

数字图像处理大作业

图片自动拼接

计05 2010011356 孙艺瀚

May 3, 2013

1 实验内容

将某一场景拍摄的多幅照片进行图像拼接，合成成为一幅全景图像。即完成一个程序，将诸如图1左边的两幅图拼接生成右图。生成的图片可以是柱面、球面或立方体全景图，实现的算法不限。如果自动生成的效果不好，可以加入手动调节功能。

2 具体实现

2.1 整体思路

这个实验中，我们要将先后拍摄的图片进行拼接，很自然地涉及到两个问题：第一，先后拍摄的图片由于拍摄角度的不同，镜头的移动，即使是相同的景物，也不会是呈现出完全一样的景象，不可能是一个直接拼接的过程，两幅图可能需要经过一定的变换。第二，即使在公共部分可以对应上，但是在两幅图的交界处，必然会有光照等问题带来的细微的色差，于是需要做交界处的融合。

对于第一个问题，注意到，在镜头的缓慢移动过程中，景物进行的变换符合投影变换，因此，我们希望找到这样一个投影变换，使得两个图片中的公共部分得以对应。



Figure 1: 示例：拼接前后

对于第二个问题，直觉上我们应该选取两幅图片公共部分中的一条“缝”进行拼接，此外，我们还希望选取高频分量尽量少的地方进行拼接。其次，对于高频分量，我们在一个更小的区域上融合，以防出现拼接处的高频分量（如台阶，花纹等）对不上的情况。而相对来说低频分量则应该在更大的区域上融合，以免出现拼接处颜色差别过大的情况。

在本实验中，我采用角点作为图像的特征，根据角点的匹配来进行图片对应点的拼接。总体来说，我将利用Harris和ANMS进行角点的选取，并以其领域为角点的特征进行对应角点的匹配，然后使用RANSAC找到一个投影变换，使得尽量多的角点符合该投影变换。至此完成图片的变换和对应工作。

在融合的过程中，我留出一条宽为20的“缝”，采用五层的高斯金字塔，第一层为该点本身（即高斯矩阵只有最中间的点为1，其余点都为0），其余第*i*层是这条缝与大小为 19×19 ， σ 分别为 2^{i-1} 和 2^{i-2} 的高斯矩阵卷积的差。对于每一层，在以拼接点为中心，宽度为 $[-2^{i-1}, 2^{i-1}]$ 的区域上进行融合。这样就实现了对于高频分量更集中的区域，在更小的区域上融合。

考虑到实际情况中的应用，我只对一个维度上（这里的测例都是水平拼接）进行了处理，在后面的章节中我会提到，多个位置的融合也是简单易

行的。

下面我将对每一部分的算法进行具体的介绍。我在matlab中写了一个UI用来详细展示我的程序的工作过程。这里也使用一些截图来进行说明。

2.2 Harris角点检测

通常情况下，可以将区域内的点分为3类：

1. 平坦的点
2. 边缘上的点
3. 角点

若对于这3类点分别求取 I_x , I_y （即x轴和y轴上的偏导数）,很显然，1类点的 I_x 和 I_y 都很小，1类点则是 I_x 和 I_y 有一个稍大一个稍小，角点3则是两个值都很大。所以根据这种性质，可以区分出角点来。如图2所示。

Harris角点检测[1]即是根据这个原理，计算每一个点的准则函数 R ，从而判断该点成为角点的可能性。准则函数的计算方法如下：

$$\mathbf{M} = \begin{pmatrix} I_{xx}(i, j) & I_{xy}(i, j) \\ I_{xy}(i, j) & I_{yy}(i, j) \end{pmatrix} \quad (1)$$

$$R(i, j) = \det(\mathbf{M}) - k \times (\text{trace}(\mathbf{M}))^2; \quad (2)$$

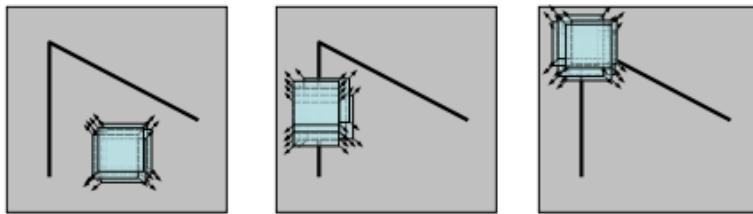
这里 $R(i, j)$ 标识了一个点可能成为角点的权重。在我的算法中，我选取了所有 $R(i, j)$ 大于 $0.02 \times \max(R(i, j))$ 的点，作为可能被选取的角点。

这样选取出的角点数大概在几万个到几十万个左右。观察可以发现，角点非常集中（容易想到，如果一个点是角点，那么它周围的八个点极可能也是角点）。然而我们需要表示整张图片的特征，距离非常接近的角点是没有用的。因此我们使用了下面介绍到的ANMS算法进行角点的选取。

2.3 ANMS

[2]中介绍了一种叫做ANMS(Adaptive Non-Maximal Suppression)的算法进行角点的挑选。它是说，对于选定的角点，在其周围的规定大小的

Harris Detector: Basic Idea



“flat” region:
no change in
all directions

“edge”:
no change along
the edge direction

“corner”:
significant change
in all directions

Figure 2: Harris角点检测原理

邻域内，只选择一个，以期减少角点的数目。在此算法下，我们可以选择k（k为50-250左右）个分布相对均匀的角点作为图片的特征。

图3显示了使用harris选取权重最大的100个角点和使用ANMS对于所有角点进行重新挑选之后的角点分布。可以看出，使用ANMS挑选后的角点分布更均匀，更有代表性。

2.4 角点匹配

对于两张图片选取出的角点，需要对其做对应的匹配。具体方法是说，对于每两个点，可以计算其邻域的SSD(Sum of Squared Differences)。即对两个角点周围 40×40 的邻域，将其下采样到 8×8 邻域，对于这个向量进行类似距离衡量的操作。计算公式如下：

$$SSD(\mathbf{a}, \mathbf{b}) = |\mathbf{a} - \mathbf{b}|^2 \quad (3)$$



Figure 3: 使用harris选取权重最大的100个角点（上图）和使用ANMS对于所有角点进行重新挑选之后的角点（下图）分布，可以看出，使用ANMS挑选后的角点分布更均匀，更有代表性。

对于每一个点，计算其和所有其它点的SSD，如果与其最接近的点（the closest）与次接近的点（the second-closest）的SSD，若最优解远好于次优解（距离在其一半以下），则直接匹配。

对于剩下的点，计算相互之间的SSD，并将其与最优解匹配。

至此可以完成角点之间的对应。通过后面的实验结果可以看出，虽然存在一些不对应的角点，但是对应角点的匹配度非常高。

2.5 RANSAC

RANSAC(RANdom SAmple Consensus)[3], 又叫随机抽样一致性算法。它可以从一组包含“局外点”的观测数据集中，通过迭代方式估计数学模型的参数。它是一种不确定的算法——它有一定的概率得出一个合理的结果；为了提高概率必须提高迭代次数。

简单来说，在这里，为了得到两张图片的匹配程度，我们对其对应角点相对的匹配程度进行度量。

考虑到，当缓慢移动相机的过程中，场景不变，整个视频满足投影变换。因此，对于输入图片，我们随机选取四个对应的角点，并由此得到一个投影变换。在此变换下，对图片1中其余所有角点进行映射，得到的结果与实际的输入图片2里的对应角点匹配。若位置距离不超过某一个固定的常数，则判定匹配成功。最后记录所有匹配成功的点数。显然，匹配点数越多，两张图片在该投影变换下越相似，该变换越有可能是我们需要的变换。

我们进行N次随机四点组的选取，并记录其中最好结果作为最终结果和对应的单应性矩阵(homography)，设为H。伪代码如表1所示：图4展示了一些我自己找的图片中，最好的投影变换及其匹配上的点。可以看出，准确率非常高。

最后，将图片2根据H进行变换，方法是对于图2的四个点先进行变换，得到坐标的范围。在这个范围内的所有点，经过H的逆变换后，得到了在原图里的对应点。如果落在原图的图片内部（横坐标在1到h之间，纵坐标在1到w之间），则认为可以从图1中取对应的颜色填充。注意这里的坐标不一定是整数，因此要进行插值。同时为了使变换后的坐标都为正，还需要给坐标一定的偏移。

此时两张照片中公共部分里，相同坐标对应的点就完全相同了。只需要在中间某处拼接即可。几组示例如图5所示。

2.6 单应性矩阵计算

RANSAC里涉及到的一个问题是，已知四个点和它们的对应点，如何计算它们之间的单应性矩阵。注意到，这里都必须使用射影坐标，我们设它

Algorithm RANSAC(picture1, picture2)

best = 0

for i=1:20000

 随机选取picture1里的四个角点，根据其对应picture2中的四个角点，
 得到仿射变换矩阵H

 tot = 0

 for picture1中所有角点c:

 设其在picture2中原来对应的角点为b

 计算其在H变换下的点为a

 计算a和b之间的距离，若小于 η ，认为匹配成功，tot+1

 if tot > best:

 更新best为tot

 记录下这四个点和可以匹配上的点

得到最适合的矩阵H。

Table 1: RANSAC算法

将 (x_1, y_1, z_1) 的射影坐标变换成了 (x_2, y_2, z_2) 。那么有：

$$\begin{pmatrix} x_2 \\ y_2 \\ z_2 \end{pmatrix} = \begin{pmatrix} H_{11} & H_{12} & H_{13} \\ H_{21} & H_{22} & H_{23} \\ H_{31} & H_{32} & H_{33} \end{pmatrix} \begin{pmatrix} x_1 \\ y_1 \\ z_1 \end{pmatrix} \quad (4)$$

不失一般性，我们可以假设 $z_1 = z_2 = 1$ 。那么就有：

$$x_2 = \frac{H_{11}x_1 + H_{12}y_1 + H_{13}}{H_{31}x_1 + H_{32}y_1 + H_{33}} \quad (5)$$

$$y_2 = \frac{H_{21}x_1 + H_{22}y_1 + H_{23}}{H_{31}x_1 + H_{32}y_1 + H_{33}} \quad (6)$$

即：

$$x_2(H_{31}x_1 + H_{32}y_1 + H_{33}) = H_{11}x_1 + H_{12}y_1 + H_{13} \quad (7)$$

$$y_2(H_{31}x_1 + H_{32}y_1 + H_{33}) = H_{21}x_1 + H_{22}y_1 + H_{23} \quad (8)$$



Figure 4: 最优投影变换和对应点匹配示例

由此可以得到,

$$\mathbf{a}_x^T \mathbf{h} = \mathbf{0} \quad (9)$$

$$\mathbf{a}_y^T \mathbf{h} = \mathbf{0} \quad (10)$$

其中,



Figure 5: 变换后的两张图

$$\mathbf{h} = (H_{11}, H_{12}, H_{13}, H_{21}, H_{22}, H_{23}, H_{31}, H_{32}, H_{33})^T \quad (11)$$

$$\mathbf{a}_x = (-x_1, -y_1, -1, 0, 0, 0, x_2x_1, x_2y_1, x_1)^T \quad (12)$$

$$\mathbf{a}_y = (0, 0, 0, -x_1, -y_1, -1, y_2x_1, y_2y_1, y_2)^T \quad (13)$$

这就把问题转化成了解线性方程组：

$$A\mathbf{h} = \mathbf{0} \quad (14)$$

其中，

$$A = \begin{pmatrix} a_{x1}^T \\ a_{y1}^T \\ \vdots \\ a_{x4}^T \\ a_{y4}^T \end{pmatrix} \quad (15)$$

这里有八个方程，可以确定八个未知数。不妨设 $H_{33} = 1$ ，那么方程变为：

$$A'\mathbf{h}' = -\mathbf{b} \quad (16)$$

其中 \mathbf{A}' 是 \mathbf{A} 矩阵除去最后一列的部分， \mathbf{h}' 是 \mathbf{h} 中除了 H_{33} 的部分， \mathbf{b} 是 \mathbf{A} 矩阵的最后一列。至此，在MATLAB里直接解这个线性方程组即可。

2.7 图片融合

这里只考虑两张图片在水平维度上的融合，即两张图的中间有一定的对应部分，两侧不同，我们在中间选取一条竖直的“缝”。将其连接。

显然，这条缝的左侧完全选取图1的左侧，而右边全部取自于图2的右侧。中间则留有一道宽度为20个像素的缝待拼接。

我的程序中，采用了五层的高斯金字塔。第一层为该点本身（即高斯矩阵只有最中间的点为1，其余点都为0），其余第*i*层是这条缝与大小为 19×19 ， σ 分别为 2^{i-1} 和 2^{i-2} 的高斯矩阵卷积的差。对于每一层，在以拼接点（设为水平方向上坐标

处）为中心，宽度为 $[-2^{i-1}, 2^{i-1}]$ 的区域上进行线性融合。其余部分用图1的左侧或图2的右侧直接加入。即距离

水平距离为j的地方，左边的图的金字塔该层以 $\frac{2^{i-1}+j}{2^i}$ 的权重加入，而右图的金字塔该层以 $\frac{2^{i-1}-j}{2^i}$ 的权重加入。伪代码如表2所示。

这样就实现了对于高频分量更集中的区域，在更小的区域上融合。图6中是几个融合后的例子。

2.8 全景图变换

得到上面的结果后，一般来说，会是一个中间平，两边凸出的图片，整个图片呈一个“凹”字形。这是由于我们的图片都是景物投影到屏幕处造成的。如图7所示。

我们目前的工作是将其全部摊开到了一个平面上，但是我们希望得到的是一个全景图，图8展示了一个平面变成全景图的样子，那么如果我们将已经得到的“凹”字形平面变成一个圆柱的全景图，刚好应该得到一个近似的长方形才对。

转换成全景图的方法和上面将图2进行变换的方法类似，都是先确定新图的坐标范围，然后对于范围内的每一个点，回到原图中找对应点，对于不是整数的点进行插值。一般来说，在拼成的图片中，有一张是不经过变换的，即是正对着的。我们的相机在水平位置上一定是正对着这张照片的

Algorithm blending(picture1, picture2, p)

```
level=5
g = zeros(level, 19, 19);
for i=2:level
    g(i, :, :) = fspecial('gaussian', [19 19], 2i-2);
    g(1, 10, 10) = 1;
    res = p1;
    res(:, p + 2level-1 : w, :) = p2(:, p + 2^(level - 1) : w, :);
    res(:, p - 2level-1 : p + 2level-1, :) = 0;
    totalW = 2level-1;
    for i=1:level
        curW = 2i-1;
        if i==level
            tmp1 = imfilter(p1(:, p-width:p+width, :), g(i, :, :), 'same');
            tmp2 = imfilter(p2(:, p-width:p+width, :), g(i, :, :), 'same');
        else
            tmp1 = imfilter(p1(:, p-width:p+width, :), g(i, :, :), 'same')
                - imfilter(p1(:, p-width:p+width, :), g(i+1, :, :), 'same');
            tmp2 = imfilter(p2(:, p-width:p+width, :), g(i, :, :), 'same')
                - imfilter(p2(:, p-width:p+width, :), g(i+1, :, :), 'same');
        res(:, p-totalW:p-curW, :) = res(:, p-totalW:p-curW, :)
            + tmp1(:, (width+1)-totalW:(width+1)-curW, :);
        res(:, p+curW:p+totalW, :) = res(:, p+curW:p+totalW, :)
            + tmp2(:, (width+1)+curW:(width+1)+totalW, :);
    for j=-curW+1:curW-1
        res(:, p+j, :) = res(:, p+j, :) + tmp2(:, (width+1)+j, :) * (j+curW)/(2curW)
            + tmp1(:, (width+1)+j, :) * (curW-j)/(2curW);
```

Table 2: blending算法



Figure 6: 融合后的结果

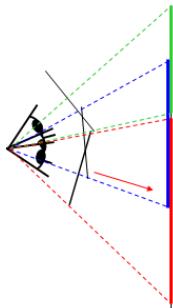


Figure 7: 相机屏幕图示

中心的。另外相机的垂直位置一般很难确定，多数情况下我设置在了图片的中间，其它特殊的情况针对不同的图片进行调整即可。以相机为原点，那么x和y坐标应该表示成：

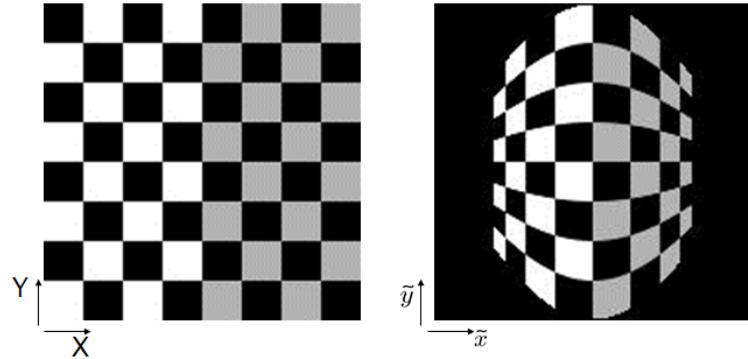


Figure 8: 圆柱形全景图

$$c = \sqrt{r^2 - y^2}; \quad (17)$$

$$x' = \frac{x}{c} \times d; \quad (18)$$

$$y' = \frac{y}{c} \times d; \quad (19)$$

这里d为相机到景物的距离。d的计算依赖于圆柱的半径。由于半径究竟是多少是未知的，只能经验地设置和调整，尽量使得上边缘和下边缘水平即可。

2.9 总体算法

实验的总体算法框架如表3所示。

最终的实验结果在下一部分里展示。

3 实验结果

图9展示了我自己一些图片的实验结果。拼接处在边缘处非常清晰，因此可以看出，在图片内部，拼接的交界处吻合的非常好，几乎看不出痕迹。

图10展示了国家大剧院内部一处的景观，由三幅照片拼接而成。上面的图片是拼接后的结果，下面的是全景图效果。

Algorithm main(picture1, picture2, p)

调用harris和ANMS找到两幅图对应的k个角点

对k个角点进行匹配

调用RANSAC计算出最优的投影变换H

将图2通过 H^{-1} 进行变换，并对图一相应做一些偏移

在p处调用blending融合

将得到的结果映射成全景图

Table 3: 总体算法

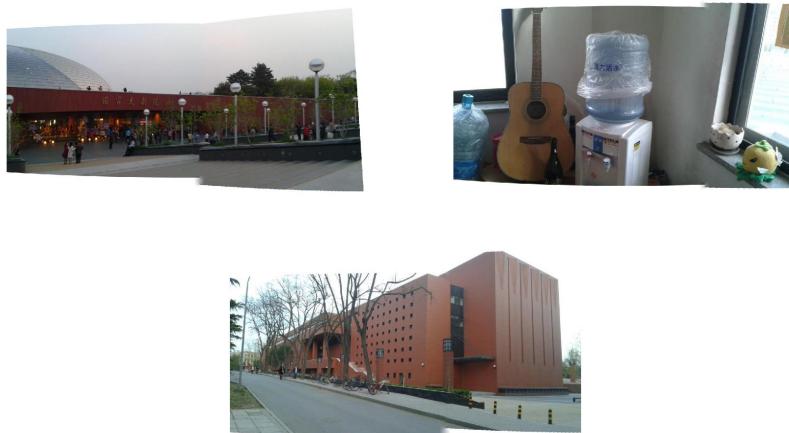


Figure 9: 实验结果

图11展示了美国某校园的一处景观，由5幅照片拼接而成。同样地，上面的图片是拼接后的结果，下面的是全景图效果。

4 实验结果分析

4.1 全景图现象分析

在以上的图中，可以看出，明显较远景的图片变换后下面的弧度要更大一些，这是因为拍远景图的时候，人并非在整个图片的中间，而其实是中



Figure 10: 三幅图片拼接结果

间偏下甚至几乎是最下面，因此当上边缘几乎水平时，下边缘会有比较明显的弧度。下边缘水平时，上边缘可能有一些凹入。

4.2 RANSAC循环次数分析

此外，在RANSAC的计算过程中，我选择了20000次作为计算次数。这是因为在两幅图中，我都找出了100个角点。100个角点中，四元组有 $\frac{100 \times 99 \times 98 \times 97}{4 \times 3 \times 2 \times 1} = 3921225$ 个，这说明，如果两幅图中有10个可以对应上的点的话，取得到这个最优的投影变换中的四元组的可能组合有 $\frac{10 \times 9 \times 8 \times 7}{4 \times 3 \times 2 \times 1} =$



Figure 11: 五幅图片拼接结果

210个，相应的概率约为 $\frac{1}{18672.5}$ ，因此我采用了20000次循环。而在多幅图片的拼接中，由于角点数取多，这个概率降低，则要提高循环次数。

4.3 多幅图片输入顺序分析

在实验中，一个很明显的感受是，当拼接多幅图片的时候，输入顺序非常重要。如果从一头向另一头拼接，则拼到后来很容易出现因为公共部分占当前图片的比例过小，在这一部分找到的角点不够多，从而造成找不到合适的投影变换（如只有5个点或者6个点对应成功）的情况。此时就会出

现异常。因此比较好的一个方法是，采用类似归并的方法，尽量先把原始图片两两组合，然后再将这些图片组合起来，保证每一次拼接都有尽量足够大的公共部分。在我的5幅图片拼接过程中，我刚开始采用的方法是：

1. 图1和图2合成result1.其中图2不动，图1变换。
2. 图4和图5合成result2.其中图4不动，图5变换。
3. result1和图3合成result3.其中图三不动，result1变换。
4. result2和result3合成result4，其中result3不动，result2变换。

这样可以达到比较好的效果。

后面用了4.4中的方法后，便可以依次往原图上拼接，最终我使用的是这种方法。顺序为：

1. 图3和图4合成result1.其中图3不动，图4变换。
2. result1和图2合成result2.其中result1不动，图2变换。
3. result2和图1合成result3.其中result2不动，图1变换。
4. result3和图5合成result4，其中result3不动，图5变换。

4.4 多幅图像特征点选取分析

上一部分提到的问题，改变输入顺序只是一个方面，还有一个比较简单的方法是，对于左边的图片，只选取右边部分的特征点，右边的图片只选取左半部分的特征点，这样把特征点集中在公共区域，方便匹配。这种情况下，程序的鲁棒性和准确度都提高了很多。

4.5 全景图变换

这里比较大的一个困难是调整相机的位置和半径的大小。首先容易知道，正对着的图片的水平方向上的中心一定是相机正对的方向。然而相机的高度变化非常随机，也几乎不可能计算出来，因此需要进行调整。此外

半径的大小也需要调节，调节的时候，可以通过设定相机到景物的距离d来计算，这个d可以是一个关于原图片大小的函数。调整的标准是，当得到的图片原本“凹”字形向外张开的部分变平，则基本上调节成功。

可以对比图9和图6中的图片，可以发现基本上上下边缘都已经水平，包括多幅图片的拼接（图11和图10），几乎达到了一个矩形。说明半径和相机位置的调整还是比较成功的。

4.6 算法优势

理论上来说，如果进行了全景图的变换，可以直接通过平移进行拼接。但是全景图的半径是很难得到的，同时在真正拍照的过程中，相机的移动也是非常复杂和随机的，因此，还是需要先计算单应性矩阵，再进行拼接，最后转化成全景图。这样更加稳定。

该算法的另一个优势在于，对于输入照片的要求很低。甚至两幅图之间如果发生了旋转，也完全可以纠正（投影变换中有旋转）。考虑到在拿相机的过程中，拍多幅照片很难不旋转或者移动，所以这个特点非常重要。

还有一个重要的特点是，在转换到全景图之前，整个算法基本上不用任何手动调节的部分（除了融合的位置，也可以自动设定为公共部分的某一个地方，从结果看来，并没有非常大的影响）。

4.7 算法局限性

当然，该算法也有如下的局限性：

1. 由于投影变换本身的局限性，在镜头背后的东西是无法映射到前端的。它对于全景图的支持最大只能达到180度，并不能达到360度。
2. RANSAC自身的随机性使得程序每一次的结果不见得相同，同时当图比较大，而RANSAC循环次数又不够多的时候，经常会出现找不到合适的投影变换的情况。
3. 我这里并没有给出最合适的拼接点如何自动得出的算法，而是在UI中通过手动设置。手动调节的方法比自动调节要好很多，因为对于不同

的图和不同的变换，很难找出一个普适的寻找拼接点的算法。当然也可以简单地设置公共部分中的某一处，虽然从结果来看差别不是很大，但是这并不能每一次都达到最好的效果。

4. 该算法对角点特征的要求较高。如果公共部分处的角点不多，或者角点的强度不如非公共部分的大，则很难进行匹配或者很难找到精确的投影变换。一个极端的情况就是公共部分的角点不足4个，这样是无论如何也找不到对应的投影变换的。当多幅图片拼接的过程中，非公共部分越来越大，这个缺点变得十分明显。使用了4.4中的方法后，效果会好很多。

4.8 其它优化和说明

虽然这里给出的所有测例都是左右拼接，事实上，上下拼接也是完全可以做到的。只需做简单的修改。在此基础上，任意形状的拼接也可以完成（比如重合的位置是一个角）。只需要把融合的部分从一条缝改为两条垂直的缝即可。

此外由于程序是用MATLAB写成的，效率难免比较低。我在程序写完之后已经把所有的for语句全部改成了矩阵运算。效率提高了不少。最终的结果基本都能在半分钟左右出解。

5 可视化

我在matlab里写的界面如图12左所示，当在下拉菜单中选择对应的测例，则会载入两幅图片到图片框中：

单击“Transform”可以将其进行变换，并将变换后的图重新载入。此时对应点的连线会在figure1中出现。然后调节上面的滑块可以设定融合的水平位置。具体坐标在左边通过文本给出，下面有一个推荐的值，是我事先针对这几组数据调好的参数。如图12右所示。

设定好融合的位置后，单击“blending”即可进行融合，融合后的图在figure4中给出。

然后单击“Cylindrical Projection”得到圆柱全景图，在figure5中。

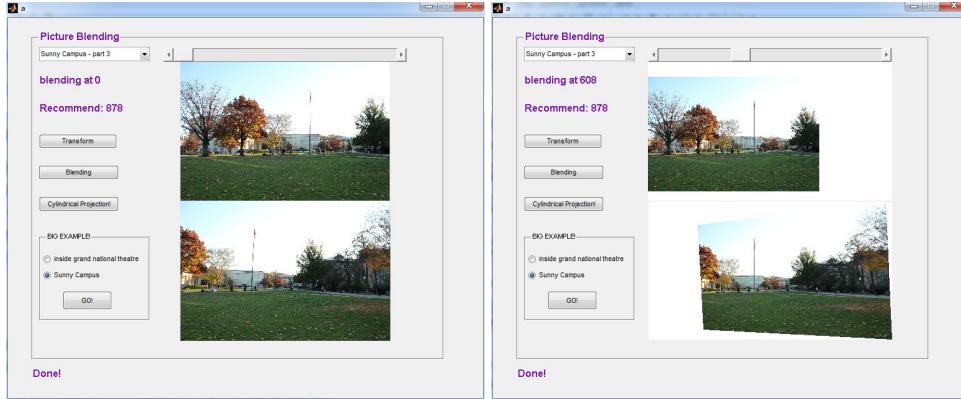


Figure 12: 可视化工具

下面有一个“Big Example”的面板，可以选定其中一个看到多幅图片拼接的示例。上面的是国家大剧院内部，三幅图片拼接的例子，下面是美国校园景色，五幅图片拼接的示意。选择对应选项点“Go!”即可。

6 文件列表

文件列表和说明如表4。

如果想测试我的程序，只需调用doit函数。函数的调用方法为：

```
doit(inname1, inname2, part, result)
```

inname1和inname2是输入文件名，part是融合位置（默认水平融合），result为输出文件名。该函数默认每张图片找200个角点，当两张图片中有一张的宽度超过1500，则RANSAC循环50000次，否则循环20000次。

References

- [1] C. Harris and M. Stephens. A combined corner and edge detector.
Proceedings of the 4th Alvey Vision Conference, pp. 147 to 151.

文件名	说明
autoMatch.m	角点匹配
blending.m	图片融合
computeH.m	计算单应性矩阵H
docampus.m	五幅图片的校园风景拼接
doit.m	对两幅图片完成拼接的全过程
dotheater.m	三幅图片的大剧院内部拼接
extract.m	Harris角点抽取和ANMS
main.m	找到两幅图片的单应性矩阵并变换
RANSAC.m	RANSAC算法
tocylin.m	转换成全景图
a.m/a.fig	界面文件

Table 4: 文件说明

- [2] Matthew Brown et al. Multi-Image Matching using Multi-Scale Oriented Patches. CVPR, 2005.
- [3] Martin A. Fischler and Robert C. Bolles. Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography. Comm. of the ACM 24 (6): 381 - 395.