Visual Odomentry 代码阅读报告

唐誉铭

2019 年 1 月 29 日

1 论文概要

1.1 论文要解决的问题

论文中提出了一种基于双镜摄像头拍摄的视频进行三维重建的方法。3D 感知是计算机视觉和机器人技术的核心课题之一。在实践中，相机的分辨率严重受限，并且对生成结果的准确性有着较高的要求。并且，该实时系统要求很高的计算性能，很多嵌入式设备不能提供那么高的计算性能，特别是无法使用 FPGA 的情况下。

论文中提到的处理方法实现了可以运行数千个特征点匹配的实时场景计算，一种简单而快速的视觉测距算法并且能推算出拍摄主体的自我运动。

1.2 论文采用的主要方法

在处理过程中，程序每次读入一帧左右两张图片，把图片加入 ringbuffer 中，并且用图 1 两种滤波器检测特征。

-1	-1	-1	-1	-1
-1	+1	+1	+1	-1
-1	+1	+8	+1	-1
-1	+1	+1	+1	-1
-1	-1	-1	-1	-1

(a) Blob detector

-1	-1	0	+1	+1
-1	-1	0	+1	+1
0	0	0	0	0
+1	+1	0	-1	-1
+1	+1	0	-1	-1

(b) Corner detector

图 1: 采用的滤波器

分别求取结果较少的稀疏特征值和结果比较多的稠密特征值，并且分别用非极大值抑制和非极小值抑制处理得到的结果。

下面的步骤是稀疏特征值匹配，首先对图像分成各个区域（bin）对每个特征点使用“环形匹配”，既先对于“前一帧”的左视图的每一个点，在“前一帧”右视图相似的位置找到对应的特征点，再找到该对应特征点在“当前帧”右视图对应的特征点，再寻找“当前帧”左视图对应的特征点，最后寻找“上一

帧”左视图中的特征点对应的“当前帧”左视图对应的特征点。如果最后找到的特征点就是初始特征点，则“环形匹配”成功，否则失败。过程如图 2,3 所示。

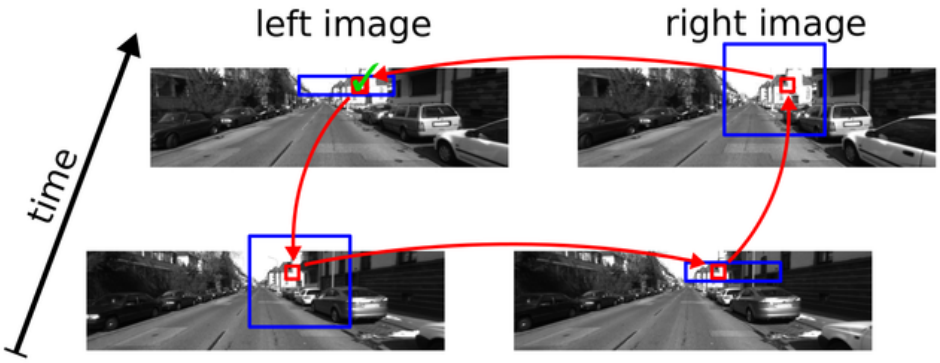

图 2: 环形匹配过程

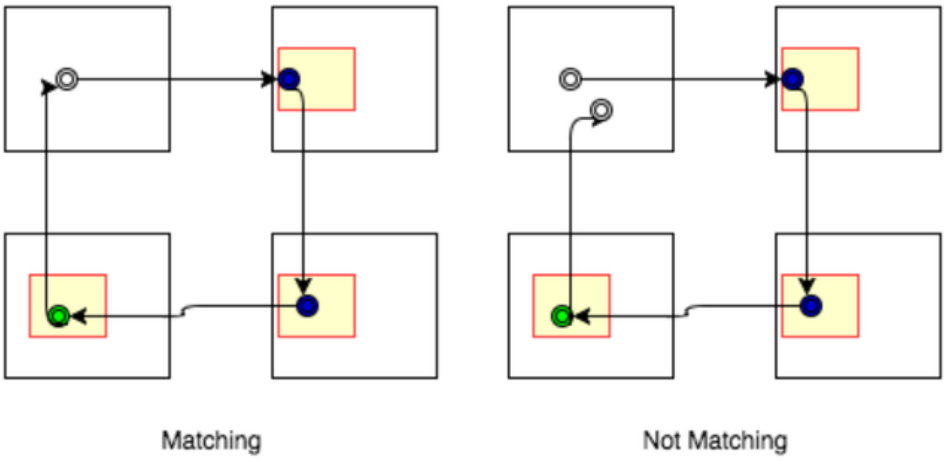

图 3: 环形匹配是否成功

对匹配的结果使用 2d Delaunay 三角剖分来消去异常值，接下来对每一个匹配进行分析，统计每一个匹配完成的特征点在另外三张图的偏移距离大小，从而计算出每一个区域（bin）偏移范围，通过偏移范围确定下次稠密特征匹配的搜索范围。

然后进行一次稠密点特征“环形匹配”，与稀疏特征匹配不同的是，后者的搜索范围是人工规定的比较大的区域，而前者的搜索范围是由后者统计出偏移范围决定的。然后通过抛物线拟合的子像素细化来进一步改进特征定位，再使用三角剖分去除异常值。

对匹配完成的结果再进行“桶”操作，把图像分为若干个桶，每个桶只随机保留若干个特征点，这样保证了特征点数量不会过多，并且较为均匀的分布在整个图像上。

完成特征匹配后再进行动作推测。论文中提到的方法使用了 RANSAC 算法。在每轮 RANSAC 算法中，先随机选择 3 个点作为初始点，对这 3 个点建立动作预测模型，迭代求算出最优的位置调整参数，然后对所有特征点使用该动作预测模型，观测值与预测值只差小于某个阈值的特征点被标记为 inlier，inlier 越多代表模型越好，如果下一个模型的 inlier 数量比当前多，就更新模型。

2 代码分析

2.1 主要类以及数据结构分析

2.1.1 VisualOdometryStereo

主要功能类，包括动作预测，动作参数计算等功能

```
class VisualOdometryStereo : public VisualOdometry {

public:

    struct parameters : public VisualOdometry::parameters {
        // 相关参数
        double  base;              // baseline (meters)
        int32_t ransac_iters;      // RANSAC迭代次数
        double  inlier_threshold; // inlier要求的偏差阈值
        bool    reweighting;
    };

    bool process (uint8_t *I1,uint8_t *I2,int32_t* dims,bool replace=false);
    // 主要处理函数，读入两张照片

    using VisualOdometry::process;



private:

    std::vector<double>  estimateMotion (std::vector<Matcher::p_match> p_matched);
    // 预测动作
    enum                 result { UPDATED, FAILED, CONVERGED };
    result               updateParameters(std::vector<Matcher::p_match>
    ↪  &p_matched,std::vector<int32_t> &active,std::vector<double> &tr,double
    ↪  step_size,double eps);
    // 更新动作变化参数
    void                 computeObservations(std::vector<Matcher::p_match>
    ↪  &p_matched,std::vector<int32_t> &active);
    void                 computeResidualsAndJacobian(std::vector<double>
    ↪  &tr,std::vector<int32_t> &active);
    // 计算相关参数
    std::vector<int32_t> getInlier(std::vector<Matcher::p_match>
    ↪  &p_matched,std::vector<double> &tr);
```

```
    // 计算inliers (内群点)

    double *X,*Y,*Z;    // 3d points
    double *p_residual; // residuals (p_residual=p_observe-p_predict)

    // parameters
    parameters param;
};
```

2.1.2 VisualOdometry

VisualOdometryStereo 的基类

```
class VisualOdometry {

public:

    // 镜头相关参数
    struct calibration {
        double f;  // focal length (in pixels)
        double cu; // principal point (u-coordinate)
        double cv; // principal point (v-coordinate)
        calibration () {
            ...
        }
    };

    // "桶"操作参数
    struct bucketing {
        int32_t max_features;  // maximal number of features per bucket
        double  bucket_width;  // width of bucket
        double  bucket_height; // height of bucket
        bucketing () {
            ...
        }
    };

    // general parameters
    struct parameters {
        Matcher::parameters         match;            // matching parameters
        VisualOdometry::bucketing   bucket;           // bucketing parameters
        VisualOdometry::calibration calib;            // camera calibration parameters
```

```
30      };
31
32
33      // returns previous to current feature matches from internal matcher
34      std::vector<Matcher::p_match> getMatches () { return matcher->getMatches(); }
35
36      // streams out the current transformation matrix Tr_delta
37      friend std::ostream& operator<< (std::ostream &os,VisualOdometry &viso) {
38          Matrix p = viso.getMotion();
39          os << p.val[0][0] << " " << p.val[0][1] << " "  << p.val[0][2]  << " "  <<
            ↪  p.val[0][3] << " ";
40          os << p.val[1][0] << " " << p.val[1][1] << " "  << p.val[1][2]  << " "  <<
            ↪  p.val[1][3] << " ";
41          os << p.val[2][0] << " " << p.val[2][1] << " "  << p.val[2][2]  << " "  <<
            ↪  p.val[2][3];
42          return os;
43      }
44
45  protected:
46
47      // 更新动作
48      bool updateMotion ();
49
50      // 计算位置变化矩阵
51      Matrix transformationVectorToMatrix (std::vector<double> tr);
52
53      // 预测动作
54      virtual std::vector<double> estimateMotion (std::vector<Matcher::p_match>
        ↪  p_matched) = 0;
55
56      // 获得一个随机取样
57      std::vector<int32_t> getRandomSample (int32_t N,int32_t num);
58
59      Matrix                       Tr_delta;   // transformation (previous -> current
        ↪  frame)
60      bool                         Tr_valid;   // motion estimate exists?
61      Matcher                     *matcher;    // feature matcher
62      std::vector<int32_t>         inliers;    // inlier set
63      double                      *J;          // jacobian
64      double                      *p_observe;  // observed 2d points
65      double                      *p_predict;  // predicted 2d points
```

```
66      std::vector<Matcher::p_match>  p_matched;  // feature point matches
67
68  private:
69
70      parameters                     param;      // common parameters
71  };
```

2.1.3 Matcher

提供维护一个 ringbuffer，计算输入图片特征值，对特征值进行匹配，进行环形比配，对匹配结果进行桶操作，优化，去除无效值等特征计算以及匹配相关操作

```
1   class Matcher {
2
3   public:
4
5       // 各项参数
6       struct parameters {
7
8           int32_t nms_n;                  // non-max-suppression: min. distance between
            ↪  maxima (in pixels)
9           int32_t nms_tau;                // non-max-suppression: interest point
            ↪  peakiness threshold
10          int32_t match_binsize;          // matching bin width/height (affects
            ↪  efficiency only)
11          int32_t match_radius;           // matching radius (du/dv in pixels)
12          int32_t match_disp_tolerance;   // dv tolerance for stereo matches (in pixels)
13          int32_t outlier_disp_tolerance; // outlier removal: disparity tolerance (in
            ↪  pixels)
14          int32_t outlier_flow_tolerance; // outlier removal: flow tolerance (in pixels)
15          int32_t multi_stage;            // 0=disabled,1=multistage matching (denser
            ↪  and faster)
16          int32_t half_resolution;        // 0=disabled,1=match at half resolution,
            ↪  refine at full resolution
17          int32_t refinement;             // refinement (0=none,1=pixel,2=subpixel)
18          double  f,cu,cv,base;           // calibration (only for match prediction)
19
20          parameters () {
21              ...
22          }
23      };
24
```

```
25      // 记录环形匹配完成后比配的四个点
26      struct p_match {
27          float   u1p,v1p; // u,v-coordinates in previous left  image
28          int32_t i1p;     // feature index (for tracking)
29          float   u2p,v2p; // u,v-coordinates in previous right image
30          int32_t i2p;     // feature index (for tracking)
31          float   u1c,v1c; // u,v-coordinates in current  left  image
32          int32_t i1c;     // feature index (for tracking)
33          float   u2c,v2c; // u,v-coordinates in current  right image
34          int32_t i2c;     // feature index (for tracking)
35      };
36
37      // 读入一帧两张照片并且计算特征值
38      void pushBack (uint8_t *I1,uint8_t* I2,int32_t* dims,const bool replace);
39
40      // 匹配特征值
41      void matchFeatures(int32_t method, Matrix *Tr_delta = 0);
42
43      // 进行“桶”操作
44      void bucketFeatures(int32_t max_features,float bucket_width,float bucket_height);
45
46      // return vector with matched feature points and indices
47      std::vector<Matcher::p_match> getMatches() { return p_matched_2; }
48
49      private:
50
51      // structure for storing interest points
52      struct maximum {
53          int32_t u;   // u-coordinate
54          int32_t v;   // v-coordinate
55          int32_t val; // value
56          int32_t c;   // class
57          int32_t d1,d2,d3,d4,d5,d6,d7,d8; // descriptor
58          maximum() {}
59          maximum(int32_t u,int32_t v,int32_t val,int32_t c):u(u),v(v),val(val),c(c) {}
60      };
61
62      // 记录统计出的搜索范围
63      struct range {
64          float u_min[4];
65          float u_max[4];
```

```
66          float v_min[4];
67          float v_max[4];
68      };
69
70      // 统计每个匹配点位置在不同图中的变化值
71      struct delta {
72          float val[8];
73          delta () {}
74          delta (float v) {
75          for (int32_t i=0; i<8; i++)
76              val[i] = v;
77          }
78      };
79
80      // 非极大值抑制
81      void nonMaximumSuppression (int16_t* I_f1,int16_t* I_f2,const int32_t*
        ↪  dims,std::vector<Matcher::maximum> &maxima,int32_t nms_n);
82
83      // descriptor functions
84      inline uint8_t saturate(int16_t in);
85      void filterImageAll (uint8_t* I,uint8_t* I_du,uint8_t* I_dv,int16_t*
        ↪  I_f1,int16_t* I_f2,const int* dims);
86      void filterImageSobel (uint8_t* I,uint8_t* I_du,uint8_t* I_dv,const int* dims);
87      inline void computeDescriptor (const uint8_t* I_du,const uint8_t* I_dv,const
        ↪  int32_t &bpl,const int32_t &u,const int32_t &v,uint8_t *desc_addr);
88      inline void computeSmallDescriptor (const uint8_t* I_du,const uint8_t* I_dv,const
        ↪  int32_t &bpl,const int32_t &u,const int32_t &v,uint8_t *desc_addr);
89      void computeDescriptors (uint8_t* I_du,uint8_t* I_dv,const int32_t
        ↪  bpl,std::vector<Matcher::maximum> &maxima);
90
91      void getHalfResolutionDimensions(const int32_t *dims,int32_t *dims_half);
92      uint8_t* createHalfResolutionImage(uint8_t *I,const int32_t* dims);
93
94      // 计算图片特征
95      void computeFeatures (uint8_t *I,const int32_t* dims,int32_t* &max1,int32_t
        ↪  &num1,int32_t* &max2,int32_t &num2,uint8_t* &I_du,uint8_t* &I_dv,uint8_t*
        ↪  &I_du_full,uint8_t* &I_dv_full);
96
97      void computePriorStatistics (std::vector<Matcher::p_match> &p_matched,int32_t
        ↪  method);
```

```
98      void createIndexVector (int32_t* m,int32_t n,std::vector<int32_t> *k,const
        ↪  int32_t &u_bin_num,const int32_t &v_bin_num);
99      inline void findMatch (int32_t* m1,const int32_t &i1,int32_t* m2,const int32_t
        ↪  &step_size, std::vector<int32_t> *k2,const int32_t &u_bin_num,const int32_t
        ↪  &v_bin_num,const int32_t &stat_bin, int32_t& min_ind,int32_t stage,bool
        ↪  flow,bool use_prior,double u_=-1,double v_=-1);
100     void matching (int32_t *m1p,int32_t *m2p,int32_t *m1c,int32_t *m2c, int32_t
        ↪  n1p,int32_t n2p,int32_t n1c,int32_t n2c, std::vector<Matcher::p_match>
        ↪  &p_matched,int32_t method,bool use_prior,Matrix *Tr_delta = 0);
101
102     void removeOutliers (std::vector<Matcher::p_match> &p_matched,int32_t method);
103
104     void refinement (std::vector<Matcher::p_match> &p_matched,int32_t method);
105
106     // parameters
107     parameters param;
108     int32_t    margin;
109
110     ...
111 };
```

2.2 主要函数过程分析

2.2.1 demo 程序分析

```
1  VisualOdometryStereo::parameters param;
2
3  // calibration parameters for sequence 2010_03_09_drive_0019
4  param.calib.f  = 645.24; // focal length in pixels
5  param.calib.cu = 635.96; // principal point (u-coordinate) in pixels
6  param.calib.cv = 194.13; // principal point (v-coordinate) in pixels
7  param.base     = 0.5707; // baseline in meters
8  //新建一个参数，并且设置参数
9  // init visual odometry
10 VisualOdometryStereo viso(param);
11 //通过参数初始化一个VisualOdometryStereo实例
```

新建参数并且初始化一个实例

```
1  int32_t dims[] = {width,height,width};
2  if (viso.process(left_img_data,right_img_data,dims)) {
3
4  // on success, update current pose
```

```
pose = pose * Matrix::inv(viso.getMotion());

// output some statistics
double num_matches = viso.getNumberOfMatches();
double num_inliers = viso.getNumberOfInliers();
cout << ", Matches: " << num_matches;
cout << ", Inliers: " << 100.0*num_inliers/num_matches << " %" << ", Current pose: "
    << endl;
cout << pose << endl << endl;

} else {
cout << " ... failed!" << endl;
}
```

分别从从左右镜头读入一帧，把图片中的信息存入数组，调用 VisualOdemetryStereo 的 process 方法进行处理，处理完成后输出匹配成功的点，内围点的占比和当前摄像机的姿态

2.2.2 VisualOdometryStereo::process

输入一帧左右两张图像，存入 ringbuffer，计算特征，匹配特征值，进行“桶”操作，最后更新动作

```
bool VisualOdometryStereo::process (uint8_t *I1,uint8_t *I2,int32_t* dims,bool
    replace) {

    matcher->pushBack(I1,I2,dims,replace);
    // 把左右两帧图片加入ringbuffer，并计算特征值

    // 如果是处理第一张图片就先建立动作预测
    if (!Tr_valid) {
        matcher->matchFeatures(2);
        matcher->bucketFeatures(param.bucket.max_features,param.bucket.bucket_width,p
            aram.bucket.bucket_height);
        p_matched = matcher->getMatches();
        updateMotion();
    }

    if (Tr_valid) matcher->matchFeatures(2,&Tr_delta);
    else          matcher->matchFeatures(2);
    // 使用环形匹配法匹配图像特征
    matcher->bucketFeatures(param.bucket.max_features,param.bucket.bucket_width,param
        .bucket.bucket_height);
    // 减少特征数量，并使特征值较为均匀的分布在图像中
    p_matched = matcher->getMatches();
```

```
    return updateMotion();
    // 更新动作
}
```

2.2.3 Matcher::pushBack

把左右各一帧图像放入 ringbuffer，并且计算两张图像的各个特征

```
void Matcher::pushBack (uint8_t *I1,uint8_t* I2,int32_t* dims,const bool replace) {

    // 定义图片大小
    int32_t width  = dims[0];
    int32_t height = dims[1];
    int32_t bpl    = dims[2];

    // sanity check
    if (width<=0 || height<=0 || bpl<width || I1==0) {
        cerr << "ERROR: Image dimension mismatch!" << endl;
        return;
    }

    if (replace) {
        ... //如果规定了replace, 释放上上张图片的各项参数
    } else {
        ... //释放上上张图片的各项参数
        ... //将“当前”图片的各项信息设为“上张”图片的各项信息
    }

    // 对齐内存
    dims_c[0] = width;
    dims_c[1] = height;
    dims_c[2] = width + 15-(width-1)%16;

    I1c = (uint8_t*)_mm_malloc(dims_c[2]*dims_c[1]*sizeof(uint8_t),16);
    I2c = (uint8_t*)_mm_malloc(dims_c[2]*dims_c[1]*sizeof(uint8_t),16);
    ... // 对齐放置图片信息

    // 计算图片各项特征
    computeFeatures(I1c,dims_c,m1c1,n1c1,m1c2,n1c2,I1c_du,I1c_dv,I1c_du_full,I1c_dv_f
    ↪  ull);
    if (I2!=0)
```

```
33          computeFeatures(I2c,dims_c,m2c1,n2c1,m2c2,n2c2,I2c_du,I2c_dv,I2c_du_full,I2c_」
            ↪  dv_full);
34      }
```

2.2.4 Matcher::computeFeatures

用各个滤波器提取图片的特征值，进行非极大化抑制，计算出稀疏特征以及稠密特征

```
1   void Matcher::computeFeatures (...) {
2
3       ...
4
5       if (!param.half_resolution) {
6           ... // demo不涉及这个部分
7       } else {
8           uint8_t* I_matching = createHalfResolutionImage(I,dims);
9           getHalfResolutionDimensions(dims,dims_matching);
10          // 将图像缩小一半
11          ... // 为各个滤波结果分配空间
12          filter::sobel5x5(I_matching,I_du,I_dv,dims_matching[2],dims_matching[1]);
13          // 对缩略图进行sobel滤波求梯度
14          filter::sobel5x5(I,I_du_full,I_dv_full,dims[2],dims[1]);
15          // 对原图图进行sobel滤波求梯度
16          filter::blob5x5(I_matching,I_f1,dims_matching[2],dims_matching[1]);
17          // 对缩略图使用blob滤波
18          filter::checkerboard5x5(I_matching,I_f2,dims_matching[2],dims_matching[1]);
19          // 对缩略图求边界
20          _mm_free(I_matching);
21      }
22
23      // 使用非极大抑制提取稀疏极大值
24      vector<Matcher::maximum> maxima1;
25      if (param.multi_stage) {
26          int32_t nms_n_sparse = param.nms_n*3;
27          if (nms_n_sparse>10)
28              nms_n_sparse = max(param.nms_n,10);
29          nonMaximumSuppression(I_f1,I_f2,dims_matching,maxima1,nms_n_sparse);
30          computeDescriptors(I_du,I_dv,dims_matching[2],maxima1);
31      }
32
33      // 使用非极大抑制提取稠密极大值
34      vector<Matcher::maximum> maxima2;
```

```
35      nonMaximumSuppression(I_f1,I_f2,dims_matching,maxima2,param.nms_n);
36      computeDescriptors(I_du,I_dv,dims_matching[2],maxima2);
37
38      ...
39
40      if (num1!=0) {
41          ... // 将稀疏最大值对齐内存存入返回变量
42      }
43
44      if (num2!=0) {
45          ... // 将稠密最大值对齐内存存入返回变量
46      }
47  }
```

2.2.5 Matcher::matchFeatures

匹配特征值，首先匹配稀疏特征值，消除异常值后计算出搜索范围，然后利用搜索范围进行稠密特征匹配，并且进行精化，消除异常值

```
1   void Matcher::matchFeatures(int32_t method, Matrix *Tr_delta) {
2
3       ... // 检查完整性
4
5       // double pass matching
6       if (param.multi_stage) {
7
8           // 1st pass 稀疏特征匹配
9           matching(m1p1,m2p1,m1c1,m2c1,n1p1,n2p1,n1c1,n2c1,p_matched_1,method,false,Tr_」
            ↪  delta);
10          // 使用2d Delaunay三角剖分消除异常值
11          removeOutliers(p_matched_1,method);
12
13          // 为加速第二次处理计算搜索范围优先级数据
14          computePriorStatistics(p_matched_1,method);
15
16          // 2nd pass 稠密特征匹配
17          matching(m1p2,m2p2,m1c2,m2c2,n1p2,n2p2,n1c2,n2c2,p_matched_2,method,true,Tr_d」
            ↪  elta);
18          if (param.refinement>0)
19              // 通过抛物线拟合的子像素细化来进一步改进特征定位
20              refinement(p_matched_2,method);
21          removeOutliers(p_matched_2,method);
```

```
22
23      // single pass matching
24      } else {
25          ...
26      }
27  }
```

2.2.6 Matcher::matching

对当前 4 张图片使用环形匹配

```
1   void Matcher::matching (int32_t *m1p,int32_t *m2p,int32_t *m1c,int32_t *m2c, int32_t
    ↪  n1p,int32_t n2p,int32_t n1c,int32_t n2c, vector<Matcher::p_match>
    ↪  &p_matched,int32_t method,bool use_prior,Matrix *Tr_delta) {
2
3       // loop variables
4       int32_t* M = (int32_t*)calloc(dims_c[0]*dims_c[1],sizeof(int32_t));
5       int32_t i1p,i2p,i1c,i2c,i1c2,i1p2;
6       int32_t u1p,v1p,u2p,v2p,u1c,v1c,u2c,v2c;
7
8       double t00,t01,t02,t03,t10,t11,t12,t13,t20,t21,t22,t23;
9       if (Tr_delta) {
10          ... // 如果存在之前的姿态变化量，就初始化变量
11      }
12
13      ////////////////////////////////////////////////////
14      // method: flow
15      if (method==0) {
16          ... // demo不涉及这个部分
17      }
18
19      ////////////////////////////////////////////////////
20      // method: stereo
21      } else if (method==1) {
22          ... // demo不涉及这个部分
23      }
24
25      ////////////////////////////////////////////////////
26      // method: quad matching
27      } else {
28
29          // create position/class bin index vectors
```

```
30          createIndexVector(m1p,n1p,k1p,u_bin_num,v_bin_num);
31          createIndexVector(m2p,n2p,k2p,u_bin_num,v_bin_num);
32          createIndexVector(m1c,n1c,k1c,u_bin_num,v_bin_num);
33          createIndexVector(m2c,n2c,k2c,u_bin_num,v_bin_num);
34
35          // for all points do
36          for (i1p=0; i1p<n1p; i1p++) {
37              // 对所有“前一帧”左边视图的所有特征点执行下面的操作
38
39              // 读取每个特征点的坐标
40              u1p = *(m1p+step_size*i1p+0);
41              v1p = *(m1p+step_size*i1p+1);
42
43              // compute row and column of statistics bin to which this observation
                ↪  belongs
44              int32_t u_bin = min((int32_t)floor((float)u1p/(float)param.match_binsize)」
                ↪  ,u_bin_num-1);
45              int32_t v_bin = min((int32_t)floor((float)v1p/(float)param.match_binsize)」
                ↪  ,v_bin_num-1);
46              int32_t stat_bin = v_bin*u_bin_num+u_bin;
47
48              // match in circle
49              // 开始环形匹配
50              findMatch(m1p,i1p,m2p,step_size,k2p,u_bin_num,v_bin_num,stat_bin,i2p,
                ↪  0,false,use_prior);
51              // 匹配“前一帧”左视图和“前一帧”右视图的稀疏特征
52
53              u2p = *(m2p+step_size*i2p+0);
54              v2p = *(m2p+step_size*i2p+1);
55
56              if (Tr_delta) {
57
58                  double d = max((double)u1p-(double)u2p,1.0);
59                  double x1p = ((double)u1p-param.cu)*param.base/d;
60                  double y1p = ((double)v1p-param.cv)*param.base/d;
61                  double z1p = param.f*param.base/d;
62
63                  double x2c = t00*x1p + t01*y1p + t02*z1p + t03 - param.base;
64                  double y2c = t10*x1p + t11*y1p + t12*z1p + t13;
65                  double z2c = t20*x1p + t21*y1p + t22*z1p + t23;
66
```

```
67                  double u2c_ = param.f*x2c/z2c+param.cu;
68                  double v2c_ = param.f*y2c/z2c+param.cv;
69
70                  // 如果有之前运动矩阵，就沿用这个运动矩阵
71                  // 这里假设运动是连续的，运动的变化较小
72                  findMatch(m2p,i2p,m2c,step_size,k2c,u_bin_num,v_bin_num,stat_bin,i2c,
                    ↪ 1,true ,use_prior,u2c_,v2c_);
73              } else {
74                  findMatch(m2p,i2p,m2c,step_size,k2c,u_bin_num,v_bin_num,stat_bin,i2c,
                    ↪ 1,true ,use_prior);
75              }
76              // 匹配“前一帧”右侧图像和“当前帧”右侧图像
77              findMatch(m2c,i2c,m1c,step_size,k1c,u_bin_num,v_bin_num,stat_bin,i1c,
                ↪ 2,false,use_prior);
78              // 匹配”当前帧“左侧图像和”当前帧“右侧图像
79              if (Tr_delta)
80                  findMatch(m1c,i1c,m1p,step_size,k1p,u_bin_num,v_bin_num,stat_bin,i1p2
                    ↪ ,3,true
                    ↪ ,use_prior,u1p,v1p);
81              else
82                  findMatch(m1c,i1c,m1p,step_size,k1p,u_bin_num,v_bin_num,stat_bin,i1p2
                    ↪ ,3,true
                    ↪ ,use_prior);
83              // 匹配”当前帧“左侧图像和”前一帧“右侧图像
84              if (i1p2==i1p) {
85
86                  u2c = *(m2c+step_size*i2c+0); v2c = *(m2c+step_size*i2c+1);
87                  u1c = *(m1c+step_size*i1c+0); v1c = *(m1c+step_size*i1c+1);
88                  if (u1p>=u2p && u1c>=u2c) {
89
90                      // 如果匹配成功，就把匹配的结果放入p_match中
91                      p_matched.push_back(Matcher::p_match(u1p,v1p,i1p,u2p,v2p,i2p,u1c,
                        ↪ v1c,i1c,u2c,v2c,i2c));
92                  }
93              }
94          }
95      }
96
97      // free memory
98      ...
99  }
```

2.2.7 Matcher::bucketFeatures

对匹配完成的结果进行“桶”操作，对每个局部区域只随机保留若干个特征点

```
void Matcher::bucketFeatures(int32_t max_features,float bucket_width,float
↪  bucket_height) {

    // 找到”当前帧“左侧图像u-v坐标做大的特征点
    float u_max = 0;
    float v_max = 0;
    for (vector<p_match>::iterator it = p_matched_2.begin(); it!=p_matched_2.end();
    ↪  it++) {
        if (it->u1c>u_max) u_max=it->u1c;
        if (it->v1c>v_max) v_max=it->v1c;
    }

    // 分配需要的buckets
    int32_t bucket_cols = (int32_t)floor(u_max/bucket_width)+1;
    int32_t bucket_rows = (int32_t)floor(v_max/bucket_height)+1;
    vector<p_match> *buckets = new vector<p_match>[bucket_cols*bucket_rows];

    // 把每个特征点放入其位置对应的bukect中
    for (vector<p_match>::iterator it=p_matched_2.begin(); it!=p_matched_2.end();
    ↪  it++) {
        int32_t u = (int32_t)floor(it->u1c/bucket_width);
        int32_t v = (int32_t)floor(it->v1c/bucket_height);
        buckets[v*bucket_cols+u].push_back(*it);
    }

    p_matched_2.clear();
    // 清空原特征点
    for (int32_t i=0; i<bucket_cols*bucket_rows; i++) {
        // 对每个bucket

        std::random_shuffle(buckets[i].begin(),buckets[i].end());

        // 每个bucket随机填入max_features个特征点
        int32_t k=0;
        for (vector<p_match>::iterator it=buckets[i].begin(); it!=buckets[i].end();
        ↪  it++) {
            p_matched_2.push_back(*it);
            k++;
```

```
            if (k>=max_features)
                break;
        }
    }

    // free buckets
    delete []buckets;
}
```

2.2.8 Matcher::computePriorStatistics

分析偏移结果，统计每个区域最大偏移量，为下一部稠密特征匹配约束搜索范围

```
void Matcher::computePriorStatistics (vector<Matcher::p_match> &p_matched,int32_t
↪ method) {

    // 计算区域 (bin) 数量
    int32_t u_bin_num = (int32_t)ceil((float)dims_c[0]/(float)param.match_binsize);
    int32_t v_bin_num = (int32_t)ceil((float)dims_c[1]/(float)param.match_binsize);
    int32_t bin_num   = v_bin_num*u_bin_num;

    ...

    for (vector<Matcher::p_match>::iterator it=p_matched.begin();
    ↪ it!=p_matched.end(); it++) {
        // 对每一组配对好的点组
        // method flow: compute position delta
        if (method==0) {
            ...
        } else if (method==1) {
            ...
        } else {
            delta_curr.val[0] = it->u2p - it->u1p;
            delta_curr.val[1] = 0;
            delta_curr.val[2] = it->u2c - it->u2p;
            delta_curr.val[3] = it->v2c - it->v2p;
            delta_curr.val[4] = it->u1c - it->u2c;
            delta_curr.val[5] = 0;
            delta_curr.val[6] = it->u1p - it->u1c;
            delta_curr.val[7] = it->v1p - it->v1c;
            // 计算这个点组的各个偏移量
        }
```

```
28
29          // 计算哪些区域 (bin) 包含了这个点组
30          int32_t u_bin_min,u_bin_max,v_bin_min,v_bin_max;
31
32          // flow + stereo: use current left image as reference
33          if (method<2) {
34              ...
35
36          } else {
37              u_bin_min = min(max((int32_t)floor(it->u1p/(float)param.match_binsize)-1,
                ↪  0),u_bin_num-1);
38              u_bin_max = min(max((int32_t)floor(it->u1p/(float)param.match_binsize)+1,
                ↪  0),u_bin_num-1);
39              v_bin_min = min(max((int32_t)floor(it->v1p/(float)param.match_binsize)-1,
                ↪  0),v_bin_num-1);
40              v_bin_max = min(max((int32_t)floor(it->v1p/(float)param.match_binsize)+1,
                ↪  0),v_bin_num-1);
41          }
42
43          // 在相关区域对应的accumulator中加入计算出的偏移量
44          for (int32_t v_bin=v_bin_min; v_bin<=v_bin_max; v_bin++)
45              for (int32_t u_bin=u_bin_min; u_bin<=u_bin_max; u_bin++)
46                  delta_accu[v_bin*u_bin_num+u_bin].push_back(delta_curr);
47      }
48
49      ranges.clear();
50
51      // 对每个区域 (bin) 计算最大偏移量
52      for (int32_t v_bin=0; v_bin<v_bin_num; v_bin++) {
53          for (int32_t u_bin=0; u_bin<u_bin_num; u_bin++) {
54
55              // 如果此区域 (bin) 没有相关记录, 就使用默认值
56              delta delta_min(-param.match_radius);
57              delta delta_max(+param.match_radius);
58
59              // 通过每个区域 (bin) 的accumulator中的各个偏移值计算出最大偏移值
60              if (delta_accu[v_bin*u_bin_num+u_bin].size()>0) {
61
62                  // init displacements 'delta' to 'infinite'
63                  delta_min = delta(+1000000);
64                  delta_max = delta(-1000000);
```

```
65
66                  // find minimum and maximum displacements
67                  for (vector<Matcher::delta>::iterator
                    ↪  it=delta_accu[v_bin*u_bin_num+u_bin].begin();
68                      it!=delta_accu[v_bin*u_bin_num+u_bin].end(); it++) {
69                      for (int32_t i=0; i<num_stages*2; i++) {
70                          if (it->val[i]<delta_min.val[i]) delta_min.val[i] =
                            ↪  it->val[i];
71                          if (it->val[i]>delta_max.val[i]) delta_max.val[i] =
                            ↪  it->val[i];
72                      }
73                  }
74              }
75
76              // 将最大偏移值进一步处理为搜索范围
77              range r;
78              for (int32_t i=0; i<num_stages; i++) {
79
80                  // bound minimum search range to 20x20
81                  float delta_u = delta_max.val[i*2+0]-delta_min.val[i*2+0];
82                  if (delta_u<20) {
83                      delta_min.val[i*2+0] -= ceil((20-delta_u)/2);
84                      delta_max.val[i*2+0] += ceil((20-delta_u)/2);
85                  }
86                  float delta_v = delta_max.val[i*2+1]-delta_min.val[i*2+1];
87                  if (delta_v<20) {
88                      delta_min.val[i*2+1] -= ceil((20-delta_v)/2);
89                      delta_max.val[i*2+1] += ceil((20-delta_v)/2);
90                  }
91
92                  // set range for this bin
93                  r.u_min[i] = delta_min.val[i*2+0];
94                  r.u_max[i] = delta_max.val[i*2+0];
95                  r.v_min[i] = delta_min.val[i*2+1];
96                  r.v_max[i] = delta_max.val[i*2+1];
97              }
98              ranges.push_back(r);
99          }
100     }
101
102     // free bin accumulator memory
```

```
103     delete []delta_accu;
104 }
```

2.2.9 VisualOdometry::updateMotion

计算主体动作，并将其转换为矩阵

```
1  bool VisualOdometry::updateMotion () {
2
3    // 调用estimateMotion预测当前姿态变换
4    vector<double> tr_delta = estimateMotion(p_matched);
5
6    // on failure
7    if (tr_delta.size()!=6)
8      return false;
9
10   // set transformation matrix (previous to current frame)
11   Tr_delta = transformationVectorToMatrix(tr_delta);
12   Tr_valid = true;
13
14   // success
15   return true;
16 }
```

2.2.10 VisualOdometryStereo::estimateMotion

使用 RANSAC 算法计算最优的投影参数

```
1  vector<double> VisualOdometryStereo::estimateMotion (vector<Matcher::p_match>
   ↪  p_matched) {
2
3      // compute minimum distance for RANSAC samples
4      double width=0,height=0;
5      for (vector<Matcher::p_match>::iterator it=p_matched.begin();
       ↪  it!=p_matched.end(); it++) {
6          if (it->u1c>width)  width  = it->u1c;
7          if (it->v1c>height) height = it->v1c;
8      }
9      double min_dist = min(width,height)/3.0;
10
11     // get number of matches
12     int32_t N  = p_matched.size();
13     if (N<6)
14         return vector<double>();
```

```
15
16      // allocate dynamic memory
17      ...
18
19      // project matches of previous image into 3d
20      // 把之前一帧的匹配点投影在3D坐标内
21      /*
22      d代表水平视差，B代表baseline(水平基线)
23      d = max(u_l - u_r, 0.0001)
24      Z = (f×B)/d
25      X = (u - c_u)B/d
26      Y = (v - c_v)B/d
27      */
28      for (int32_t i=0; i<N; i++) {
29          double d = max(p_matched[i].u1p - p_matched[i].u2p,0.0001f);
30          X[i] = (p_matched[i].u1p-param.calib.cu)*param.base/d;
31          Y[i] = (p_matched[i].v1p-param.calib.cv)*param.base/d;
32          Z[i] = param.calib.f*param.base/d;
33      }
34
35      // loop variables
36      vector<double> tr_delta;
37      vector<double> tr_delta_curr;
38      tr_delta_curr.resize(6);
39
40      // clear parameter vector
41      inliers.clear();
42
43      // 进行ransac_iters轮RANSAC算法匹配
44      for (int32_t k=0;k<param.ransac_iters;k++) {
45
46          // 随机选择3个观测点作为初始inlier
47          vector<int32_t> active = getRandomSample(N,3);
48
49          // clear parameter vector
50          for (int32_t i=0; i<6; i++)
51          tr_delta_curr[i] = 0;
52
53          // minimize reprojection errors
54          VisualOdometryStereo::result result = UPDATED;
55          int32_t iter=0;
```

```
56          // 迭代寻找到最优的投影参数
57          while (result==UPDATED) {
58              result = updateParameters(p_matched,active,tr_delta_curr,1,1e-6);
59              if (iter++ > 20 || result==CONVERGED)
60                  break; //迭代20次或者结果收敛就停止迭代
61          }
62
63          // overwrite best parameters if we have more inliers
64          if (result!=FAILED) {
65              vector<int32_t> inliers_curr = getInlier(p_matched,tr_delta_curr);
66              // 获取符合当前模型的内点（inlier）
67              if (inliers_curr.size()>inliers.size()) {
68                  inliers = inliers_curr;
69                  tr_delta = tr_delta_curr;
70                  // 模型的优劣以能匹配到的inlier数量决定
71                  // 保留能匹配到更多inlier的模型
72              }
73          }
74      }
75
76      // 最后根据匹配到的inlier修正模型
77      if (inliers.size()>=6) {
78          int32_t iter=0;
79          VisualOdometryStereo::result result = UPDATED;
80          while (result==UPDATED) {
81          result = updateParameters(p_matched,inliers,tr_delta,1,1e-8);
82          if (iter++ > 100 || result==CONVERGED)
83              break;
84          }
85
86          // not converged
87          if (result!=CONVERGED)
88              success = false;
89
90      // not enough inliers
91      } else {
92          success = false;
93      }
94
95      ...
96
```

```
97      // parameter estimate succeeded?
98      if (success) return tr_delta;
99      else         return vector<double>();
100 }
```

2.2.11 VisualOdometryStereo::updateParameters

迭代计算投影和动作参数，返回当前参数迭代成功，收敛，或失败

```
1  VisualOdometryStereo::result
   ↪ VisualOdometryStereo::updateParameters(vector<Matcher::p_match>
   ↪ &p_matched,vector<int32_t> &active,vector<double> &tr,double step_size,double
   ↪ eps) {
2
3       // we need at least 3 observations
4       if (active.size()<3)
5           return FAILED;
6
7       // extract observations and compute predictions
8       computeObservations(p_matched,active);
9       //将active点的“当前帧”坐标放入p_observation中
10      computeResidualsAndJacobian(tr,active);
11      // 计算残差与雅克比行列式
12
13      // init
14      Matrix A(6,6);
15      Matrix B(6,1);
16
17      // fill matrices A and B
18      for (int32_t m=0; m<6; m++) {
19          for (int32_t n=0; n<6; n++) {
20              double a = 0;
21              for (int32_t i=0; i<4*(int32_t)active.size(); i++) {
22                  a += J[i*6+m]*J[i*6+n];
23              }
24              A.val[m][n] = a;
25          }
26          double b = 0;
27          for (int32_t i=0; i<4*(int32_t)active.size(); i++) {
28              b += J[i*6+m]*(p_residual[i]);
29          }
30          B.val[m][0] = b;
```

```
    }

    // perform elimination
    if (B.solve(A)) { //如果模型合理
        bool converged = true;
        for (int32_t m=0; m<6; m++) {
            tr[m] += step_size*B.val[m][0];
            if (fabs(B.val[m][0])>eps)
                converged = false;
                // 如果某项参数更优
        }
        if (converged)
            return CONVERGED;
        else
            return UPDATED;
    } else {
        return FAILED;
    }
}
```

2.2.12 VisualOdometryStereo::getInlier

对每个特征点使用当前模型进行匹配，把观测和预计的差别小于阈值的点加入 inlier 中

```
vector<int32_t> VisualOdometryStereo::getInlier(vector<Matcher::p_match>
↪ &p_matched,vector<double> &tr) {

    // 把所有观测值标记为active
    vector<int32_t> active;
    for (int32_t i=0; i<(int32_t)p_matched.size(); i++)
        active.push_back(i);

    computeObservations(p_matched,active);
    computeResidualsAndJacobian(tr,active);

    vector<int32_t> inliers;
    for (int32_t i=0; i<(int32_t)p_matched.size(); i++)
        if (pow(p_observe[4*i+0]-p_predict[4*i+0],2)+pow(p_observe[4*i+1]-p_predict[4
        ↪ *i+1],2)
        ↪ +
```

```
            pow(p_observe[4*i+2]-p_predict[4*i+2],2)+pow(p_observe[4*i+3]-p_predict[4
            ↪ *i+3],2) <
            ↪ param.inlier_threshold*param.inlier_threshold)
        inliers.push_back(i);
        // 如果某个点观测值与预测值的偏差小于阈值，就将其标记为inlier
    return inliers;
}
```

2.2.13 VisualOdometryStereo::transformationVectorToMatrix

将移动向量转化为矩阵

```
Matrix VisualOdometry::transformationVectorToMatrix (vector<double> tr) {

    ...

    // 计算姿态变换矩阵
    Matrix Tr(4,4);
    Tr.val[0][0] = +cy*cz;          Tr.val[0][1] = -cy*sz;          Tr.val[0][2] =
    ↪ +sy;    Tr.val[0][3] = tx;
    Tr.val[1][0] = +sx*sy*cz+cx*sz; Tr.val[1][1] = -sx*sy*sz+cx*cz; Tr.val[1][2] =
    ↪ -sx*cy; Tr.val[1][3] = ty;
    Tr.val[2][0] = -cx*sy*cz+sx*sz; Tr.val[2][1] = +cx*sy*sz+sx*cz; Tr.val[2][2] =
    ↪ +cx*cy; Tr.val[2][3] = tz;
    Tr.val[3][0] = 0;               Tr.val[3][1] = 0;               Tr.val[3][2] = 0;
    ↪       Tr.val[3][3] = 1;
    return Tr;
}
```

3 实验运行结果

libviso2 有 matlab warpper 版本也有 c++ 原生版本，我们这里运行了两个 demo

3.1 MATLAB 版本

使用 matlab 运行 mex.m 进行编译，再运行 demo_viso_steroe.m 运行 demo 程序，结果如图 4

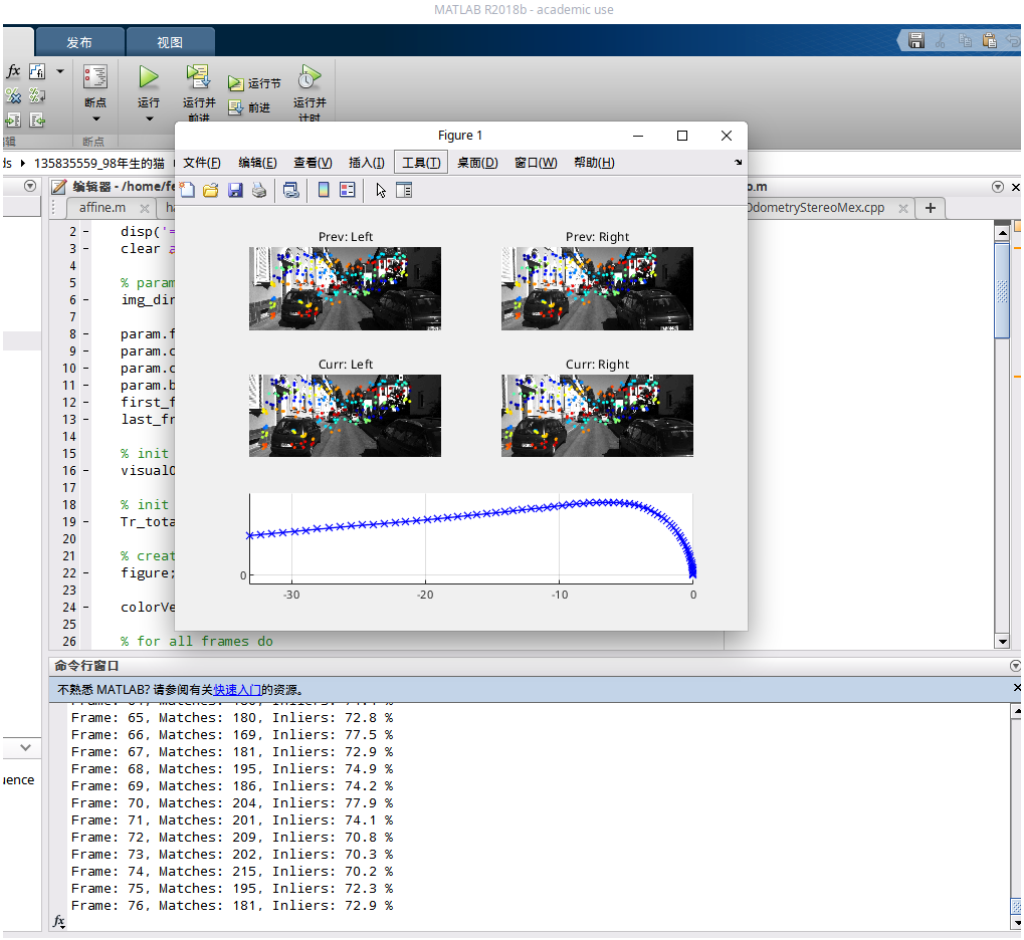

图 4: matlab 版程序运行结果

程序试试显示当前匹配点的数量，inlier 的比例，和主体的运动轨迹

3.2 c++ 版本

使用 cmake 生成 makefile，通过 make 进行编译，然后下载官网上的 demo 进行测试，结果如图 5

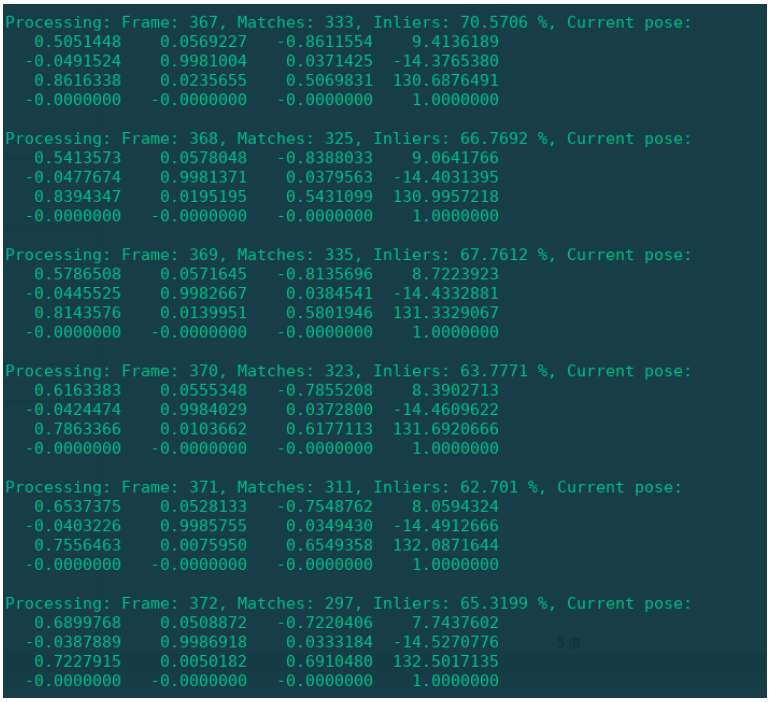

图 5: c++ 版程序运行的结果

输出的结果与 matlab 版本相似，但是会直接输出位置矩阵，不会输出轨迹