SOUSIC-第 2 章作业

1. 设置 IMU 仿真代码中的不同参数，生成 Allan 方差标定曲线

1.1 生成数据

- 仿真设置的 noise 和 bias

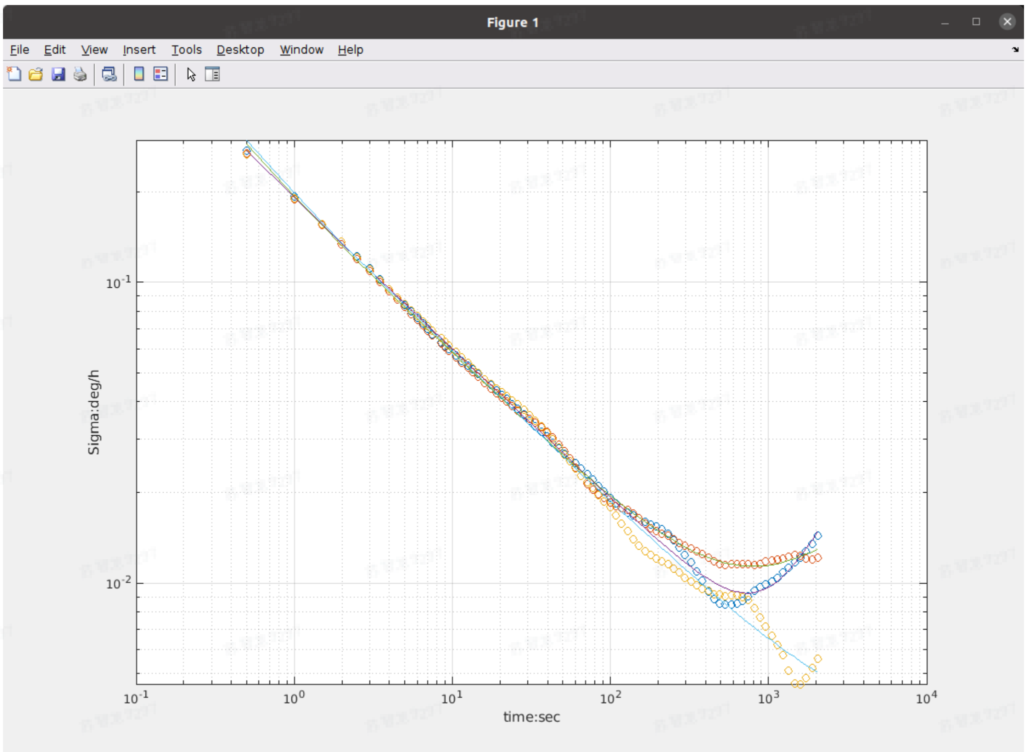
```
// noise
double gyro_bias_sigma = 0.00005;
double acc_bias_sigma = 0.0005;

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

double gyro_noise_sigma = 0.015;    // rad/s * 1/sqrt(hz)
double acc_noise_sigma = 0.019;      //  m/(s^2) * 1/sqrt(hz)
```

- 生成 imu.bag

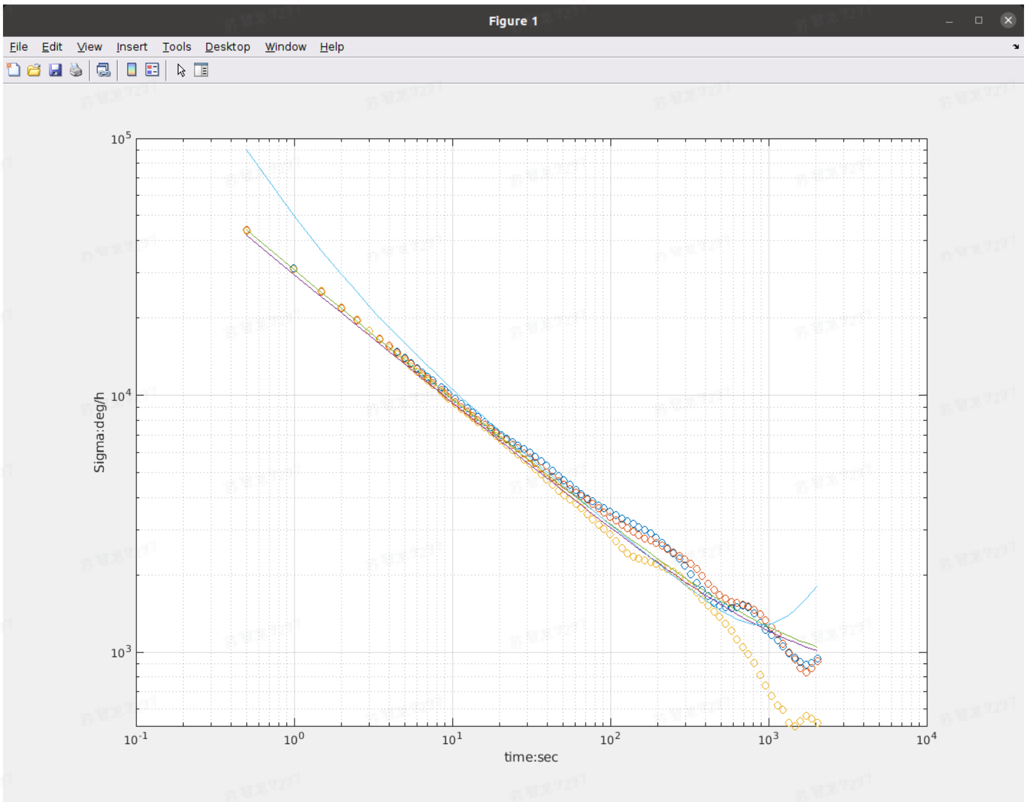
```
root@zhilong-ubuntu:/home/zhilong/Documents/vio_homework/VIO_Hw/ch2/course2_hw_new/vio_sim_ws# rosrun vio_data_simulation vio_data_simulation_node
Start generate data, please waiting...
Done, save to /root/imu.bag
root@zhilong-ubuntu:/home/zhilong/Documents/vio_homework/VIO_Hw/ch2/course2_hw_new/vio_sim_ws#
```

1.2 imu_utils 标定

- acc 曲线



- gyr 曲线



- 使用 imu_utils 生成的 yaml 文件

sim_imu_imu_param.yaml [Read-Only]
~/Documents/vio_homework/VIO_Hw/ch2/co..._sim_ws/src/imu_utils/analysis_imu_data


```yaml
%YAML:1.0
---
type: IMU
name: sim_imu
Gyr:
   unit: " rad/s"
   avg-axis:
      gyr_n: 2.5269485695121152e-01
      gyr_w: 4.8477797168896648e-03
   x-axis:
      gyr_n: 2.0300116403561616e-01
      gyr_w: 4.8477797168896648e-03
   y-axis:
      gyr_n: 2.1142475756573628e-01
      gyr_w: 4.8477797168896648e-03
   z-axis:
      gyr_n: 3.4365864925228212e-01
      gyr_w: 4.8477797168896648e-03
Acc:
   unit: " m/s^2"
   avg-axis:
      acc_n: 2.7440761325972324e-01
      acc_w: 8.5627907101857859e-03
   x-axis:
      acc_n: 2.7081807634147342e-01
      acc_w: 9.2541949283251844e-03
   y-axis:
      acc_n: 2.7259165354816717e-01
      acc_w: 1.1382110874508673e-02
   z-axis:
      acc_n: 2.7981310988952923e-01
      acc_w: 5.0520663277235003e-03
```


YAML Tab Width: 8 Ln 1, Col 1 INS

- 计算得到的白噪声 gyro_noise_sigma 和 acc_noise_sigma 与设定的值（gyr: 0.15, acc:0.19）相差不大，但是 bias 随机游走就不准了

```python
from math import sqrt

gyr_noise = 2.5269485695121152e-01
acc_noise = 2.7440761325972324e-01
gyr_bias = 4.8477797168896648e-03
acc_bias = 8.5627907101857859e-03

gyr_noise_sigma = gyr_noise * (1/sqrt(200))
acc_noise_sigma = acc_noise * (1/sqrt(200))
gyr_bias_sigma = gyr_bias * sqrt(200)
acc_bias_sigma = acc_bias * sqrt(200)

print('gyr_noise_sigma: ', gyr_noise_sigma)
print('acc_noise_sigma: ', acc_noise_sigma)
print('gyr_bias_sigma', gyr_bias_sigma)
print('acc_bias_sigma', acc_bias_sigma)
```

```
gyr_noise_sigma:  0.017868224692116626
acc_noise_sigma:  0.019403548414516587
gyr_bias_sigma 0.06855795823022567
acc_bias_sigma 0.12109614754107086
[Finished in 19ms]
```

1.3 kalibar_allan

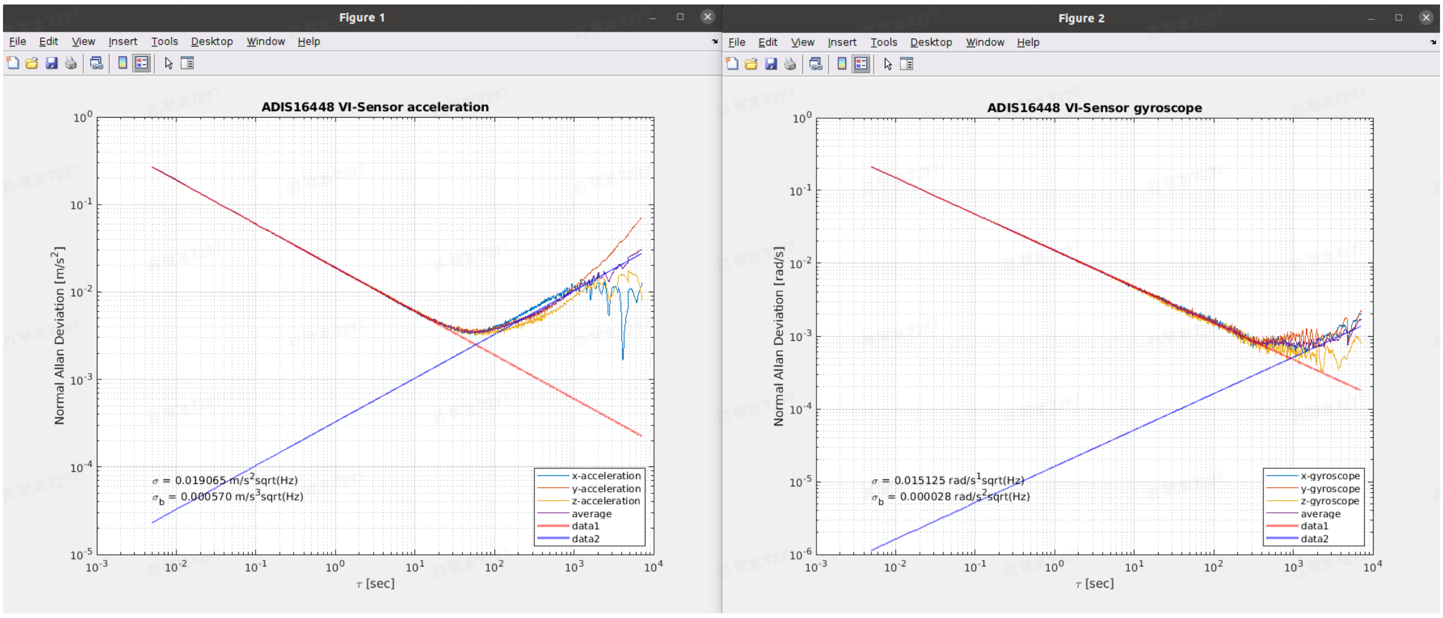
- imu.bag 转为 .mat 文件

```
root@zhilong-ubuntu:/home/zhilong/Documents/vio_homework/VIO_Hw/ch2/course2_hw_new/vio_sim_ws# rosrun bagconvert bagconvert /root/imu.bag imu
[ INFO] [1673063894.743341724]: Starting up
[ INFO] [1673063895.044160235]: BAG Path is: /root/imu.bag
[ INFO] [1673063895.044177904]: MAT Path is: /root/imu.mat
[ INFO] [1673063895.044181453]: Reading in rosbag file...
[ INFO] [1673063896.386286647]: Done processing bag
[ INFO] [1673063899.392771153]: Done processing IMU data
```

- Matlab 计算结果：白噪声与设定的值（gyr: 0.15, acc:0.19）相差很小，bias 随机游走也比 imu_utils 好很多

```
>> SCRIPT_process_results
=> opening the mat file.
=> plotting accelerometer.
Warning: MATLAB has disabled some advanced graphic
information, click here.
tau = 1.00 | tauid1 = 1089
h_fit1 slope = -0.5000 | y-intercept = -3.9599
h_fit2 slope = 0.5000 | y-intercept = -8.0175
tau = 2.99 | tauid2 = 1201
=> plotting gyroscope.
tau = 1.00 | tauid1 = 1089
h_fit1 slope = -0.5000 | y-intercept = -4.1914
h_fit2 slope = 0.5000 | y-intercept = -11.0206
tau = 2.99 | tauid2 = 1201
=> final results
accelerometer_noise_density = 0.01906498
accelerometer_random_walk   = 0.00056998
gyroscope_noise_density     = 0.01512469
gyroscope_random_walk       = 0.00002829
>>
```

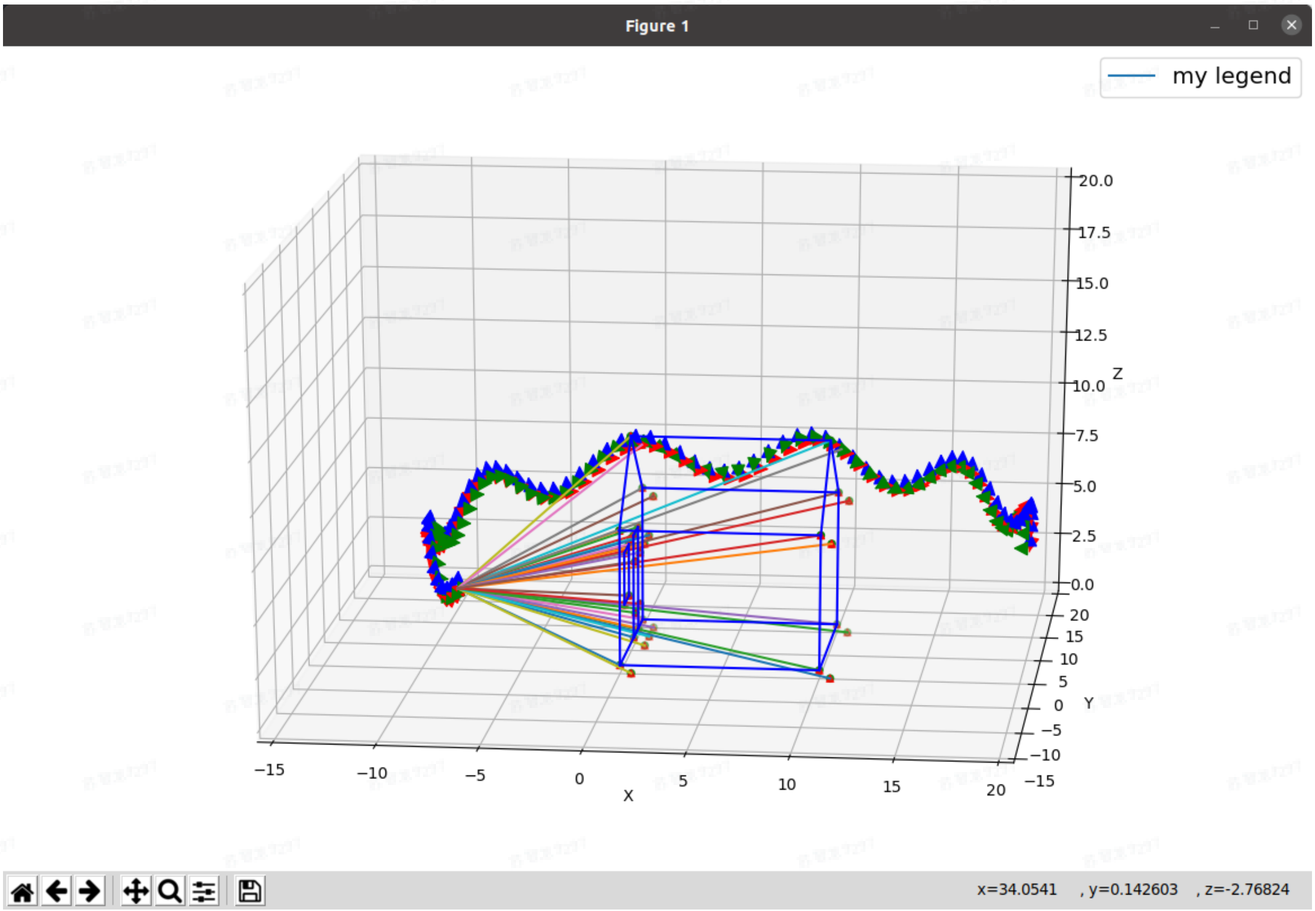
- Allan 方差标定曲线



2. 将 IMU 仿真代码中的欧拉积分替换成中值积分

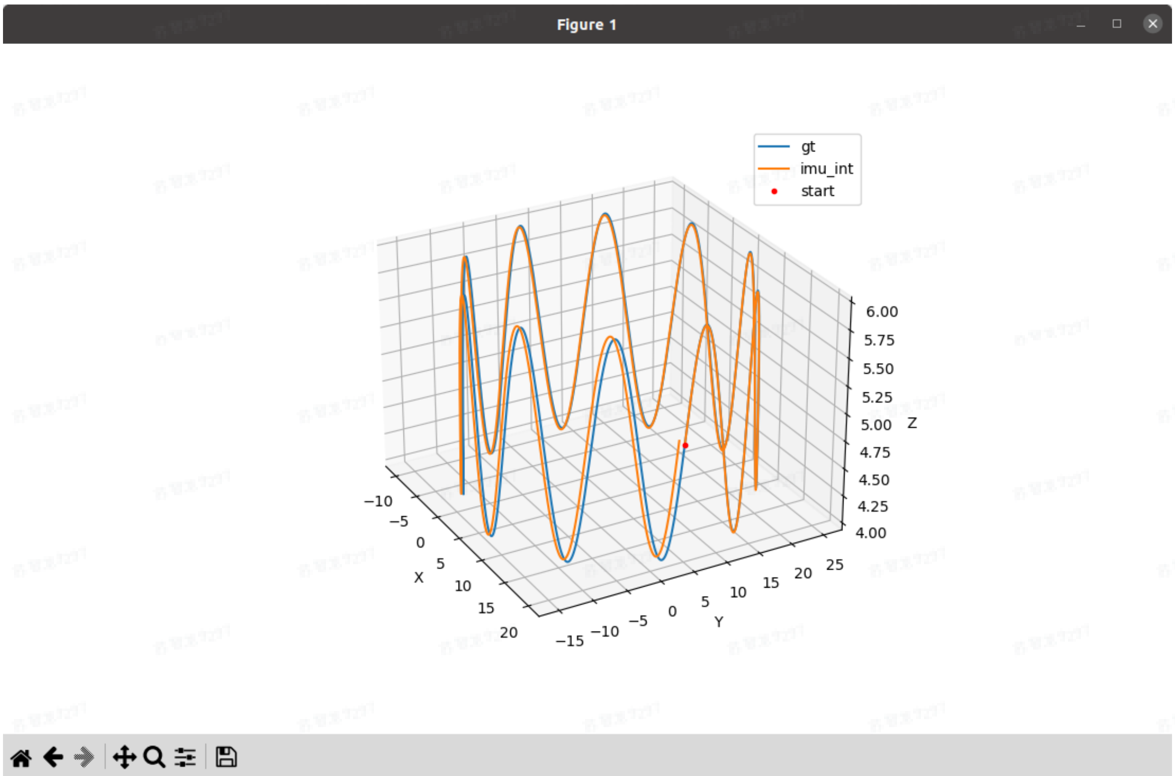
2.1 运行 draw_points.py

可以画出 IMU 的运动轨迹：



2.2 欧拉法

使用欧拉法积分，轨迹和真值有一些偏移



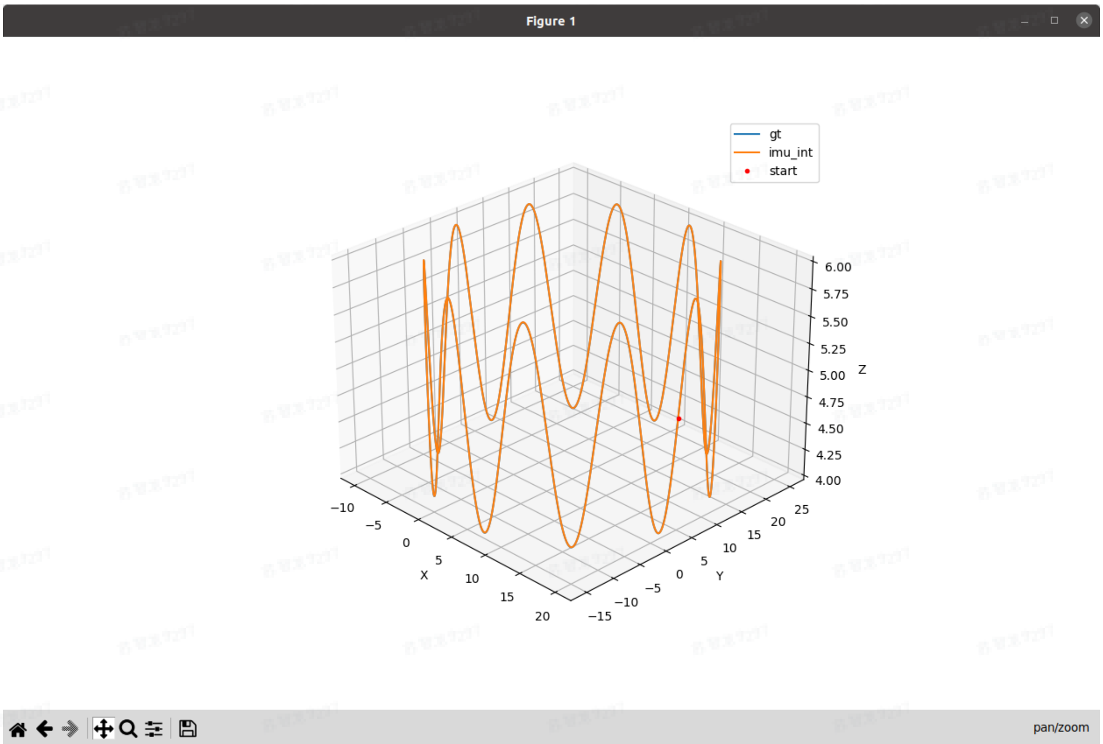
2.2 中值法

- 中值法代码修改

home > zhilong > Documents > vio_homework > VIO_Hw > ch2 > course2_hw_new > vio_data_simulation > src > imu.cpp > testImu(std::string, std::string)

```cpp
//用来验证数据以及模型的有效性。
void IMU::testImu(std::string src, std::string dist)
{
    std::vector<MotionData>imudata;
    LoadPose(src,imudata);

    std::ofstream save_points;
    save_points.open(dist);

    double dt = param_.imu_timestep;
    Eigen::Vector3d Pwb = init_twb_;              // position :    from  imu measurements
    Eigen::Quaterniond Qwb(init_Rwb_);            // quaternion:  from imu measurements
    Eigen::Vector3d Vw = init_velocity_;          // velocity  :   from imu measurements
    Eigen::Vector3d gw(0,0,-9.81);    // ENU frame
    Eigen::Vector3d temp_a;
    Eigen::Vector3d theta;
    for (int i = 1; i < imudata.size(); ++i) {

        /***** eulur *****/
        // MotionData imupose = imudata[i];    You, 3 days ago • feat(ch2): add chapter 2 homework …

        // //delta_q = [1 , 1/2 * thetax , 1/2 * theta_y, 1/2 * theta_z]
        // Eigen::Quaterniond dq;
        // Eigen::Vector3d dtheta_half =  imupose.imu_gyro * dt /2.0;
        // dq.w() = 1;
        // dq.x() = dtheta_half.x();
        // dq.y() = dtheta_half.y();
        // dq.z() = dtheta_half.z();
        // dq.normalize();

        // /// imu 动力学模型 欧拉积分
        // Eigen::Vector3d acc_w = Qwb * (imupose.imu_acc) + gw;  // aw = Rwb * ( acc_body - acc_bias ) + gw
        // Qwb = Qwb * dq;
        // Pwb = Pwb + Vw * dt + 0.5 * dt * dt * acc_w;
        // Vw = Vw + acc_w * dt;

        /***** median *****/
        /// 中值积分
        MotionData imupose_pre = imudata[i-1]; // 上一时刻的数据
        MotionData imupose = imudata[i]; // 下一时刻的数据
        Eigen::Quaterniond dq;
        Eigen::Vector3d dtheta_half = (imupose_pre.imu_gyro + imupose.imu_gyro) * dt / 4.0;
        dq.w() = 1;
        dq.x() = dtheta_half.x();
        dq.y() = dtheta_half.y();
        dq.z() = dtheta_half.z();
        dq.normalize(); // 归一化
        Eigen::Quaterniond Qwb_pre = Qwb;
        // 下一时刻对应的转换矩阵是 Qwb * dq, 不是 Qwb
        Qwb = Qwb * dq;
        Eigen::Vector3d acc_w = (Qwb_pre * imupose.imu_acc + gw + Qwb * imupose_pre.imu_acc + gw) / 2;
        Qwb = Qwb * dq;
        Pwb = Pwb + Vw * dt + 0.5 * dt * dt * acc_w;
        Vw = Vw + acc_w * dt;

        //  按着imu postion, imu quaternion , cam postion, cam quaternion 的格式存储，由于没有cam，所以imu存了两次
        save_points<<imupose.timestamp<<" "
```

- 运行结果：使用中值法积分，轨迹比欧拉法更接近真值



3. 论文阅读

> BMVC, Steven Lovegrove ,Spline Fusion: A continuous-timerepresentation for visual-inertial fusion withapplication to rolling shutter cameras

主要介绍了用样条插值的方法来解决 rolling shutter 的问题。同时，样条插值的方法也用到了 imu 数据的仿真中，给定特定时刻的姿态，能从姿态插值生成线速度，加速度等数据。其中：

1. Introduction：介绍了一些前人的相关工作。
2. Continuous-time representation：使用离散的相机位姿，通过 B-Spline 曲线得到连续的时间轨迹，这条时间轨迹任一点上都可以对时间求导。
3. Generative model of visual-inertial data：通过迭代的非线性最小二乘，求解样条、相机内参、逆深度、相机到 IMU 的外参、IMU 零偏(spline control poses, camera intrinsics, landmark inverse depth values, camera to IMU transformation and IMU biases)。
4. Projection into a rolling shutter camera：卷帘同步相机的每一行像素都是在不同阶段曝光得到的，当相机在运动的时候，卷帘同步会造成图像的畸变。使用连续时间模型可以把每一行像素的值精准定位到它曝光时的值。这里把 y 轴当做对于时间连续的参数，使用 sub-pixel 对不同时间进行插值。
5. Experiments：使用仿真数据和真实数据做了实验。