第五讲作业

崔华坤

达闼科技（北京）有限公司

2018.3.27

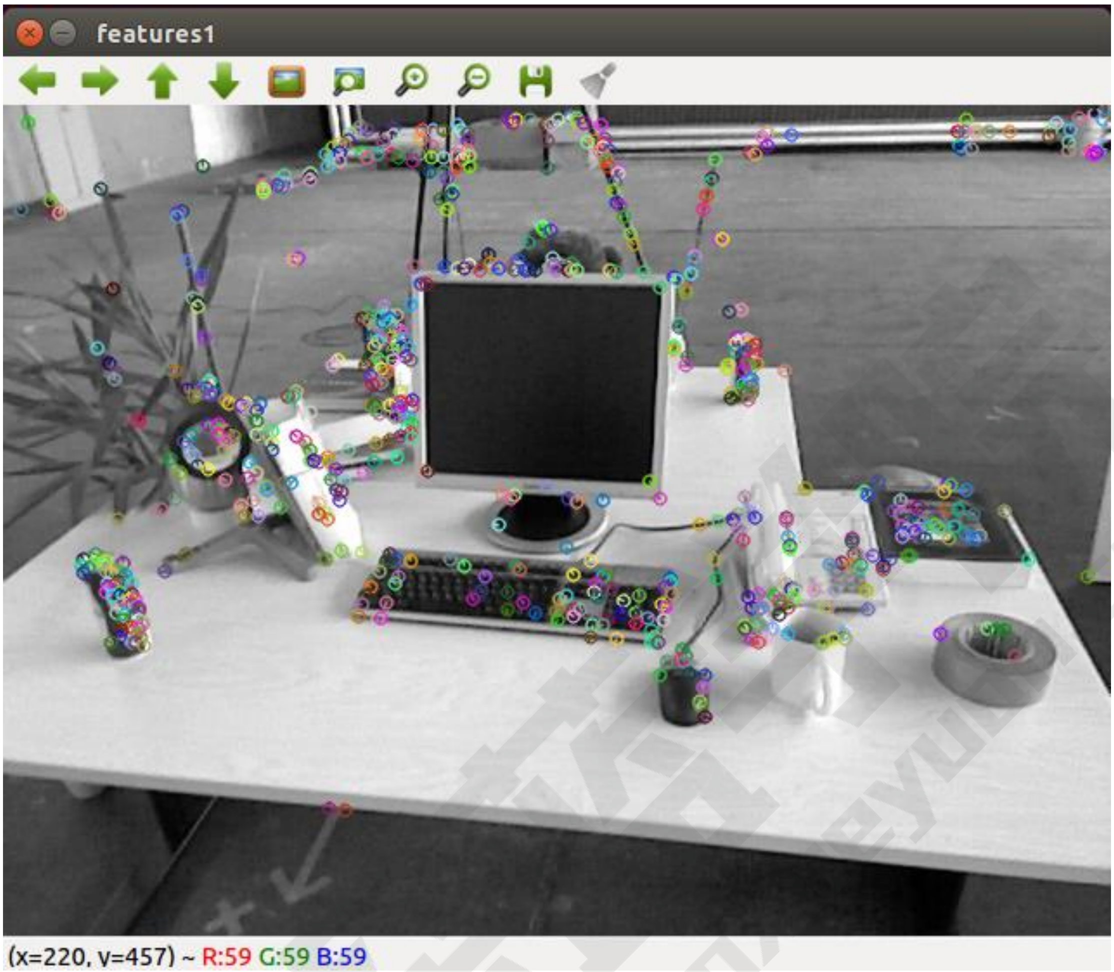
二、ORB 特征点

2.1 ORB 提取

```
// --------------------------------------------------------------------------------------------------
// 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 u = kp.pt.x;
        int v = kp.pt.y;

        if(!bInImage(u-8,v-8,image.cols,image.rows) ||
           !bInImage(u+7,v+7,image.cols,image.rows)) {
          cout<<"Keypoint is near to edge! "<< kp.pt<<endl;
         // keypoints.erase(remove(keypoints.begin(),keypoints.end(),kp),keypoints.end());
          continue;
        }
        double m10 = 0;
        double m01 = 0;
        for(int j = -8; j < 8; j++) {
          for(int i = -8; i < 8; i++){
            double Gray = (double)image.at<uchar>(v+j,u+i);
            m10 += i * Gray;
            m01 += j * Gray;
          }
        }

        kp.angle = (float)atan2(m01,m10)*180.0/pi;
        // END YOUR CODE HERE
    }
    return;
}
```



2.2 ORB 描述

```
// Caculate the position after compensating the keypoint's rotation
// x,y[int]: the position when keypoint's angle is 0
// angle[int]: keypoint's angle, unit: degree
// [out]: the position after rotating the angle in the same coordinate
void GetPosAfterRotation(int x_in, int y_in, float angle, int &x_out, int &y_out)
{
    float fCos = (float)cos(angle * pi / 180.0);
    float fSin = (float)sin(angle * pi / 180.0);
    float x_temp = (float)x_in * fCos - (float)y_in * fSin;
    float y_temp = (float)x_in * fSin + (float)y_in * fCos;
    x_out = (int)x_temp;
    y_out = (int)y_temp;

}
// compute the descriptor
void computeORBDesc(const cv::Mat &image, vector<cv::KeyPoint> &keypoints, vector<DescType> &desc) {
    int px = 0;
    int py = 0;
    int qx = 0;
    int qy = 0;

    for (auto &kp: keypoints) {
        DescType d(256, false);
        for (int i = 0; i < 256; i++) {
            // START YOUR CODE HERE (~7 lines)
            d[i] = 0;  // if kp goes outside, set d.clear()
            GetPosAfterRotation(ORB_pattern[4*i], ORB_pattern[4*i+1], kp.angle, px, py);
            px += kp.pt.x;
            py += kp.pt.y;
            GetPosAfterRotation(ORB_pattern[4*i+2], ORB_pattern[4*i+3], kp.angle, qx, qy);
            qx += kp.pt.x;
            qy += kp.pt.y;

            if(!bInImage(px,py,image.cols,image.rows)  ||
                !bInImage(qx,qy,image.cols,image.rows)) {
                d.clear();
                break;
            } else {
                d[i] = (image.at<uchar>(py,px) > image.at<uchar>(qy,qx))? 0 : 1;
            }
            // END YOUR CODE HERE
        }
        desc.push_back(d);
    }

    int bad = 0;
    for (auto &d: desc) {
        if (d.empty()) bad++;
    }
```

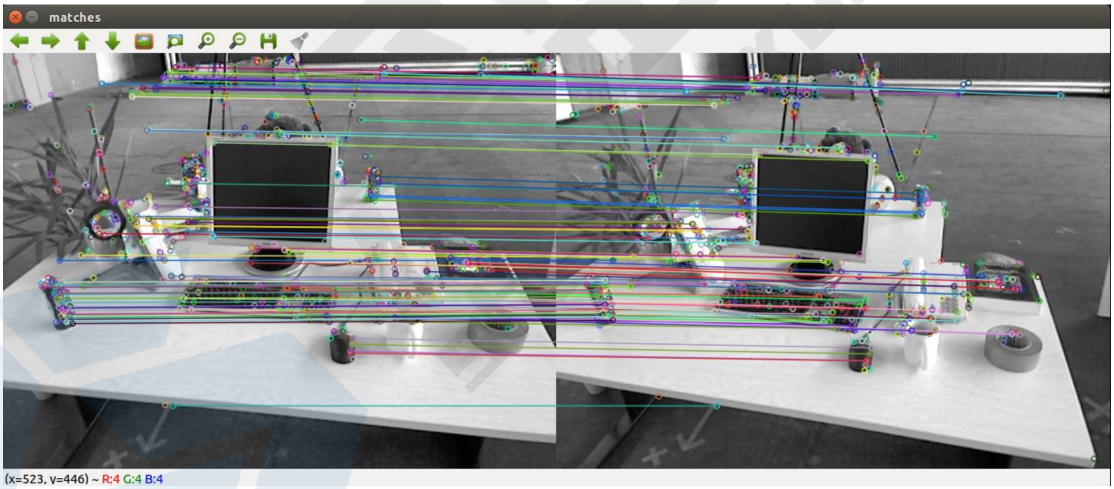
2.3 暴力匹配

```cpp
// 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.
    int d1_num = -1;
    for (auto &d1: desc1)
    {
        d1_num++;
        if(d1.empty()) continue;
        vector<vector<int>> d1_match(0,vector<int>(2));

        int d2_num = -1;
        for(auto &d2: desc2)
        {
            d2_num++;
            if(d2.empty()) continue;

            int HammingDist = 0;
            for(int n = 0; n<256; n++) {
                HammingDist += (d1[n] == d2[n])?0:1;
            }
            vector<int> d2_HamDis(2);
            d2_HamDis =  {HammingDist, d2_num};
            d1_match.push_back(d2_HamDis);
        }
        sort(d1_match.begin(), d1_match.end());
        int dist = d1_match[0][0];
        if(dist <= d_max){
            DMatch m(d1_num,d1_match[0][1],dist);
            matches.push_back(m);
        }
    }
    cout<<"matches size: "<<matches.size()<<endl;
    // END YOUR CODE HERE
    for (auto &m: matches) {
        cout << m.queryIdx << ", " << m.trainIdx << ", " << m.distance << endl;
    }
    return;
}
```



```
stevencui@ubuntu:~/Project/SLAMCourse/l5-2-orb-detect/build$ ./orb-detect
init done
opengl support available
1 image1 keypoints: 638
Keypoint is near to edge! [630, 5]
Keypoint is near to edge! [327, 6]
Keypoint is near to edge! [291, 7]
Keypoint is near to edge! [293, 7]
Keypoint is near to edge! [331, 7]
Keypoint is near to edge! [621, 7]
Keypoint is near to edge! [635, 8]
Keypoint is near to edge! [636, 22]
Keypoint is near to edge! [636, 24]
bad/total: 41/638
image2 keypoints: 595
Keypoint is near to edge! [635, 43]
bad/total: 6/595
matches size: 107 Match time cost= 0.154373 seconds.
36, 21, 42
48, 42, 48
53, 33, 34
58, 58, 28
59, 70, 45
60, 46, 39
63, 73, 33
74, 52, 27
79, 71, 30
82, 56, 23
89, 61, 36
92, 60, 30
101, 77, 20
```

问题回答如下：

1. ORB 的描述子是用许多对像素点坐标（此处坐标需要经过旋转校正）所在位置的灰度值的大小对比来组成，对比结果为 0 和 1，假设有 256 对像素点，那么 ORB 的特征点描述子便有 256 个 0 或 1 组成，因此称 ORB 是一种二进制特征；
2. ORB 匹配时是计算两个描述子之前的汉明距离，即计算两个二进制值在每一位上不相等的个数，当然我们希望这个不相等的个数越少越好，当个数为 0，则说明两个描述子的特征完全一致，随着个数越多，说明两个特征不一致的程度也就越高，因此若将阈值取小，则匹配点对的数量越少，但是误匹配几率越小，反之，若将阈值调高，则匹配出的点对数量增加，但误匹配几率也增加。
3. 匹配出 107 组点，耗时为 3329.67ms 。若在 CMakeLists 中增加-O2 编译优化，则耗时可减小至 160.675ms，若改为-O3 编译优化，耗时可继续减小至 154.373ms。

三、从 E 恢复 R, t

```
U:
 0.0334451  -0.568383  -0.822084
  0.993853   0.105773 -0.0326974
  0.105539  -0.815937   0.568426
V:
  0.551696  0.0831262   0.829892
 -0.155034   0.987901 0.00411052
   0.81951   0.130929  -0.557908
Sigma:
    0.707107 -9.71445e-17  1.11022e-16
-1.38778e-17     0.707107  9.71445e-17
 5.95412e-17  6.51463e-17  9.34203e-17
Sigma after fix:
0.707107        0        0
       0 0.707107        0
       0        0        0
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
1 t^R = -0.0203619  -0.400711 -0.0332407
  0.393927  -0.035064   0.585711
-0.00678849  -0.581543 -0.0143826
2 t^R = -0.0203619  -0.400711 -0.0332407
  0.393927  -0.035064   0.585711
-0.00678849  -0.581543 -0.0143826
3 t^R = 0.0203619  0.400711 0.0332407
-0.393927  0.035064 -0.585711
0.00678849  0.581543 0.0143826
4 t^R = 0.0203619  0.400711 0.0332407
-0.393927  0.035064 -0.585711
0.00678849  0.581543 0.0143826
```

```
int main(int argc, char **argv) {
    cout<<"E2Rt main..."<<endl;
    // 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<MatrixXd> svd(E, ComputeThinU | ComputeThinV);
    Matrix3d U = svd.matrixU();
    Matrix3d V = svd.matrixV();
    Matrix3d Sigma = U.inverse() * E * V.transpose().inverse();
    cout<<"U:\n"<<U<<"\nV:\n"<<V<<"\nSigma:\n"<<Sigma<<endl;
    vector<double> tao = {Sigma(0,0), Sigma(1,1), Sigma(2,2)};
    sort(tao.begin(),tao.end());
    Matrix3d SigmaFix = Matrix3d::Zero();
    double tao_mean = (tao[1]+tao[2])*0.5;
    SigmaFix(0,0) = tao_mean;
    SigmaFix(1,1) = tao_mean;
    cout<<"Sigma after fix: \n"<<SigmaFix<<endl;
    // END YOUR CODE HERE

    // set t1, t2, R1, R2
    // START YOUR CODE HERE
    Matrix3d R_Z1 = AngleAxisd(M_PI/2,Vector3d(0,0,1)).matrix();
    Matrix3d R_Z2 = AngleAxisd(-M_PI/2,Vector3d(0,0,1)).matrix();

    Matrix3d t_wedge1 = U * R_Z1 * SigmaFix * U.transpose();
    Matrix3d t_wedge2 = U * R_Z2 * SigmaFix * U.transpose();

    Matrix3d R1 = U * R_Z1.transpose() * V.transpose();
    Matrix3d R2 = U * R_Z2.transpose() * V.transpose();

    // END YOUR CODE HERE

    cout << "R1 = " << R1 << endl;
    cout << "R2 = " << R2 << endl;
    cout << "t1 = " << SO3::vee(t_wedge1).transpose() << endl;
    cout << "t2 = " << SO3::vee(t_wedge2).transpose() << endl;
```

四、用 G-N 实现 Bundle Adjustment

```
ifstream ifp3d, ifp2d;
string sP3d, sP2d;

int main(int argc, char **argv) {
    cout<<"PnP BA main..."<<endl;
    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;
    cout<<"K: \n"<<K<<endl;
    // load points in to p3d and p2d
    // START YOUR CODE HERE
    ifp3d.open(p3d_file.c_str());
    if(!ifp3d.is_open()) {
        cerr<<"ifp3d is not open! p3d_file: "<<p3d_file.c_str()<<endl;
        return -1;
    }
    ifp2d.open(p2d_file.c_str());
    if(!ifp2d.is_open()) {
        cerr<<"ifp2d is not open! p2d_file: "<<p2d_file.c_str()<<endl;
        return -1;
    }

    while(getline(ifp3d,sP3d) && !sP3d.empty()) {
        istringstream issP3d(sP3d);
        Vector3d vP3d;
        issP3d >> vP3d[0] >> vP3d[1] >> vP3d[2];
        p3d.push_back(vP3d);
    }
//     cout<<"p3d size: "<<p3d.size()<<endl;

    while(getline(ifp2d, sP2d) && !sP2d.empty()) {
        istringstream issP2d(sP2d);
        Vector2d vP2d;
        issP2d >> vP2d[0] >> vP2d[1];
        p2d.push_back(vP2d);
    }
//     cout<<"p2d size: "<<p2d.size()<<endl;

    // END YOUR CODE HERE
    assert(p3d.size() == p2d.size());
```

```
    int iterations = 10;
    double cost = 0, lastCost = 0;
    int nPoints = p3d.size();
    cout << "points: " << nPoints << endl;

    Sophus::SE3 T_esti(Matrix3d::Identity(), Vector3d::Zero()); // estimated pose
    cout<<"T_esti start: \n"<<T_esti.matrix()<<endl;
    for (int iter = 0; iter < iterations; iter++) {

        Matrix<double, 6, 6> H = Matrix<double, 6, 6>::Zero();
        Vector6d b = Vector6d::Zero();
        Vector2d e;
        cost = 0;
        // compute cost
        for (int i = 0; i < nPoints; i++) {
            // compute cost for p3d[I] and p2d[I]
            // START YOUR CODE HERE
            Vector3d Pc = T_esti * p3d[i];
            if(DEBUG) cout<<"K * Pc: \n" << K*Pc<<endl;
            Vector3d v3_e = Vector3d(p2d[i][0], p2d[i][1], 1) - K * Pc / Pc[2];
            e[0] = v3_e[0];
            e[1] = v3_e[1];
            if(DEBUG) cout<<"v3_e:"<<v3_e.transpose()<<" e: "<<e.transpose()<<" Pc: "<<
            double x = Pc[0]; double y = Pc[1]; double z = Pc[2];
            double x2 = x * x;
            double y2 = y * y;
            double z2 = z * z;

            //compute jacobi
            Matrix<double, 2, 6> J = Matrix<double, 2, 6>::Zero();
            J(0,0) = -fx / z;
            J(0,2) = fx * x / z2;
            J(0,3) = fx * x * y / z2;
            J(0,4) = -fx - fx * x2 / z2;
            J(0,5) = fx * y / z;
            J(1,1) = -fy / z;
            J(1,2) = fy * y / z2;
            J(1,3) = fy + fy * y2 / z2;
            J(1,4) = -fy * x * y / z2;
            J(1,5) = -fy * x / z;
            // END YOUR CODE HERE
            if(DEBUG) cout<<"J: \n"<<J<<endl;

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

            cost += 0.5 * e.transpose() * e;
            if(DEBUG) cout<<"cost: "<<cost<<endl;
        }
        if(DEBUG) cout<<"H : \n"<<H<< "\n b: \n"<<b<<endl;
```

```
        // solve dx
        Vector6d dx;
        // START YOUR CODE HERE
        dx = H.ldlt().solve(b);
        cout<<"iter: "<< iter <<"dx: "<<dx.transpose()<<endl;
        // END YOUR CODE HERE

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

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

        // update your estimation
        // START YOUR CODE HERE
        T_esti = 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;
```

```
stevencui@ubuntu:~/Project/SLAMCourse/l5-4-pnp/build$ ./pnp_ba
PnP BA main...
K:
520.9     0 325.1
    0   521 249.7
    0     0     1
points: 76
T_esti start:
1 0 0 0
0 1 0 0
0 0 1 0
0 0 0 1
iter: 0dx:   -0.121534 -0.00625545   0.0624059   -0.0260197   0.0379231   0.049552
iteration 0 cost=622769.1141257
iter: 1dx: -0.00730621053146 0.00108128881576 -0.00374197890306 -0.00052085459371
iteration 1 cost=12206.604278533
iter: 2dx: 2.52359402886e-05 -1.26017542736e-05 -4.24306741566e-06 -6.34719603115
iteration 2 cost=12150.675965788
iter: 3dx: 2.29320271917e-08 2.92055292543e-08 -5.85083751549e-09 1.71616802208e-
iteration 3 cost=12150.6753269
iter: 4dx: 1.95956969106e-10 3.06507122881e-11 -2.95082895258e-11 1.53638120777e-
iteration 4 cost=12150.6753269
iter: 5dx: 7.08612653177e-13 6.19465674549e-13 -7.5620611483e-14 3.64401118231e-1
iteration 5 cost=12150.6753269
iter: 6dx: 4.49876150176e-15 1.86746378308e-15 -5.33124654598e-16 1.0501562107e-1
cost: 150.675, last cost: 150.675
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
```

问题回答如下：

1. 重投影误差：

$$e(\xi) = P_{uv} - \frac{1}{z_c} K exp(\xi^\wedge) P_w$$

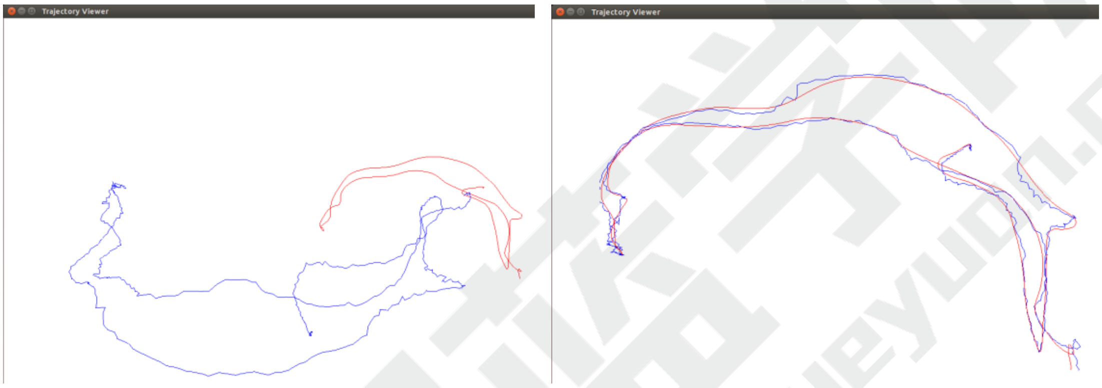
2. 雅可比矩阵为：

$$J(\xi) = -\begin{bmatrix} \frac{f_x}{z_c} & 0 & -\frac{f_x x_c}{{z_c}^2} & -\frac{f_x x_c y_c}{{z_c}^2} & f_x + \frac{f_x {x_c}^2}{{z_c}^2} & -\frac{f_x y_c}{z_c} \\ 0 & \frac{f_y}{z_c} & -\frac{f_y y_c}{{z_c}^2} & -f_y - \frac{f_y {y_c}^2}{{z_c}^2} & \frac{f_y x_c y_c}{{z_c}^2} & \frac{f_y x_c}{z_c} \end{bmatrix}$$

3. 更新之前估计：

$$T_{k+1} = exp(\Delta\xi^{\wedge})T_k$$

五、用 ICP 实现轨迹对齐



```
stevencui@ubuntu:~/Project/SLAMCourse/l5-5-ICP/build$ make
Scanning dependencies of target icp
[ 50%] Building CXX object CMakeFiles/icp.dir/icp.cpp.o
[100%] Linking CXX executable icp
[100%] Built target icp
stevencui@ubuntu:~/Project/SLAMCourse/l5-5-ICP/build$ ./icp
ICP main...
vt_w_cg size: 612 vt_w_ce size: 612
iteration= 0     chi2= 6716865.410383     time= 0.000341021      cumTime= 0.000341021
iteration= 1     chi2= 1410981.119538     time= 0.000258397      cumTime= 0.000599418
iteration= 2     chi2= 62548.433600       time= 0.000261181      cumTime= 0.000860599
iteration= 3     chi2= 3265.699587        time= 0.000271071      cumTime= 0.00113167
iteration= 4     chi2= 3262.995060        time= 0.000215336      cumTime= 0.00134701
iteration= 5     chi2= 3262.995060        time= 0.000214053      cumTime= 0.00156106
iteration= 6     chi2= 3262.995060        time= 0.000247714      cumTime= 0.00180877
iteration= 7     chi2= 3262.995060        time= 0.000435743      cumTime= 0.00224452
iteration= 8     chi2= 3262.995060        time= 0.000333575      cumTime= 0.00257809
iteration= 9     chi2= 3262.995060        time= 0.000391315      cumTime= 0.00296941
T_wg_we:
 0.923062  0.133592 -0.360707    1.5394
 0.369046 -0.571969  0.732568  0.932636
-0.108448 -0.809323 -0.577265   1.44618
        0         0         0         1
```

```
class EdgeProjectXYZRGBDPoseOnly : public g2o::BaseUnaryEdge<3, Vector3d, VertexSE3Expmap>
{
public:
    EIGEN_MAKE_ALIGNED_OPERATOR_NEW;
    EdgeProjectXYZRGBDPoseOnly(const Vector3d& p_we): _p_we( p_we ){};
    virtual void computeError()
    {
        const VertexSE3Expmap* T_wg_we = static_cast<const VertexSE3Expmap*>(_vertices[0]);
        _error = _measurement - T_wg_we->estimate().map( _p_we );
    }
    virtual void linearizeOplus()
    {
        VertexSE3Expmap* T_wg_we = static_cast<VertexSE3Expmap*>( _vertices[0]);
        Vector3d p_wg = T_wg_we->estimate().map( _p_we );
        double x = p_wg[0];
        double y = p_wg[1];
        double z = p_wg[2];

        _jacobianOplusXi(0,0) = 0;
        _jacobianOplusXi(0,1) = -z;
        _jacobianOplusXi(0,2) = y;
        _jacobianOplusXi(0,3) = -1;
        _jacobianOplusXi(0,4) = 0;
        _jacobianOplusXi(0,5) = 0;

        _jacobianOplusXi(1,0) = z;
        _jacobianOplusXi(1,1) = 0;
        _jacobianOplusXi(1,2) = -x;
        _jacobianOplusXi(1,3) = 0;
        _jacobianOplusXi(1,4) = -1;
        _jacobianOplusXi(1,5) = 0;

        _jacobianOplusXi(2,0) = -y;
        _jacobianOplusXi(2,1) = x;
        _jacobianOplusXi(2,2) = 0;
        _jacobianOplusXi(2,3) = 0;
        _jacobianOplusXi(2,4) = 0;
        _jacobianOplusXi(2,5) = -1;
    }
    bool read(istream& in) {};
    bool write(ostream& out) const {};
protected:
    Vector3d _p_we;
};
```

```
void ICP_BundleAdjustment()
{
    typedef g2o::BlockSolver< g2o::BlockSolverTraits<6,3> > Block;
    Block::LinearSolverType* linearSolver = new g2o::LinearSolverCSparse<Block::PoseMatrixType>();
    Block* solver_ptr = new Block(linearSolver);
    g2o::OptimizationAlgorithmLevenberg* solver = new g2o::OptimizationAlgorithmLevenberg( solver_ptr);

    g2o::SparseOptimizer optimizer;
    optimizer.setAlgorithm(solver);

    //vertex T_wg_we
    g2o::VertexSE3Expmap* T_wg_we = new g2o::VertexSE3Expmap();
    T_wg_we -> setId(0);
    T_wg_we -> setEstimate(g2o::SE3Quat(R_wg_we, t_wg_we ));
    optimizer.addVertex(T_wg_we);

    //edge
    int index = 1;
    vector<EdgeProjectXYZRGBDPoseOnly*> edges;
    for(size_t i = 0; i < vt_w_ce.size(); i++)
    {
        EdgeProjectXYZRGBDPoseOnly* edge = new EdgeProjectXYZRGBDPoseOnly(vt_w_ce[i]);
        edge->setId(index);
        edge->setVertex(0, dynamic_cast<VertexSE3Expmap*>(T_wg_we));
        edge->setMeasurement(vt_w_cg[i]);
        edge->setInformation(Matrix3d::Identity() * 1e4);
        optimizer.addEdge(edge);
        index++;
        edges.push_back(edge);
    }

    //optimize
    optimizer.setVerbose(true);
    optimizer.initializeOptimization();
    optimizer.optimize(10);
    isoT_wg_we = Isometry3d(T_wg_we->estimate());
    cout<< "T_wg_we: \n"<< isoT_wg_we.matrix()<<endl;
}
```

```
int main()
{
    cout<<"ICP main..."<<endl;
    ifFile.open(sFilePath.c_str());
    if(!ifFile.is_open()) {
        cerr<<"isFile is not open! sFilePath: " << sFilePath.c_str() << endl;
        return -1;
    }
    string sFile;
    while(getline(ifFile,sFile) && !sFile.empty()) {
        istringstream iss(sFile);
        iss >> timestamp >> t_w_ce[0] >> t_w_ce[1] >> t_w_ce[2] >>
                q_w_ce.x() >> q_w_ce.y() >> q_w_ce.z() >> q_w_ce.w() >>
                timestamp >> t_w_cg[0] >> t_w_cg[1] >> t_w_cg[2] >>
                q_w_cg.x() >> q_w_cg.y() >> q_w_cg.z() >> q_w_cg.w();
        //cout << "t_we: " << t_we.transpose() << "q_we: " << q_we.coeffs().transpose() << endl;
        //cout << "t_wg: " << t_wg.transpose() <<" q_wg: "<< q_wg.coeffs().transpose() << endl;
        vt_w_cg.push_back(t_w_cg);
        vt_w_ce.push_back(t_w_ce);
    }

    cout<<"vt_w_cg size: "<<vt_w_cg.size() <<" vt_w_ce size: "<<vt_w_ce.size()<<endl;

//    DrawTrajectory(vt_w_cg, vt_w_ce);

    ICP_BundleAdjustment();
    for(int i = 0; i< vt_w_ce.size();i++) {
        vt_w_ce2.push_back(isoT_wg_we * vt_w_ce[i]);
    }
    DrawTrajectory(vt_w_cg, vt_w_ce2);
    return 0;
}
```