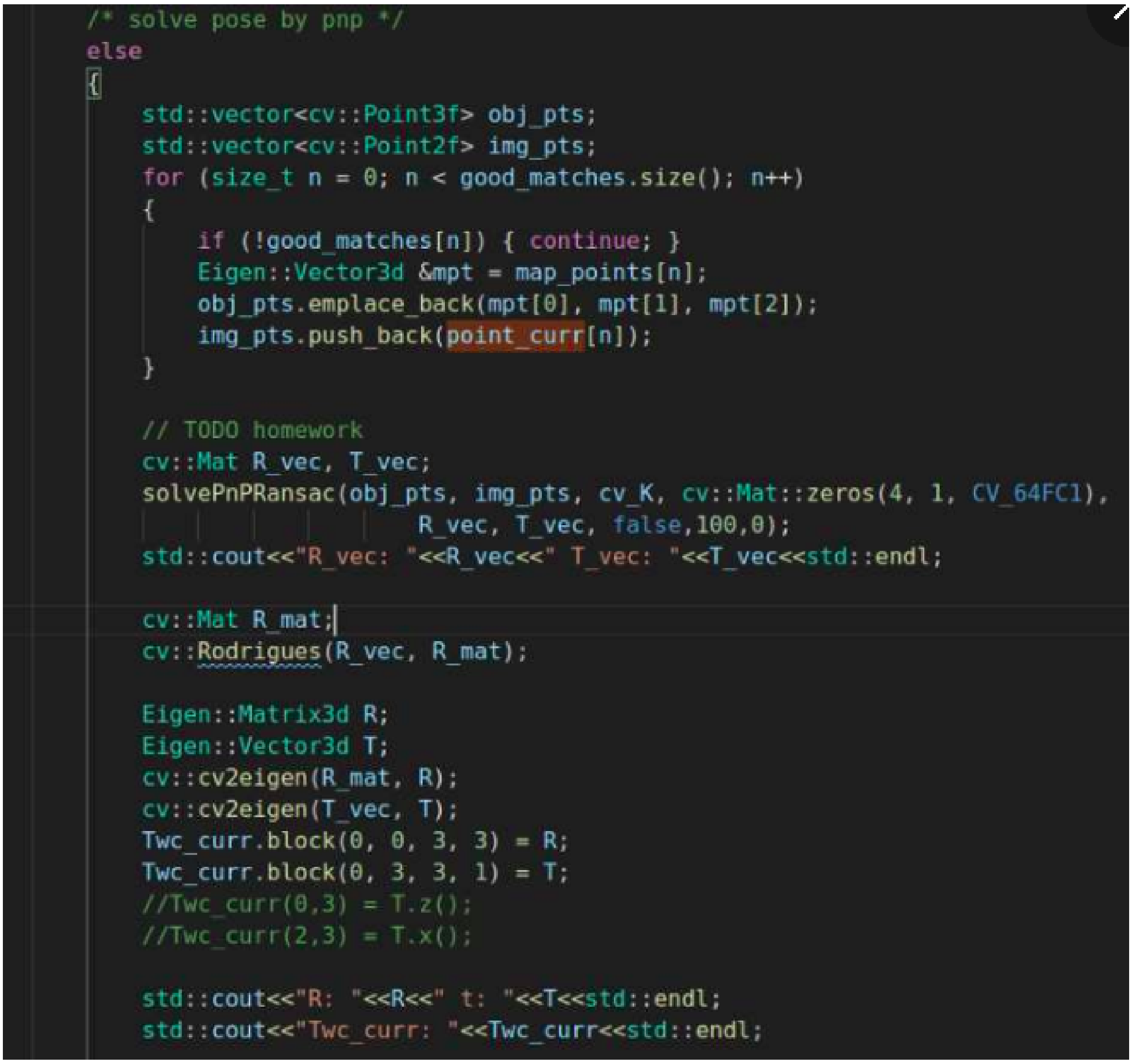
说明： 代码补充完成，但是计算结果一直有问题，排除中暂时还找不到问题点。。



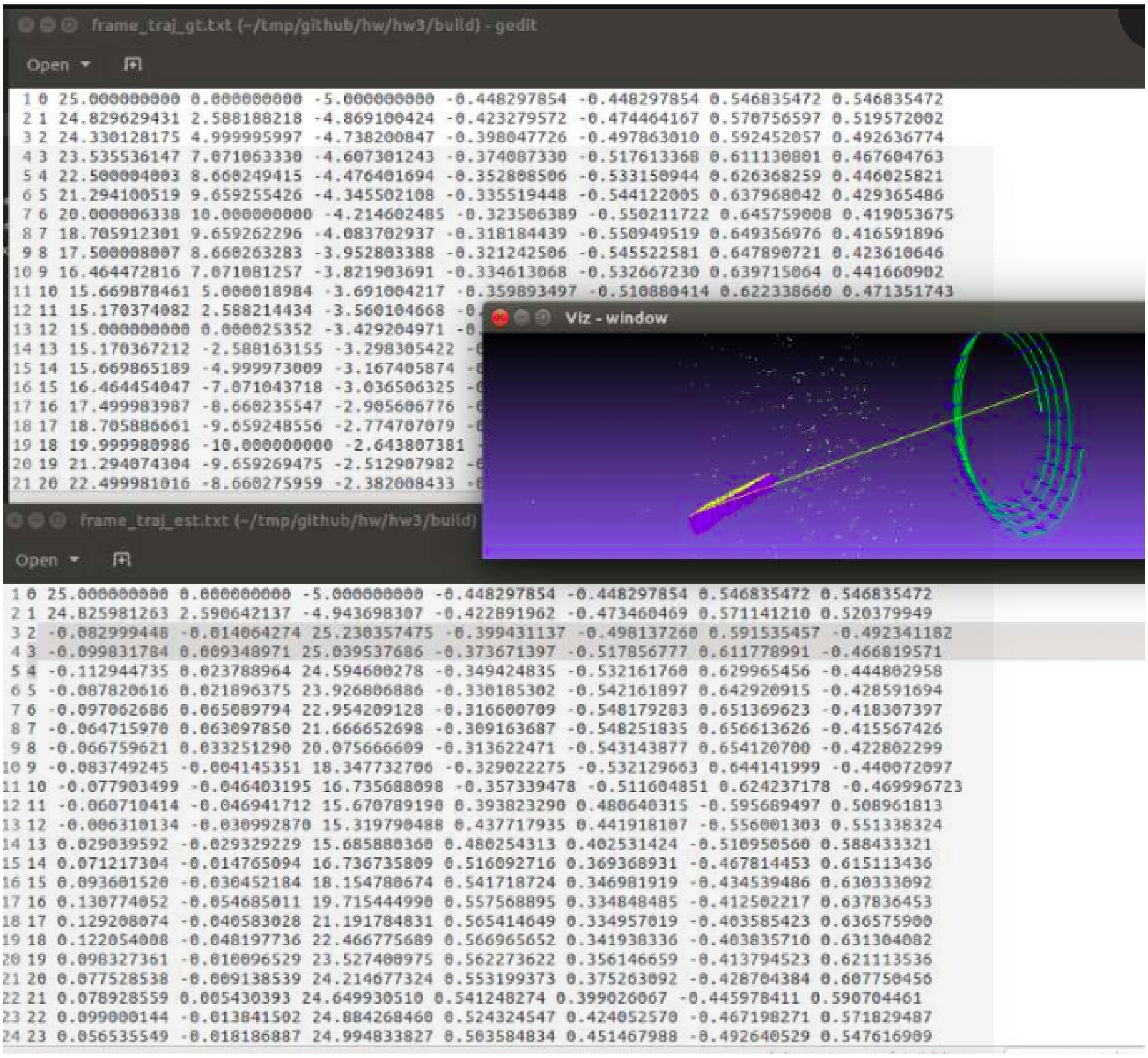
```
/* solve pose by pnp */
else
{
    std::vector<cv::Point3f> obj_pts;
    std::vector<cv::Point2f> img_pts;
    for (size_t n = 0; n < good_matches.size(); n++)
    {
        if (!good_matches[n]) { continue; }
        Eigen::Vector3d &mpt = map_points[n];
        obj_pts.emplace_back(mpt[0], mpt[1], mpt[2]);
        img_pts.push_back(point_curr[n]);
    }

    // TODO homework
    cv::Mat R_vec, T_vec;
    solvePnPRansac(obj_pts, img_pts, cv_K, cv::Mat::zeros(4, 1, CV_64FC1),
                                        R_vec, T_vec, false,100,0);
    std::cout<<"R_vec: "<<R_vec<<" T_vec: "<<T_vec<<std::endl;

    cv::Mat R_mat;
    cv::Rodrigues(R_vec, R_mat);

    Eigen::Matrix3d R;
    Eigen::Vector3d T;
    cv::cv2eigen(R_mat, R);
    cv::cv2eigen(T_vec, T);
    Twc_curr.block(0, 0, 3, 3) = R;
    Twc_curr.block(0, 3, 3, 1) = T;
    //Twc_curr(0,3) = T.z();
    //Twc_curr(2,3) = T.x();

    std::cout<<"R: "<<R<<" t: "<<T<<std::endl;
    std::cout<<"Twc_curr: "<<Twc_curr<<std::endl;
```


frame_traj_gt.txt (~/tmp/github/hw/hw3/build) - gedit
Open
Viz - window
frame_traj_est.txt (~/tmp/github/hw/hw3/build)
Open

```
0 25.000000000 0.000000000 -5.000000000 -0.448297854 -0.448297854 0.546835472 0.546835472
1 24.829629431 2.588188218 -4.869100424 -0.423279572 -0.474464167 0.570756597 0.519572002
2 24.330128175 4.999995997 -4.738200847 -0.398047726 -0.497863010 0.592452057 0.492636774
3 23.535536147 7.071063330 -4.607301243 -0.374087330 -0.517613368 0.611130801 0.467604763
4 22.500004003 8.660249415 -4.476401694 -0.352808506 -0.533150944 0.626368259 0.446025821
5 21.294100519 9.659255426 -4.345502108 -0.335519448 -0.544122005 0.637968042 0.429365486
6 20.000006338 10.000000000 -4.214602485 -0.323506389 -0.550211722 0.645759008 0.419053675
7 18.705912301 9.659262296 -4.083702937 -0.318184439 -0.550949519 0.649356976 0.416591896
8 17.500008007 8.660263283 -3.952803388 -0.321242506 -0.545522581 0.647890721 0.423610646
9 16.464472816 7.071081257 -3.821903691 -0.334613068 -0.532667230 0.639715064 0.441660902
10 15.669878461 5.000018984 -3.691004217 -0.359893497 -0.510880414 0.622338660 0.471351743
11 15.170374082 2.588214434 -3.560104668 -0.
12 15.000000000 0.000025352 -3.429204971 -0.
13 15.170367212 -2.588163155 -3.298305422 -
14 15.669865189 -4.999973009 -3.167405874 -
15 16.464454047 -7.071043718 -3.036506325 -
16 17.499983987 -8.660235547 -2.905606776 -
17 18.705886661 -9.659248556 -2.774707079 -
18 19.999980986 -10.000000000 -2.643807381
19 21.294074304 -9.659269475 -2.512907982 -
20 22.499981016 -8.660275959 -2.382008433 -
```

```
0 25.000000000 0.000000000 -5.000000000 -0.448297854 -0.448297854 0.546835472 0.546835472
1 24.825981263 2.590642137 -4.943698307 -0.422891962 -0.473460469 0.571141210 0.520379949
2 -0.082999448 -0.014064274 25.230357475 -0.399431137 -0.498137260 0.591535457 -0.492341182
3 -0.099831784 0.009348971 25.039537686 -0.373671397 -0.517856777 0.611778991 -0.466819571
4 -0.112944735 0.023788964 24.594600278 -0.349424835 -0.532161760 0.629965456 -0.444802958
5 -0.087820616 0.021896375 23.926806886 -0.330185302 -0.542161897 0.642920915 -0.428591694
6 -0.097062686 0.065089794 22.954209128 -0.316600709 -0.548179283 0.651369623 -0.418307397
7 -0.064715970 0.063097850 21.666652698 -0.309163687 -0.548251835 0.656613626 -0.415567426
8 -0.066759621 0.033251290 20.075666609 -0.313622471 -0.543143877 0.654120700 -0.422802299
9 -0.083749245 -0.004145351 18.347732706 -0.329022275 -0.532129663 0.644141999 -0.440072097
10 -0.077903499 -0.046403195 16.735688098 -0.357339478 -0.511604851 0.624237178 -0.469996723
11 -0.060710414 -0.046941712 15.670789190 0.393823290 0.480640315 -0.595689497 0.508961813
12 -0.006310134 -0.030992870 15.319790488 0.437717935 0.441918107 -0.556001303 0.551338324
13 0.029039592 -0.029329229 15.685880360 0.480254313 0.402531424 -0.510950560 0.588433321
14 0.071217304 -0.014765094 16.736735809 0.516092716 0.369368931 -0.467814453 0.615113436
15 0.093601520 -0.030452184 18.154780674 0.541718724 0.346981919 -0.434539486 0.630333092
16 0.130774052 -0.054685011 19.715444990 0.557568895 0.334848485 -0.412502217 0.637836453
17 0.129208074 -0.040583028 21.191784831 0.565414649 0.334957019 -0.403585423 0.636575900
18 0.122054008 -0.048197736 22.466775689 0.566965652 0.341938336 -0.403835710 0.631304082
19 0.098327361 -0.010096529 23.527400975 0.562273622 0.356146659 -0.413794523 0.621113536
20 0.077528538 -0.009138539 24.214677324 0.553199373 0.375263092 -0.428704384 0.607750456
21 0.078928559 0.005430393 24.649930510 0.541248274 0.399026067 -0.445978411 0.590704461
22 0.099000144 -0.013841502 24.884268460 0.524324547 0.424052570 -0.467198271 0.571829487
23 0.056535549 -0.018186887 24.994833827 0.503584834 0.451467988 -0.492640529 0.547616909
```

================================记录点

landmark

```
createLandmarks  200

    std::normal_distribution<double> d_x{0.0, 4.0};
    std::normal_distribution<double> d_y{0.0, 10.0};
    std::normal_distribution<double> d_z{0.0, 10.0};
```

createCameraPose(v_Twc, Eigen::Vector3d(0,0,0));

```
    float x_offset = 20;
    float y_offset = 0;
    float z_offset = -5;
```

```
detectFeatures(v_Twc[0], K, landmarks, features_last, features_matched_last);
detectFeatures(v_Twc[i], K, landmarks, features_curr, features_matched_cur);

features_curr ==> point_curr
        std::vector<cv::Point2f> point_curr;
        std::vector<cv::Point2f> point_last;
```

std::vectorEigen::Vector3d map_points; map_points.resize(landmarks.size());

```
init_flag = createInitMap(point_curr, point_last, cv_K, T_curr_last, map_points,
good_matches);
```

初始化F， 三角化的map_points是带有尺度的

```
            double t_scale = 1.0;
            {
                Eigen::Matrix4d gt_Twc_last = Twc_last;
                Eigen::Matrix4d gt_Twc_curr = v_Twc[i];
                Eigen::Matrix4d gt_T_cur_last = gt_Twc_curr.inverse() *
gt_Twc_last;
                t_scale = gt_T_cur_last.block(0, 3, 3, 1).norm();
            }
```

map_points恢复到世界坐标系

```
            for (size_t n = 0; n < map_points.size(); n++)
            {
                if (!good_matches[n]) continue;
                Eigen::Vector3d &mpt = map_points[n];
                mpt *= t_scale;
                mpt = Twc_last.block(0, 0, 3, 3) * mpt + Twc_last.block(0, 3, 3,
1);
            }
```