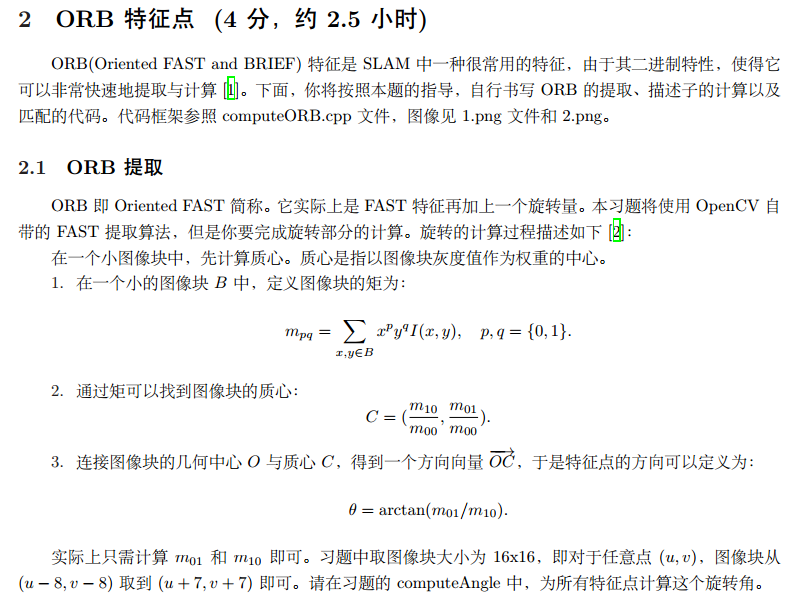
**2018年第5次课习题答案**

**——Vance吴方熠**



解：

计算思路在题中已经给出，特征点及其方向的可视化结果如下图所示：



图1 特征点及其方向的可视化结果

特征点旋转角计算的自编代码部分如下所示：

|  |  |
| --- | --- |
| …115  116  117  118  119  120  121  122  123  124  125  126  127  128  129  130  131  132  133  134  … | ……  // compute the angle  void computeAngle**(**const cv**::**Mat **&**image**,** vector**<**cv**::**KeyPoint**>** **&**keypoints**)** **{**  int half\_patch\_size **=** 8**;**  **for** **(**auto **&**kp **:** keypoints**)** **{**  // START YOUR CODE HERE (~7 lines)  int m\_10 **=** 0**,** m\_01 **=** 0**;**  **for** **(**int u **=** **-**half\_patch\_size**;** u **<=** half\_patch\_size**-**1**;** **++**u**)** **{** // col  **for** **(**int v **=** **-**half\_patch\_size**;** v **<=** half\_patch\_size**-**1**;** **++**v**)** **{** // row  int u1 **=** cvRound**(**kp**.**pt**.**x**)** **+** u**,** v1 **=** cvRound**(**kp**.**pt**.**y**)** **+** v**;** // u1, kp.pt.x(col); v1,kp.pt.y(row)  **if** **(**u1 **>=** 0 **&&** u1 **<** image**.**cols **&&** v1 **>=** 0 **&&** v1 **<** image**.**rows**)** **{**  m\_10 **+=** u **\*** image**.**at**<**uchar**>(**v1**,** u1**);**  m\_01 **+=** v **\*** image**.**at**<**uchar**>(**v1**,** u1**);**  **}** **else** **continue;**  **}**  **}**  kp**.**angle **=** cv**::**fastAtan2**((**float**)**m\_01**,** **(**float**)**m\_10**);**  // END YOUR CODE HERE  **}**  **return;**  **}**  …… |

编程过程的心得：

1. 质心计算以特征点为中心，故在计算过程中任意点(u,v)的坐标不需要加上特征点的坐标；
2. 注意image**.**at**<**uchar**>(**y**,** x**)** 的使用，x**,** y容易搞反。
3. cv**::**fastAtan2 可直接返回0~360°的值。



解：

特征点描述坏点计算结果截图如下所示：



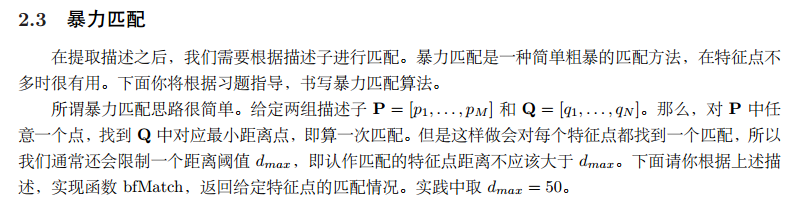
图2 特征点描述坏点计算结果截图

计算出两幅图的坏点/特征点数量分别为：43/638, 7/595，这些坏点产生的原因是因为该特征坏点接近于图像的边界，故其pattern容易超出图像边界，按程序中的处理，其描述子设为空，将其视为坏点。

特征点描述的自编代码部分如下所示：

|  |  |
| --- | --- |
| …398  399  400  401  402  403  404  405  406  407  408  409  410  411  412  413  414  415  416  417  418  419  420  421  422  423  … | ……  // compute the descriptor  void computeORBDesc**(**const cv**::**Mat **&**image**,** vector**<**cv**::**KeyPoint**>** **&**keypoints**,** vector**<**DescType**>** **&**desc**)** **{**  **for** **(**auto **&**kp **:** keypoints**)** **{**  DescType d**(**256**,** **false);**  **for** **(**int i **=** 0**;** i **<** 256**;** i**++)** **{**  // START YOUR CODE HERE (~7 lines)  int up **=** ORB\_pattern**[**4**\***i**];**  int vp **=** ORB\_pattern**[**4**\***i**+**1**];**  int uq **=** ORB\_pattern**[**4**\***i**+**2**];**  int vq **=** ORB\_pattern**[**4**\***i**+**3**];**  double angle **=** kp**.**angle**\***pi**/**180**;**  int u\_p **=** cvRound**(** up**\***cos**(**angle**)** **-** vp**\***sin**(**angle**)** **);**  int v\_p **=** cvRound**(** up**\***sin**(**angle**)** **+** vp**\***cos**(**angle**)** **);**  int u\_q **=** cvRound**(** uq**\***cos**(**angle**)** **-** vq**\***sin**(**angle**)** **);**  int v\_q **=** cvRound**(** uq**\***sin**(**angle**)** **+** vq**\***cos**(**angle**)** **);**  **if** **(**kp**.**pt**.**x**+**u\_p **<** 0 **||** kp**.**pt**.**x**+**u\_p **>** image**.**cols**-**1 **||** kp**.**pt**.**x**+**u\_q **<** 0 **||** kp**.**pt**.**x**+**u\_q **>** image**.**cols**-**1 **||** kp**.**pt**.**y**+**v\_p **<** 0 **||** kp**.**pt**.**y**+**v\_p **>** image**.**rows**-**1 **||** kp**.**pt**.**y**+**v\_q **<** 0 **||** kp**.**pt**.**y**+**v\_q **>** image**.**rows**-**1**)** **{**  d**.**clear**();** // if kp goes outside, set d.clear()  **break;** // next keypoint  **}** **else** **if** **(** image**.**at**<**uchar**>(**cvRound**(**kp**.**pt**.**y**)+**v\_p**,** cvRound**(**kp**.**pt**.**x**)+**u\_p**)** **<=** image**.**at**<**uchar**>(**cvRound**(**kp**.**pt**.**y**)+**v\_q**,** cvRound**(**kp**.**pt**.**x**)+**u\_q**)** **)** **{**  d**[**i**]** **=** 1**;**  **}**  **}**  // END YOUR CODE HERE  desc**.**push\_back**(**d**);**  **}**  …… |

注意：对于pattern中取的点对，应先旋转，后相加，否则旋转中心将会变成像素坐标系的(0, 0)点。



解：

暴力匹配的结果如图3，图4所示。

共匹配到99个特征点，总共花费时长约4.2秒。

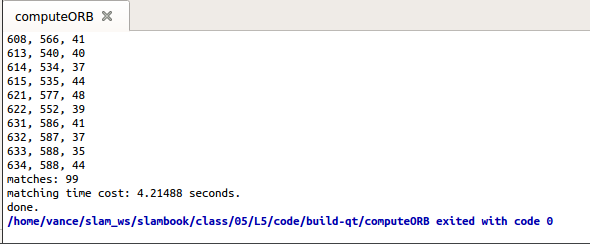


图3 暴力匹配运行结果截图

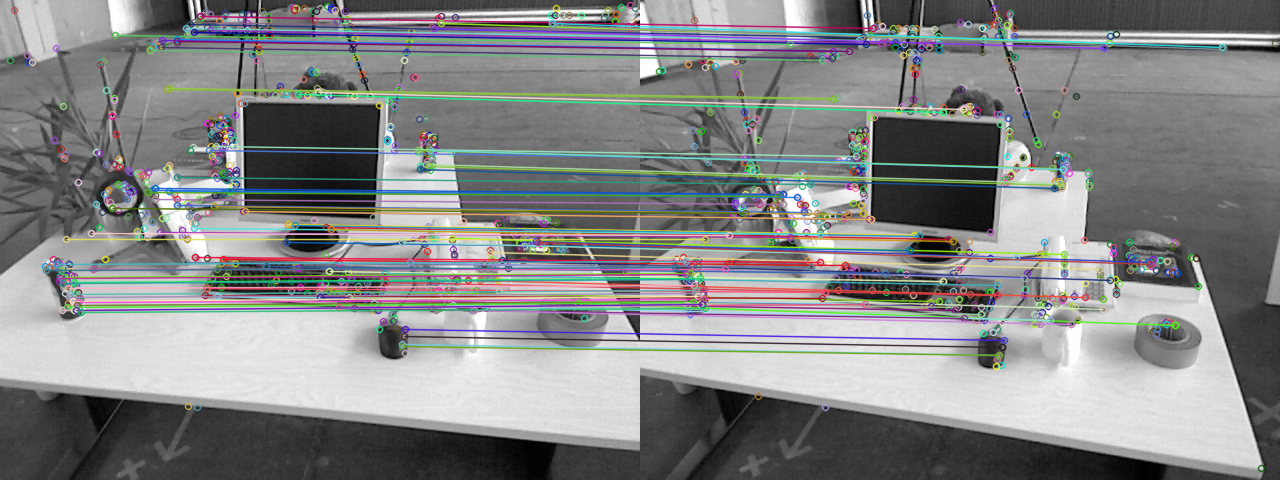


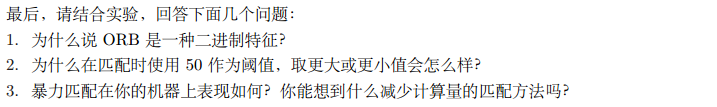
图4 暴力匹配效果可视化（阈值为50）

代码编写心得：

1. 要注意对空描述子的跳过处理；
2. 循环内部的中间变量要及时保存在容器中，同样在每个特征点处理结束后又要及时清空容器。

暴力匹配的自编代码部分如下所示：

|  |  |
| --- | --- |
| …432  433  434  435  436  437  438  439  440  441  442  443  444  445  446  447  448  449  450  451  452  453  454  455  456  457  458  459  460  461  462  463  464  465  466  … | ……  // brute-force matching  void bfMatch**(**const vector**<**DescType**>** **&**desc1**,** const vector**<**DescType**>** **&**desc2**,** vector**<**cv**::**DMatch**>** **&**matches**)** **{**  int d\_max **=** 50**;**  // START YOUR CODE HERE (~12 lines)  // find matches between desc1 and desc2.  vector**<**int**>** ham\_dis**,** ham\_dis\_index**;**  cv**::**DMatch match**;**  **for** **(**int i**=**0**;** i**<**desc1**.**size**();** i**++)** **{**  **if** **(**desc1**[**i**].**empty**())** **continue;**  **for** **(**int j**=**0**;** j**<**desc2**.**size**();** j**++)** **{**  **if** **(**desc2**[**j**].**empty**())** **continue;**  int d **=** 0**;**  **for** **(**int k**=**0**;** k**<**256**;** k**++)** **{**  **if** **(**desc1**[**i**][**k**]** **!=** desc2**[**j**][**k**])** d**++;**// 计算汉明距离  **}**  ham\_dis**.**push\_back**(**d**);** // desc1中的第i个有效描述子与desc2中所有有效描述子的汉明距离  ham\_dis\_index**.**push\_back**(**j**);** // 对应的索引  **}**  vector**<**int**>::**iterator d\_min **=** min\_element**(**ham\_dis**.**begin**(),** ham\_dis**.**end**());** // 最小值  int n **=** distance**(**ham\_dis**.**begin**(),** d\_min**);**// 最小值索引位置  **if** **(\***d\_min **<=** d\_max**)** **{**  match**.**queryIdx **=** i**;**  match**.**trainIdx **=** ham\_dis\_index**[**n**];**  match**.**distance **=** **\***d\_min**;**  matches**.**push\_back**(**match**);**  **}**  ham\_dis**.**clear**();**  ham\_dis\_index**.**clear**();**  **}**  // END YOUR CODE HERE  …… |



答：

1.主要是因为ORB特征采用的描述子为带旋转的BRIEF描述。所谓BRIEF描述是指由0-1组成的字符串，即二进制串的特征描述符，故有此说。

2.若取更小值（如图5，取为30），则会出现匹配的点数量过于稀少，不利于后期的相机位姿变换的计算；若取更大值（如图6，取为70），则会出现误匹配现象，这对后续的计算更加不利。当然，做完特征点的匹配后，我们还可以通过**RANSAC方法**计算透视变换矩阵来筛选符合相同透视的特征点，这样做可以去除很多错误的匹配。

3.在本人机器人匹配99个点用时4.2秒，感觉时间有点长。对于减少计算量的匹配方法，可以使用**FLANN快速近似最近邻算法**来寻找匹配关系。

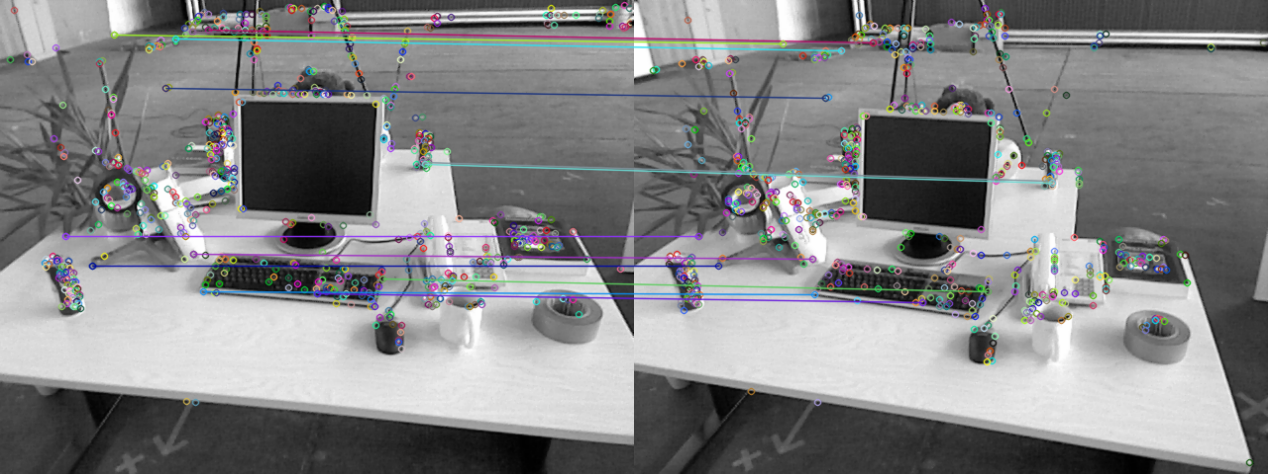
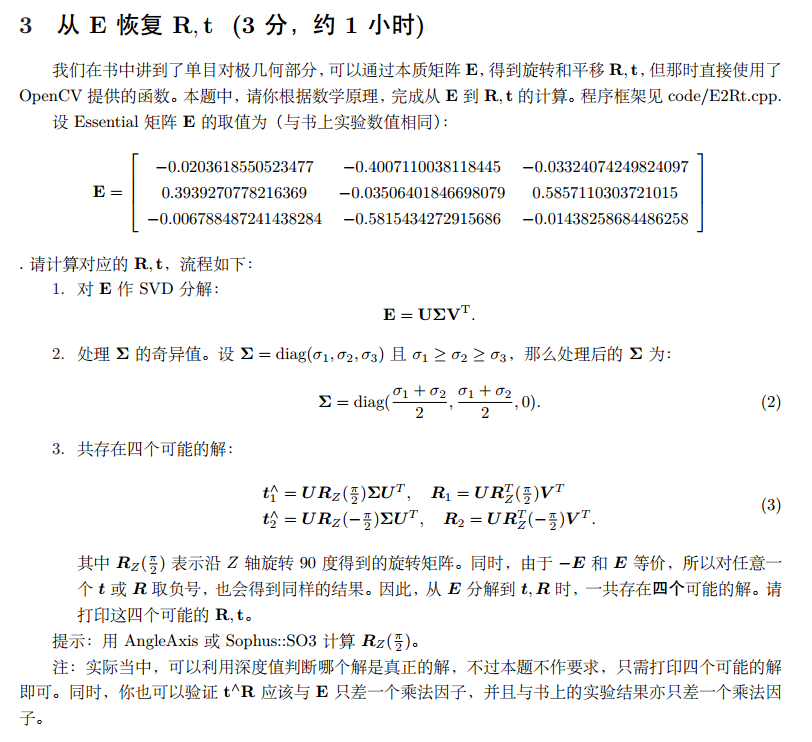


图5 阈值为30时的匹配效果



图6 阈值为70时的匹配效果



解：

解题思路已由题中给出，程序运行结果如下所示，可以看出SVD分解结果正确，**t**^**R**乘上一个乘法因子s后与**E**几乎一致。

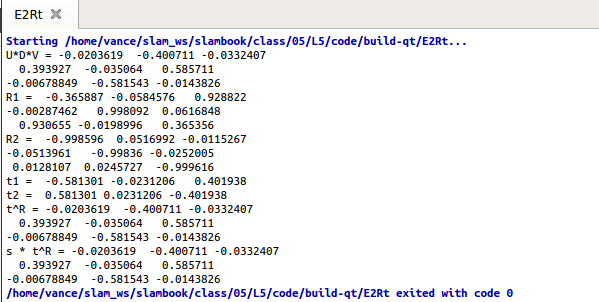
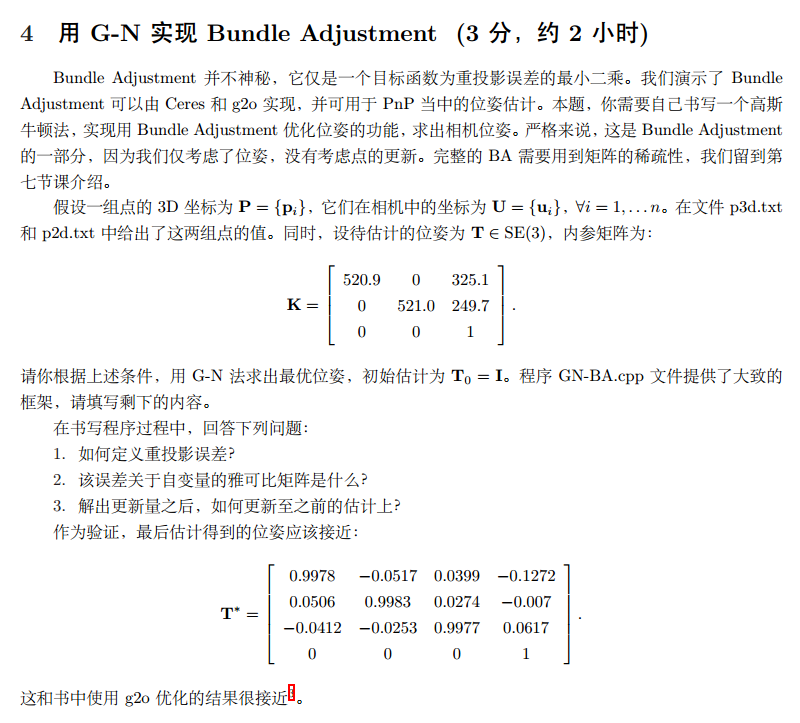


图7 由**E**恢复**R,t**的程序运行结果截图

自编代码部分如下所示：

|  |  |
| --- | --- |
| …  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  … | ……  #include <algorithm>  bool compare**(**double a**,** double b**)**  **{**  **return** a **>** b**;**  **}**  int main**(**int argc**,** char **\*\***argv**)** **{**  // 给定Essential矩阵  Matrix3d E**;**  E **<<** **-**0.0203618550523477**,** **-**0.4007110038118445**,** **-**0.03324074249824097**,**  0.3939270778216369**,** **-**0.03506401846698079**,** 0.5857110303721015**,**  **-**0.006788487241438284**,** **-**0.5815434272915686**,** **-**0.01438258684486258**;**  // 待计算的R,t  Matrix3d R**;**  Vector3d t**;**  // SVD and fix sigular values  // START YOUR CODE HERE  JacobiSVD**<**Matrix3d**>** svd**(**E**,** ComputeFullV **|** ComputeFullU **);**  MatrixXd U **=** svd**.**matrixU**(),** V **=** svd**.**matrixV**(),** sv **=** svd**.**singularValues**();**  double DD**[**3**]** **=** **{**sv**(**0**),** sv**(**1**),** sv**(**2**)};**  sort**(**DD**,** DD**+**2**,** compare**);**  Matrix3d D **=** Matrix3d**::**Zero**();**  D**(**0**)** **=** **(**DD**[**0**]+**DD**[**1**])/**2**;**  D**(**4**)** **=** D**(**0**);**  cout **<<** "U\*D\*V = " **<<** U**\***D**\***V**.**transpose**()** **<<** endl**;**  // END YOUR CODE HERE  // set t1, t2, R1, R2  // START YOUR CODE HERE  Matrix3d t\_wedge1**;**  Matrix3d t\_wedge2**;**  Matrix3d R1**;**  Matrix3d R2**;**  Matrix3d R\_z1 **=** AngleAxisd**(**M\_PI**/**2**,** Vector3d**(**0**,**0**,**1**)).**toRotationMatrix**();**  Matrix3d R\_z2 **=** AngleAxisd**(-**M\_PI**/**2**,** Vector3d**(**0**,**0**,**1**)).**toRotationMatrix**();**  R1 **=** U **\*** R\_z1**.**transpose**()** **\*** V**.**transpose**();**  R2 **=** U **\*** R\_z2**.**transpose**()** **\*** V**.**transpose**();**  t\_wedge1 **=** U **\*** R\_z1 **\*** D **\*** U**.**transpose**();**  t\_wedge2 **=** U **\*** R\_z2 **\*** D **\*** U**.**transpose**();**  // END YOUR CODE HERE  cout **<<** "R1 = " **<<** R1 **<<** endl**;**  cout **<<** "R2 = " **<<** R2 **<<** endl**;**  cout **<<** "t1 = " **<<** Sophus**::**SO3d**::**vee**(**t\_wedge1**).**transpose**()** **<<** endl**;**  cout **<<** "t2 = " **<<** Sophus**::**SO3d**::**vee**(**t\_wedge2**).**transpose**()** **<<** endl**;**  // check t^R=E up to scale  Matrix3d tR **=** t\_wedge1 **\*** R1**;**  cout **<<** "t^R = " **<<** tR **<<** endl**;**  cout **<<** "s \* t^R = " **<<** E**(**0**)/**tR**(**0**)** **\*** tR **<<** endl**;**  **return** 0**;**  **}**  …… |



答：

运行结果截图如下所示：

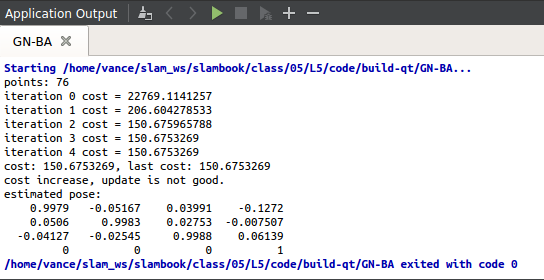


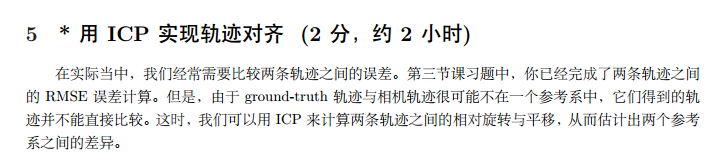
图8 BA优化的结果截图

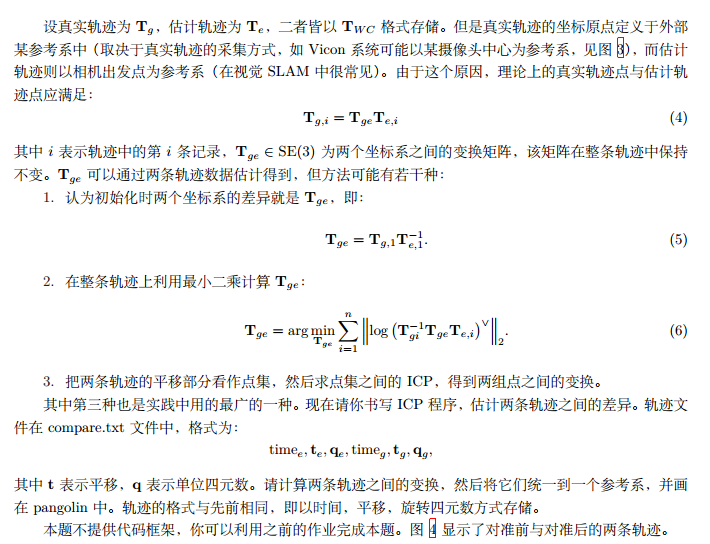
1.重投影误差是将像素坐标（观测值，这里即p2d.txt中的数据）与3D点（这里即p3d.txt中的数据）按当前位姿（这里即每次迭代更新的T）估计出的投影相比较得到的误差。在程序中的第62行定义了重投影误差（这里是一个二维向量），在第76行对其进行了赋值。

2.误差关于自变量的雅克比矩阵，可以参照《视觉SLAM十四讲》中公式（7.36）及（7.45），是误差函数（这里是重投影误差e）关于自变量增量（这里是相机位姿T）的一阶变化关系。

3.关于更新量的更新问题，由于更新量计算出来是一个6维向量，即李代数的左乘扰动，将其转换到李群上进行左乘即可更新位姿。代码实现位于第124行。

因程序代码过长，此处不再显示，详见附件“GN-BA.cpp”文档。





解：

因程序代码过长，此处不再显示，详见附件“ICP\_trajectory.cpp”文档。程序思路如下：

1. 读取两个文档的数据；
2. 用SVD求解ICP；
3. 依照求解出变换关系配准位姿；
4. 画图；

运行结果截图如下：

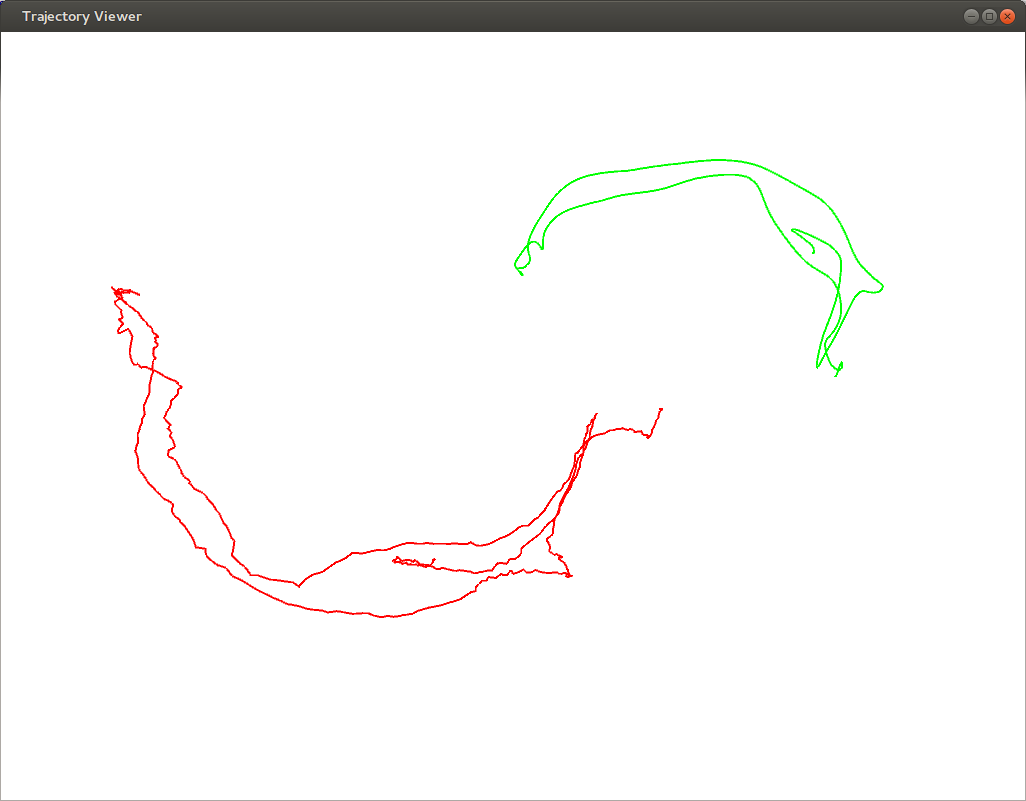


图9 两轨迹对齐前的相对位置（红：groundtruth，绿：estimate）



图10 用ICP对齐轨迹后的效果（红：groundtruth，绿：estimate）