



深蓝学院
shenlanxueyuan.com

视觉SLAM进阶课程 从零开始手写VIO-第5期

第7讲 VINS系统构建 作业讲评



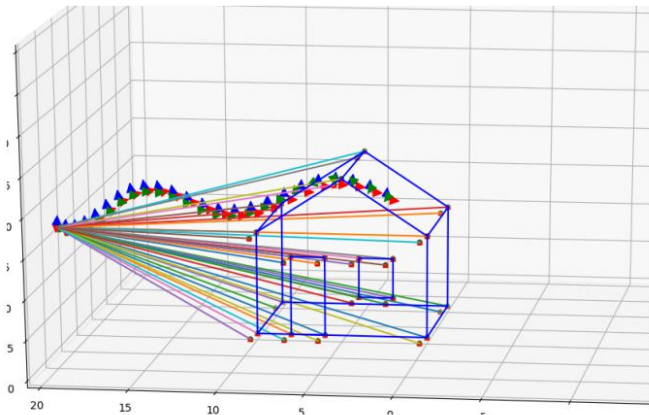
评讲人 吕华龙



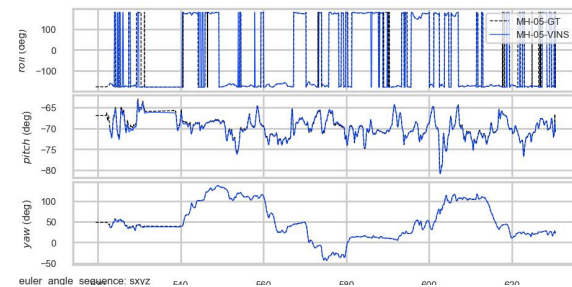
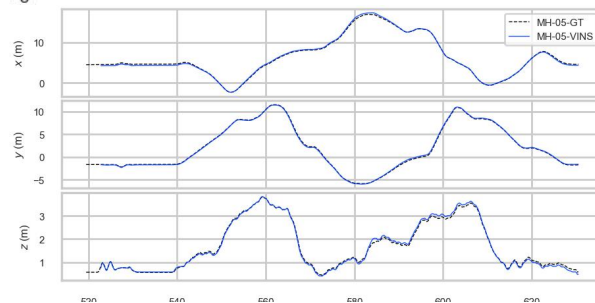
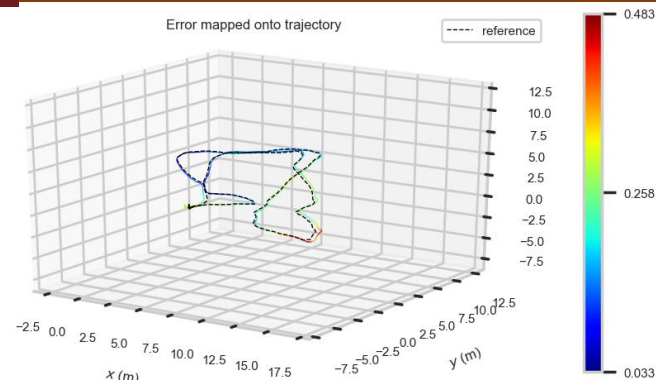
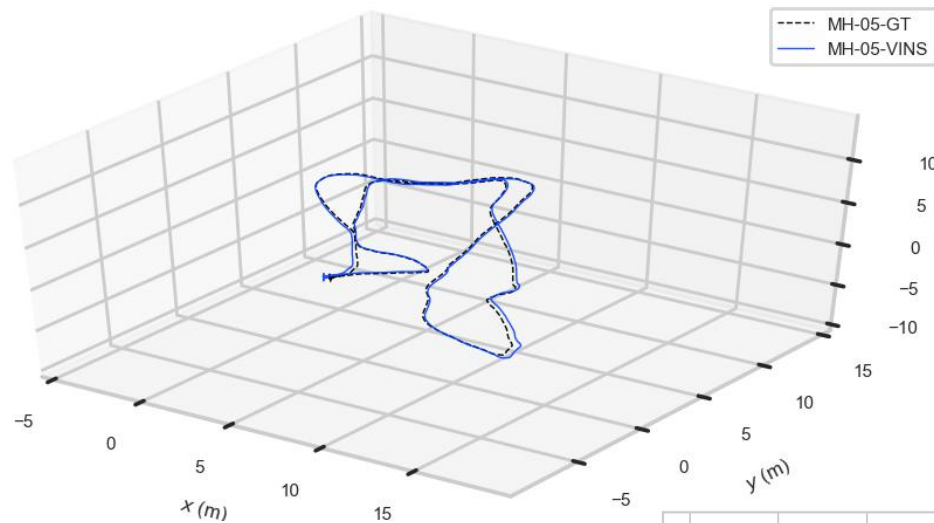
【作业】

作业

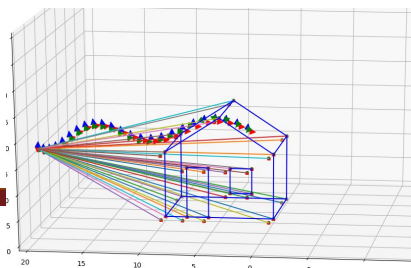
- ① 将第二讲的仿真数据集（视觉特征，imu 数据）接入我们的 VINS 代码，并运行出轨迹结果。
 - 仿真数据集无噪声
 - 仿真数据集有噪声（不同噪声设定时，需要配置 vins 中 imu noise 大小。）



Test VINS Course Project with Euroc-MH-05



仿真数据的输入



imu_data.txt

1	0.000000	0.0000000000000000	0.23064426760421	0.292623102962106	-1.480440795421600	0.979365817305384	9.760990861377433
2	0.004999999888241	-0.000814155811218	0.230631557795240	0.292623484203109	-1.490096205742578	0.962957570887890	9.605341064950775
3	0.009999999776483	-0.001628290544444	0.230329508690953	0.292624627888175	-1.499442351721024	0.940528986841282	9.449723853644117
4	0.014999999664724	-0.002442324762221	0.230338664859335	0.292626533903476	-1.508470232111480	0.930085104505438	9.294173242968126
5	0.019999999552905	-0.003256410568471	0.230318524784657	0.292629202659317	-1.517267014782552	0.913630966636026	9.13872809375363
6	0.025000000372529	-0.004070365806177	0.230292706093361	0.292632632098054	-1.525626038017597	0.897171601898899	8.98347118828036
7	0.030000001152093	-0.004884211823630	0.230261151283429	0.292636082365628	-1.533736805576910	0.880712840517941	8.828326845284552
8	0.03500000149012	-0.005697923490531	0.230223861052679	0.292641776336450	-1.541539808327014	0.864257306196874	8.674309535038080
9	0.039999999109510	-0.006511510526439	0.230188636291953	0.292647409639846	-1.549034430957723	0.847812299244593	8.518813445155237
10	0.044999998062849	-0.007324922261824	0.230132078041440	0.292653962997845	-1.556227142748103	0.831382312992268	8.36448652454973
11	0.049999997019768	-0.008138148055856	0.230077587479952	0.292661195766169	-1.563113298717841	0.814972025686058	8.210497270705302
12	0.054999995976686	-0.008951167266809	0.230017305924928	0.292669187224937	-1.56969624710815	0.798586498518548	8.056883208643271
13	0.059999994833665	-0.009783959295950	0.229951414832395	0.292677336378034	-1.575977478941722	0.782233673366803	7.903802638024766
14	0.0649999937615814	-0.010570584009619	0.229879735741409	0.292687424964350	-1.581958868910632	0.765909460199857	7.749932974442089
15	0.0700000000298023	-0.011388780289313	0.229802338432112	0.292697705429104	-1.587641696779544	0.749627768624761	7.598672101205579
16	0.075000002980232	-0.012206767484042	0.229719200776150	0.2927087322951678	-1.593028507350811	0.733390474241301	7.446937575891119
17	0.080000003662441	-0.01301244907431	0.229638348783848	0.292720483453466	-1.598121279253500	0.717262418272877	7.295768820741496
18	0.085000004344650	-0.013823702201903	0.229557766031172	0.292733018780859	-1.603222338447670	0.70106842423953	7.14539714280277
19	0.090000010162659	-0.014634788539261	0.229435486521690	0.292746294525280	-1.607434152056207	0.68499327761828	6.995265501447838
20	0.095000013789068	-0.015454132421277	0.229329489864535	0.292760320563366	-1.611659373120833	0.66898173861131	6.840609155099388
21	0.100000016391277	-0.016255646280269	0.229217762495363	0.292775709542692	-1.615600793289538	0.653038513218677	6.697464715416415
22	0.105000019073406	-0.01705456559687	0.229110033811012	0.292790817645604	-1.619321364429215	0.637168300340666	6.54968701906316
23	0.110000021755955	-0.017874853714694	0.228977197760914	0.292800885674168	-1.622644192568237	0.621375741718644	6.402657803172782
24	0.115000024479904	-0.018683787212749	0.228848357326136	0.292823897893418	-1.62575236368256	0.605654400647414	6.256467998160103
25	0.120000027120113	-0.0194923246534182	0.228713815610233	0.292841052609997	-1.628598905526087	0.590841972221268	6.11135379646273
26	0.125000029802322	-0.020300211172780	0.228575735873743	0.292860184805621	-1.631195501064082	0.57599847948540	5.966085715102177
27	0.130000032639251	-0.021107659043566	0.228422641729210	0.292879328263044	-1.633465500856499	0.559972570313457	5.823104840301601
28	0.135000036265579	-0.021915772843399	0.228270916507520	0.292899353641678	-1.635511486436114	0.543737546139666	5.680837810885392
29	0.1400000395497208	-0.022720928539993	0.228118703871932	0.292920059904542	-1.637301513186825	0.528596155998076	5.539089742955545
30	0.1450000410728836	-0.023526708476831	0.227955707638365	0.292941499098047	-1.638839719303351	0.513383729297790	5.398575451177407
31	0.150000043506464	-0.024311891423738	0.227793813728789	0.292963669052999	-1.640138032920034	0.498374545585404	5.259120500480186
32	0.1550000461152993	-0.025136456967457	0.227612680247866	0.292986567622109	-1.641177921613016	0.48342829183218	5.12076211386055
33	0.159999948623721	-0.025940384712226	0.227432657397877	0.293010192481406	-1.64198888551066	0.468712749143978	4.98357957490697
34	0.164999951053530	-0.026743652480348	0.227246967548644	0.29303451291367	-1.642561964023971	0.454068418040577	4.84754333717651
35	0.169999953868978	-0.027546245312764	0.227055615189458	0.293059011628062	-1.642907971329485	0.43953389578837	4.712710909808648
36	0.174999957118607	-0.02834817406927	0.226858664951999	0.293084809095997	-1.6430264054831020	0.425172157325850	4.579115432720180
37	0.179999975750235	-0.029149310430872	0.226655941605249	0.293111906826854	-1.642932687588787	0.410930156445337	4.446789718092480
38	0.184999972581063	-0.029949743896782	0.226447638055411	0.293139126482747	-1.642621666520421	0.39682874085079	4.315766250102880
39	0.189999967813492	-0.030749417580563	0.226233675345825	0.293167057253481	-1.642102111929532	0.382872742003872	4.186077197153095
40	0.19499996345120	-0.031548311448904	0.226040864585820	0.293195374090339	-1.641378460911875	0.369685976560613	4.05752374090339
41	0.199999958270749	-0.03234644642550	0.225788579887870	0.293223518246471	-1.64045268846649	0.355481214647171	3.938292829203187
42	0.204999953068377	-0.033143677556863	0.225550884747043	0.293251808894761	-1.639347286532341	0.341014170272775	3.803392874326257

IMU测量数据

keyframe

Name

all_lines_0.txt	
all_lines_1.txt	
all_lines_2.txt	
all_lines_3.txt	
all_lines_4.txt	
all_lines_5.txt	
all_lines_6.txt	
all_lines_7.txt	
all_lines_8.txt	
all_lines_9.txt	
all_lines_10.txt	
all_lines_11.txt	
all_lines_12.txt	
all_lines_13.txt	
all_lines_14.txt	
all_lines_15.txt	
all_lines_16.txt	
all_lines_17.txt	
all_lines_18.txt	
all_lines_19.txt	

all_lines_0.txt (-/Downloads/course2_hw_new/vio_data_0)

Open Save

0.275022	-0.224731	0.242657	0.0978391
0.548675	-0.448344	0.484107	0.195191
-0.441378	-0.54768	0.505947	0.0958546
-0.22124	-0.274523	-0.253605	0.0480468
0.275022	-0.224731	0.548675	-0.448344
0.548675	-0.448344	-0.441378	-0.54768
-0.441378	-0.54768	-0.22124	-0.274523
-0.22124	-0.274523	0.275022	-0.224731
0.242657	0.0978391	-0.0229011	0.246634
-0.0229011	0.246634	-0.253605	0.0480468
0.484107	0.195191	-0.0456883	0.492042
-0.0456883	0.492042	-0.585947	0.0958546
-0.0229011	0.246634	-0.0456883	0.492042
0.242657	0.0978391	0.484107	0.195191
-0.253605	0.0480468	-0.505947	0.0958546
-0.022735	-0.254606	-0.121987	-0.264655
-0.121987	-0.264655	-0.146883	-0.0164339
-0.146883	-0.0164339	-0.0476311	-0.00647348
-0.0476311	-0.00647348	-0.027235	-0.254606
-0.163321	-0.110624	0.0646692	-0.120582
0.0646692	-0.120582	0.0516212	0.00348296
0.0516212	0.00348296	0.150873	0.0134414
0.150873	0.0134414	0.163321	-0.110624

Plain Text Tab Width: 8 Ln 23, Col 39 INS

视觉观测数据

run_simulation pub_IMU

```
void PubImageData()
{
    string sImage_file = sConfig_path + "MH_05_cam0.txt";

    cout << "I PubImageData start sImage_file: " << sImage_file << endl;

    ifstream fsImage;
    fsImage.open(sImage_file.c_str());
    if (!fsImage.is_open())
    {
        cerr << "Failed to open image file! " << sImage_file << endl;
        return;
    }

    std::string sImage_line;
    double dStampNSec;
    string sImgFileName;

    // cv::namedWindow("SOURCE IMAGE", CV_WINDOW_AUTOSIZE);
    while (std::getline(fsImage, sImage_line) && !sImage_line.empty())
    {
        std::istringstream ssImuData(sImage_line);
        ssImuData >> dStampNSec >> sImgFileName;
        // cout << "Image t : " << fixed << dStampNSec << " Name: " << sImgFileName << endl;
        string imagePath = sData_path + "cam0/data/" + sImgFileName;

        Mat img = imread(imagePath.c_str(), 0);
        if (img.empty())
        {
            cerr << "image is empty! path: " << imagePath << endl;
            return;
        }
        pSystem->PubImageData(dStampNSec / 1e9, img);
        // cv::imshow("SOURCE IMAGE", img);
        // cv::waitKey(0);
        usleep(50000 * nDelayTimes);
    }
    fsImage.close();
}
```



```
void PubImageData()
{
    string timeStamp_file = sData_path + "cam_pose_tum.txt";

    cout << "I PubImageData start sImage_file: " << timeStamp_file << endl;

    ifstream ftime;
    ftime.open(timeStamp_file.c_str());
    if (!ftime.is_open())
    {
        cerr << "Failed to open image file! " << timeStamp_file << endl;
        return;
    }

    std::string sTime_line;
    double dStampNSec;

    int cnt = 0;
    while (std::getline(ftime, sTime_line) && !sTime_line.empty())
    {
        std::istringstream ssTimeData(sTime_line);
        ssTimeData >> dStampNSec;
        string feature_points_path =
            sData_path + "/keyframe/" + "all_points_" + std::to_string(cnt) + ".txt";
        std::vector<cv::Point2f> feature_points;

        ReadFeatureFromFile(feature_points_path, feature_points);

        if (feature_points.empty())
        {
            cout << "image feature points is empty! path: " << feature_points_path << endl;
            return;
        }

        pSystem->PubFeatureData(dStampNSec, feature_points);

        usleep(50000 * nDelayTimes);
        cnt++;
    }
}
```


run_simulation pub_Camera

```
void System::PubImageData(double dStampSec, Nat img)  
{  
    if (!init feature)  
    {  
        // detect unstable camera stream  
        if (dStampSec - last_image_time > 1.0 || dStampSec < last_image_time)  
        {  
            last_image_time = dStampSec;  
            // frequency control  
            if (round(1.0 * pub_count / (dStampSec - first_image_time)) <= FREQ)  
            {  
                pub_count++;  
            }  
            else  
            {  
                // skip the first image; since no optical speed on first image  
                if (!init pub)  
                {  
                    // init pub  
                    m_buf.lock();  
                    feature_buf.push(feature_points);  
                    // cout << "feature buf size: " << feature_buf.size() << endl;  
                    m_buf.unlock();  
                    con.notify_one();  
                }  
            }  
        }  
    }  
    // cout << "PubImageData + " << dStampSec << endl;  
    trackerData[0].readImage(img, dStampSec);  
    for (unsigned int i = 0; i++)  
    {  
        if (PUB_THIS_FRAME)  
        {  
            pub_count++;  
            // cout << "PubImageData + " << dStampSec << endl;  
            shared_ptr<IMG_MSG> feature_points(new IMG_MSG());  
            feature_points->header = dStampSec;  
            vector<set<int>> hash_ids(NUM_OF_CAM);  
            for (int i = 0; i < NUM_OF_CAM; i++)  
            {  
                auto &un_pts = trackerData[i].cur_un_pts;  
                auto &cur_pts = trackerData[i].cur_pts;  
                auto &ids = trackerData[i].ids;  
                auto &pts_velocity = trackerData[i].pts_velocity;  
                for (unsigned int j = 0; j < ids.size(); j++)  
                {  
                    if (trackerData[i].track_cnt[j] > 1)  
                    {  
                        int p_id = ids[j];  
                        hash_ids[i].insert(p_id);  
                        double x = un_pts[j].x;  
                        double y = un_pts[j].y;  
                        double z = 1;  
                        feature_points->points.push_back(Vector3d(x, y, z));  
                        feature_points->id_of_point.push_back(p_id * NUM_OF_CAM + i);  
                        feature_points->w_of_point.push_back(cur_pts[j].x);  
                        feature_points->w_of_point.push_back(cur_pts[j].y);  
                        feature_points->velocity_x_of_point.push_back(pts_velocity[j].x);  
                        feature_points->velocity_y_of_point.push_back(pts_velocity[j].y);  
                    }  
                }  
            }  
            // skip the first image; since no optical speed on first image  
            if (!init pub)  
            {  
                // init pub  
                m_buf.lock();  
                feature_buf.push(feature_points);  
                // cout << "feature buf size: " << feature_buf.size() << endl;  
                m_buf.unlock();  
                con.notify_one();  
            }  
        }  
    }  
}
```



```
void System::PubFeatureData(double dStampSec,  
                             std::vector<cv::Point2f> &feature_points)  
{  
    if (!init feature)  
    {  
        // detect unstable camera stream  
        if (dStampSec - last_image_time > 1.0 || dStampSec < last_image_time)  
        {  
            last_image_time = dStampSec;  
            // frequency control  
            if (round(1.0 * pub_count / (dStampSec - first_image_time)) <= FREQ)  
            {  
                pub_count++;  
            }  
            else  
            {  
                // skip the first image; since no optical speed on first image  
                if (!init pub)  
                {  
                    // init pub  
                    m_buf.lock();  
                    feature_buf.push(feature_points);  
                    // cout << "feature buf size: " << feature_buf.size() << endl;  
                    m_buf.unlock();  
                    con.notify_one();  
                }  
            }  
        }  
    }  
    // cout << "PubFeatureData + " << dStampSec << endl;  
    trackerData[0].ReadFeature(feature_points, dStampSec);  
    for (unsigned int i = 0; i++)  
    {  
        if (PUB_THIS_FRAME)  
        {  
            pub_count++;  
            shared_ptr<IMG_MSG> feature_points(new IMG_MSG());  
            feature_points->header = dStampSec;  
            vector<set<int>> hash_ids(NUM_OF_CAM);  
            for (int i = 0; i < NUM_OF_CAM; i++)  
            {  
                auto &un_pts = trackerData[i].cur_un_pts;  
                auto &cur_pts = trackerData[i].cur_pts;  
                auto &ids = trackerData[i].ids;  
                auto &pts_velocity = trackerData[i].pts_velocity;  
                for (unsigned int j = 0; j < ids.size(); j++)  
                {  
                    if (trackerData[i].track_cnt[j] > 1)  
                    {  
                        int p_id = ids[j];  
                        hash_ids[i].insert(p_id);  
                        double x = un_pts[j].x;  
                        double y = un_pts[j].y;  
                        double z = 1;  
                        feature_points->points.push_back(Vector3d(x, y, z));  
                        feature_points->id_of_point.push_back(p_id * NUM_OF_CAM + i);  
                        feature_points->w_of_point.push_back(cur_pts[j].x);  
                        feature_points->w_of_point.push_back(cur_pts[j].y);  
                        feature_points->velocity_x_of_point.push_back(pts_velocity[j].x);  
                        feature_points->velocity_y_of_point.push_back(pts_velocity[j].y);  
                    }  
                }  
            }  
            // skip the first image; since no optical speed on first image  
            if (!init pub)  
            {  
                // init pub  
                m_buf.lock();  
                feature_buf.push(feature_points);  
                // cout << "feature buf size: " << feature_buf.size() << endl;  
                m_buf.unlock();  
                con.notify_one();  
            }  
        }  
    }  
}
```

run_simulation pub_Camera

```
void FeatureTracker::readImage(const cv::Mat &img, double _cur_time)
{
    cv::Mat img;
    TicToc t_r;
    cur_time = _cur_time;

    if (EQUALIZE)
    {
        ~
    }
    else
    {
        img = _img;
    }

    if (forw_img.empty())
    {
        ~
    }
    else
    {
        ~
    }

    forw_pts.clear();

    if (cur_pts.size() > 0)
    {
        TicToc t_o;
        vector<uchar> status;
        vector<float> err;
        cv::calcOpticalFlowPyrLK(cur_img, forw_img, cur_pts, forw_pts, status, err, cv::Size(21, 21), 3);

        for (int i = 0; i < int(forw_pts.size()); i++)
        {
            if (status[i] && !inBorder(forw_pts[i]))
            {
                status[i] = 0;
                reduceVector(prev_pts, status);
                reduceVector(cur_pts, status);
                reduceVector(forw_pts, status);
                reduceVector(ids, status);
                reduceVector(cur_un_pts, status);
                reduceVector(track_cnt, status);
                //ROS_DEBUG("temporal optical flow costs: %fms", t_o.toc());
            }
        }

        for (auto &n : track_cnt)
            n++;
    }
}
```

```
if (PUB_THIS_FRAME)
{
    rejectWithF();
    //ROS_DEBUG("set mask begins");
    TicToc t_m;
    setMask();
    //ROS_DEBUG("set mask costs %fms", t_m.toc());

    //ROS_DEBUG("detect feature begins");
    TicToc t_t;
    int n_max_cnt = MAX_CNT - static_cast<int>(forw_pts.size());
    if (n_max_cnt > 0)
    {
        if (mask.empty())
        {
            cout << "mask is empty " << endl;
        }
        if (mask.type() != CV_8UC1)
        {
            cout << "mask type wrong " << endl;
        }
        if (mask.size() != forw_img.size())
        {
            cout << "wrong size " << endl;
        }
        cv::goodFeaturesToTrack(forw_img, n_pts, MAX_CNT - forw_pts.size(), 0.01, MIN_DIST, mask);
    }
    else
    {
        n_pts.clear();
        //ROS_DEBUG("detect feature costs: %fms", t_t.toc());

        //ROS_DEBUG("add feature begins");
        TicToc t_a;
        addPoints();
        //ROS_DEBUG("selectFeature costs: %fms", t_a.toc());
    }

    prev_img = cur_img;
    prev_pts = cur_pts;
    prev_un_pts = cur_un_pts;
    cur_img = forw_img;
    cur_pts = forw_pts;
    undistortedPoints();
    prev_time = cur_time;
}
```

run_simulation pub_Camera

```
void FeatureTracker::ReadFeature(
    const std::vector<cv::Point2f> &feature_points, double _cur_time)
{
    TicToc t_r;
    cur_time = _cur_time;

    forw_pts.clear();

    auto phnhole_cam = dynamic_cast<PinholeCamera *>(m_camera.get());
    PinholeCamera::Parameters parameters = phnhole_cam->getParameters();

    if (cur_pts.size() > 0)
    {
        for (auto feature : feature_points)
        {
            cv::Point2f pixel_feature;
            pixel_feature.x = parameters.fx() * feature.x + parameters.cx();
            pixel_feature.y = parameters.fy() * feature.y + parameters.cy();
            forw_pts.push_back(pixel_feature);
        }
        std::vector<uchar> status(feature_points.size(), 1);

        reduceVector(prev_pts, status);
        reduceVector(cur_pts, status);
        reduceVector(forw_pts, status);
        reduceVector(ids, status);
        reduceVector(cur_un_pts, status);
        reduceVector(track_cnt, status);
    }

    for (auto &n : track_cnt)
        n++;
```

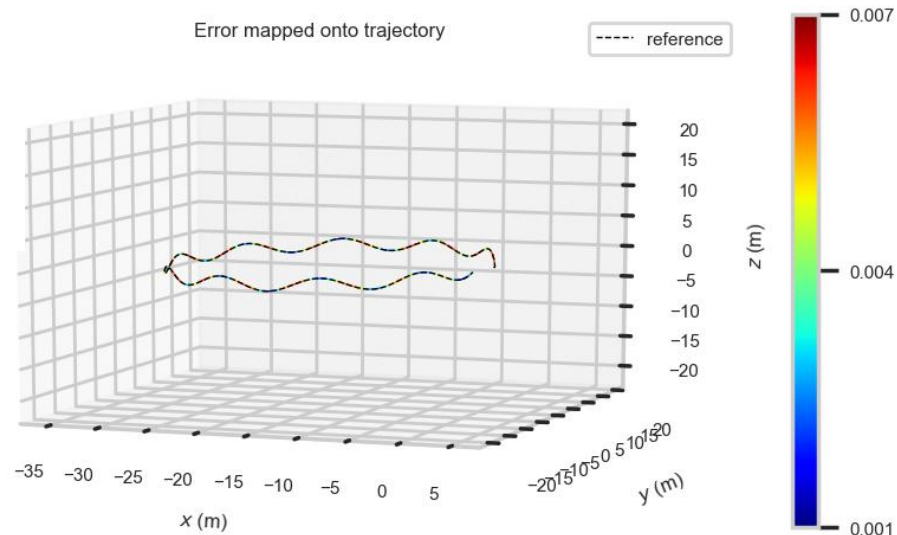
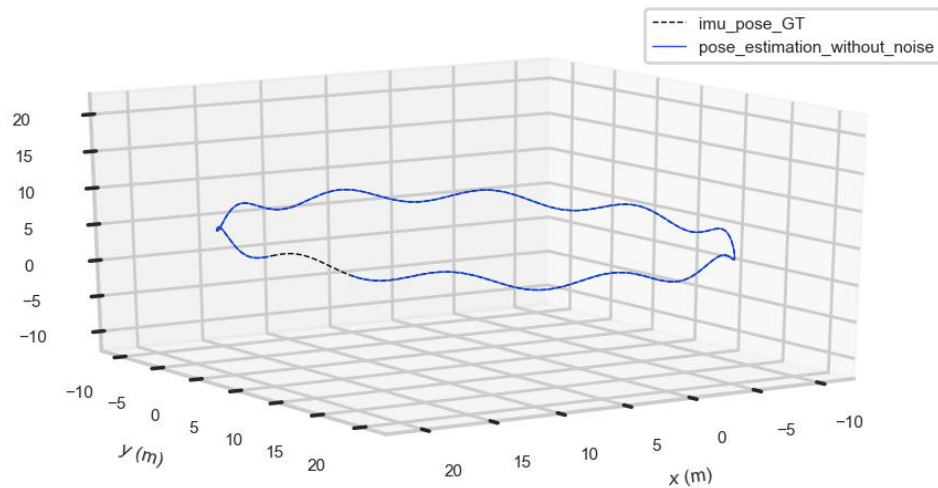
```
if (PUB_THIS_FRAME)
{
    TicToc t_m;
    // ROS_DEBUG("detect feature begins");
    int n_max_cnt = feature_points.size() - static_cast<int>(forw_pts.size());
    if (n_max_cnt > 0)
    {
        for (auto it = 0; it < feature_points.size(); it++)
        {
            cv::Point2f pixel_feature;
            pixel_feature.x =
                parameters.fx() * feature_points[it].x + parameters.cx();
            pixel_feature.y =
                parameters.fy() * feature_points[it].y + parameters.cy();
            n_pts.push_back(pixel_feature);
        }
    }
    else
    {
        n_pts.clear();
    }
    TicToc t_t;

    // ROS_DEBUG("add feature begins");
    TicToc t_a;
    addPoints();
    // ROS_DEBUG("selectFeature costs: %fms", t_a.toc());
}

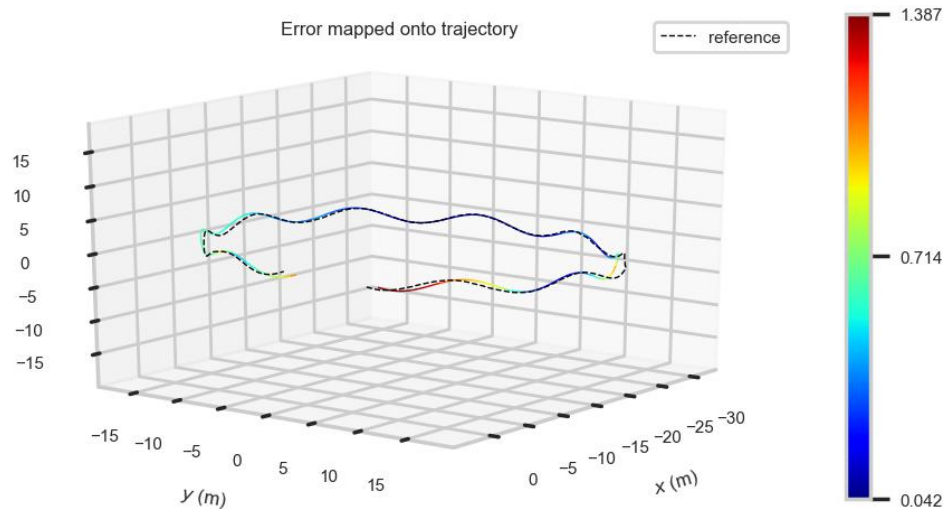
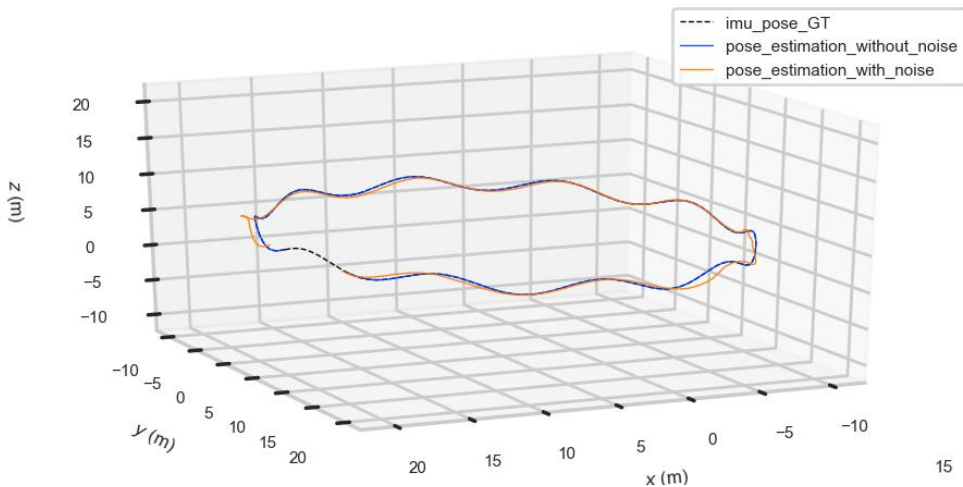
prev_pts = cur_pts;
prev_un_pts = cur_un_pts;

cur_pts = forw_pts;
undistortedPoints();
prev_time = cur_time;
```


使用无噪声IMU测量数据

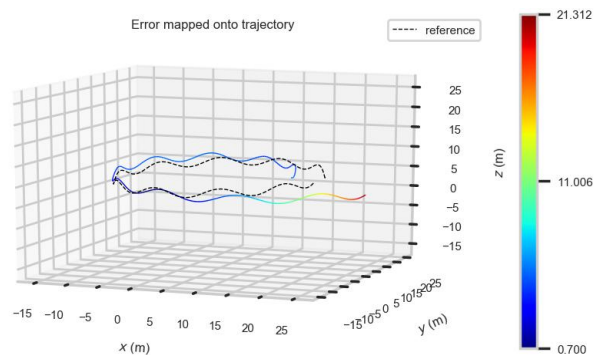


使用带噪声IMU测量数据

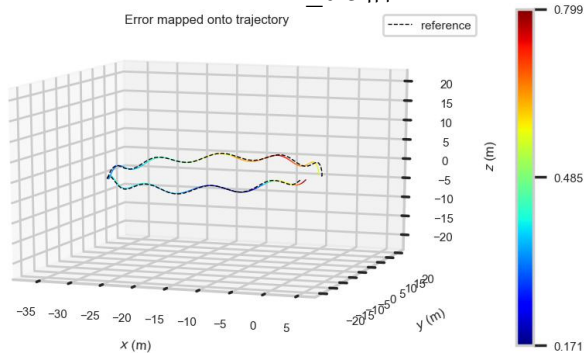


设定噪声的不同config参数

noise_0



noise_0.5倍



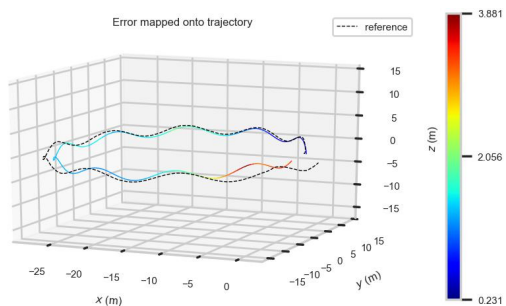
gyro_noise = 0.015

gyro_noise = 0.019

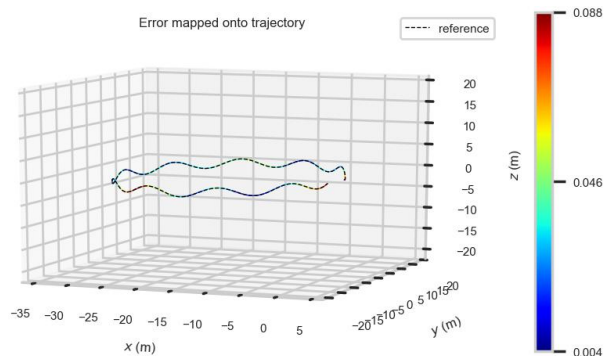
gyro_bias_sigma = 1.0e-5;

acc_bias_sigma = 0.0001;

noise_2倍



noise_10倍



感谢各位聆听 !
Thanks for Listening !

