# 数字图像处理第一次大作业实验报告

自 42 张博文 2014011455

# 目录

# 1.方法原理的简要说明

我编写的图像拼接程序在 img_stitch.m 文件中。其中 img_stitch 函数为主函数，另外五个功能函数的名字和功能分别如下表

| 文件名 | 函数调用方式 | 函数功能 |
| --- | --- | --- |
| point2matrix | T=point2matrix(point1,point2) | 求仿射变换矩阵。其中 point1 为原图控制点，point2 为新图控制点，仿射变换矩阵为新图像素点逆映射回原图像素点的变换矩阵。 |
| myTransform1 | [img_new left up bottom] =myTransform1(points1,points2,img1) | 求最左边的图片(img1)的仿射变换之后的图像。其中 points1 为 img1 上的控制点，points2 为中间图片(img2)上的控制点。left 为图像真实的左边界，up 为图像真实的上边界，bottom 为图像真实的下边界。 |
| myTransform2 | [img_new right up bottom] =myTransform3(points3,points2,img3) | 求最右边的图片(img3)的仿射变换之后的图像。其中 points3 为 img3 上的控制点，points2 为中间图片(img2)上的控制点。right 为图像真实的左边界, up 为图像真实的上边界，bottom 为图像真实的下边界。 |
| combine3img | newimg=combine3img(img1,img2,img3) | 将输入的三张图片从左至右拼接起来 |
| crop | newimg=crop(img,left,right,up,bottom) | 按照真实图像的上下左右裁剪图片，去掉黑边 |

函数 img_stitch 是整个程序的入口，img_stitch 函数按照顺序完成了以下功能。
①读取三张图片
②手动选取左边图和中间图对应的控制点
③对左边图进行仿射变换
④手动选取右边图和中间图对应的控制点
⑤对右边图进行仿射变换
⑥将仿射变换后的图片拼接到中间图上，并显示
⑦去掉图片的黑边，保存并显示

# 2.实验思路

## 2.1 整体实验思路

整个图像拼接程序关键是：选取控制点求取仿射变换矩阵，对图像进行仿射变换，图像

对接这三个步骤。

## 2.2 选取控制点求取仿射变换矩阵

在我的程序中，我采用了手动选取控制点的方式。主要采取了 cpselect 函数来选取两幅图像的控制点，且在脚本中写好了可以更改的预制控制点。为了保证变换矩阵比较准确，我一共预制了 5 对控制点。最后会得到原图控制点$(x_i, y_i), i = 1,2,3,4,5$，新图控制点$(x_i', y_i'), i = 1,2,3,4,5$。为了后边程序的逻辑，这里需要求出的是新图控制点映射回原图控制点的矩阵 T，应满足以下关系

$$\begin{bmatrix} x_1 & x_2 & x_3 & x_4 & x_5 \\ y_1 & y_2 & y_3 & y_4 & y_5 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix} = T \begin{bmatrix} x_1' & x_2' & x_3' & x_4' & x_5' \\ y_1' & y_2' & y_3' & y_4' & y_5' \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

在 matlab 中，设

$$A = \begin{bmatrix} x_1 & x_2 & x_3 & x_4 & x_5 \\ y_1 & y_2 & y_3 & y_4 & y_5 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix}, \quad \mathrm{B} = \begin{bmatrix} x_1' & x_2' & x_3' & x_4' & x_5' \\ y_1' & y_2' & y_3' & y_4' & y_5' \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

则

$$\mathrm{T} = A/B$$

## 2.3 对图像进行仿射变换

### 2.3.1 对控制点的一些调整

分别对左、中、右三张图片明明为 img1,img2,img3。则我的程序是将 img2 作为基准的。即对 img2 不做变换，而将 img1 和 img3 经过仿射变换之后的图片拼接到 img2 上。由于对 img3 进行仿射变换的思路与对 img1 的操作相仿，所以下面我以 img1 的仿射变换为例讲述我的思路。

在进行仿射变换之前，已经从 img1 和 img2 图片上找对了几对相应的控制点。根据需要，img1 在进行仿射变换之后，得到 newimg1，newimg1 上的那些由 img1 控制点映射而来的点应当与 img2 相应的点重合。

而 img2 此时图片数组的范围是 1:height×1:width，其左边并没有位置可以插入 newimg1 的数据。为了后续拼接过程操作方便，此时需要对 img2 进行一些操作。将其左右两边各拓宽 width 长的位置，形成一张拓展后的 img2 图，示意图如下图：

### 2.3.2 根据逆映射的仿射变换矩阵插值

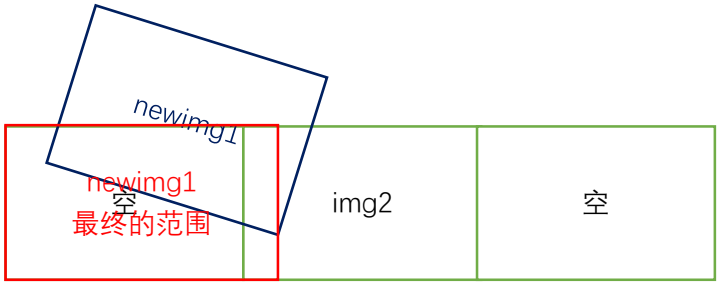
因此在对 img1 和 img2 进行仿射变换时，需要将 img2 上的控制点 point2 的横坐标加 width。这样，得到的 newimg1 图像，可以直接将其放到 3 倍宽度的新图矩阵 newimg 中。

为了保证仿射变换之后的图像的清晰度，仿射变换的过程并不是由 img1 各像素点通过仿射变换矩阵转变为 newimg1 图像的，而是要逆着映射，从 newimg1 矩阵的(1,1)点开始，到(newimg1_height, newimg1_width)点结束，分别将坐标映射回 img1 图中，之后根据 interp2 函数的双线性插值，得到 img1 图中相应像素点的 RGB 值，并赋值在 newimg1 矩阵相应的位置。这也是为什么之前的 point2matrix 函数求得的是逆映射的矩阵。

这部分功能具体的代码分析见 3.2 节。

### 2.3.3 确定新图的边界范围

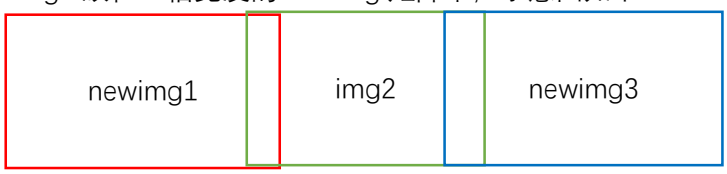
另外，newimg1 图像的长宽尺寸也应当确定下来。为了保证最后在图像拼接的过程中有一个比较良好的视觉效果，我将 newimg1 图像所占空间的轮廓定位矩形。由于 newimg1 之后所处的位置位于 newimg 矩阵的左侧，所以 newimg1 的高度已经定下为 height，宽度左侧从第一列开始，右侧则需要通过 img1 右侧的两个边界点，根据正着映射的仿射变换矩阵，求最小值得到。整个过程如下图所示：




对于 newimg1 范围中 img1 未映射到的一些范围并不需要特别处理，因为 interp2 函数会将范围在外部的数据插值为 0，即显示黑色。另外，由于任务所给图片的一些透视关系，我发现在后续处理中央主楼的过程中效果并不是很好，为了保证最终的视觉效果，newimg1 的右侧边界不宜超过 img2 的 1/7 宽度处。即整个 newimg1 图像的宽度最大不能超过 width+width×1/7，所以对上一步得到的宽度，仍需再做处理。

## 2.4 图像对接

在分别对 img1 和 img3 进行仿射变换得到 newimg1 和 newimg3 之后，需要将其与 img2 对接在一起。由于在仿射变换时已经对 img2 的坐标进行调整，所以此时可直接将 newimg1 和 newimg3 放在 3 倍宽度的 newimg 矩阵中，示意图如下：



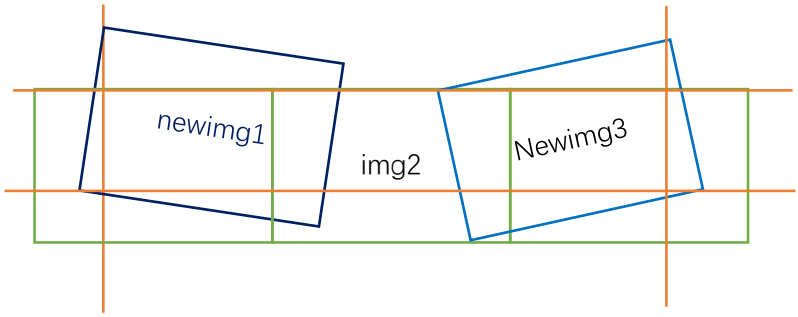

由示意图可发现，newimg1、newimg3 与 img2 均存在着一部分重合部分，这一部分的图像仍需处理，使最终的图像得到一个渐变的比较平滑的过渡。以 newimg1 和 img2 为例

说明我的设计思路。

首先将 newimg 的第 1 列至第 width 列赋值为 newimg1 相应位置的值，其次在 newimg1 和 img2 重叠的部分，生成一个系数矩阵 A。其中 A 在 img2 的左边界点值为 1，在 newimg1 右边界点值为 0。之后采取 A×newimg1+（1-A）×img2 的方式为该重合区域赋值，以达到图像平滑过渡的目的。

## 2.5 图像裁剪

由于在 2.3.3 过程中仿射变换图像的尺寸较大，所以映射之后的图像不能填满区域，会出现一些黑色部分，最后为了图片的效果，将该部分黑边去掉。在对图片作仿射变换的过程中，已经求出图片真实的上下左右界，此时只需将黑色部分截掉即可，示意图如下：




# 3.程序结果与性能

## 3.1 主楼效果图

未截去黑边

截去黑边

## 3.2 向量化编程与运算时间

为了减少程序运行时间，就要尽量使用矩阵运算，减少循环语句的使用。在整个程序中，需要进行大量运算的有两个地方：求原图各个坐标映射到新图上的坐标，根据仿射矩阵求新图各点像素值。

对于第一个求各个坐标的映射的问题。以对 img1 变换为例，我使用了如下方法：

①使用 meshgrid 函数生成网格点。其中 X_img2(i, j)代表新图中(i,j)位置的横坐标，即 i；Y_img2(i, j)代表新图中(i, j)位置的纵坐标，即 j。

```
[X_img2 Y_img2]=meshgrid(1:min_x,1:height);
```

②直接用生成的网格矩阵进行运算，求出映射之后的坐标值。如下图所示：

```
new2old_X=double(X_img2)*T(1,1)+double(Y_img2)*T(1,2)+T(1,3);
new2old_Y=double(X_img2)*T(2,1)+double(Y_img2)*T(2,2)+T(2,3);
```

例如其中的第(i , j)位置，满足

new2old_X(i , j)=X_img2(i , j)*T(1,1)+Y_img2(i , j)*T(1,2)+T(1,3)

因此 new2old_X(i , j)即为新图的(i , j)点逆映射之后，在原图中点的横坐标；

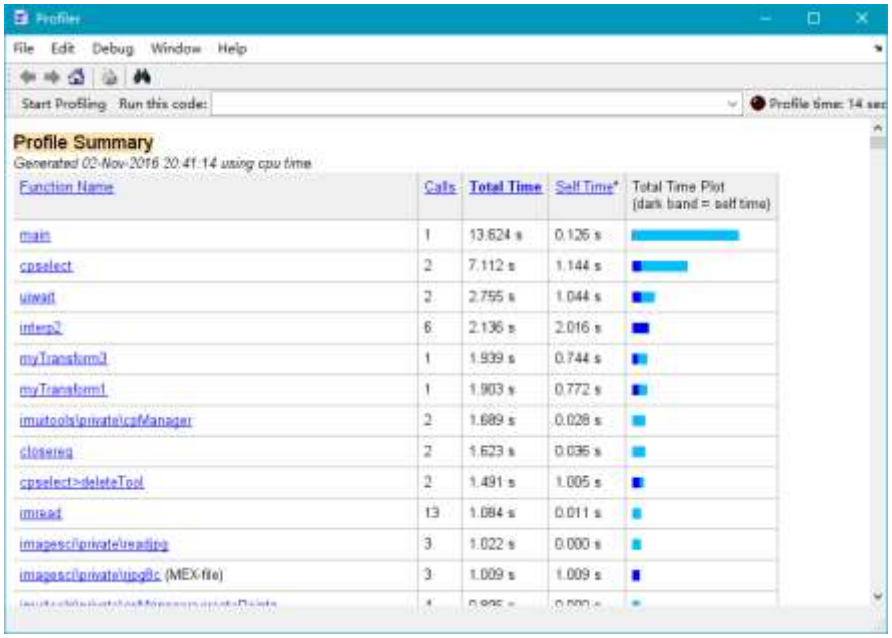
同理 new2old_Y(i ，j)即为该点在原图中对应点的纵坐标。因此 new2old_X，new2old_Y 即为映射回原图点之后的坐标网格矩阵。

在求出坐标映射矩阵之后，由于 new2old_X，new2old_Y 中的数据类型为 double 型，所以无法直接将原图的(new2old_X(i , j), new2old_Y(i , j))处的像素值赋值给新图的(i , j)点，而是需要进行插值。如果使用双线性插值，则需要根据周围的四个点计算该点的 RBG 值。这里我直接使用了 matlab 自带的二维插值函数 interp2，如下图所示：

```
[X_img2 Y_img2]=meshgrid(1:width,1:height);
%使用插值函数算出各点
img_new(:,:,1)=interp2(X_img2,Y_img2,double(img1(:,:,1)),new2old_X,new2old_Y);
img_new(:,:,2)=interp2(X_img2,Y_img2,double(img1(:,:,2)),new2old_X,new2old_Y);
img_new(:,:,3)=interp2(X_img2,Y_img2,double(img1(:,:,3)),new2old_X,new2old_Y);
```

interp2 函数调用的参数有五个，以第一条指令为例，实现的功能是：X_img2, Y_img2 为 img1(:,:,1)的坐标矩阵，最终 img_new(i , j)处的值为(new2old_X(i , j), new2old_Y(i , j))在原图中插值所得到的值。

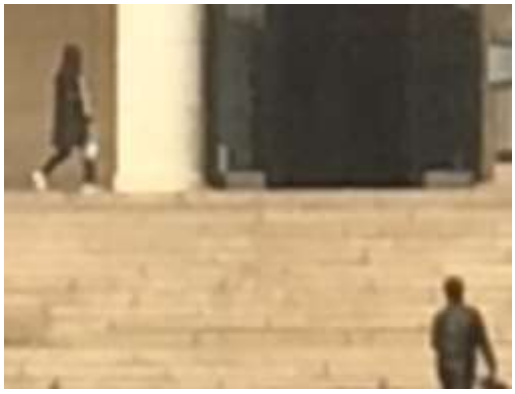
由于在程序中避免了循环的使用，所以整体运算速度还算较快。




## 3.3 图像的清晰度

由于原图的视角一些问题，以及手动匹配的不精确，会出现如下图所示的一些出现虚影的问题。

另外关于插值方法，在 interp2 函数中，我试着用了双线性和三次插值，发现并无太大区别。下图左侧为双线性插值，右侧为三次插值。



# 4.其它方法的对比讨论

由于时间问题，此次作业并未来得及写自动拼接图像的功能，但是我通过在 github 以及一些博客上的搜索，获得了一些其它人的算法程序，在这里做一些对比分析。

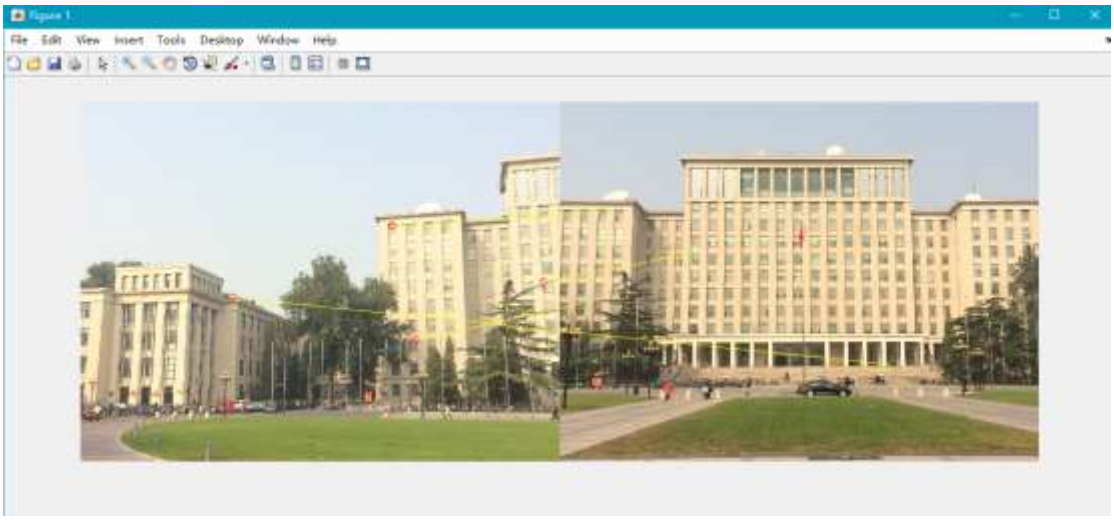
由于在获得控制点之后的拼接过程大同小异，所以我只研究了自动识别图像特征点和匹配特征点的算法。

## 4.1SIFT 算法

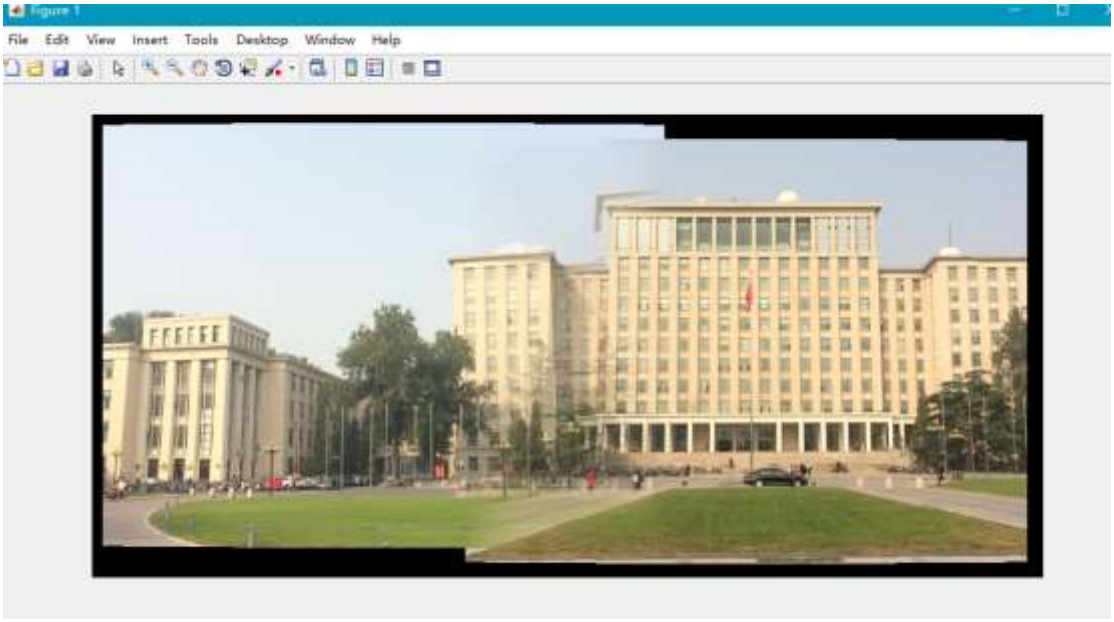
SIFT 算法的目的是找到图像的局部最值，之后将不同图像上的局部最值进行匹配。其提取图像局部最值的方法是使用图像金字塔方法。

分别以两幅图片的地面作四棱锥，那么每一个截面与原图像相似，所以两个金字塔中必然会有包含大小一致的物体的无穷个截面，但应用只能是离散的，所以只能构造有限层金字塔。有了图像金字塔就可以对每一层求出局部最值，之后则需要使用某种方法抑制去除一部分点，但又使得同一尺度下的稳定点得以保存。

我找到的一段 SIFT 算法最终的匹配结果如下，可以看到结果并不是特别准确




最终，img1 和 img2 的拼接结果如下：




另外根据 http://blog.csdn.net/CXP2205455256/article/details/41747325 的博客指出，SIFT 算法存在着以下缺陷：

(1)SIFI 在求主方向阶段太过于依赖局部区域像素的梯度方向，有可能使得找到的主方向不准确，后面的特征向量提取以及匹配都严重依赖于主方向，即使不大偏差角度也可以造成后面特征匹配的放大误差，从而匹配不成功；

(2)图像金字塔的层取得不足够紧密也会使得尺度有误差，后面的特征向量提取同样依赖相应的尺度，发明者在这个问题上的折中解决方法是取适量的层然后进行插值。

(3)我们知道同样的景物在不同的照片中可能出现不同的形状、大小、角度、亮度，甚至扭曲；计算机视觉的知识表明通过光学镜头获取的图像，对于平面形状的两个物体它们之间可以建立射影对应，对于像人脸这种曲面物体在不同角度距离不同相机参数下获取的两幅图像，它们之间不是一个线性对应关系，就是说我们即使获得两张图像中的脸上若干匹配好的点对，还是无法从中推导出其他点的对应。

## 4.2Harris 算法

Harris 算法提取的是图像的角点，即图像在水平、竖直方向变换较大的点。Harris 角点检测一般分为以下几个步骤：

①计算图像在 x 和 y 方向的梯度$I_x, I_y$

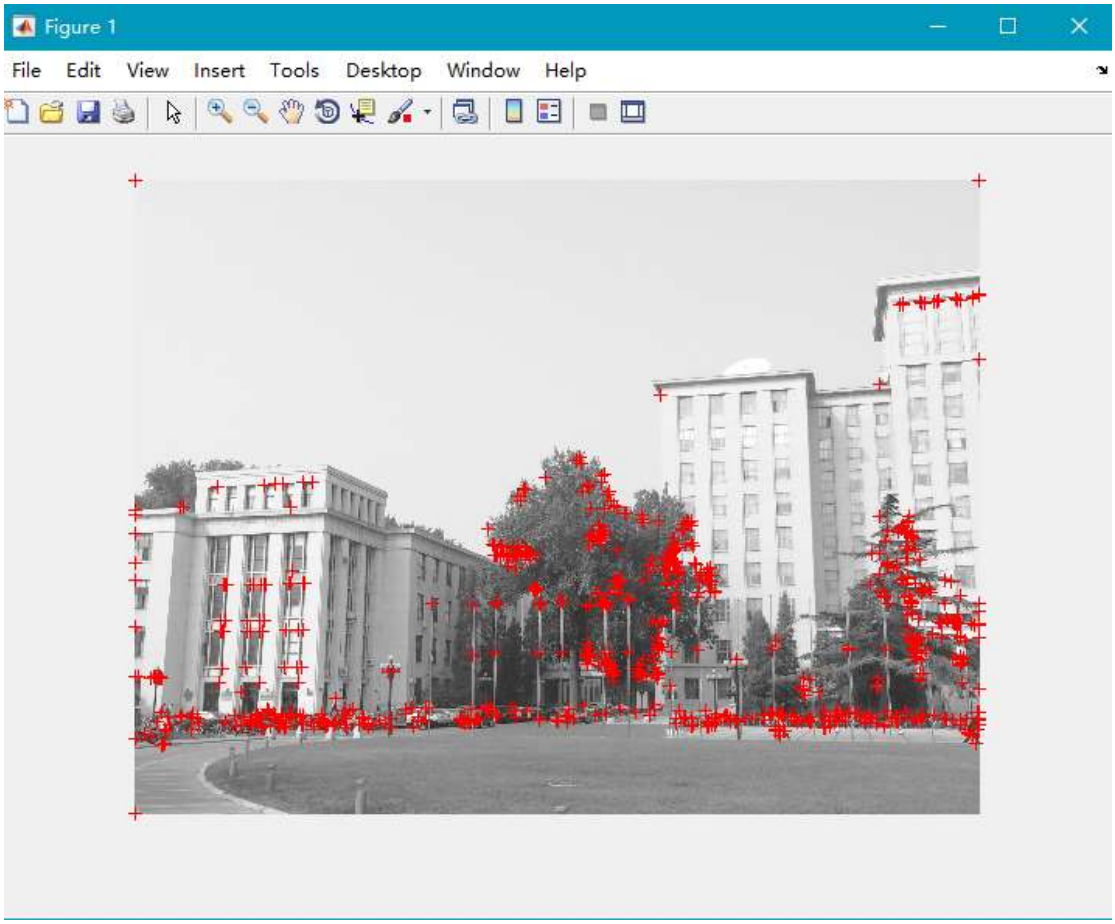
②计算梯度方向的乘积$I_x^2 = I_x \cdot I_x \quad I_y^2 = I_y \cdot I_y \quad I_{xy} = I_x \cdot I_y$

③使用高斯核对$I_x^2, I_y^2, I_{xy}$进行加权

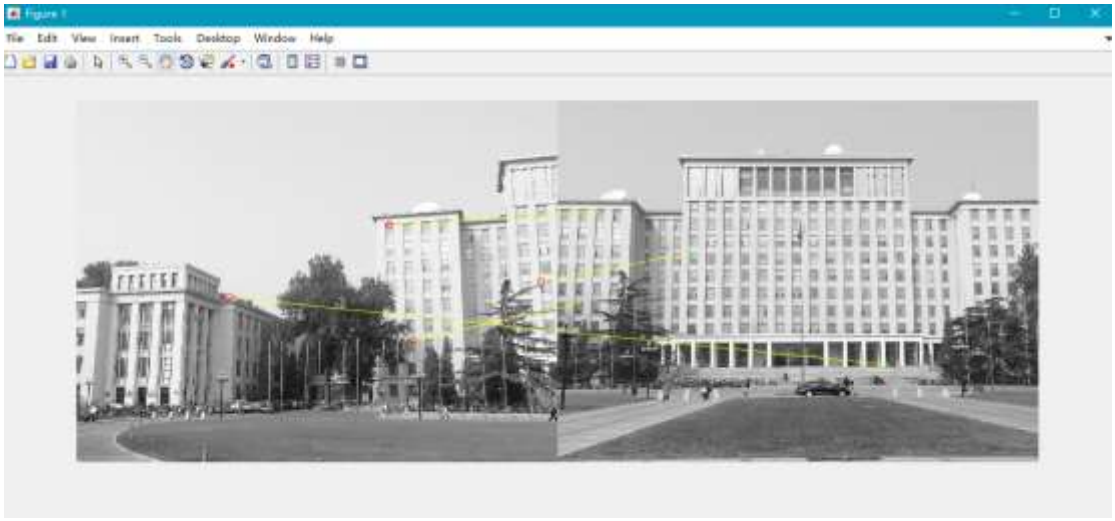
④根据计算角点量，遍历图中所有点，当该点的角点量小于阈值的时候，证明不是候选角点

⑤进行局部极大值抑制

在其中的一个代码的运行结果中，可以看到提取出了图像中大量的角点，但是这些角点大多集中在树和楼下的自行车上，这是由于这些物体的轮廓不均匀且颜色较深，因此边缘处存在着较大的颜色变化梯度，被识别为了角点。




在另外一个程序中，可以看到最终的特征点匹配结果如下



由于图片中主楼的窗户较多，且大部分窗户几乎相同，所以在匹配的过程中会出现很多问题，所以 Harris 角点能抗旋转，抗仿射变换，但是不能抗尺度的影响。