# 一. 使用 VINS 系统跑仿真数据

将第二讲中的仿真数据集接入课程中的 VINS 系统，并运行处轨迹结果。

1. 无噪声参数

答:

主要的不同就是: 跑数据集的 VINS 是通过对图片的处理提取出特征点，然后送到后面的函数去计算。而仿真中的特征点是直接得到的，然后我们又通过相机模型之类的得到一系列的点归一化坐之类的。因此在实际使用中，需要把系统读取数据的函数修改了。仿真数据要求用第二章的数据，具体形式如图 1 所示，是贺博画的一个房子，由几个特征点和这些点的连线组成。VINS 系统的轨迹就是绕着房子走了个椭圆 (X-Y 平面上投影是椭圆,Z 轴上是正弦波)。NFT 抽象派大师了, 属于是。

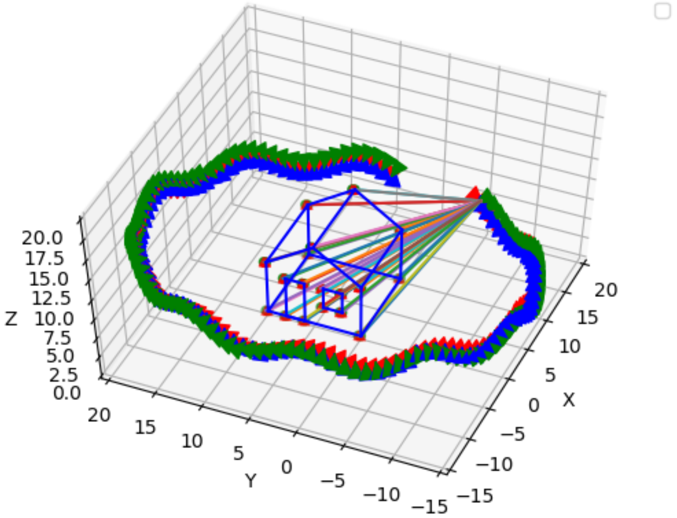


图 1: ratio vary with sigma

这周比较忙，先抄前面一个叫高洪臣的兄弟的作业用一下。

主要修改的地方有三处

- 添加 config 文件 sim_config.yaml 和 run_sim.cpp 文件 (内容和 run_euroc.cpp 一样)，只不过其中调用的函数不一样。他这里面把里面的函数重载了，重载的两个函数如下，它们的参数改变，然后就可以接受仿真数据了。

- 重载函数 System::PubImageData();
- 重载函数 Featureracker::readImage();

下面分别是两个重载函数的内容，第一个函数把 PubImageData() 的第二个参数改成了 string filename，这样就可以从仿真数据中读点了。第二个函数的修改也是一样。

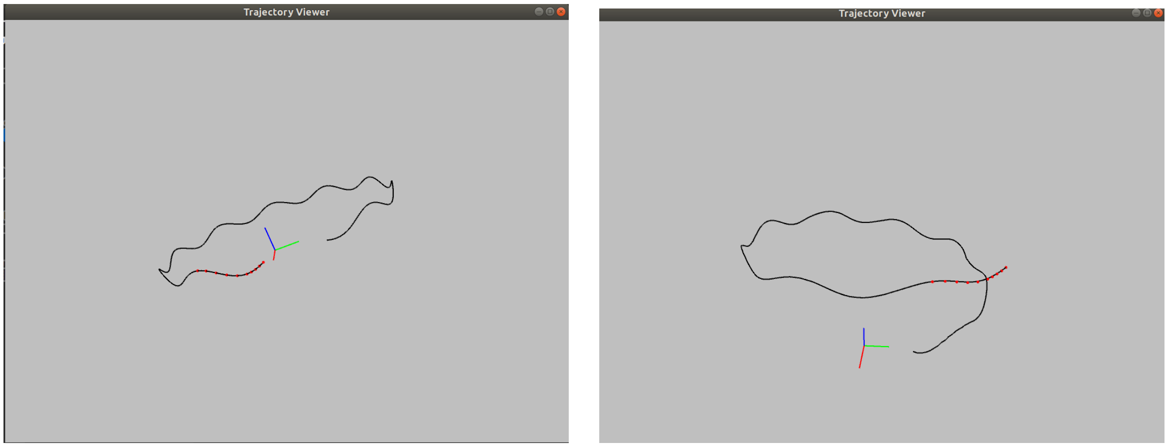
```
void System::PubImageData(double dStampSec, std::string filename)
{
    if (!init_feature)
    {
        cout << "1 PubImageData skip the first detected feature,
        which doesn't contain optical flow speed" << endl;
        init_feature = 1;
        return;
    }
    if (first_image_flag)
    {
        cout << "2 PubImageData first_image_flag" << endl;
        first_image_flag = false;
        first_image_time = dStampSec;
        last_image_time = dStampSec;
        return;
    }
    // detect unstable camera stream
    if (dStampSec - last_image_time > 1.0 || dStampSec < last_image_time)
    {
        cerr << "3 PubImageData image discontinue!
        reset the feature tracker!" << endl;
        first_image_flag = true;
        last_image_time = 0;
        pub_count = 1;
        return;
    }
```

```
last_image_time = dStampSec;
// frequency control
if (round(1.0 * pub_count / (dStampSec - first_image_time)) <= FREQ)
{
    PUB_THIS_FRAME = true;
    // reset the frequency control
    if (abs(1.0 * pub_count / (dStampSec - first_image_time) - FREQ)
    < 0.01 * FREQ)
    {
        first_image_time = dStampSec;
        pub_count = 0;
    }
}
else
{
    PUB_THIS_FRAME = false;
}
TicToc t_r;
// cout << "3 PubImageData t : " << dStampSec << endl;
trackerData[0].readImage(filename, dStampSec);

for (unsigned int i = 0;; i++)
{
    bool completed = false;
    completed |= trackerData[0].updateID(i);

    if (!completed)
        break;
}
if (PUB_THIS_FRAME)
{
    pub_count++;
```

```
        shared_ptr<IMG_MSG> feature_points(new IMG_MSG());
        feature_points->header = dStampSec;
        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;

            for (unsigned int j = 0; j < un_pts.size(); j++)
            {
                //if (trackerData[i].track_cnt[j] > 1)
                {
                    // int p_id = ids[j];
                    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(j * NUM_OF_CAM + i);
                    feature_points->u_of_point.push_back(cur_pts[j].x);
                    feature_points->v_of_point.push_back(cur_pts[j].y);
                    feature_points->velocity_x_of_point.push_back(0);
                    feature_points->velocity_y_of_point.push_back(0);
                }
            }
            // skip the first image; since no optical speed on frist image
            if (!init_pub)
            {
                cout << "4 PubImage init_pub skip the first image!" << endl;
                init_pub = 1;
            }
            else
            {
                m_buf.lock();
                feature_buf.push( feature_points );
                // cout << ”5 PubImage t : ” << fixed << feature_points->header
                //      << ” feature_buf size: ” << feature_buf.size() << endl;
                m_buf.unlock();
                con.notify_one();
            }
        }
    }
}
```

```
void FeatureTracker::readImage(std::string filename,double _cur_time) {
    cur_time = _cur_time;

    ids.clear();
    cur_un_pts.clear();
    track_cnt.clear();

        ifstream fsFeatures;
        fsFeatures.open(filename.c_str());
        if (!fsFeatures.is_open())
        {
                cerr << "Failed to open fsFeatures file! " << filename << endl;
                return;
        }
        std::string sFeature_line;
        double dStampNSec = 0.0;
    Eigen::Vector4d p;
    Eigen::Vector2d f;
        while (std::getline(fsFeatures, sFeature_line) && !sFeature_line.empty())
        {
                std::istringstream ssFeatureData( sFeature_line );
                ssFeatureData >> p(0) >> p(1) >> p(2) >> p(3) >> f(0) >> f(1);

        cur_un_pts.push_back(cv::Point2f(f(0), f(1)));

        float u = FOCAL_LENGTH * f(0) + COL / 2.0;
        float v = FOCAL_LENGTH * f(1) + ROW / 2.0;
        cur_pts.push_back(cv::Point2f(u,v));

        ids.push_back(-1);
        track_cnt.push_back(1);
        }
        fsFeatures.close();

    prev_time = cur_time;
}
```

然后分别运行有无噪声的 IMU 仿真数据，得到的结果如图 2 所示。其中图 2.(a) 是没有噪声的数据，可以看出，最后还是一个椭圆，而且能够形成一个圈，图 2(b) 中则是有噪声的数据，可以看到预测轨迹明显有偏差了。这周属实有点没时间，先把这个版本提交了，后面整理资料的时候再搞一个完整一点的



(a) with no noise

(b) with noise

图 2: VINS pose estimation