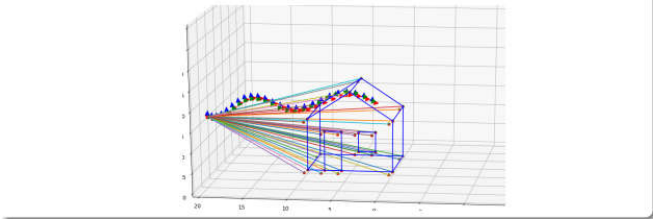
第七次作业

作业

❶ 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。）



解:

分为两部分：仿真数据准备和开源的 vins 系统

1. 数据准备

工具采用的是 **vio_data_simulation-master**

a.Param.h 参数的设定为

```
class Param{

public:

    Param();

    // time
    int imu_frequency = 200;
    int cam_frequency = 30;
    double imu_timestep = 1./imu_frequency;
    double cam_timestep = 1./cam_frequency;
    double t_start = 0.;
    double t_end = 20;  //  20 s

    // noise
    double gyro_bias_sigma = 1.0e-5;
    double acc_bias_sigma = 0.0001;

    double gyro_noise_sigma = 0.015;    // rad/s
    double acc_noise_sigma = 0.019;      //  m/(s^2)

    double pixel_noise = 1;              // 1 pixel noise

    // cam f
    double fx = 460;
    double fy = 460;
    double cx = 255;
    double cy = 255;
    double image_w = 640;
    double image_h = 640;


    // 外参数
    Eigen::Matrix3d R_bc;   // cam to body
    Eigen::Vector3d t_bc;     // cam to body

};
```

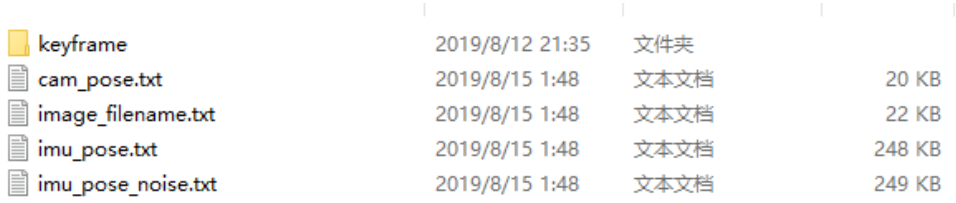
外参：

```
Param::Param()
{
    Eigen::Matrix3d R;   // 把body坐标系朝向旋转一下,得到相机坐标系，好让它看到landmark,
    // 相机朝着轨迹里面看， 特征点在轨迹外部， 这里我们采用这个
    R << 0, 0, -1,
         -1, 0, 0,
         0, 1, 0;
    R_bc = R;
    t_bc = Eigen::Vector3d(0.05,0.04,0.03);

}
```

b.文件格式

文件如



keyframe	2019/8/12 21:35	文件夹	
cam_pose.txt	2019/8/15 1:48	文本文档	20 KB
image_filename.txt	2019/8/15 1:48	文本文档	22 KB
imu_pose.txt	2019/8/15 1:48	文本文档	248 KB
imu_pose_noise.txt	2019/8/15 1:48	文本文档	249 KB

1）imu_pose.txt 与 imu_pose_noise.txt: （imu 数据）

每一行：时间戳，陀螺仪向量，加速度计向量

2）image_filename.txt（图像名文件）

每一行：时间戳，一帧图像名字

3）cam_pose.txt(图像实际的姿态)

每一行：x,y,z

c.最后系统的配置文件为：

myslam_config.yaml

```yaml
%YAML:1.0

#common parameters
imu_topic: "/imu0"
image_topic: "/cam0/image_raw"
output_path: "../output"

#camera calibration
model_type: PINHOLE
camera_name: camera
image_width: 640
image_height: 640
distortion_parameters:
   k1: -2.917e-03
   k2: 8.228e-03
   p1: 5.333e-05
   p2: -1.578e-04
projection_parameters:
   fx: 4.60e+02
   fy: 4.60e+02
   cx: 2.55e+02
   cy: 2.55e+02

# Extrinsic parameter between IMU and Camera.
estimate_extrinsic: 1   # 0  Have an accurate extrinsic parameters. We will trust the follow
                        # 1  Have an initial guess about extrinsic parameters. We will optim
                        # 2  Don't know anything about extrinsic parameters. You don't need
#If you choose 0 or 1, you should write down the following matrix.
#Rotation from camera frame to imu frame, imu^R_cam
extrinsicRotation: !!opencv-matrix
   rows: 3
   cols: 3
   dt: d
   data: [ 0, 0, -1,
           -1, 0, 0,
           0, 1, 0]
#Translation from camera frame to imu frame, imu^T_cam
extrinsicTranslation: !!opencv-matrix
   rows: 3
   cols: 1
   dt: d
   data: [0.05,0.04,0.03]

#feature traker paprameters
max_cnt: 150            # max feature number in feature tracking
min_dist: 30            # min distance between two features
freq: 10                # frequence (Hz) of publish tracking result. At least 10Hz for good
F_threshold: 1.0        # ransac threshold (pixel)
show_track: 1           # publish tracking image as topic
```

```
equalize: 1            # if image is too dark or light, trun on equalize to find enough f
fisheye: 0             # if using fisheye, trun on it. A circle mask will be loaded to re

#optimization parameters
max_solver_time: 0.04  # max solver itration time (ms), to guarantee real time
max_num_iterations: 8   # max solver itrations, to guarantee real time
keyframe_parallax: 10.0 # keyframe selection threshold (pixel)

#imu parameters       The more accurate parameters you provide, the better performance
acc_n: 0.019         # accelerometer measurement noise standard deviation. #0.2   0.04
gyr_n: 0.015        # gyroscope measurement noise standard deviation.     #0.05  0.004
acc_w: 0.0001         # accelerometer bias random work noise standard deviation.  #0.02
gyr_w: 1.0e-5       # gyroscope bias random work noise standard deviation.     #4.0e-5
g_norm: 9.81007     # gravity magnitude

#loop closure parameters
loop_closure: 0                    # start loop closure
load_previous_pose_graph: 0        # load and reuse previous pose graph; load from 'pose_g
fast_relocalization: 0             # useful in real-time and large project
pose_graph_save_path: "/home/vslam/VIO_Tutorial/7.vins_system/vins_sys_code/data/" # save

#unsynchronization parameters
estimate_td: 1                      # online estimate time offset between camera and imu
td: 0.0                             # initial value of time offset. unit: s. readed image

#rolling shutter parameters
rolling_shutter: 0                  # 0: global shutter camera, 1: rolling shutter camera
rolling_shutter_tr: 0               # unit: s. rolling shutter read out time per frame (fr

#visualization parameters
save_image: 0                   # save image in pose graph for visualization prupose; you
visualize_imu_forward: 0        # output imu forward propogation to achieve low latency an
visualize_camera_size: 0.4      # size of camera marker in RVIZ
```

2. 开源 vins

a.imu 数据输入

```cpp
//将IMU数据载入system
void PubImuData()
{
    string sImu_data_file = sData_path + imu_data;
    cout << "1 PubImuData start sImu_data_filea: " << sImu_data_file << endl;
    ifstream fsImu;
    fsImu.open(sImu_data_file.c_str());
    if (!fsImu.is_open())
    {
        cerr << "Failed to open imu file! " << sImu_data_file << endl;
        return;
    }

    std::string sImu_line;
    double dStampNSec = 0.0;
    Vector3d vAcc;
    Vector3d vGyr;

    double lastTime;

    while (std::getline(fsImu, sImu_line) && !sImu_line.empty()) // read imu data  every line
    {
        std::istringstream ssImuData(sImu_line);
        ssImuData >> dStampNSec >> vGyr.x() >> vGyr.y() >> vGyr.z() >> vAcc.x() >> vAcc.y() >> vAcc.z();
        pSystem->PubImuData(dStampNSec, vGyr, vAcc);

        lastTime = dStampNSec;
        usleep(4500 * nDelayTimes);
    }
    fsImu.close();
}
```

b.图像数据输入

```cpp
//将image载入system
void PubImageData()
{
    string sImage_file = sData_path + "image_filename.txt";

    cout << "1 PubImageData start sImage_file: " << sImage_file << endl;

    ifstream fsImage;
    fsImage.open(sImage_file.c_str()); //所有图像特征的文件名列表  600个
    if (!fsImage.is_open())
    {
        cerr << "Failed to open image file! " << sImage_file << endl;
        return;
    }

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

    while (std::getline(fsImage, sImage_line) && !sImage_line.empty()) // 读取每一行，每一行都是包含一副图片所有的特征点文
    {
        std::istringstream ssImuData(sImage_line);
        ssImuData >> dStampNSec >> sImgFileName;

        string imagePath = sData_path + sImgFileName;

        //读取每个camera提取的特征点
        ifstream featuresImage;
        featuresImage.open(imagePath.c_str());
        if (!featuresImage.is_open())
        {
            cerr << "Failed to open features file! " << imagePath << endl;
            return;
        }
        std::string featuresImage_line;
        std::vector<int> feature_id;
        int ids = 0;

        std::vector<Vector2d> featurePoint;
        std::vector<Vector2d> observation_feature;
        std::vector<Vector2d> featureVelocity;
        static double lastTime;
        static std::vector<Vector2d> lastfeaturePoint(50);

        cv::Mat show_img(640, 640, CV_8UC3, cv::Scalar(0, 0, 0));

        while (std::getline(featuresImage, featuresImage_line) && !featuresImage_line.empty())
        {
            {
                Vector2d current_featurePoint;        //归一化相机坐标
                Vector3d current_observation_feature; //像素坐标
                Vector2d current_featureVelocity;    //归一化相机坐标下点的运动速度

                Eigen::Matrix3d K;
                K << 640.0, 0, 255,
                    0, 640.0, 255,
                    0, 0, 0;

                std::istringstream ssfeatureData(featuresImage_line);
                ssfeatureData >> current_featurePoint.x() >> current_featurePoint.y();
                featurePoint.push_back(current_featurePoint);
                feature_id.push_back(ids);

                current_featureVelocity.x() = (current_featurePoint.x() - lastfeaturePoint[ids].x()) / (dStampNSec - lastTime);
                current_featureVelocity.y() = (current_featurePoint.y() - lastfeaturePoint[ids].y()) / (dStampNSec - lastTime);
                featureVelocity.push_back(current_featureVelocity);

                current_observation_feature = Vector3d(current_featurePoint.x(), current_featurePoint.y(), 1);
                current_observation_feature = K * current_observation_feature;

                observation_feature.push_back(Vector2d(current_observation_feature.x(), current_observation_feature.y()));

                //可视化图像
                cv::circle(show_img, cv::Point2f(current_observation_feature.x(), current_observation_feature.y()), 2, cv::Scalar(255, 225, 255),
                ids++;
            }
            featuresImage.close();
            lastTime = dStampNSec;
            lastfeaturePoint = featurePoint;
            pSystem->PubFeatureData(dStampNSec, feature_id, featurePoint, observation_feature, featureVelocity); //带时间戳的feature point数据载入系

            //是否可视化追踪过程
            if (1)
            {
                cv::namedWindow("IMAGE", CV_WINDOW_AUTOSIZE);
                cv::imshow("IMAGE", show_img);
                cv::waitKey(1);
            }
            usleep(50000 * nDelayTimes);
        }
    }
```

c.真实数据的显示（groudturth）

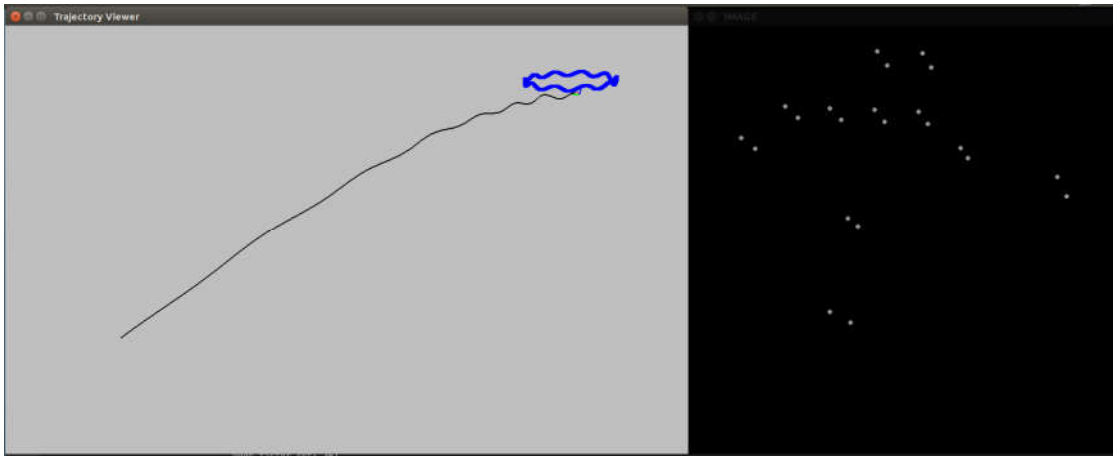
```cpp
void PubRealData()
{
    string sRealData_file = sData_path + "cam_pose.txt";
    ifstream fsPose;
    fsPose.open(sRealData_file.c_str());
    if (!fsPose.is_open())
    {
        cerr << "Failed to open realPose file! " << sRealData_file << endl;
        return;
    }
    std::string sPose_line;
    double temp;
    Vector3d current_pose;__
    while (std::getline(fsPose, sPose_line) && !sPose_line.empty())
    {
        std::istringstream ssPoseData(sPose_line);
        ssPoseData >> temp >> temp >> temp >> temp >> temp >> current_pose.x() >> current_pose.y() >> current_pose.z() >> temp >> temp >> temp >> temp

        pSystem->real_poses.push_back(current_pose);
        usleep(50000 * nDelayTimes);
    }
}
```

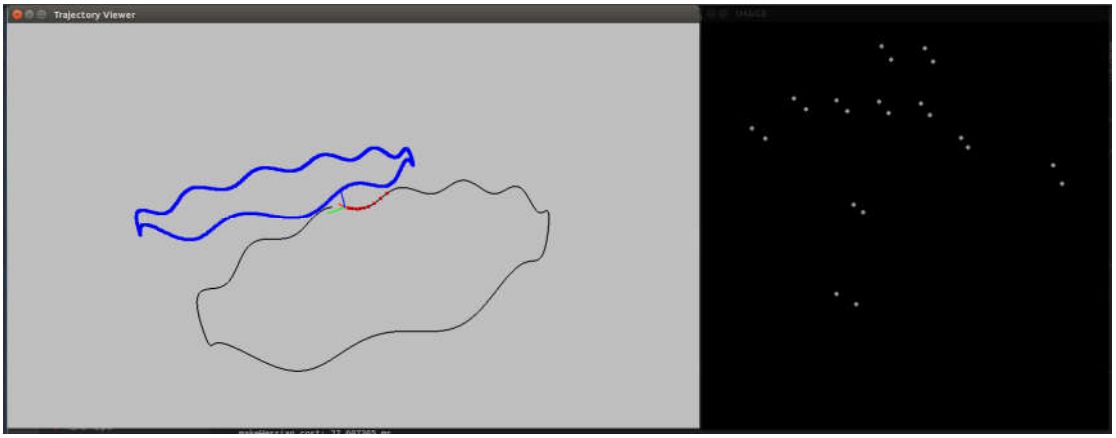
d．可视化处理

```cpp
// points
if (estimator.solver_flag == Estimator::SolverFlag::NON_LINEAR)
{
    glPointSize(5);
    glBegin(GL_POINTS);
    for (int i = 0; i < WINDOW_SIZE + 1; ++i)
    {
        Vector3d p_wi = estimator.Ps[i];
        glColor3f(1, 0, 0);
        glVertex3d(p_wi[0], p_wi[1], p_wi[2]);
    }
    glEnd();
}
  // show the real pose
{
    glPointSize(5);
    glBegin(GL_POINTS);
    for (size_t i = 0; i < real_poses.size(); ++i)
    {
        Vector3d p_wi = real_poses[i];
        glColor3f(0, 0, 1);
        glVertex3d(p_wi[0], p_wi[1], p_wi[2]);
    }
    glEnd();
}
```

有噪声的数据运行结果如图（dift）：



无噪声的数据运行结果如图：


Trajectory Viewer