使用 Allan 方差法进行 IMU 标定

在 IMU 采集数据时，会产生两种误差：确定性误差和随机性误差，为获得精确的数据，需要对上述两种误差进行标定。

1、确定性误差

确定性误差主要包括 bias(偏置)、scale(尺度)、misalignment(坐标轴互相不垂直)等多种。常使用六面静置法标定**加速度计**和**陀螺仪**的确定性误差。

2、随机误差

随机误差主要包括：高斯白噪声、bias 随机游走。**加速度计**和**陀螺仪**随机误差的标定通常使用 Allan 方差法，Allan 方差法是 20 世纪 60 年代由美国国家标准局的 David Allan 提出的基于时域的分析方法。

3、Allan 方差图读取误差系数

Allan 方差法可用于 5 种随机误差的标定：

量化噪声(Quantization Noise)：误差系数为 Q，Allan 方差双对数曲线上斜率为-1 的直线延长线与 t=10° 的交点的纵坐标读数为$\sqrt{3}$Q；

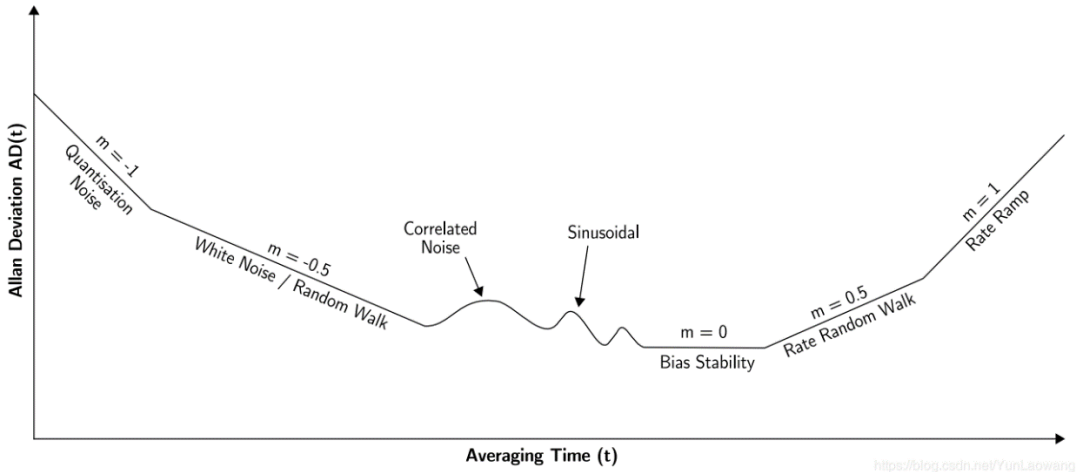
角度随机游走(Angle Random Walk)：其误差系数 N，Allan 方差双对数曲线上斜率为-1/2 的线的延长线与 t=10° 交点的纵坐标读数即为 N；

零偏不稳定性(Bias Instability)：其误差系数 B，Allan 方差双对数曲线上斜率为 0 的线的延长线与 t=10° 交点的纵坐标读数为$\sqrt{\frac{2\ln 2}{\pi}}$B，一般常取底部平坦区的最小值或取 $t=10^1$ 或 $t=10^2$ 处的值；

角速率随机游走(Rate Random Walk)：其误差系数 K，斜率为+1/2 的线的延长线与 t=10˚交点的纵坐标读数为$\frac{K}{\sqrt{3}}$;

角速率斜坡(Rate Ramp)：其误差系数 R，斜率为+1 的线的延长线与 t=10˚交点的纵坐标读数为$\frac{R}{\sqrt{2}}$;

噪声类型	参数	单位	Allan 标准差
量化噪声	Q	μrad	$\sigma_Q=\sqrt{3}\,Q/\tau$
角度随机游走	N	$°/\sqrt{h}$	$\sigma_{arw}=N/\sqrt{\tau}$
零偏不稳定性	B	°/h	$\sigma_b=B/0.6648$
角速度随机游走	K	$(°/h)/\sqrt{h}$	$\sigma_{rrw}=K\sqrt{\tau/3}$
速率斜坡	R	°/h/h	$\sigma_{rr}=R\tau/\sqrt{2}$
指数相关噪声	q_e	$°/\sqrt{h}$	$\sigma_M=q_cT_c/\sqrt{\tau}$ $(\tau>>T_c)$ $\sigma_M=q_c\sqrt{\tau/3}$ $(\tau<<T_c)$
正弦噪声	ω_0	°/h	$\sigma_s=\omega_0(\sin^2(\pi f_0\tau)/(\pi f_0\tau))$



IMU 标定 Allan 方差工具

常用的 Allan 方差工具，主要有以下两种：

- https://github.com/gaowenliang/imu_utils
- https://github.com/rpng/kalibr_allan

imu_utils

使用 Ubuntu 18.04 + ros-melodic-desktop-full

1、安装 ROS

初始化 catkin 工作空间

- mkdir –p ~/catkin_ws/src
- cd ~/catkin_ws/src
- catkin_init_workspace //初始化工作空间
- cd ..
- catkin_make
- source devel/setup.bash //设置 ros 环境

2、安装 ceres-solver

注意：https://github.com/ceres-solver/ceres-solver.git，编译之前需要安装一些基础

库：

- sudo apt-get install -y liblapack-dev libsuitesparse-dev libcxsparse3 libgflags-dev libgoogle-glog-dev libgtest-dev

3、编译 code_utils

在 catkin 工作空间中：

- cd ~/catkin_ws/src
- git clone https://github.com/gaowenliang/code_utils.git
- cd ~/catkin_ws
- catkin_make

在编译之前需要对源码进行修改两种方法：

(1) 在~/catkin_ws/src/code_utils/CMakeLists.txt 中，添加：
include_directories("include/code_utils");

(2) 修改~/catkin_ws/src/code_utils/src/sumpixel_test.cpp 文件中的#include "backward.hpp"为#include ";

4、编译 imu_utils

- cd ~/catkin_ws/srcgit clone https://github.com/gaowenliang/imu_utils.git
- cd ~/catkin_ws
- catkin_make

5、生成 imu.bag

- roscore
- source devel/setup.bash
- rosrun vio_data_simulation vio_data_simulation_node

默认位置在～/根目录中

6、写 launch 文件

进入 catkin_ws/src/imu_utils/launch 文件夹，新建 imu.launch 文件：

```
<launch>
<node pkg="imu_utils" type="imu_an" name="imu_an" output="screen">
<param name="imu_topic" type="string" value= "/imu"/>
<param name="imu_name" type="string" value= "imutest"/>
<param name="data_save_path" type="string" value= "$(find
imu_utils)/data/"/>
<param name="max_time_min" type="int" value= "120"/>
<param name="max_cluster" type="int" value= "100"/>
</node>
</launch>
```

7、重新编译：

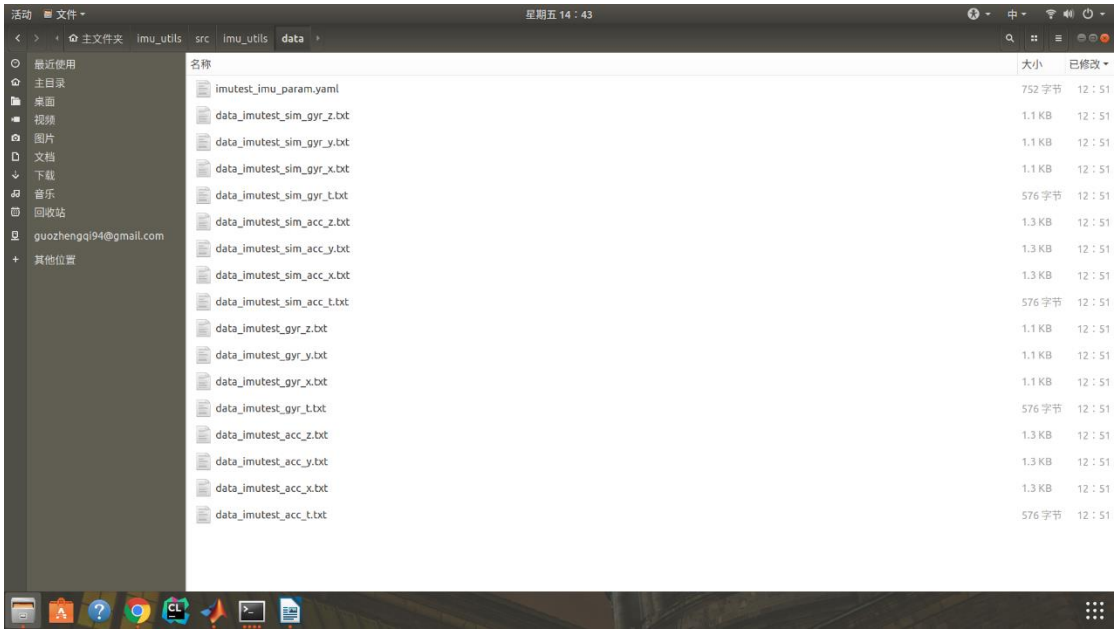
- cd ~/catkin_ws
- catkin_make
- source ./devel/setup.bash

8、生成 Allan 方差

rosbag 倍速回放 imu.bag 信息，并运行 launch 文件：

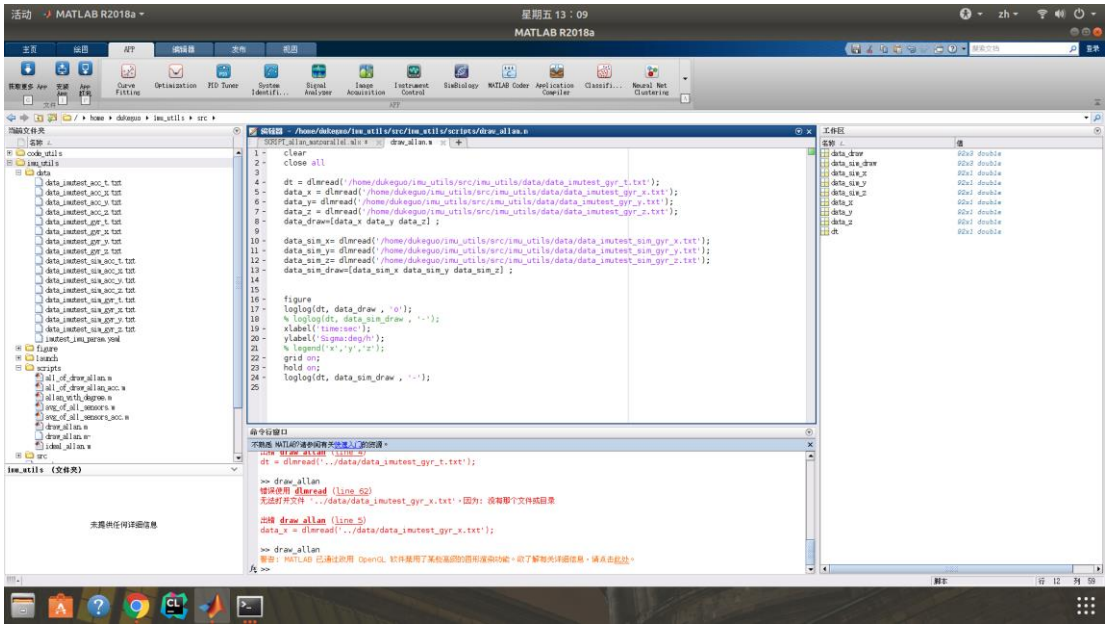
- rosbag play -r 200 imu.bag
- roslaunch imu_utils imu.launch

在 imu_utils/data 文件夹下，会生成 16 个 txt 文件：

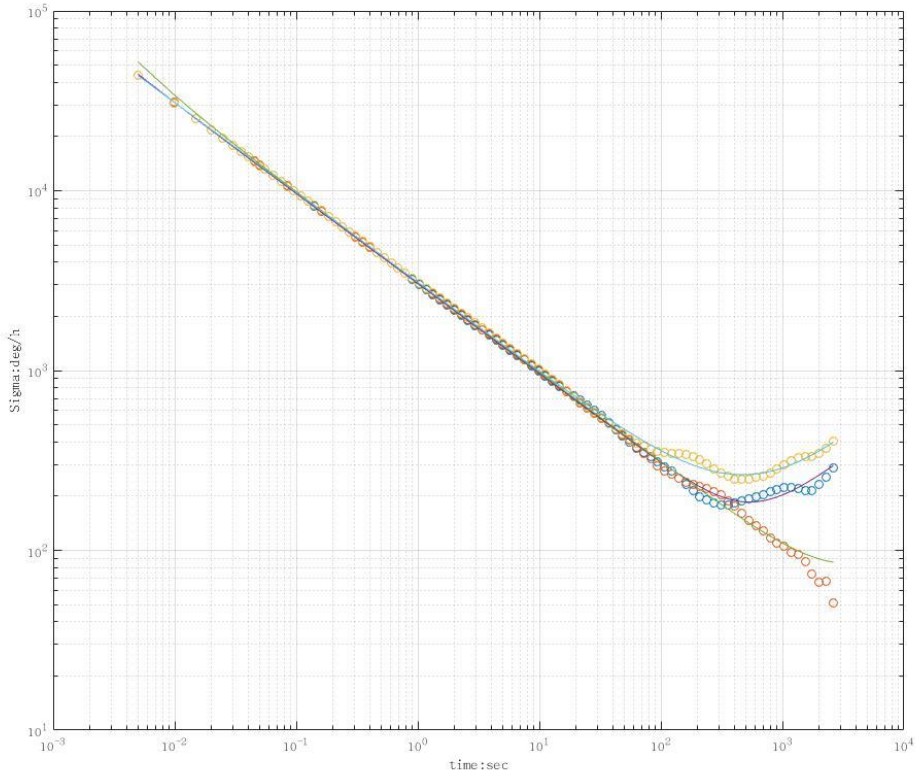


9. 绘制 Allan 方差图

修改 draw_allan.m 中文件路径：



运行结果：



根据 Allan 方差图即可读出相应的误差。

kalibr_allan

使用 Matlab2018

1、编译 kalibr_allan

- cd ~/catkin_ws/src
- git clone https://github.com/rpng/kalibr_allan.git
- cd ..
- catkin_make

2、bag 文件转换成 mat

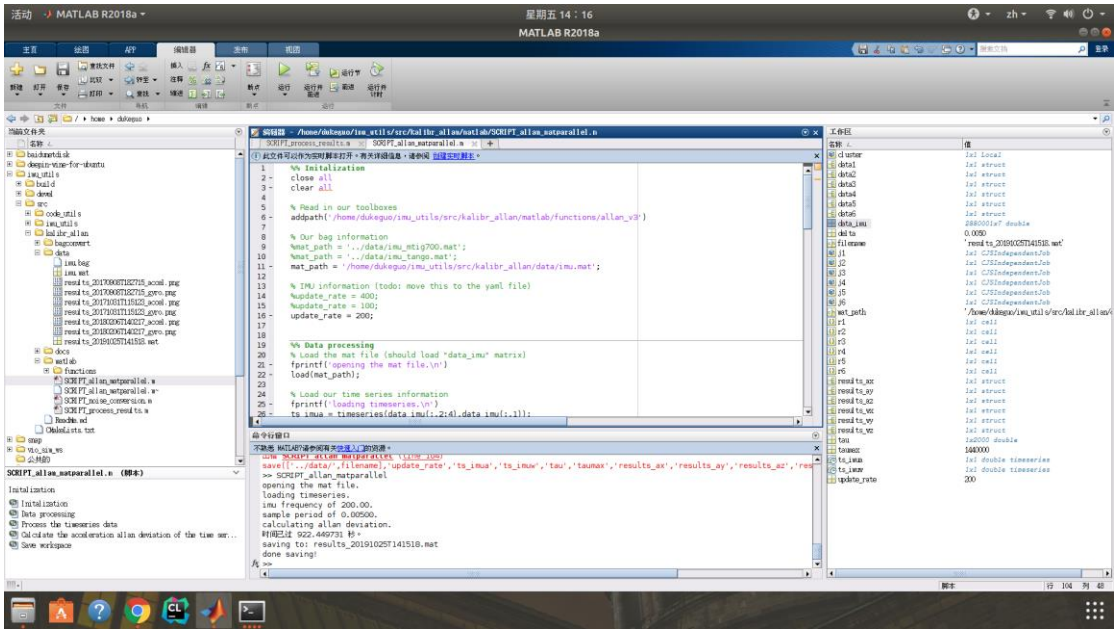
拷贝上述过程生成的 imu.bag 拷贝到~/catkin_ws/src/kalibr_allan/data 文件夹中，参考：https://github.com/rpng/kalibr_allan ，使用 bagconvert 将.bag 转换成.mat 文件：

- rosrun bagconvert bagconvert /home/dukeguo/imu_utils/src/kalibr_allan/data/imu.bag imu

在 imu.bag 文件的位置生成转换的结果 imu.mat，许多文件是“/imu0”，记得修改。

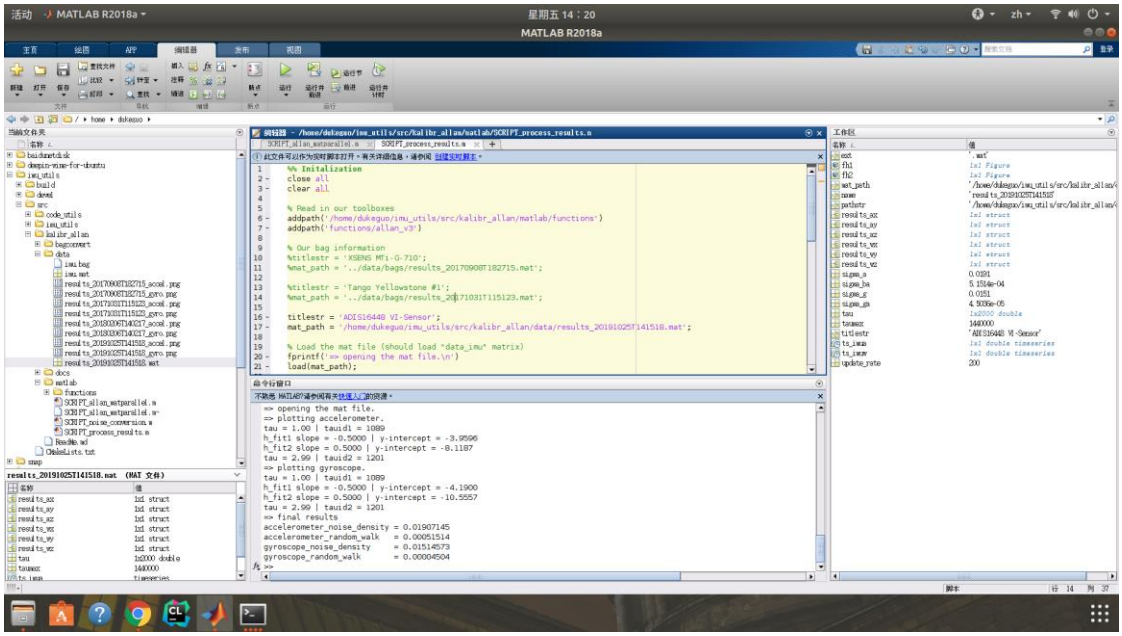
3、生成曲线参数文件

修改~/catkin_ws/src/kalibr_allan/matlab 文件夹下的 SCRIPT_process_results.m 中.mat 路径，即可画出 allan 曲线。

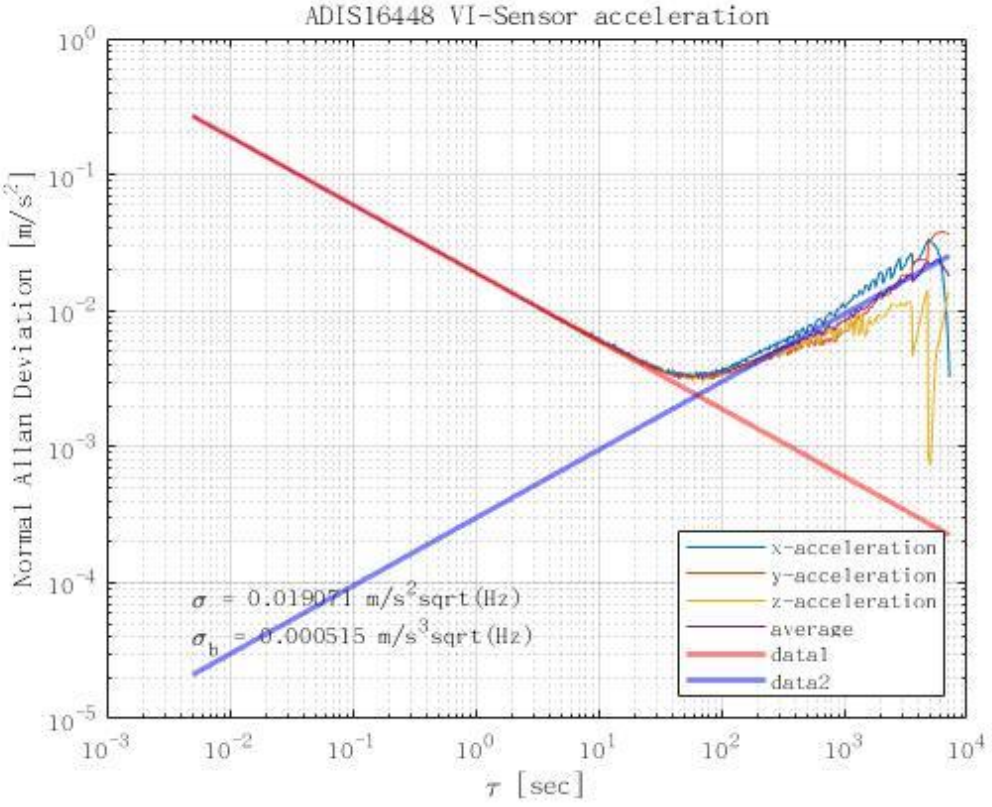


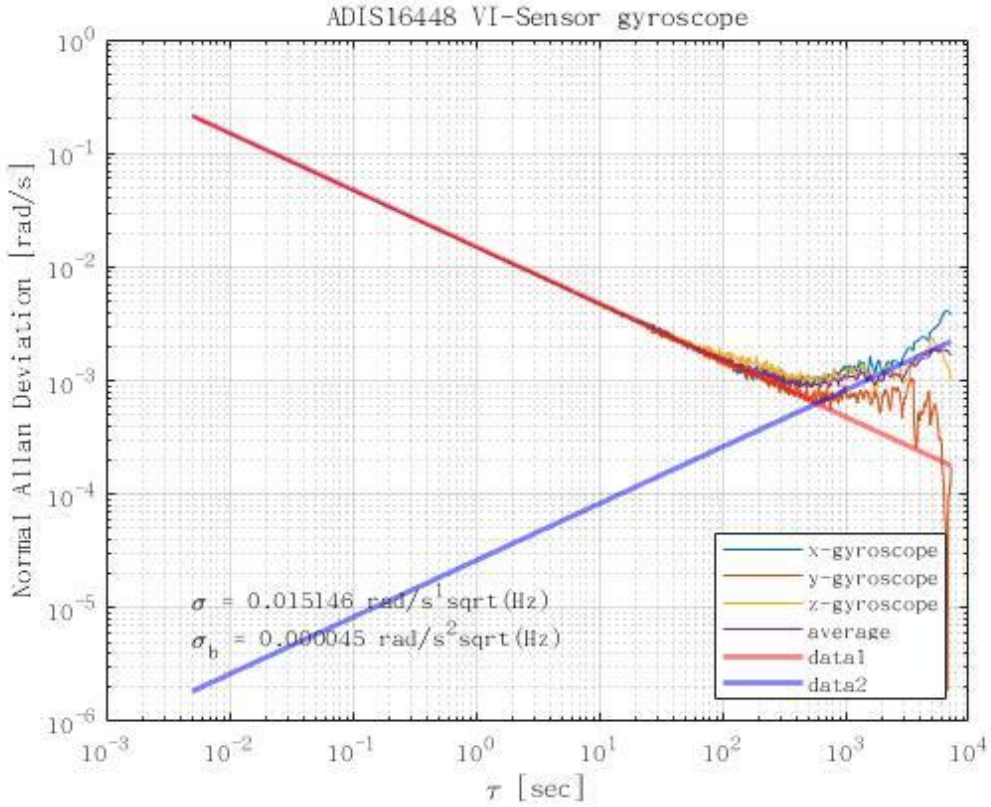
运行 SCRIPT_allan_matparallel.m 生成 results_20191025T141518.mat 文件，用时 922s。

4、绘制 Allan 曲线：



运行 SCRIPT_process_results.m 生成 Allan 结果


ADIS16448 VI-Sensor acceleration
Normal Allan Deviation [m/s^2]
τ [sec]
σ = 0.019071 m/s^2sqrt(Hz)
σb = 0.000515 m/s^3sqrt(Hz)
x-acceleration
y-acceleration
z-acceleration
average
data1
data2


ADIS16448 VI-Sensor gyroscope
Normal Allan Deviation [rad/s]
τ [sec]
σ = 0.015146 rad/s^1sqrt(Hz)
σb = 0.000045 rad/s^2sqrt(Hz)
x-gyroscope
y-gyroscope
z-gyroscope
average
data1
data2

使用欧拉积分和中值积分进行 IMU 模拟

在进行 IMU 数据仿真的时候，主要的积分方法有欧拉积分和中值积分两种。

欧拉积分

运动模型的离散积分——欧拉法

使用欧拉法，即两个相邻时刻 k 到 k+1 的位姿是用第 k 时刻的测量值 $\mathbf{a}, \boldsymbol{\omega}$ 来计算。

$$\begin{aligned}
\mathbf{p}_{wb_{k+1}} &= \mathbf{p}_{wb_k} + \mathbf{v}_k^w \Delta t + \frac{1}{2}\mathbf{a}\Delta t^2 \\
\mathbf{v}_{k+1}^w &= \mathbf{v}_k^w + \mathbf{a}\Delta t \\
\mathbf{q}_{wb_{k+1}} &= \mathbf{q}_{wb_k} \otimes \begin{bmatrix} 1 \\ \frac{1}{2}\boldsymbol{\omega}\delta t \end{bmatrix}
\end{aligned} \tag{30}$$

其中，

$$\begin{aligned}
\mathbf{a} &= \mathbf{q}_{wb_k}\left(\mathbf{a}^{b_k} - \mathbf{b}_k^a\right) - \mathbf{g}^w \\
\boldsymbol{\omega} &= \boldsymbol{\omega}^{b_k} - \mathbf{b}_k^g
\end{aligned} \tag{31}$$

代码实现为：

```
MotionData imupose = imudata[i];

//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;
```

中值积分

中值积分与欧拉积分不同的是，从第二项开始，每一项都是与前项的平均值：

运动模型的离散积分——中值法

使用 mid-point 方法，即两个相邻时刻 k 到 k+1 的位姿是用两个时刻的测量值 $\mathbf{a}, \boldsymbol{\omega}$ 的平均值来计算。

$$
\begin{aligned}
\mathbf{p}_{wb_{k+1}} &= \mathbf{p}_{wb_k} + \mathbf{v}_k^w \Delta t + \frac{1}{2}\mathbf{a}\Delta t^2 \\
\mathbf{v}_{k+1}^w &= \mathbf{v}_k^w + \mathbf{a}\Delta t \\
\mathbf{q}_{wb_{k+1}} &= \mathbf{q}_{wb_k} \otimes \begin{bmatrix} 1 \\ \frac{1}{2}\boldsymbol{\omega}\delta t \end{bmatrix}
\end{aligned} \tag{32}
$$

其中，

$$
\begin{aligned}
\mathbf{a} &= \frac{1}{2}\left[\mathbf{q}_{wb_k}\left(\mathbf{a}^{b_k} - \mathbf{b}_k^a\right) - \mathbf{g}^w + \mathbf{q}_{wb_{k+1}}\left(\mathbf{a}^{b_{k+1}} - \mathbf{b}_k^a\right) - \mathbf{g}^w\right] \\
\boldsymbol{\omega} &= \frac{1}{2}\left[\left(\boldsymbol{\omega}^{b_k} - \mathbf{b}_k^g\right) + \left(\boldsymbol{\omega}^{b_{k+1}} - \mathbf{b}_k^g\right)\right]
\end{aligned} \tag{33}
$$

代码实现为：

```
if (i > 1) {
    MotionData imupose_before = imudata_mid[i - 1];
    Eigen::Vector3d dtheta_half = ((imupose.imu_gyro + imupose_before.imu_gyro) / 2.0) * dt / 2.0; //1/2 * w * delta(t)
    dq.w() = 1;
    dq.x() = dtheta_half.x();
    dq.y() = dtheta_half.y();
    dq.z() = dtheta_half.z();
    dq.normalize();
    Eigen::Vector3d acc_w = Qwb * (imupose.imu_acc) + gw;  // aw = Rwb * ( acc_body - acc_bias ) + gw
    Eigen::Vector3d acc_w_before = Qwb * (imupose_before.imu_acc) + gw;
    acc_w_mid = (acc_w + acc_w_before) / 2.0;
    Qwb = Qwb * dq;
    Pwb = Pwb + Vw * dt + 0.5 * dt * dt * acc_w_mid;
    Vw = Vw + acc_w_mid * dt;
```

如果将两个结果放在一起的话，可以明显看出，中值积分的结果更接近 groundtruth：