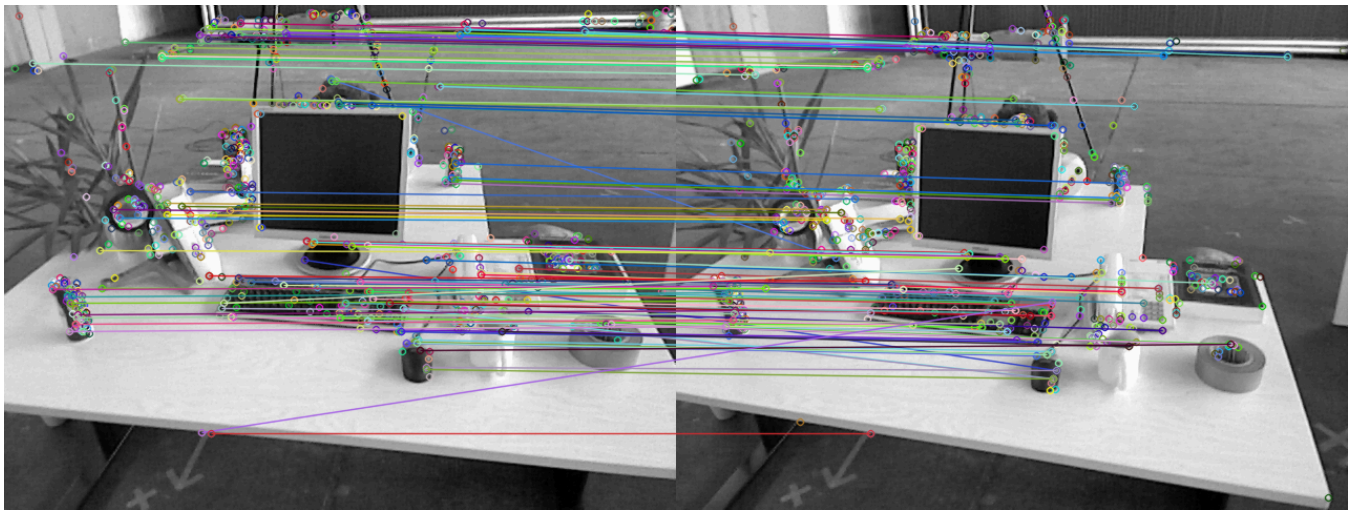
2.ORB特征点

ORB提取并计算角度如下：

暴力匹配结果如下：



2.4 多线程ORB

1.为什么说ORB是一种二进制特征？

因为ORB采用二进制的描述子来描述每个特征点的特征信息

2.为什么在匹配时使用50作为阈值，取更大或者更小值会怎么样？

50是一个经验值，如果取大了会带来更多的误匹配，取小了匹配的点数会比较少，需要折中取值

3.暴力匹配在你的机器上表现如何？是否有减少计算量的匹配方法？

在我的机器上需要100-130ms，可以采用快速最近邻(FLANN)的方法减少匹配时间

```
627, 584, 33
628, 583, 35
629, 573, 43
633, 587, 34
634, 588, 23
636, 470, 40
637, 593, 36
方法 bf match 平均调用时间/次数: 109.184/1 毫秒.
matches: 106
done.
```

4.多线程版本相比单线程版本是否有提升？在你的机器上大约能提升多少性能？

计算特征点质心角度的函数我参考的ORB_SLAM3的写法，计算比较快，提速不明显，跑一次均在0.3+ms。计算描述子的函数耗时较多，多线程版本相比单线程版本快速很多，单线程大约3ms，多线程只需0.5ms

```
xwl@xwl-System-Product-Name:~/Documents/视觉SLAM基础与VIO进阶/L5/compute_ORB/bin$ ./computeORB
keypoints: 638
image.rows: 480 image.cols: 640
方法 compute angle 平均调用时间/次数: 0.360621/1 毫秒.
方法 compute angle mt 平均调用时间/次数: 0.3263/1 毫秒.
bad/total: 43/638
方法 compute orb descriptor 平均调用时间/次数: 2.88189/1 毫秒.
bad/total: 43/638
方法 compute orb descriptor mt 平均调用时间/次数: 0.552686/1 毫秒.
keypoints: 595
image.rows: 480 image.cols: 640
bad/total: 7/595
```

3.从E恢复R,t

程序运行结果为：

```
xwl@xwl-System-Product-Name:~/Documents/视觉SLAM基础与VIO进阶/L5/E2Rt/bin$ ./E2Rt
R1 =   -0.365887  -0.0584576    0.928822
-0.00287462    0.998092   0.0616848
   0.930655  -0.0198996    0.365356
R2 =  -0.998596  0.0516992 -0.0115267
-0.0513961   -0.99836 -0.0252005
 0.0128107  0.0245727  -0.999616
t1 =  -0.581301
-0.0231206
  0.401938
t2 =  0.581301
0.0231206
-0.401938
t^R =  -0.0203619   -0.400711  -0.0332407
   0.393927   -0.035064    0.585711
-0.00678849   -0.581543  -0.0143826
```

4.用G-N实现Bundle Adjustment中的位姿估计

1.如何定义重投影误差？

3D点投影到相机成像平面中的位置与观测位置作差

$$\boldsymbol{e_i} = \boldsymbol{u_i} - \frac{1}{\boldsymbol{s_i}}\boldsymbol{K}exp(\hat{\xi})P_i$$

2.该误差关于自变量的雅可比矩阵是什么？

$$\frac{\partial \boldsymbol{e}}{\partial \delta \boldsymbol{\xi}} = -\begin{bmatrix} \frac{f_x}{Z'} & 0 & -\frac{f_x X'}{Z'^2} & -\frac{f_x X' Y'}{Z'^2} & f_x + \frac{f_x X^2}{Z'^2} & -\frac{f_x Y'}{Z'} \\ 0 & \frac{f_y}{Z'} & -\frac{f_y Y'}{Z'^2} & -f_y - \frac{f_y Y^2}{Z'^2} & \frac{f_y X' Y'}{Z'^2} & \frac{f_y X'}{Z'} \end{bmatrix}$$

3.解出更新量之后，如何更新至之前的估计上？

左乘或者右乘微小扰动$exp(dx)$均可

左乘结果：

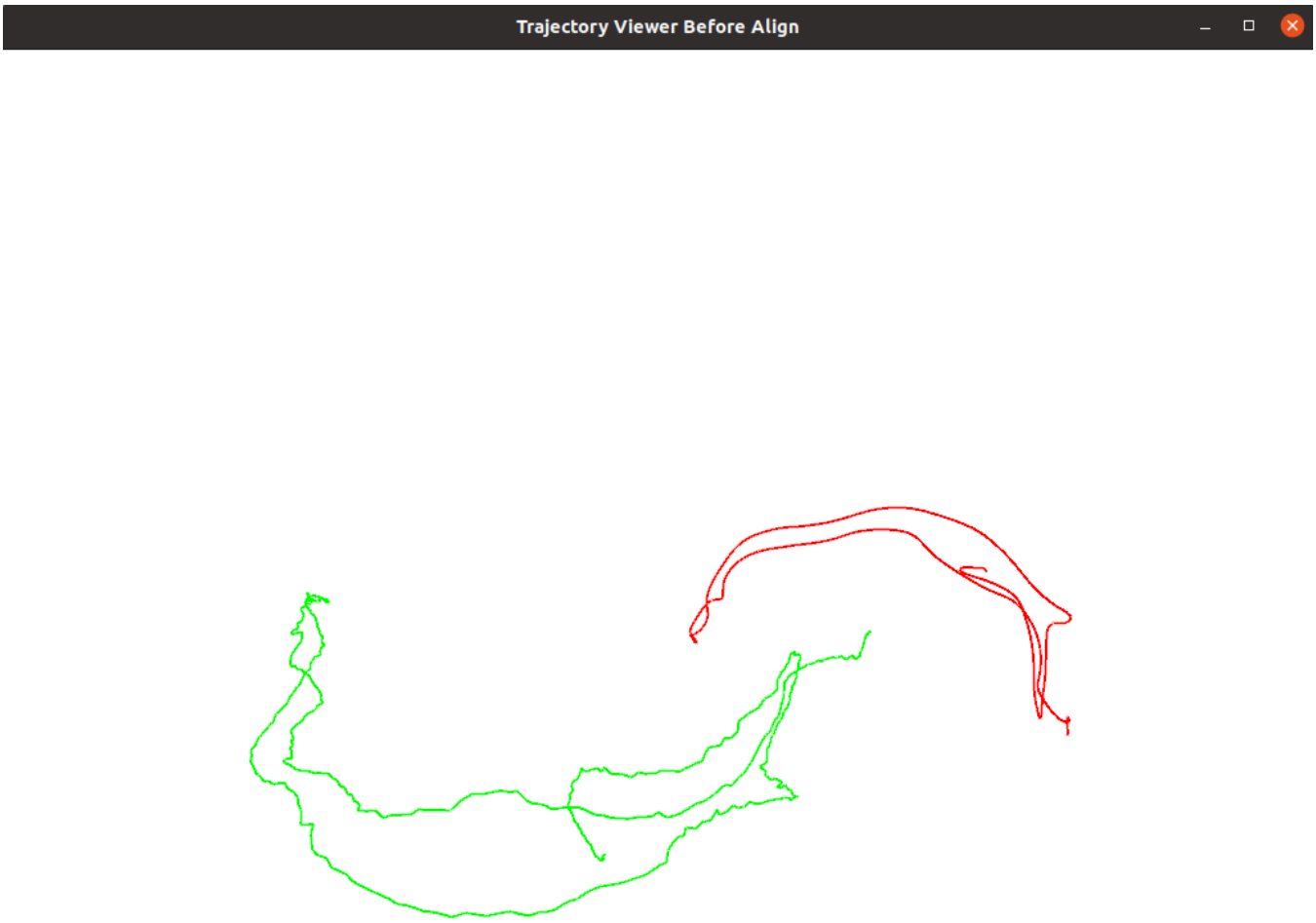
```
xwl@xwl-System-Product-Name:~/Documents/视觉SLAM基础与VIO进阶/L5/GN_BA/bin$ ./GN-BA
points: 76
iteration 0 cost=622769.1141257
iteration 1 cost=12206.604278533
iteration 2 cost=12150.675965788
iteration 3 cost=12150.6753269
iteration 4 cost=12150.6753269
cost: 150.6753269, last cost: 150.6753269
estimated pose:
   0.997866186837  -0.0516724392947   0.0399128072711      -0.127226621
   0.050595918872    0.998339770315   0.027527368229 -0.00750679765341
 -0.0412689491074  -0.0254492048097   0.998823914318    0.061386084881
                0                 0                0                 1
```

右乘结果：

```
xwl@xwl-System-Product-Name:~/Documents/视觉SLAM基础与VIO进阶/L5/GN_BA/bin$ ./GN-BA
points: 76
iteration 0 cost=622769.1141257
iteration 1 cost=12206.604278533
iteration 2 cost=12151.324611545
iteration 3 cost=12150.678510591
iteration 4 cost=12150.67535943
iteration 5 cost=12150.67532706
iteration 6 cost=12150.675326902
iteration 7 cost=12150.6753269
iteration 8 cost=12150.6753269
iteration 9 cost=12150.6753269
cost: 150.6753269, last cost: 150.6753269
estimated pose:
   0.997866186837  -0.0516724392948   0.0399128072707      -0.127226621
   0.050595918872    0.998339770315   0.0275273682287 -0.00750679765308
  -0.041268949107  -0.0254492048094    0.998823914318   0.0613860848806
                0                 0                 0                 1
```

5.*用ICP实现轨迹对齐

轨迹对齐之前：



轨迹对齐之后：


Trajectory Viewer After Align