0. 说明

本PDF文档为自动生成，如有遗漏的格式错误请及时告知！

---

1. ORB特征点

问1-1：ORB提取

function： computeAngle()

```
// 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)
        kp.angle = 0; // compute kp.angle
        cv::Point2f p = kp.pt;
        if(p.x<half_patch_size || p.x>image.cols-half_patch_size ||
           p.y<half_patch_size || p.y>image.rows-half_patch_size)
           continue;

        double m01=0,m10=0;
        for(int i= -half_patch_size;i<half_patch_size;i++){
            for(int j= -half_patch_size;j<half_patch_size;j++){
                m01 += j*image.at<uchar>(p.y+j,p.x+i);       //pay attention to
the order
                m10 += i*image.at<uchar>(p.y+j,p.x+i);
            }
        }

        kp.angle = atan(m01/m10)*180/pi;
        // END YOUR CODE HERE
    }
    return;
}
```

问1-2：ORB描述

function： computeORBDesc()

```
// 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)
            float c = cos(kp.angle*pi/180);
            float s = sin(kp.angle*pi/180);

            cv::Point2f p_dot(c*ORB_pattern[4*i]-s*ORB_pattern[4*i+1],
```

```
                            s*ORB_pattern[4*i]+c*ORB_pattern[4*i+1]);
            cv::Point2f q_dot(c*ORB_pattern[4*i+2]-s*ORB_pattern[4*i+3],
                            s*ORB_pattern[4*i+2]+c*ORB_pattern[4*i+3]);

            p_dot = p_dot+kp.pt;
            q_dot = q_dot+kp.pt;

            if(p_dot.x<0||p_dot.y<0||p_dot.x>image.cols||p_dot.y>image.rows||
               q_dot.x<0||q_dot.y<0||q_dot.x>image.cols||q_dot.y>image.rows){
               d.clear();
               break;
            }
            d[i] = image.at<uchar>(p_dot)>image.at<uchar>(q_dot)?0:1;
        // END YOUR CODE HERE
        }
        desc.push_back(d);
    }

    int bad = 0;
    for (auto &d: desc) {
        if (d.empty()) bad++;
    }
    cout << "bad/total: " << bad << "/" << desc.size() << endl;
    return;
}
```

问1-3：暴力匹配

function： bfMatch()

```
// 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.
    for(int i=0;i<desc1.size();i++){
        if(desc1[i].empty()) continue;
        int d_min=256;
        int count =-1;

        for(int j=0;j<desc2.size();j++){
            if(desc2[j].empty()) continue;
            int d=0;
            for(int k=0;k<256;k++){
                d += desc1[i][k]^desc2[j][k];
            }
            if(d<d_min){d_min=d;count=j;}
        }
        if(d_min<=d_max){
            cv::DMatch match(i,count,d_min);
            matches.push_back(match);
        }
    }
    // END YOUR CODE HERE
```

```
    for (auto &m: matches) {
        cout << m.queryIdx << ", " << m.trainIdx << ", " << m.distance << endl;
    }
    return;
}
```

- CMakeLists

```
cmake_minimum_required(VERSION 2.8.3)
SET(CMAKE_BUILD_TYPE "Release")
PROJECT (Chapter5)

add_compile_options(-std=c++11)

INCLUDE_DIRECTORIES(${PROJECT_SOURCE_DIR}/include)
INCLUDE_DIRECTORIES("/usr/include/opencv2")

find_package( OpenCV REQUIRED )

find_package(Pangolin REQUIRED)
INCLUDE_DIRECTORIES(${Pangolin_INCLUDE_DIRS})

SET(SRC_LIST ${PROJECT_SOURCE_DIR}/src/computeORB.cpp)
ADD_EXECUTABLE(computeORB ${SRC_LIST})
target_link_libraries(computeORB ${OpenCV_LIBRARIES})
```

- 运行结果

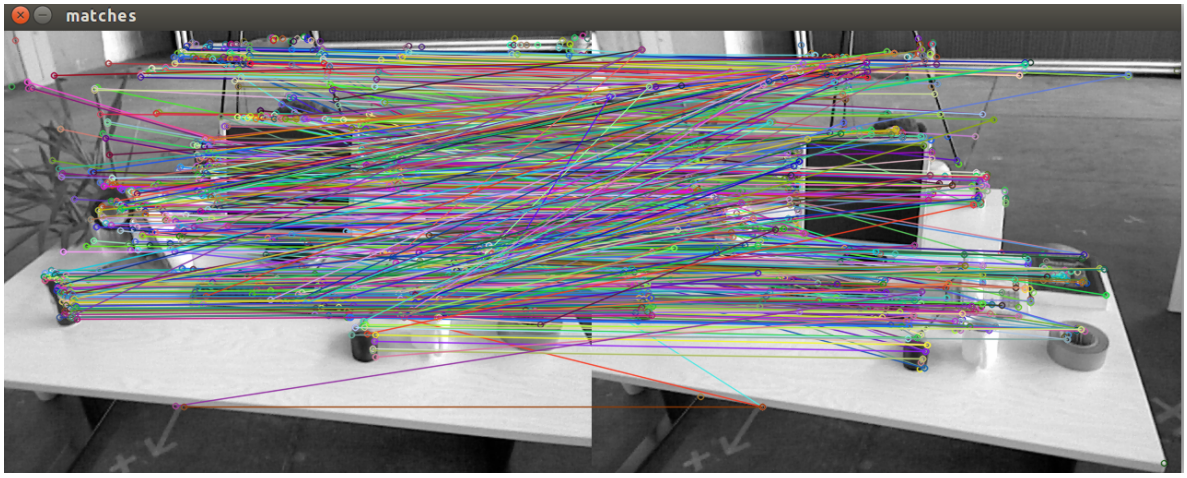
问1-4：结合实验回答下面问题

- 为什么说ORB是一种二进制特征？

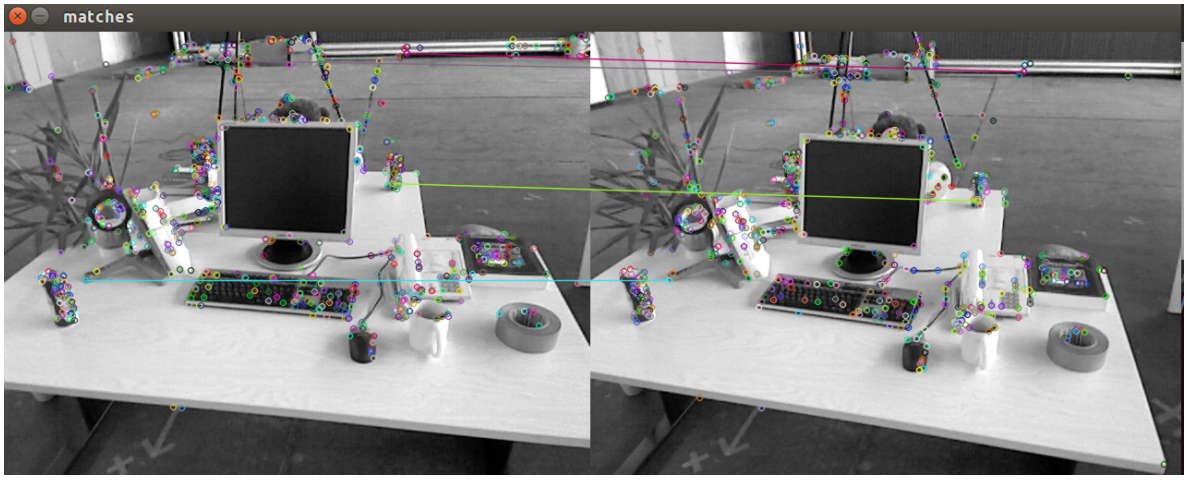
  因为ORB使用一种二进制描述子BRIEF。

- 匹配时为什么选取50作为阈值，取更大或者更小会怎么样？

  取100作为阈值，匹配到的特征点过多。



取25作为阈值，匹配到的特征点过少。



- 暴力匹配在你的机器上表现如何？你能想到什么减少计算量的匹配方法吗？

  time：0.122705s

  可以采用快速近似最近邻（FLANN）算法。

2. 从 E 恢复 R, t

- E2Rt.cpp

```
// 本程序演示如何从Essential矩阵计算R,t
//

#include <Eigen/Core>
#include <Eigen/Dense>
#include <Eigen/Geometry>

using namespace Eigen;

#include <sophus/so3.h>

#include <iostream>

using namespace std;

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<Eigen::MatrixXd> svd(E, ComputeFullU | ComputeFullV );
    Vector3d sigma = svd.singularValues();
    Matrix3d U = svd.matrixU();
    Matrix3d V = svd.matrixV();

    Matrix3d SIGMA;
    SIGMA<<(sigma(0,0)+sigma(1,0))/2, 0, 0,
            0, (sigma(0,0)+sigma(1,0))/2, 0,
            0, 0, 0;

    // END YOUR CODE HERE

    // set t1, t2, R1, R2
    // START YOUR CODE HERE
    Matrix3d t_wedge1;
    Matrix3d t_wedge2;

    Matrix3d R1;
    Matrix3d R2;

    Matrix3d Rz1=AngleAxisd(M_PI/2,Vector3d(0,0,1)).toRotationMatrix();
    Matrix3d Rz2=AngleAxisd(-M_PI/2,Vector3d(0,0,1)).toRotationMatrix();
```

```
    t_wedge1=U*Rz1*SIGMA*U.transpose();
    t_wedge2=U*Rz2*SIGMA*U.transpose();
    R1=U*Rz1.transpose()*V.transpose();
    R2=U*Rz2.transpose()*V.transpose();
    // END YOUR CODE HERE

    cout << "R1 = \n" << R1 << endl;
    cout << "R2 = \n" << R2 << endl;
    cout << "t1 = \n" << Sophus::SO3::vee(t_wedge1) << endl;
    cout << "t2 = \n" << Sophus::SO3::vee(t_wedge2) << endl;

    // check t^R=E up to scale
    Matrix3d tR = t_wedge1 * R1;
    cout << "t^R = \n" << tR << endl;

    return 0;
}
```

- CMakeLists

```
cmake_minimum_required(VERSION 2.8.3)
SET(CMAKE_BUILD_TYPE "Release")
PROJECT (Chapter5)

add_compile_options(-std=c++11)

INCLUDE_DIRECTORIES(${PROJECT_SOURCE_DIR}/include)
INCLUDE_DIRECTORIES("/usr/include/opencv2")
INCLUDE_DIRECTORIES("/usr/include/eigens3")

find_package( OpenCV REQUIRED )

find_package(Sophus REQUIRED)
include_directories(${Sophus_INCLUDE_DIRS})

SET(SRC_LIST ${PROJECT_SOURCE_DIR}/src/computeORB.cpp)
ADD_EXECUTABLE(computeORB ${SRC_LIST})
target_link_libraries(computeORB ${OpenCV_LIBRARIES})

ADD_EXECUTABLE(E2Rt ${PROJECT_SOURCE_DIR}/src/E2Rt.cpp)
target_link_libraries(E2Rt ${Sophus_LIBRARIES})
```

- 运行结果

```
iusl@iusl-OptiPlex-7060:~/lyq/github/SLAM/Homeworks/Chapter5/build$ ./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
```

---

3. 用 G-N 实现 Bundle Adjustment 中的位姿估计

问3-1：编写程序，用G-N法求出最优位姿

- GN-BA.cpp

```cpp
#include <Eigen/Core>
#include <Eigen/Dense>

using namespace Eigen;

#include <vector>
#include <fstream>
#include <iostream>
#include <iomanip>

#include "sophus/se3.h"

using namespace std;

typedef vector<Vector3d, Eigen::aligned_allocator<Vector3d>> VecVector3d;
typedef vector<Vector2d, Eigen::aligned_allocator<Vector3d>> VecVector2d;
typedef Matrix<double, 6, 1> Vector6d;

string p3d_file = "./p3d.txt";
string p2d_file = "./p2d.txt";

int main(int argc, char **argv) {

    VecVector2d p2d;
    VecVector3d p3d;
    Matrix3d K;
    double fx = 520.9, fy = 521.0, cx = 325.1, cy = 249.7;
    K << fx, 0, cx, 0, fy, cy, 0, 0, 1;

    // load points in to p3d and p2d
    // START YOUR CODE HERE
```

```
    ifstream p2d_read(p2d_file);
    ifstream p3d_read(p3d_file);
    if(!p2d_read && !p3d_read){
        cout<<"open file fail!"<<endl;
    }

    string oneline;
    Vector2d v2d;
    Vector3d v3d;
    while(getline(p2d_read,oneline)){
        istringstream temp(oneline);
        temp >>v2d[0]>>v2d[1];
        p2d.push_back(v2d);
    }
    while(getline(p3d_read,oneline)){
        istringstream temp(oneline);
        temp >>v3d[0]>>v3d[1]>>v3d[2];
        p3d.push_back(v3d);
    }
    // END YOUR CODE HERE
    assert(p3d.size() == p2d.size());

    int iterations = 100;
    double cost = 0, lastCost = 0;
    int nPoints = p3d.size();
    cout << "points: " << nPoints << endl;

    Sophus::SE3 T_esti; // estimated pose

    for (int iter = 0; iter < iterations; iter++) {

        Matrix<double, 6, 6> H = Matrix<double, 6, 6>::Zero();
        Vector6d b = Vector6d::Zero();

        cost = 0;
        // compute cost
        for (int i = 0; i < nPoints; i++) {
            // compute cost for p3d[I] and p2d[I]
            // START YOUR CODE HERE
           Vector2d p2 = p2d[i];
           Vector3d p3 = p3d[i];
           Vector3d P = T_esti * p3;
           double x = P[0];
           double y = P[1];
           double z = P[2];

           Vector2d p2_dot = {fx * ( x/z ) + cx, fy * ( y/z ) + cy};
           Vector2d e = p2-p2_dot;
           cost += e.squaredNorm()/2;
        // END YOUR CODE HERE

        // compute jacobian
            Matrix<double, 2, 6> J;
            // START YOUR CODE HERE
            J(0,0)=fx/z;
            J(0,1)=0;
            J(0,2)=-fx*x/(z*z);
            J(0,3)=-fx*x*y/(z*z);
```

```
            J(0,4)=fx+fx*x*x/(z*z);
            J(0,5)=-fx*y/z;

            J(1,0)=0;
            J(1,1)=fy/z;
            J(1,2)=-fy*y/(z*z);
            J(1,3)=-fy-fy*y*y/(z*z);
            J(1,4)=fy*x*y/(z*z);
            J(1,5)=fy*x/z;
            J = -J;
        // END YOUR CODE HERE

            H += J.transpose() * J;
            b += -J.transpose() * e;
        }

    // solve dx
        Vector6d dx;

        // START YOUR CODE HERE
        dx=H.ldlt().solve(b);
        // END YOUR CODE HERE

        if (isnan(dx[0])) {
            cout << "result is nan!" << endl;
            break;
        }

        if (iter > 0 && cost >= lastCost) {
            // cost increase, update is not good
            cout << "cost: " << cost << ", last cost: " << lastCost << endl;
            break;
        }

        // update your estimation
        // START YOUR CODE HERE
        T_esti=Sophus::SE3::exp(dx)*T_esti;
        // END YOUR CODE HERE

        lastCost = cost;

        cout << "iteration " << iter << " cost=" << cout.precision(12) <<
cost << endl;
    }

    cout << "estimated pose: \n" << T_esti.matrix() << endl;
    return 0;
}
```

- CMakeLists

```
cmake_minimum_required(VERSION 2.8.3)
SET(CMAKE_BUILD_TYPE "Release")
PROJECT (Chapter5)
```

```
add_compile_options(-std=c++11)

INCLUDE_DIRECTORIES(${PROJECT_SOURCE_DIR}/include)
INCLUDE_DIRECTORIES("/usr/include/opencv2")
INCLUDE_DIRECTORIES("/usr/include/eigens3")

find_package( OpenCV REQUIRED )

find_package(Sophus REQUIRED)
include_directories(${Sophus_INCLUDE_DIRS})

SET(SRC_LIST ${PROJECT_SOURCE_DIR}/src/computeORB.cpp)
ADD_EXECUTABLE(computeORB ${SRC_LIST})
target_link_libraries(computeORB ${OpenCV_LIBRARIES})

ADD_EXECUTABLE(E2Rt ${PROJECT_SOURCE_DIR}/src/E2Rt.cpp)
target_link_libraries(E2Rt ${Sophus_LIBRARIES})

ADD_EXECUTABLE(GN-BA ${PROJECT_SOURCE_DIR}/src/GN-BA.cpp)
target_link_libraries(GN-BA ${Sophus_LIBRARIES})
```

- 运行结果

```
lusl@lusl-OptiPlex-7060:~/lyq/github/SLAM/Homeworks/Chapter5/build$ ./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
iteration 5 cost=12150.6753269
cost: 150.6753269, last cost: 150.6753269
estimated pose:
   0.997866186837  -0.0516724392948   0.0399128072707   -0.127226620999
  0.0505959188721    0.998339770315   0.0275273682287 -0.00750679765283
  -0.041268949107  -0.0254492048094    0.998823914318   0.0613860848809
                0                 0                 0                 1
```

问3-2：问答

- 如何定义重投影误差？

  实际观测值-估计值=重投影误差

- 该误差关于自变量的雅克比矩阵是什么？

$$-\begin{bmatrix} \frac{f_x}{Z'} & 0 & -\frac{f_x X'}{Z'^2} & -\frac{f_x X' Y'}{Z'^2} & f_x + \frac{f_x X^2}{Z'^2} & -\frac{f_x Y'}{Z'} \\ 0 & \frac{f_y}{Z'} & -\frac{f_y Y'}{Z'^2} & -f_y - \frac{f_y Y'^2}{Z'^2} & \frac{f_y X' Y'}{Z'^2} & \frac{f_y X'}{Z'} \end{bmatrix}$$

- 解出更新量之后，如何更新至之前的估计上？

  左乘更新

---

4. 用 ICP 实现轨迹对齐

- trajectoryICP.cpp

```cpp
#include <string>
#include <iostream>
#include <fstream>

#include <Eigen/Core>
#include <Eigen/Dense>

#include <opencv2/opencv.hpp>

#include <sophus/se3.h>

// need pangolin for plotting trajectory
#include <pangolin/pangolin.h>

using namespace std;
using namespace Eigen;
using namespace cv;

// path to trajectory file
string compare_file = "./compare.txt";

// start point is red and end point is blue
void DrawTrajectory(vector<Sophus::SE3, Eigen::aligned_allocator<Sophus::SE3>>
poses_g,
                    vector<Sophus::SE3, Eigen::aligned_allocator<Sophus::SE3>>
poses_e);

// read data
void ReadData(string FileName,
         vector<Sophus::SE3, Eigen::aligned_allocator<Sophus::SE3>> &poses_e,
         vector<Sophus::SE3, Eigen::aligned_allocator<Sophus::SE3>> &poses_g,
         vector<Point3d> &t_e,
         vector<Point3d> &t_g);

//ICP_SVD
void ICP_SVD(vector<Point3d>& pts1,
             vector<Point3d>& pts2,
             Eigen::Matrix3d& R,
             Eigen::Vector3d& t);


int main(int argc, char **argv){

    vector<Sophus::SE3, Eigen::aligned_allocator<Sophus::SE3>> poses_g;
    vector<Sophus::SE3, Eigen::aligned_allocator<Sophus::SE3>> poses_e;

    vector<Point3d> t_g,t_e;

    Eigen::Matrix3d R;
    Eigen::Vector3d t;

    ReadData(compare_file, poses_e, poses_g, t_e, t_g);

    ICP_SVD(t_g,t_e,R,t);
    Sophus::SE3 T_ge(R,t);

    for(auto &p:poses_e){
```

```
        p = T_ge*p;
    }

    DrawTrajectory(poses_e, poses_g);
    return 0;
}

void
ReadData(string FileName,
         vector<Sophus::SE3, Eigen::aligned_allocator<Sophus::SE3>> &poses_g,
         vector<Sophus::SE3, Eigen::aligned_allocator<Sophus::SE3>> &poses_e,
         vector<Point3d> &t_g,
         vector<Point3d> &t_e){

    ifstream trajectory(FileName);

    if(!trajectory){
        cout<<"open file fail!"<<endl;
    }

    string oneline;
    double timestamp1,timestamp2;
    Eigen::Vector3d t1,t2;
    Eigen::Quaterniond q1,q2;
    Sophus::SE3 T1,T2;

    while(getline(trajectory,oneline)){
        istringstream temp(oneline);
        temp >>timestamp1>>t1[0]>>t1[1]>>t1[2]>>q1.x()>>q1.y()>>q1.z()>>q1.w()
             >>timestamp2>>t2[0]>>t2[1]>>t2[2]>>q2.x()>>q2.y()>>q2.z()>>q2.w();
        t_e.push_back(Point3d(t1[0],t1[1],t1[2]));
        t_g.push_back(Point3d(t2[0],t2[1],t2[2]));

        T1 = Sophus::SE3(q1.normalized(),t1);
        T2 = Sophus::SE3(q2.normalized(),t2);
        poses_e.push_back(T1);
        poses_g.push_back(T2);
    }
    return;
}

void ICP_SVD(vector<Point3d>& pts1,
             vector<Point3d>& pts2,
             Eigen::Matrix3d& R,
             Eigen::Vector3d& t){

             Point3d p1,p2;
             int N = pts1.size();
             for(int i = 0; i<N; i++){
                 p1 += pts1[i];
                 p2 += pts2[i];
             }

             p1 = Point3d(Vec3d(p1)/ N);
             p2 = Point3d(Vec3d(p2)/ N);
             vector<Point3d> q1(N),q2(N);

             for(int i=0 ; i<N; i++){
```

```
                q1[i]=pts1[i]-p1;
                q2[i]=pts2[i]-p2;
            }

            Eigen::Matrix3d W = Eigen::Matrix3d::Zero();
            for(int i=0; i<N; i++){
                W+=Eigen::Vector3d(q1[i].x, q1[i].y,
q1[i].z)*Eigen::Vector3d(q2[i].x ,q2[i].y, q2[i].z).transpose();
            }

            Eigen::JacobiSVD<Eigen::Matrix3d> svd ( W,
Eigen::ComputeFullU|Eigen::ComputeFullV );
            Eigen::Matrix3d U = svd.matrixU();
            Eigen::Matrix3d V = svd.matrixV();

            R = U* ( V.transpose() );
            t = Eigen::Vector3d ( p1.x, p1.y, p1.z ) - R * Eigen::Vector3d (
p2.x, p2.y, p2.z );
            //R = R.inverse();
            //t = -R*t;
}



void DrawTrajectory(vector<Sophus::SE3, Eigen::aligned_allocator<Sophus::SE3>>
poses_g,
                    vector<Sophus::SE3, Eigen::aligned_allocator<Sophus::SE3>>
poses_e){
    if (poses_g.empty()||poses_e.empty()) {
        cerr << "Trajectory is empty!" << endl;
        return;
    }

    // create pangolin window and plot the trajectory
    pangolin::CreateWindowAndBind("Trajectory Viewer", 1024, 768);
    glEnable(GL_DEPTH_TEST);
    glEnable(GL_BLEND);
    glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);

    pangolin::OpenGlRenderState s_cam(
            pangolin::ProjectionMatrix(1024, 768, 500, 500, 512, 389, 0.1,
1000),
            pangolin::ModelViewLookAt(0, -0.1, -1.8, 0, 0, 0, 0.0, -1.0, 0.0)
    );

    pangolin::View &d_cam = pangolin::CreateDisplay()
            .SetBounds(0.0, 1.0, pangolin::Attach::Pix(175), 1.0, -1024.0f /
768.0f)
            .SetHandler(new pangolin::Handler3D(s_cam));

    while (pangolin::ShouldQuit() == false) {
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

        d_cam.Activate(s_cam);
        glClearColor(1.0f, 1.0f, 1.0f, 1.0f);

        glLineWidth(2);
        for (size_t i = 0; i < poses_g.size() - 1; i++) {
```

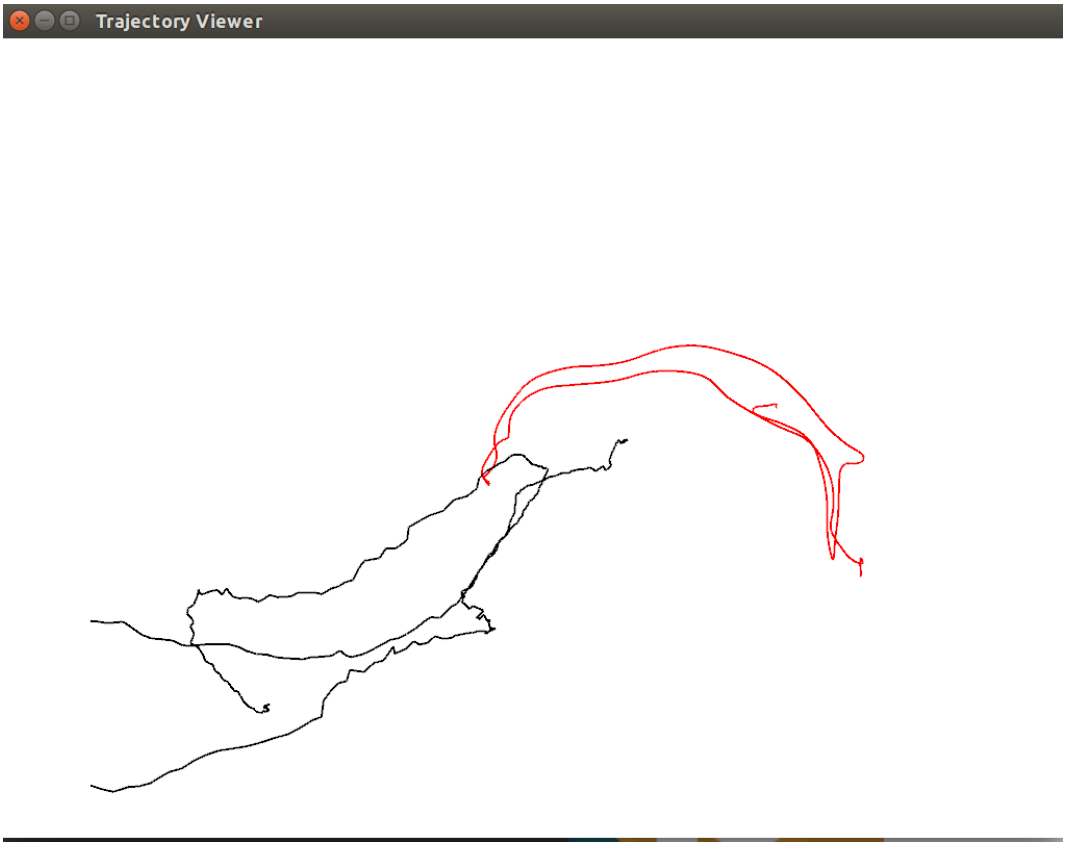
```
            //glColor3f(1 - (float) i / poses_g.size(), 0.0f, (float) i /
poses_g.size());
            glColor3f(1.0f, 0.0f, 0.0f);
            glBegin(GL_LINES);
            auto p1 = poses_g[i], p2 = poses_g[i + 1];
            glVertex3d(p1.translation()[0], p1.translation()[1],
p1.translation()[2]);
            glVertex3d(p2.translation()[0], p2.translation()[1],
p2.translation()[2]);
            glEnd();
        }

        for (size_t i = 0; i < poses_e.size() - 1; i++) {
            //glColor3f(1 - (float) i / poses_e.size(), 0.0f, (float) i /
poses_e.size());
            glColor3f(0.0f, 0.0f, 0.0f);
            glBegin(GL_LINES);
            auto p1 = poses_e[i], p2 = poses_e[i + 1];
            glVertex3d(p1.translation()[0], p1.translation()[1],
p1.translation()[2]);
            glVertex3d(p2.translation()[0], p2.translation()[1],
p2.translation()[2]);
            glEnd();
        }
        pangolin::FinishFrame();
        usleep(5000);   // sleep 5 ms
    }

}
```

- CMakeLists

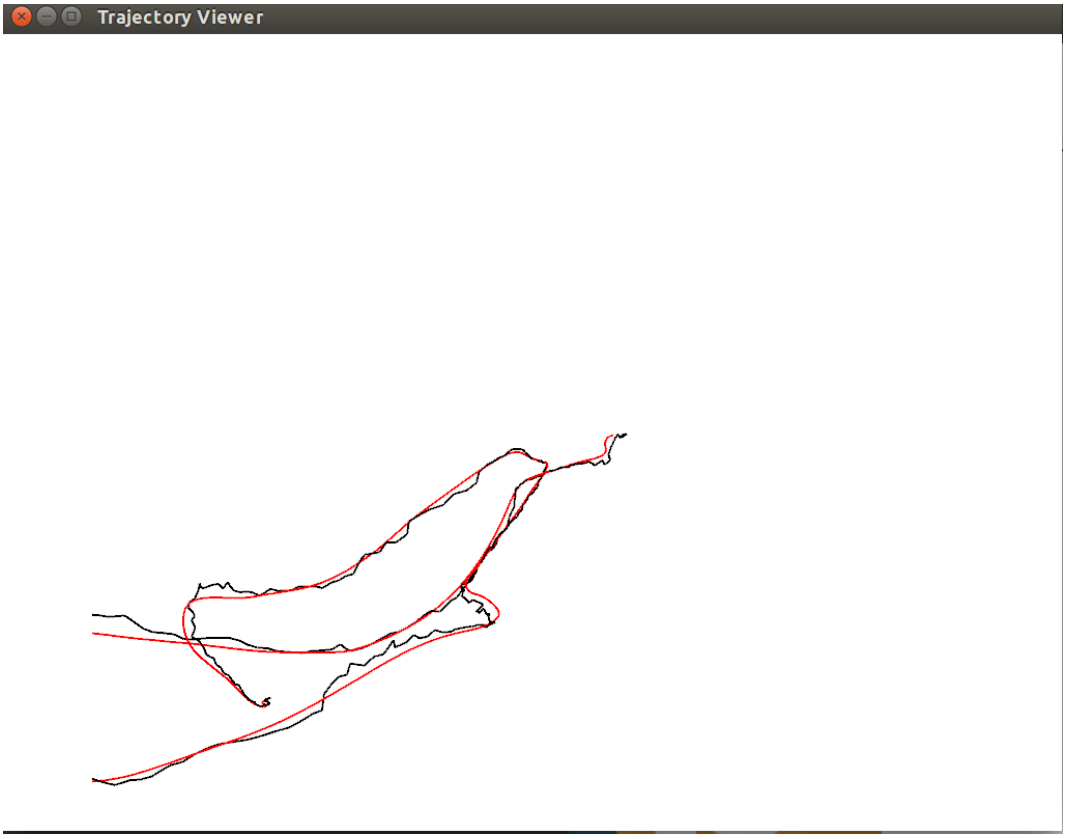
```
cmake_minimum_required(VERSION 2.8.3)
SET(CMAKE_BUILD_TYPE "Release")
PROJECT (Chapter5)

add_compile_options(-std=c++11)

INCLUDE_DIRECTORIES(${PROJECT_SOURCE_DIR}/include)
INCLUDE_DIRECTORIES("/usr/include/opencv2")
INCLUDE_DIRECTORIES("/usr/include/eigens3")

find_package( OpenCV REQUIRED )

find_package(Sophus REQUIRED)
include_directories(${Sophus_INCLUDE_DIRS})

find_package(Pangolin REQUIRED)
INCLUDE_DIRECTORIES(${Pangolin_INCLUDE_DIRS})

SET(SRC_LIST ${PROJECT_SOURCE_DIR}/src/computeORB.cpp)
ADD_EXECUTABLE(computeORB ${SRC_LIST})
target_link_libraries(computeORB ${OpenCV_LIBRARIES})

ADD_EXECUTABLE(E2Rt ${PROJECT_SOURCE_DIR}/src/E2Rt.cpp)
```

```
target_link_libraries(E2Rt ${Sophus_LIBRARIES})

ADD_EXECUTABLE(GN-BA ${PROJECT_SOURCE_DIR}/src/GN-BA.cpp)
target_link_libraries(GN-BA ${Sophus_LIBRARIES})

ADD_EXECUTABLE(trajectoryICP ${PROJECT_SOURCE_DIR}/src/trajectoryICP.cpp)
target_link_libraries(trajectoryICP ${Sophus_LIBRARIES}
${Pangolin_LIBRARIES})
```

- 运行结果
  - 原始轨迹



  - ICP轨迹


Trajectory Viewer