

计算机视觉大作业报告

课程名称： 计算机视觉

任课老师： 徐力豪

姓名： 徐嘉宁

学号： 18221787

专业： 数字媒体技术

# 前 言

《计算机视觉》这一门课程向我们介绍了计算机视觉、数字图像处理的主要内容。《计算机视觉》是一门非常有意思的课程，这一门课让我回想起了之前学习的《数字图像处理》课程，同样是对图像做一定的数字处理。在处理图片的过程中，我们对于图片被处理后的结果是不知道的，这当中充满着神秘感与新奇感，这正是“图形图像”带给人们吸引力的原因之一。但除了对图像处理的过程以外，《计算机视觉》当中包含的对图像的算法与思想才是最为精彩的。有些时候这些思想在原理上也并没有非常复杂，但是其对于目标的期望具有非常好的效果，并且在方法上也别具一格，让人惊叹，就比如说大作业中的HOG描述算法。

整一个学期下来，让我印象最为深刻的就是在课堂上看到的各种各样的图片，许许多多的处理方法以及四次作业，前两者让我感受到了图形图像这一个领域的多姿多彩，后者则让我颇具有成就感，让我仿佛在这个汪洋大海中舀到一瓢水。不及我是这样感觉的，有许多的同学都有如此感觉——我的同学马逍通说，只有写计算机视觉的作业是让他开心的，其他的课程作业都做不到。

《计算机视觉》确实是一门不太一样的课，即使最终的大作业很难，但我依然觉得在这份痛苦的经历之中，我拥有了许多收获。当然，期末其他大作业压身，期末考试多、压力大，造成了作业时间变少，这也是不争的事实，这也造成了我作业最后的效果没有那么的完美。我相信如果我有更多的时间，对于这一份作业我能够交出更满意的答卷。

以下是我的大作业报告正文。

## 一些前提

* 1. 在老师的作业要求中，我们的最终任务是实现全景图拼接，但我发现图像只对于水平方向上的图像拼接效果较好，但纵向的稍有欠缺，但在我的作业中，我对于纵向的凭借做了一些改进。
  2. 也许其中最难的部分不是特征点寻找或是特征点匹配，我认为是经过变换矩阵得到的两张图的合成以及变换矩阵的求得。难的原因应该是在于我不知道如何确定坐标系以及参考系；还有对于最小二乘法的利用以及线性代数知识的陌生——在老师的提示下，我使用了H=P1\P2的方法简单求出了变换矩阵。我推测这两个原因是导致我最后图形叠加合成的时候效果不好的主要原因。
  3. 以下所有内容的素材都是来源于我自己拍的图，当然，对于老师给的例子，我的代码同样能够运行，并且效果会比我自己拍的图要好。压缩包中有两个MXL，一个是Final\_Project，里面是原图例子，另一个是XJN\_Final\_Project，是我自己拍的图片的例子（因为防止上传的mxl不能改动的原因，我干脆写了备份）
  4. 在使用我自己的图的过程中，我发现了以下问题。（1）我的图是由手机拍的，成像质量特别好，因此图片特别大，包含的像素特别多，大概在3000\*3000级别，这个时候我发现图像中特征点寻找的角点不够准确，匹配上的特征点也非常少，最后造成了变换矩阵错误求得，这让我想起课上提到的，角点是一种对于图像缩放尺寸有要求的特征点，老师给的样图尺寸在500\*500像素级别，而如果我用同样大小的patch，那么他可能被识别成边缘。所以最后我压缩了我的图片，也推荐使用1000像素级以下的图片进行操作（2）我的程序在得到描述子信息以及匹配描述子的时候所花的的时间特别长，这是因为我将特征点匹配时计算欧式距离放在了双重循环内，这就需要我在循环中计算m\*n次欧氏距离，这导致了我速度的变慢，虽然说我也有一些解决问题的思路，但奈何时间不允许，最后我没有做出改进。（3）影响最终结果的因素有很多，主要是各种参数，这些参数的微调都能够起到莫大的作用，包括角点检测于匹配的阈值、描述符的阈值、描述符的大小等等，总之调参是一件麻烦的事情，如果图像最终效果不行，建议手动调参试试看。
  5. 请注意，我的代码仅支持patch为偶数方阵的形式！奇数会有问题奥！以及，如果最后的效果不太让人满意，第一是各个参数的问题，因为对于样图的效果还算不错，而且过程中思路都是正确的，没有错，但是调参这门技术活，确实有点儿难；第二是距离远近的问题，因为这个算法对于图像的缩放是没有办法解决的，所以最后合成的图像很容易受到图像上主体大小的影响，可能出现类似同心包围的场景。

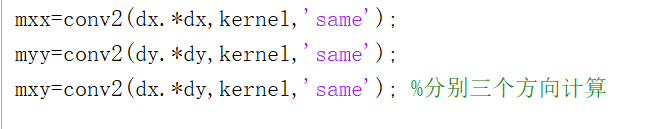
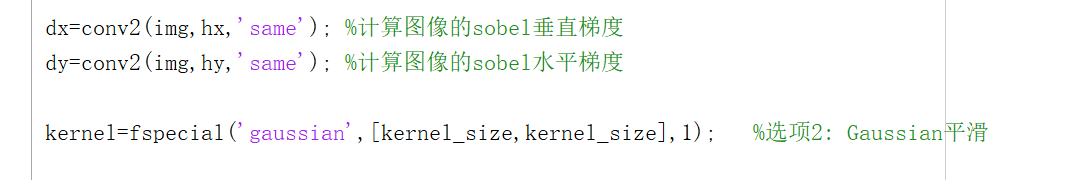
## 整体思路

* 1. 检测图像中的特征点信息，在这里我们使用角点检测
  2. 对特征点的描述，我们需要对特征点进行一个描述，其实就是给他一个标准或者说规则，让它们知道，怎么在另一张图片上找到自己。在这个过程中，我们可以使用特征点领域一维向量化的信息，也可以使用HOG算法
  3. 特征点匹配，将平行世界中的两点牵线哦！
  4. 根据特征点求出变换举证，这里我们可以使用RANSAC算法或者最小二乘法得到大概的变换矩阵
  5. 将变换矩阵作用到图片上，将得到的两张图互相叠加，得到全景图。

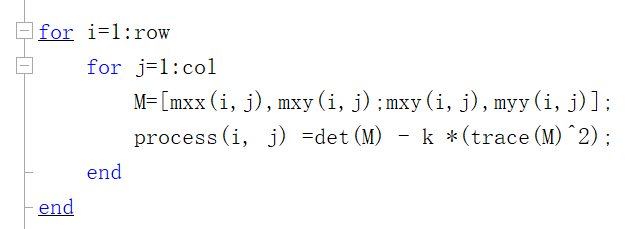
## Harris角点检测

图像中的角点是描述图像信息的非常好的要素，因为图像中的角点不会因为旋转，平移等等变换做出改变。这是我们两张图像能够匹配起来的重要基础。角点检测的算法已经给出，我也就不再强调。大致流程为：

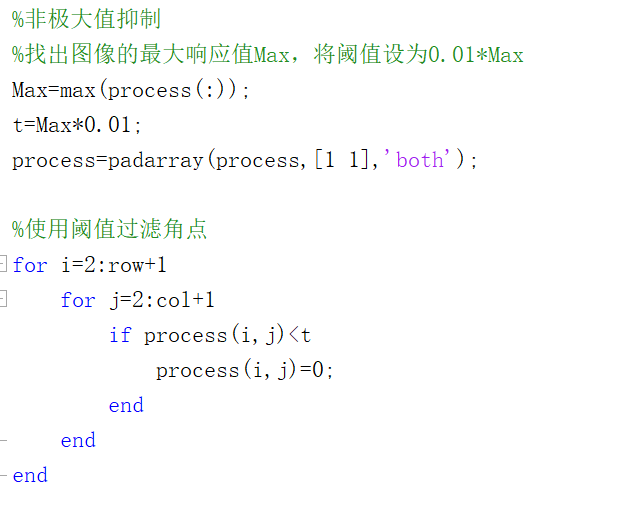
1. 利用sobel算子计算图片在水平和垂直方向上的的偏导数。
2. hx = fspecial('sobel'); %垂直方向sobel算子
3. hy = hx';%将矩阵转置得到水平方向sobel算子
4. 对于每个像素点，计算偏导数的乘积，并用高斯算子进行平滑。（如果用全1模板，效果会变差）

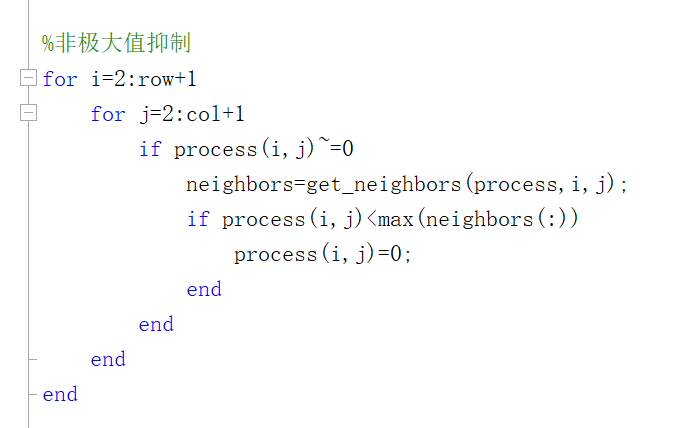


1. 对于每个像素点，计算矩阵M，M公式已经给出了
2. 对于每个像素点，计算角点响应值



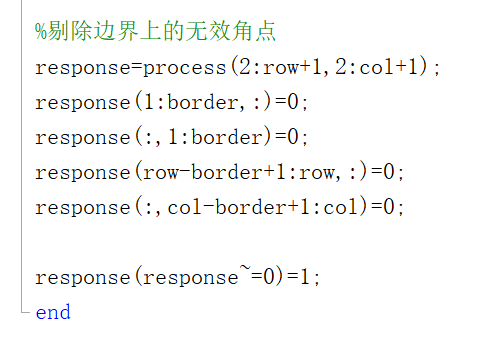
1. 先用阈值过滤掉一部分的角点，然后再用非极大值抑制进行细化





1. 输出角点响应的图片

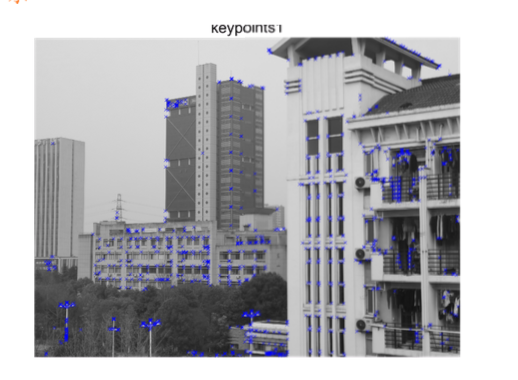
在这里重点说明一下，在这里我参考了别的同学的想法，设置的一个边缘border，含义是距离图像边缘为border以内的角点我们都不考虑，因为以后要选取特征点周围的一个领域，有可能会选到超出图片本身的范围。



最后我们将得到的角点画在图片上：

1. figure
2. imshow(img1);title('keypoints1','FontSize',20);
3. [x,y]=find(keypoints1==1);
4. hold on;
5. plot(y,x,'rx','LineWidth',1,'MarkerSize',6,'Color','b');
7. figure
8. imshow(img2);title('keypoints2','FontSize',20);
9. [x,y]=find(keypoints2==1);
10. hold on;
11. plot(y,x,'rx','LineWidth',1,'MarkerSize',6,'Color','b');

最后得到了这样两张图片：



## 关键点描述

在关键点描述中，我们会使用到两种方法,我们已经得到了特征点了，我们可以在特征点的周围取一片小区域，如果图片A与图片B上能够找到长得很像的小区域，那这不就是要找到的对应点吗！

于是第一种方法，将选定的小区域展开成一维向量。我们对A与B的特征点都这么操作，然后选A中的一个点，将他与B中所有点计算欧式距离（两个向量的欧式距离），如果有一个距离特别小的，那说明他俩就对上了！我们就找到一对匹配点了。

我们先要得到两张图的所有特征点：

1. %用describe\_keypoints函数实现特征点描述算法
2. descriptorMode=1;%设定描述算法为一维向量
3. patch\_size=8;
4. desc1 = keypoint\_description\_simple(img1, keypoints1, descriptorMode, patch\_size);
5. desc2 = keypoint\_description\_simple(img2, keypoints2, descriptorMode, patch\_size);

count表示点的个数。X,y表示点的位置

patch=image( (x(i)-patch\_size/2) : (x(i)+patch\_size/2) , (y(i)-patch\_size/2) : (y(i)+patch\_size/2) );//得到patch

最后我们将得到的描述放进keypoint\_desc中，keypoint\_desc中的数据结构是 由行向量组成的 前两个是特征点坐标，之后都是描述出来的feature（特征）。

在这里，我用了descriptorMode来决定使用哪个描述函数去得到描述子。当descriptorMode为1，则用简单的一维展开，为2，则使用HOG算法。

1. **for** i=1:count
2. **if** descriptorMode==1
3. patch=image( (x(i)-patch\_size/2) : (x(i)+patch\_size/2) , (y(i)-patch\_size/2) : (y(i)+patch\_size/2) );
4. description = [x(i),y(i),simple\_descriptor(patch)]; %得到描述
5. elseif descriptorMode==2
6. patch=image( (x(i)-patch\_size/2+1) : (x(i)+patch\_size/2) , (y(i)-patch\_size/2+1) : (y(i)+patch\_size/2) );
7. description = [x(i),y(i),HOG\_descriptor(patch,patch\_size)]; %得到描述
8. end
9. keypoint\_desc=[keypoint\_desc;description];
10. end

一维向量展开就不多介绍了。这里重点讲一下HOG。

HOG其实就是相当于将我们得到的patch分块，对每个块（cell）计算出他每个像素的梯度，然后将这些数据整理进一个直方图中，被成为block。这里为了方便，我一律将一个patch就视为一个block，而且每个patch只分成2\*2个cell,这在patch过小或者过大的时候都会有影响，但是在一般情况下已经够用了。

比如我选取16\*16的patch，分成4个cell，将360度分成9个区域，将梯度归进这9个区域图中。最后我们得到了4\*9个数据，最后我们将它们一维化、归一化，整理进一个叫特征的向量中去（实际就和一维展开的方法差不多了）。

这里要重点说一下这个，因为在算像素的梯度的时候，很有可能出现0/0的情况，会出现报错，可以这样解决！

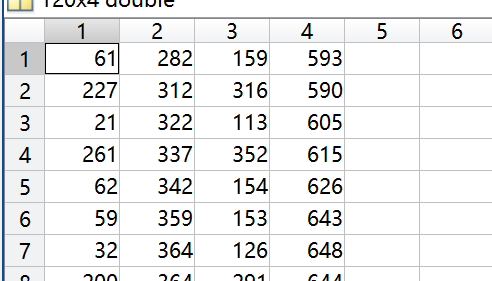
**if** isnan(tmpphase(p,q))==1  %0/0会得到nan，如果像素是nan，重设为0(防止强度为0的地方影响)

以下是所有代码：

1. function feature = HOG\_descriptor(patch,patch\_size )
2. % 这是用于HOG描述算法
3. %   输入有patch,patch\_size
4. %   这里为了简化算法 我们一律将patch视为block，一律划分成2\*2的cell
6. [m n]=size(patch);
7. patch=sqrt(patch);      %伽马校正
9. %下面是求边缘
10. fy=[-1 0 1];        %定义竖直模板
11. fx=fy';             %定义水平模板
12. Iy=imfilter(patch,fy,'replicate');    %竖直边缘
13. Ix=imfilter(patch,fx,'replicate');    %水平边缘
14. Ied=sqrt(Ix.^2+Iy.^2);              %边缘强度
15. Iphase=Iy./Ix;              %边缘斜率，有些为inf,-inf,nan，其中nan需要再处理一下
17. %下面是求cell
18. step=patch\_size/2;      %step\*step个像素作为一个单元
19. orient=9;               %方向直方图的方向个数
20. range=360/orient;       %每个方向包含的角度范围
21. Cell=cell(2,2);         %所有的角度直方图,cell是可以动态增加的，所以先设了一个
22. ii=1;
23. jj=1;
24. **for** i=1:step:m          %如果处理的m/step不是整数，最好是i=1:step:m-step
25. ii=1;
26. **for** j=1:step:n      %同上
27. tmpx=Ix(i:i+step-1,j:j+step-1);  %当前cell的横向梯度
28. tmped=Ied(i:i+step-1,j:j+step-1);%当前cell的梯度
29. tmped=tmped/sum(sum(tmped));     %局部边缘强度归一化
30. tmpphase=Iphase(i:i+step-1,j:j+step-1);%当前cell的边缘斜率
31. Hist=zeros(1,orient);            %当前step\*step像素块统计角度直方图,就是cell，一共有orient列
32. %进入直方图计算
33. **for** p=1:step
34. **for** q=1:step
35. **if** isnan(tmpphase(p,q))==1  %0/0会得到nan，如果像素是nan，重设为0(防止强度为0的地方影响)
36. tmpphase(p,q)=0;
37. end
38. ang=atan(tmpphase(p,q));    %atan求的是[-90 90]度之间，但这里求出来的可能与真正方向相反
39. ang=mod(ang\*180/pi,360);    %全部变正，-90变270
40. **if** tmpx(p,q)<0              %根据x方向确定真正的角度
41. **if** ang<90               %如果是第一象限
42. ang=ang+180;        %移到第三象限
43. end
44. **if** ang>270              %如果是第四象限
45. ang=ang-180;        %移到第二象限
46. end
47. end
48. ang=ang+0.0000001;          %防止ang为0
49. Hist(ceil(ang/range))=Hist(ceil(ang/range))+tmped(p,q);   %ceil向上取整，使用边缘强度加权
50. end
51. end
52. Hist=Hist/sum(Hist);    %方向直方图归一化
53. Cell{ii,jj}=Hist;       %放入Cell中
54. ii=ii+1;                %针对Cell的y坐标循环变量
55. end
56. jj=jj+1;                    %针对Cell的x坐标循环变量
57. end
59. %下面是求feature,2\*2个cell合成一个block
60. feature=[Cell{1,1}(:)', Cell{1,2}(:)',  Cell{2,1}(:)',  Cell{2,1}(:)'];
62. end

## 描述子匹配

首先我要说明的是我得到的描述子（matches）的数据结构非常简单，一共就4列，前两个是图片A的特征点的坐标，后两列是图B的特征点的坐标，这两个点他们匹配。



1. threshold=0.5; %设定阈值
2. matches = match\_descriptors(desc1,desc2,threshold);
3. matches(:,4)=matches(:,4)+size(img1,2);
4. plot\_match(matches,img1,img2);

在这里，我用了matches(:,4)=matches(:,4)+size(img1,2),因为我觉得两张图最后要拼在一起，所以我将图B的特征点的横坐标加上了图片A的宽度，这样他们就处在同一个坐标系当中了。

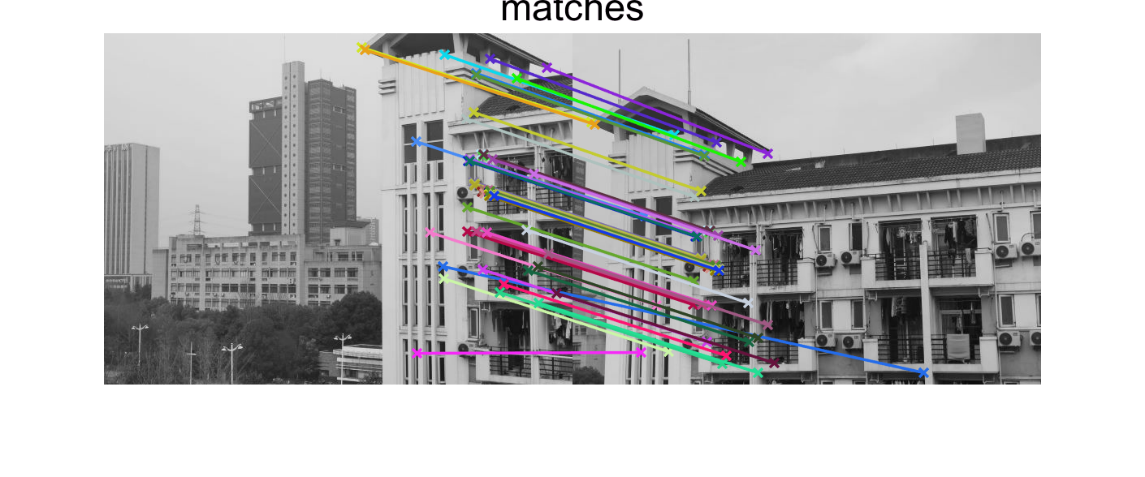
Match的方法非常简单，其实就是一个向量的构造和矩阵的构造而已。

dis=sqrt( sum( ( desc1(i,3:end)-desc2(j,3:end) ).^2 )) 用来得到欧式距离

disMap=sortrows(disMap); 这个函数是用来将表内的行向量按照第一个数从小到大排的，这可以很方便的找到最小的欧式距离是哪一对点的。

1. **for** i=1:length\_disc1
2. disMap=[];%用于存储一张距离表，其中包含点的距离和坐标
3. %     min\_dis=sqrt( sum( ( desc1(i,3:end)-disc2(1,3:end) ).^2 ));%定一个初始最小值
4. **for** j=1:length\_disc2
5. dis=sqrt( sum( ( desc1(i,3:end)-desc2(j,3:end) ).^2 ));
6. disMap=[disMap;dis,desc1(i,1:2),desc2(j,1:2) ];
7. end
9. disMap=sortrows(disMap);
10. **if** disMap(1,1)/disMap(2,1)<=threshold
11. matches=[ matches; disMap(1,2:end) ];
12. end
13. end

然后我们将点画出，画点和线用到了plot函数，这里不是学习重点我也就略过了。最后得到效果如图：



## 变换矩阵

我的变换矩阵非常简单，一开始我用最小二乘法，我觉得应该很复杂，但是在老师的提示下，我直接用’\’就完成了！

1. function H = fit\_affine\_matrix( p1,p2)
2. %   仿射变换
3. %   此处显示详细说明
4. H=p1\p2;
5. H(:,3)=[0 0 1];
6. end

这里的H(:,3)=[0 0 1];是为了防止浮点数等影响，直接将矩阵最后一列置为【0 ；0 ；1】。

然后我是配合使用了affine2d和imwarp这两个函数，得到了变换之后的图B。图A我让他保持原样，因为我认为反正变换是相对的，那么只变换一个图应该也够了。接下来只需要我们把这个方式应用在矩阵中所有的匹配点上就行了

1. add=ones(size(matches,1),1);%增加向量1的列向量
3. matches\_A=[matches(:,1:2),add];
4. matches\_B=[matches(:,3:4),add];
6. %找到仿射变换函数
7. H1=fit\_affine\_matrix(matches\_A, matches\_B)
8. % H2=fit\_affine\_matrix(matches\_B, matches\_A)
10. tform = affine2d(H1)
11. img2\_warp=imwarp(img2,tform);
13. img1\_warp=img1;

于是得到这样的结果：





我们会发现图像匹配的不是很准确，原因就在于可能有的匹配点可能匹配错了，或者不准确，那么我们就要用RANSAC算法去再一次选择，哪些匹配点对于我们来说是相对准确的。

这里重点说一下RANSAC算法。

我们首先先在所有的候选匹配点当中随机挑几对点，然后我们通过这几组得到一个变换矩阵，接着我们将这个变换矩阵应用到图A中所有的点上去，就能算出对应在图B山的点，如果这些得到的点与图B中的特征点的相对距离小于一个你设定的阈值的话，那么他就是一对有效的点，我们在一次工作内，记录最后有效匹配点对的数量。我们进入循环，做500次这样的事情，最后排个序，看看谁有效数对最多，我们就用那一组的变换矩阵！

具体的解释我有写注释。

1. function H = ransac(matches,sample,k,threshold)
2. % ransac函数
3. %   输入为循环次数 和匹配特征点，输出为一个变换矩阵
5. %进入大循环
6. matches\_size=size(matches,1); %获取match长度
8. matches\_A=[matches(:,1:2),ones(matches\_size,1)];
9. matches\_B=[matches(:,3:4),ones(matches\_size,1)];
10. countMap=zeros(k,10); %生成一个计数用的结构体，由第一列为距离，第2到10列为变换矩阵
11. **for** i=1:k
12. %随机选某几组matches
13. rander=ceil(rand(sample,1)\*matches\_size); %生成随机数
14. matches\_choice=matches( rander(:),: ); %获取对应的坐标
16. %根据这几个计算H
17. add=ones(sample,1);%增加向量1的列向量
18. matches\_a=[matches\_choice(:,1:2),add];
19. matches\_b=[matches\_choice(:,3:4),add];
21. H=fit\_affine\_matrix(matches\_a, matches\_b); %得到了变换矩阵 A\*H->B H(3\*3)
23. %将变换矩阵应用到整个matches\_A上
24. matches\_C=matches\_A\*H;
25. %将得到的数据与mateches\_A做距离比较,得到dis向量
26. dis =( matches\_C(:,1)-matches\_B(:,1) ).^2+( matches\_C(:,2)-matches\_B(:,2) ).^2;
27. %看dis里有几个小于阈值的点
28. count=sum(dis(:)<threshold);
29. %填充countMap结构体
30. countMap(i,1)=count;
31. countMap(i,2:10)=reshape(H,1,9);
33. %对MAP进行排序
34. countMap=sortrows(countMap,-1);
35. end
37. H=countMap(1,2:10);
38. H=reshape(H,3,3);
39. H(:,3)=[0 0 1];
40. end

RANSAC之后得到的图，明显比直接应用变换矩阵效果好多了（除非你直接就能找到正确的匹配点！）





## 图像叠加

最后的一步就是将得到的图像叠在一起了。这里我只说明任务最后一步给的线性图像叠加。

在这里的思想就是在文档中说到的：

1.确定融合区域的左边界和右边界。

2. 给图片1确定一个权重矩阵: - 从图片1的最左边到融合区域的最左边，weight为1 - 从融合区域的最左边到图片1的最右边，weight从1到0进行分布

3. 给图片2确定一个权重矩阵: - 从融合区域的最右边到图片2的最右边，weight为1 - 从融合区域的最左边到图片1的最右边，weight从0到1进行分布

4. 分别对左右两张图片应用权重矩阵

5. 将两张图相加

这样我们得到的图像效果就能变好了。

最重要的就是如何确定这个权重矩阵了，我使用如下方法：先得到一个向量U10(W1-dis+i)=1-1/dis\*i，再将这个向量用repmat函数扩张为一个与图片相同的矩阵，将图片与矩阵做点乘即可。

1. **if** A\_left\_corner\_W< B\_left\_corner\_W
2. U10=ones(1,W1);
3. **for** i=1:dis
4. U10(W1-dis+i)=1-1/dis\*i;
5. end
6. U10=repmat(U10,[H1,1]);
7. img1\_warp=img1\_warp.\*U10;
9. U01=ones(1,W2);
10. **for** i=1:dis
11. U01(i)=1/dis\*i;
12. end
13. U01=repmat(U01,[H2,1]);
14. img2\_warp=img2\_warp.\*U01;
15. **else**
16. U10=ones(1,W2);
17. **for** i=1:dis
18. U10(W2-dis+i)=1-1/dis\*i;
19. end
20. U10=repmat(U10,[H2,1]);
21. img2\_warp=img2\_warp.\*U10;
23. U01=ones(1,W1);
24. **for** i=1:dis
25. U01(i)=1/dis\*i;
26. end
27. U01=repmat(U01,[H1,1]);
28. img1\_warp=img1\_warp.\*U01;
29. end

我们先设定一个一定放得下两张图片拼接的大图作为底片，将图A放在底片的中心，然后计算出图B相对图A的位移，将位移作用再图B上，最后画在底片上。

1. img3( A\_left\_corner\_H : A\_left\_corner\_H+H1-1, A\_left\_corner\_W : A\_left\_corner\_W+W1-1 )=img1\_warp;
2. img3( B\_left\_corner\_H : B\_left\_corner\_H+H2-1, B\_left\_corner\_W : B\_left\_corner\_W+W2-1 )=img3( B\_left\_corner\_H : B\_left\_corner\_H+H2-1,  B\_left\_corner\_W : B\_left\_corner\_W+W2-1 )+img2\_warp;

最后我们将图片周围黑色的区域去掉即可。

1. %去掉周围的0
2. img3(all(img3==0,2),:) = [];
3. img3(:,all(img3==0,1)) = [];

我们得到了最后的结果，如图：我们可以发现，画面上好像有些错位，这是由于两张照片，人与楼的距离之间有差距引起的。（拍的时候没考虑到这个问题）



再附一张样图的结果作为参考：

