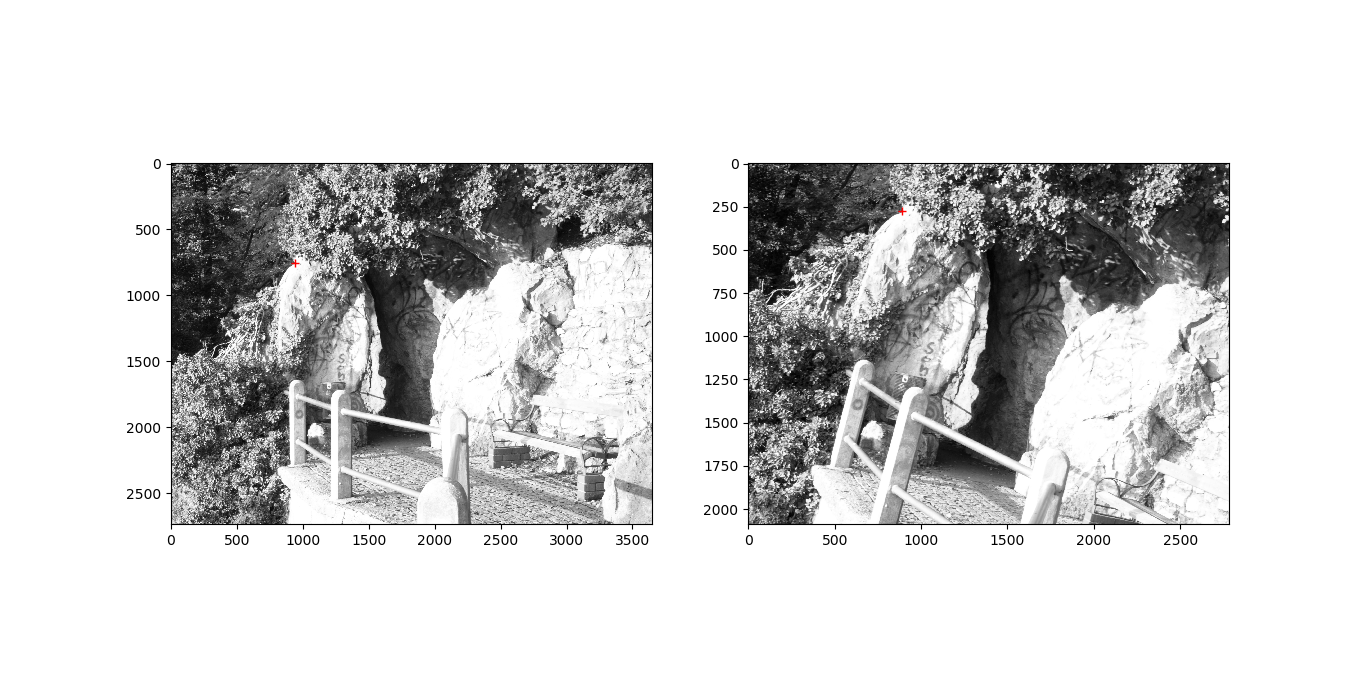
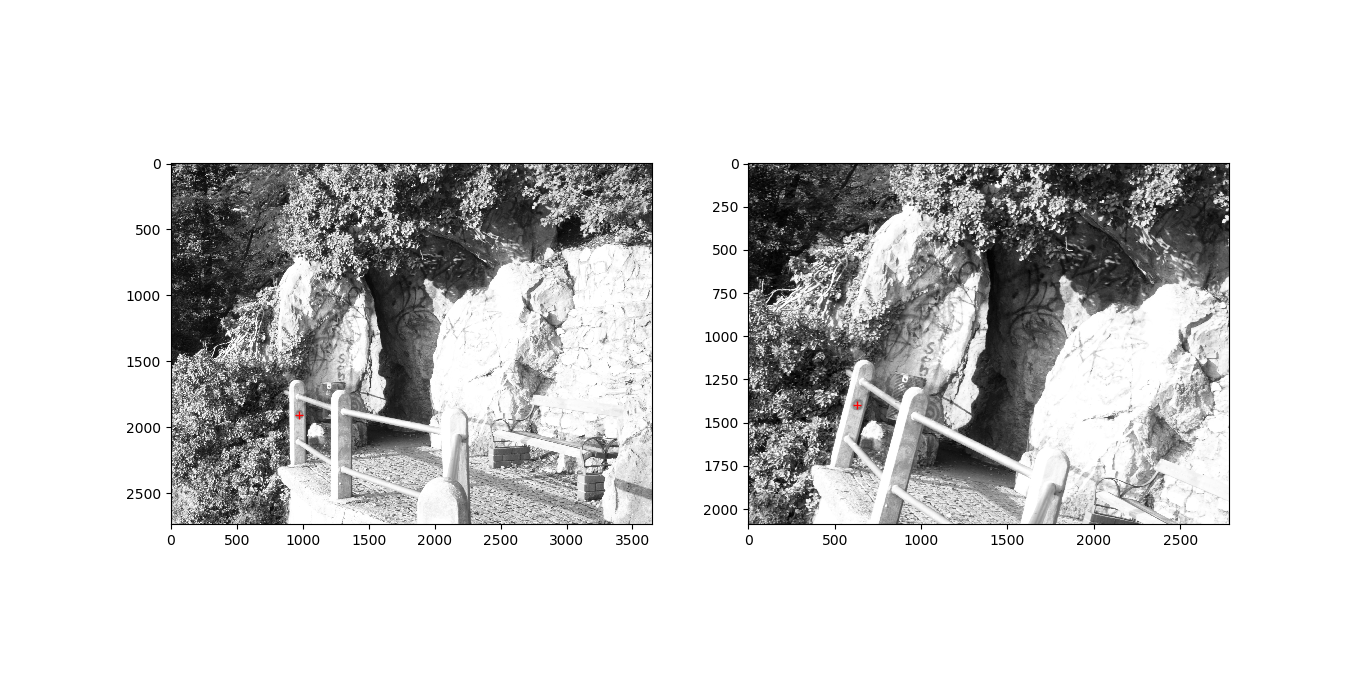
图2.二号点(台阶上侧)

图3.三号点(巨石拐角处)

图4.四号点(左上角拐角处)

图5.五号点(垃圾桶左上角)

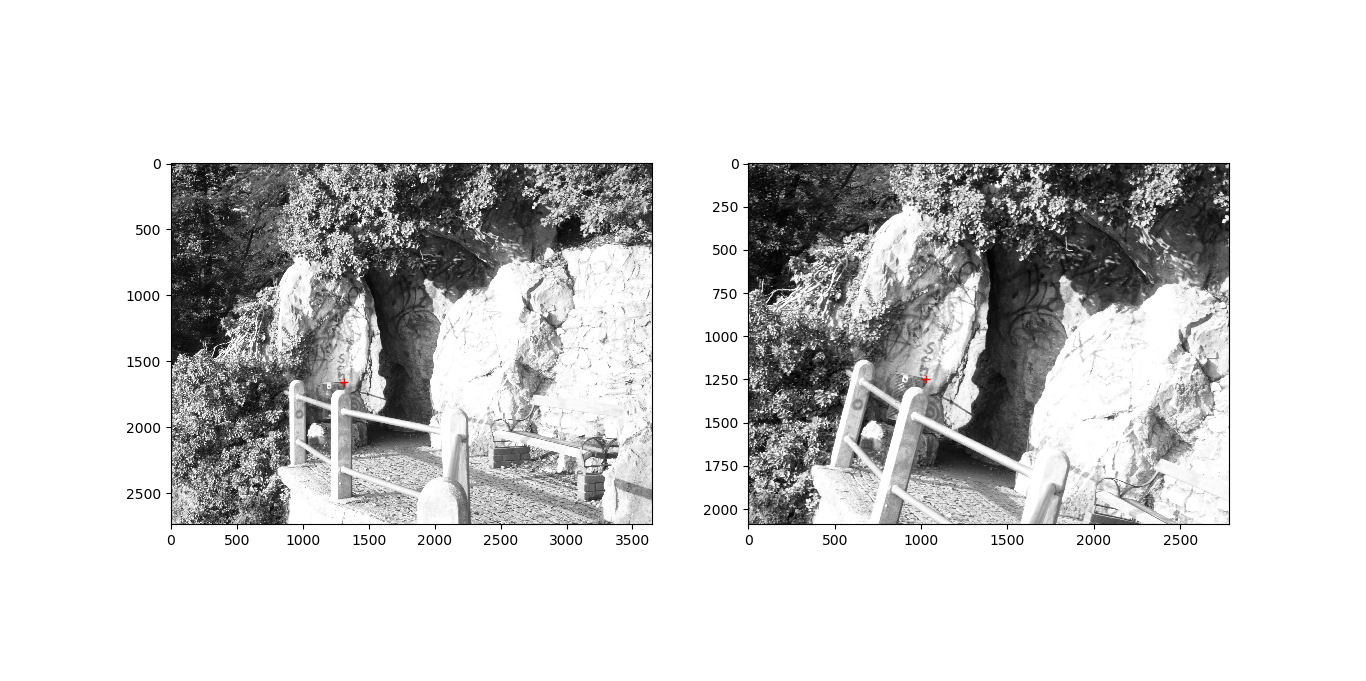
图6.六号点(第一根柱子)

图7.七号点(垃圾桶右上角)

**二．输出两幅图中对应点的坐标：**

原始图像：

Fixed =

1.0e+03 \*

1.36597088 2.36890899

2.44246944 2.16422265

2.04067773 1.20901970

0.94143624 0.75416115

1.19160844 1.69420215

0.97176014 1.90646947

1.31290405 1.65629727

旋转后的图像：

Moving =

1.0e+03 \*

0.89544053 1.95319667

1.98928931 2.02843495

1.84460032 1.00403690

0.88965298 0.27480439

0.90122810 1.24711441

0.62921279 1.39759096

1.02855441 1.24711441

**三．计算转换矩阵：**：

移动矩阵P为：

原始矩阵Q为:

根据公式有：

代入数据计算可得H为：

H的逆矩阵为：

对于P与Q的定义不同时，可能会有不同的H的矩阵，但必定是上述两个矩阵中的一个。

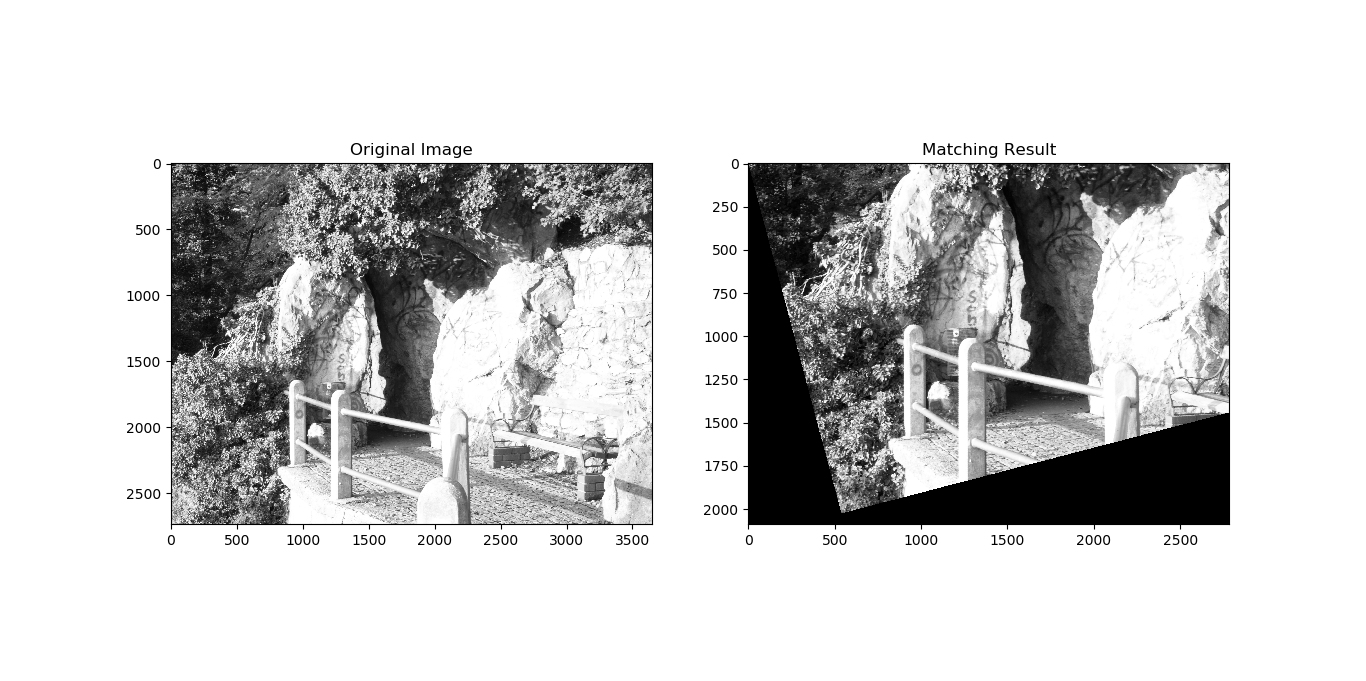
**四．输出转换之后的图像：**

图5.结果展示

转换过程：

**1.**新建一个与原始图像大小一致的零矩阵，这个矩阵便是匹配恢复后的矩阵。

**2.**对于零矩阵之中的点q，用H矩阵映射为旋转后的图像中的某个坐标p。

**3.**对p坐标四舍五入取整，对应于旋转后的图像中的某个像素，将该像素值赋给q点。

**4.**遍历完零矩阵中的所有点后，结束循环，输出图像，这时零矩阵已经被转变为匹配结果。

**五．代码示例：**

将图片复制到程序路径中，直接运行即可得到图5所示结果。由于读取图片与显示图片用到了open-cv与matplotlib，因此需要有这两个函数库。

#Code with Python

import cv2

import numpy as np

import matplotlib.pyplot as plt

class Point\_matching(object):

def \_\_init\_\_(self):

self.fixed = cv2.imread('Image A.jpg',cv2.IMREAD\_GRAYSCALE)

self.moving = cv2.imread('Image B.jpg',cv2.IMREAD\_GRAYSCALE)

def get\_point(self):

plt.figure('Point Matching')

plt.subplot(1,2,1)

plt.imshow(self.fixed,cmap='gray')

plt.subplot(1,2,2)

plt.imshow(self.moving,cmap='gray')

pos = plt.ginput(15)

print(pos)

def Matching(self):

P = np.matrix([[0.89544053, 1.98928931, 1.84460032, 0.88965298, 0.90122810, 0.62921279, 1.02855441],

[1.95319667, 2.02843495, 1.00403690, 0.27480439, 1.24711441, 1.39759096, 1.24711441],

[1, 1, 1, 1, 1, 1, 1]])

Q = np.matrix([[1.36597088, 2.44246944, 2.04067773, 0.94143624, 1.19160844, 0.97176014, 1.31290405],

[2.36890899, 2.16422265, 1.20901970, 0.75416115, 1.69420215, 1.90646947, 1.65629727],

[1, 1, 1, 1, 1, 1, 1]])

H = Q \* P.T \* (P \* P.T).I

m, n = np.shape(self.moving)

new = np.zeros((m, n))

for i in range(m):

for j in range(n):

co = H \* np.matrix([i, j, 1]).T

# use the same way convert [i, j] into P

xx = int(co[0,0])

yy = int(co[1,0])

if xx > 0 and yy > 0 and xx < m and yy < n:

new[i, j] = self.moving[xx, yy]

plt.figure('Results')

plt.subplot(1,2,1)

plt.imshow(self.fixed,cmap='gray')

plt.title('Original Image')

plt.subplot(1,2,2)

plt.imshow(new,cmap='gray')

plt.title('Matching Result')

plt.show()

if \_\_name\_\_ == '\_\_main\_\_':

h = Point\_matching()

# h.get\_point()

h.Matching()

**六．心得体会：**

在完成了两次数字图像处理作业之后，我深深的体会到了矩阵操作的重要性。仅仅就图像处理而言，矩阵的切片操作不仅使得我们可以灵活的操作图像中的像素，最重要的是矩阵之间的运算极大的提高了图像处理的速度。

相比于用循环的方式修改矩阵中的像素，矩阵运算所带来的速度提升是显著的，这在图像插值操作以及卷积操作中体现的尤为明显。

除此之外，在完成作业的同时，我体验到了这门课程的乐趣，它不仅锻炼了我的编程能力，更丰富了我对计算机处理图像的认知。