

# 从零开始手写VIO 第二课作业

边城量子 2019.6.18

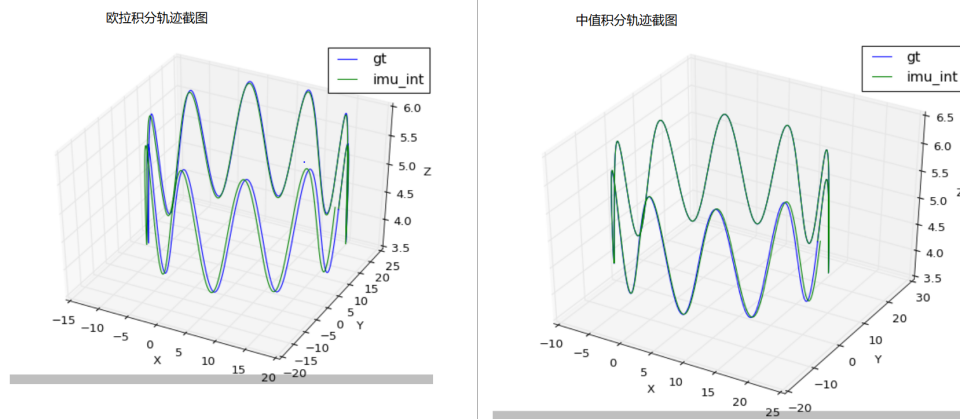
## 基础作业,必做

- 设置IMU仿真代码中的不同的参数, 生成Allan方差标定曲线.
  - Allan方差工具:
    - [https://github.com/gaowenliang/imu\\_utils](https://github.com/gaowenliang/imu_utils)
    - [https://github.com/rpng/kalibr\\_allan](https://github.com/rpng/kalibr_allan)
  - IMU仿真代码:
    - <http://www.shenlanxueyuan.com/course/160/activiy/2275/download?materialId=1552>
- 将IMU仿真代码中的欧拉积分替换成中值积分

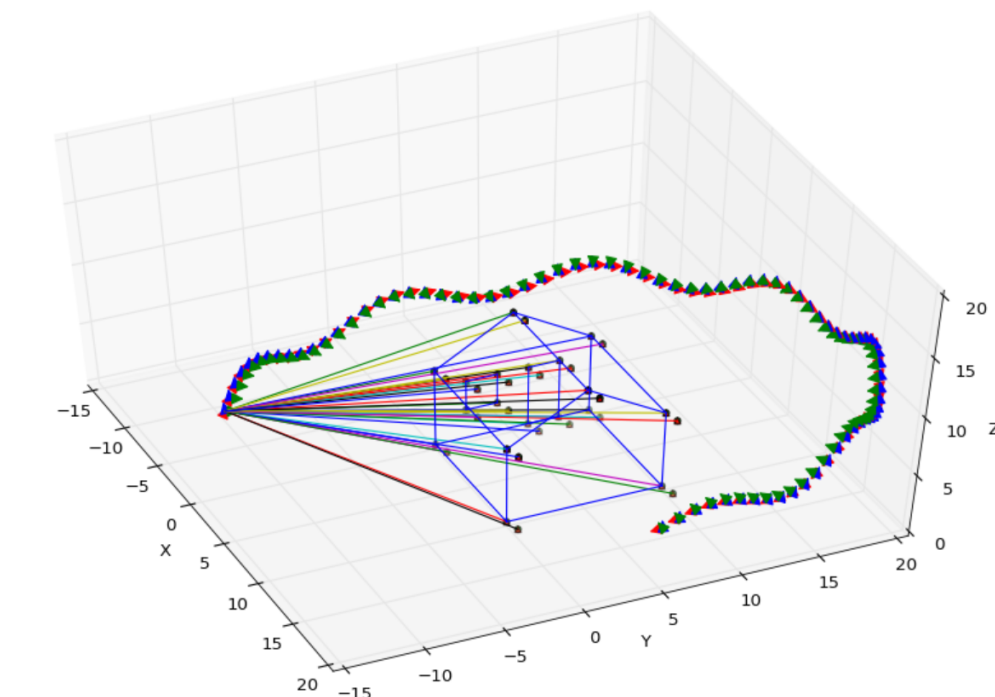
## 回答:

- **目标1: 编译非ROS版本, 绘制trajectory轨迹, 比较欧拉积分和中值积分的轨迹差异**
    - 1. 编译过程
      - 1.1 解压后编译 vio\_data\_simulation-master
        - 备注: 需要提前安装Eigen3, Sophus, Ceres等相关库
      - 1.2 在bin下新建 keyframe 文件夹: mkdir bin/keyframe
    - 2. 欧拉积分轨迹绘制
      - 2.1 进入bin目录, 运行 ./data\_gen, 在bin目录下生成轨迹数据
- ```
all_points.txt    cam_pose.txt    demo.png        imu_int_pose_noise.txt
imu_pose_noise.txt keyframe
cam_pose_tum.txt data_gen        house_model     imu_int_pose.txt
imu_pose.txt
```
- 2.2 进入python\_tool目录, 运行 python draw\_trajecory.py, 画出欧拉积分图形,图形见下文
  - 3. 中值积分轨迹绘制
    - 3.1 修改src/imu.cpp文件, 欧拉积分改为中值积分,修改后的代码如下文附录所示
    - 3.2 重新编译, 再次执行 bin/data\_gen和python\_tool下的python脚本, 画出中值积分图形,图形见下文
  - 4. 欧拉积分与中值积分的轨迹比较

- 4.1 欧拉与中值积分的轨迹截图比较如下, 可以看出中值积分误差稍小些:



- 4.2 附: 执行 draw\_points.py 会逐个描点, 并跟踪惯性系下的点(组成房子的那些点)根据带误差的轨迹反算回来后坐标的变化情况(可以看到并不是和原惯性点完全重合). 最终的效果图如下:



备注: 为了让图形绘制不至于在结束后自动退出,可以在py脚本最后加一行代码

```
plt.pause(-1)
```

- 5.附: 中值积分代码修改后如下, 代码行[50,68]:

```
//读取生成的imu数据并用imu动力学模型对数据进行计算, 最后保存imu积分以后的轨迹,
//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_;
    Eigen::Quaterniond Qwb(init_Rwb_);
    Qwb.normalize();

    // position : from imu
    // quaterniond: from imu
    measurements
    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) {

    MotionData imupose = imudata[i];
    MotionData imupose_ = imudata[i-1];

    /// imu 动力学模型 中值积分

    //delta_q = [1 , 1/2 * thetax , 1/2 * theta_y, 1/2 * theta_z]
    Eigen::Quaterniond dq;
    // i时刻与i-1时刻的gyro相加后除以2, 得到中值角速度
    Eigen::Vector3d dtheta_half = 0.5* (imupose.imu_gyro +
imupose_.imu_gyro) * dt /2.0;
    dq.w() = 1;
    dq.x() = dtheta_half.x();
    dq.y() = dtheta_half.y();
    dq.z() = dtheta_half.z();

    // 由i-1时刻Qwb和中值角速度dq, 计算i时刻的Q,并归一化
    Eigen::Quaterniond Qwb_i = Qwb*dq;
    Qwb_i.normalize();
    // i时刻与i-1时刻加速度相加后除以2, 得到中值加速度
    Eigen::Vector3d acc_w = (Qwb * imupose.imu_acc + Qwb_i
*imupose_.imu_acc) * 0.5 + gw;
    // {body}系角速度转为{world}系角速度
    Qwb = Qwb_i ;
    // {body}系中点在{world}系中的坐标的刷新 ( p_w' = p_w + v_w*t + 0.5*t*t*a_w
)

    Pwb = Pwb + Vw * dt + 0.5 * dt * dt * acc_w;
    // {body}系中的点在{world}系中的速度刷新 ( v_w' = v_w + a_w*t)
    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<<" "
        <<Qwb.w()<<" "
        <<Qwb.x()<<" "
        <<Qwb.y()<<" "
        <<Qwb.z()<<" "
        <<Pwb(0)<<" "
        <<Pwb(1)<<" "
        <<Pwb(2)<<" "
        <<Qwb.w()<<" "
        <<Qwb.x()<<" "
        <<Qwb.y()<<" "
        <<Qwb.z()<<" "
        <<Pwb(0)<<" "
        <<Pwb(1)<<" "
        <<Pwb(2)<<" "
        <<std::endl;

}

std::cout<<"test end"<<std::endl;

```

```
}
```

## • 目标2: 编译ROS版本, imu\_utils 绘制Allan曲线, 分析结果

- 1. 下载所需的ROS包
  - 1.1 下载imu\_utils: [https://codeload.github.com/gaowenliang/imu\\_utils/zip/master](https://codeload.github.com/gaowenliang/imu_utils/zip/master)
  - 1.2 下载code\_utils: [https://codeload.github.com/gaowenliang/code\\_utils/zip/master](https://codeload.github.com/gaowenliang/code_utils/zip/master)
  - 1.3 下载vio\_data\_simulation-ros\_version: <http://www.shenlanxueyuan.com/course/160/activiy/2275/download?materialId=1552>
  - 1.4 以上三个ros的package均解压后放入 `~/catkin_ws/src` 下
- 2. 修改代码bug, 准备必备文件, 填各种坑, 为编译做准备,如下:
  - 2.1 修改 `~/catkin_ws/src/imu_utils-master/package.xml`, 添加对 `code_utils` 的依赖, 避免编译时提示找不到 `code_utils` 的错误, 修改后内容如下(备注: 其中 `<buildtool_depend>` 所在行属于原有内容):

```
<buildtool_depend>catkin</buildtool_depend>
<build_depend>code_utils</build_depend>
```

- 2.2 修改 `~/catkin_ws/src/vio_data_simulation-ros_version/src/gener_alldata.cpp` 文件第18行, 将生成 `imu.bag` 的地址改为本机存在的地址, 例如 `~/Downloads/imu.bag`, 避免未来执行 `node` 时出现无法打开 `imu.bag` 的错误。修改后代码如下:

```
rosbag::Bag bag;
bag.open("/home/hadoop/Downloads/imu.bag", rosbag::bagmode::write);
```

- 2.3 修改 `~/catkin_ws/src/code_utils-master/src/sumpixel_test.cpp` 的包含文件路径, 把 `#include "backward.hpp"` 改为如下所示, 避免编译时出现找不到 `backward.hpp` 的错误, 修改后代码如下:

```
#include "code_utils/backward.hpp"
```

- 2.4 在 `~/catkin_ws/src/imu_utils-master/launch/` 下添加新的 `launch` 文件 `vio_imu.launch`, 内容如下:

```
<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= "vio_imu"/>
    <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>
```

- 3. 编译与执行
  - 3.1 执行 `catkin_make` 进行编译如上的三个包

- 3.2 在多个终端分别执行如下语句, 运行 `vio_data_simulation_node` 节点生成 `imu.bag` 文件:

```
终端1: roscore
终端2: roslaunch vio_data_simulation vio_data_simulation_node
```

- 3.3 在多个终端分别执行如下语句, 回放 `imu.bag` 并使用 `imu_utils` 进行分析得到数据:

```
终端2: roslaunch imu_utils vio_imu.launch
终端3: rosbag play -r 500 ~/Downloads/imu.bag
```

其中-r 500表示是以500倍的速率播放这120分钟的数据,一般接近播放完毕时终端2上就会显示结果和生成数据;

`~/catkin_ws/src/code_utils-master/data/` 目录下, 生成的分析数据的文件列表:

```
data_vio_imu_acc_t.txt data_vio_imu_gyr_x.txt
data_vio_imu_sim_acc_y.txt data_vio_imu_sim_gyr_z.txt
data_vio_imu_acc_x.txt data_vio_imu_gyr_y.txt
data_vio_imu_sim_acc_z.txt vio_imu_imu_param.yaml
data_vio_imu_acc_y.txt data_vio_imu_gyr_z.txt
data_vio_imu_sim_gyr_t.txt data_vio_imu_acc_z.txt
data_vio_imu_sim_acc_t.txt data_vio_imu_sim_gyr_x.txt
data_vio_imu_gyr_t.txt data_vio_imu_sim_acc_x.txt
data_vio_imu_sim_gyr_y.txt
```

- 3.4 查看结果, `~/catkin_ws/src/code_utils-master/data/vio_imu_imu_param.yaml` 文件内容,可以看出 `gyr_n`即`gyr`的噪声值为0.21, `gyr_w`即`gyr`的bias随机游走约为0.00072, `acc_n`即`acc`的噪声约为0.268, `acc_w`即`acc`的bias随机游走约为0.0036:

```
%YAML:1.0
---
type: IMU
name: vio_imu
Gyr:
  unit: " rad/s"
  avg-axis:
    gyr_n: 2.1105657814671464e-01
    gyr_w: 7.1662674189783239e-04
  x-axis:
    gyr_n: 2.1177628132040993e-01
    gyr_w: 8.5279050640430637e-04
  y-axis:
    gyr_n: 2.1246606280232497e-01
    gyr_w: 4.2323679207795977e-04
  z-axis:
    gyr_n: 2.0892739031740898e-01
    gyr_w: 8.7385292721123102e-04
Acc:
  unit: " m/s^2"
  avg-axis:
    acc_n: 2.6819340623544052e-01
    acc_w: 3.5774598464253763e-03
```

```

x-axis:
    acc_n: 2.6830418287375574e-01
    acc_w: 3.2635148262893290e-03
y-axis:
    acc_n: 2.6755589669114294e-01
    acc_w: 3.6611895452645497e-03
z-axis:
    acc_n: 2.6872013914142284e-01
    acc_w: 3.8076751677222510e-03

```

- 3.4 编写 `~/catkin_ws/src/imu_utils/scripts/draw_vio_imu_allan.m`，文件内容如下：

```

clear
close all

dt = dlmread('../data/data_vio_imu_gyr_t.txt');
data_x = dlmread('../data/data_vio_imu_gyr_x.txt');
data_y = dlmread('../data/data_vio_imu_gyr_y.txt');
data_z = dlmread('../data/data_vio_imu_gyr_z.txt');
data_draw=[data_x data_y data_z] ;

data_sim_x= dlmread('../data/data_vio_imu_sim_gyr_x.txt');
data_sim_y= dlmread('../data/data_vio_imu_sim_gyr_y.txt');
data_sim_z= dlmread('../data/data_vio_imu_sim_gyr_z.txt');
data_sim_draw=[data_sim_x data_sim_y data_sim_z] ;
figure
loglog(dt, data_draw , 'o');
% loglog(dt, data_sim_draw , '-');
xlabel('time:sec');
ylabel('Sigma:deg/h');
% legend('x','y','z');
grid on;
hold on;
loglog(dt, data_sim_draw , '-');

```

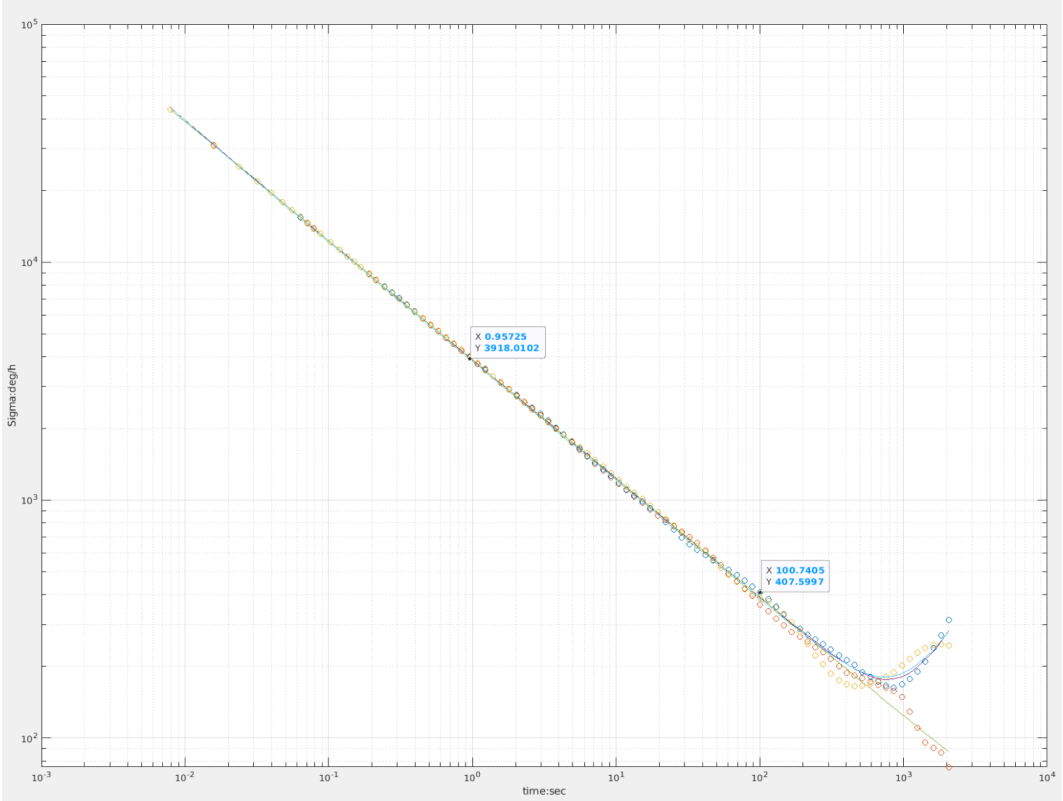
- 4. 绘制allan曲线, 分析结果
  - 4.1 安装matlab，然后执行如下脚本生成allan曲线：

```

cd ~/catkin_ws/src/imu_utils/scripts
matlab -r draw_vio_imu_allan.m

```

生成的曲线如下图所示( ):



- 5. 设定不同噪声参数的标定结果比较
  - 修改param.h中噪声和bias的设定值, 然后按照上述过程执行, 反复三次, 得到三组数据如下:
  - 5.1 第一组:

| 程序项名称            | 程序设定值   | 单位      | 工具标定名称 | 工具标定值   | 工具标定值(精确)              |
|------------------|---------|---------|--------|---------|------------------------|
| gyro_bias_sigma  | 0.00005 |         | gyr_w  | 0.00072 | 7.1662674189783239e-04 |
| acc_bias_sigma   | 0.0005  |         | acc_n  | 0.269   | 2.6819340623544052e-01 |
| gyro_noise_sigma | 0.015   | rad/s   | gyr_n  | 0.21    | 2.1105657814671464e-01 |
| acc_noise_sigma  | 0.019   | m/(s^2) | acc_w  | 0.0036  | 3.5774598464253763e-03 |

- 5.2 第二组:

| 程序项名称            | 程序设定值  | 单位      | 工具标定名称 | 工具标定值   | 工具标定值(精确)              |
|------------------|--------|---------|--------|---------|------------------------|
| gyro_bias_sigma  | 0.0001 |         | gyr_w  | 0.00147 | 1.4657805294837946e-03 |
| acc_bias_sigma   | 0.001  |         | acc_n  | 0.00623 | 6.2342927057711965e-03 |
| gyro_noise_sigma | 0.030  | rad/s   | gyr_n  | 0.428   | 4.2823808776695610e-01 |
| acc_noise_sigma  | 0.035  | m/(s^2) | acc_w  | 0.494   | 4.9445930067300020e-01 |

o 5.3 第三组:

| 程序项名称            | 程序设定值  | 单位      | 工具标定名称 | 工具标定值    | 工具标定值(精确)              |
|------------------|--------|---------|--------|----------|------------------------|
| gyro_bias_sigma  | 0.0005 |         | gyr_w  | 0.000782 | 7.8284393455931543e-04 |
| acc_bias_sigma   | 0.005  |         | acc_n  | 0.00604  | 6.0371737347246048e-03 |
| gyro_noise_sigma | 0.001  | rad/s   | gyr_n  | 0.0147   | 1.4735668185605065e-02 |
| acc_noise_sigma  | 0.005  | m/(s^2) | acc_w  | 0.0857   | 8.5741277324662635e-02 |

结论: 可以看到gyro的noise比较吻合, 但是整个bias都很差, 根据与贺博交流, 反馈说imu\_utils中yaml文件中得到的值(即allan曲线最低点)也不能代表正确的标定值, 建议还是换用kalidr\_allan工具标定就可以避免这种问题。

• 目标3: 编译ROS版本, 使用 kalidr\_allan 绘制Allan曲线, 分析标定结果

- o 1. 下载所需的ROS包
  - 1.1 下载kalibr\_allan: [https://codeload.github.com/rpng/kalibr\\_allan/zip/master](https://codeload.github.com/rpng/kalibr_allan/zip/master)
  - 1.2 下载vio\_data\_simulation-ros\_version: <http://www.shenlanxueyuan.com/course/160/activiy/2275/download?materialId=1552>
  - 1.3 以上两个ros的package均解压后放入 `~/catkin_ws/src` 下, 注意 kalibr\_allan中的bagconvert才是ROS package,需拷贝
  - 1.4 kalibr\_allan\_master 所在目录为 `~/Documents/kalibr_allan_master`
- o 2. 准备必备文件, 为编译和后续运行做准备,如下:
  - 2.1 修改 `~/Documents/kalibr_allan_master/matlab/SCRIPT_allan_matparallel.m` 第11行, 修改 `imu.mat` 文件保存位置:

```
mat_path = '/home/hadoop/Downloads/imu.mat';
```

修改104行, 指定分析结果文件保存的位置在 `~/Documents/` 下;

```
// 分析结果的保存文件
filename = ['results.mat'];
fprintf('saving to: %s\n', filename);
// 分析结果文件所保存的目录
save(['/home/hadoop/Documents/', filename], 'update_rate', 'ts_imua', 'ts_imuw', 'tau', 'taumax', 'results_ax', 'results_ay', 'results_az', 'results_wx', 'results_wy', 'results_wz');
```

- 2.2 修改 `~/Documents/kalibr_allan_master/matlab/SCRIPT_process_results.m` 第17行, 指定分析结果文件所在位置(必须与上一步指定的相同):

```
mat_path = '/home/hadoop/Documents/results.mat';
```



- 2.2 修改 `~/catkin_ws/src/vio_data_simulation-ros_version/src/gener_alldata.cpp` 文件第18行, 将生成 `imu.bag` 的地址改为本机存在的地址, 例如 `~/Downloads/imu.bag`, 避免未来执行 `node` 时出现无法打开 `imu.bag` 的错误。修改后代码如下:

```
rosbag::Bag bag;  
bag.open("/home/hadoop/Downloads/imu.bag", rosbag::bagmode::write);
```

修改66行, 把输出的topic从 `imu` 改为 `/imu`:

```
bag.write("/imu", time_now, imu_data);
```

○ 3. 运行与分析:

- 3.1 执行 `catkin_make` 进行编译如上的两个包
- 3.1 在多个终端分别执行如下语句, 运行 `vio_data_simulation_node` 节点生成 `imu.bag` 文件:

```
终端1: roscore  
终端2: rosrun vio_data_simulation vio_data_simulation_node
```

- 3.3 执行如下语句, 把 `imu.bag` 转换为 `imu.mat`, 执行时间较长, 几十分钟:

```
终端2: rosrun bagconvert bagconvert /home/hadoop/Downloads/imu.bag  
/imu  
其中/imu为topic名称
```

- 3.4 安装matlab, 执行如下脚本对 `imu.mat` 数据进行分析, 根据上面步骤的修改, 分析结果应该存入在 `/home/hadoop/Documents/results.mat`; 安装matlab, 然后执行如下脚本生成allan曲线:

```
cd ~/Documents/kalibr_allan-master/matlab/  
matlab -r SCRIPT_allan_matparallel.m
```

- 3.5 执行如下脚本获得标定结果, 并画出allan曲线.

```
cd ~/Documents/kalibr_allan-master/matlab/  
matlab -r SCRIPT_process_results.m
```

标定结果的命令行输出如下:

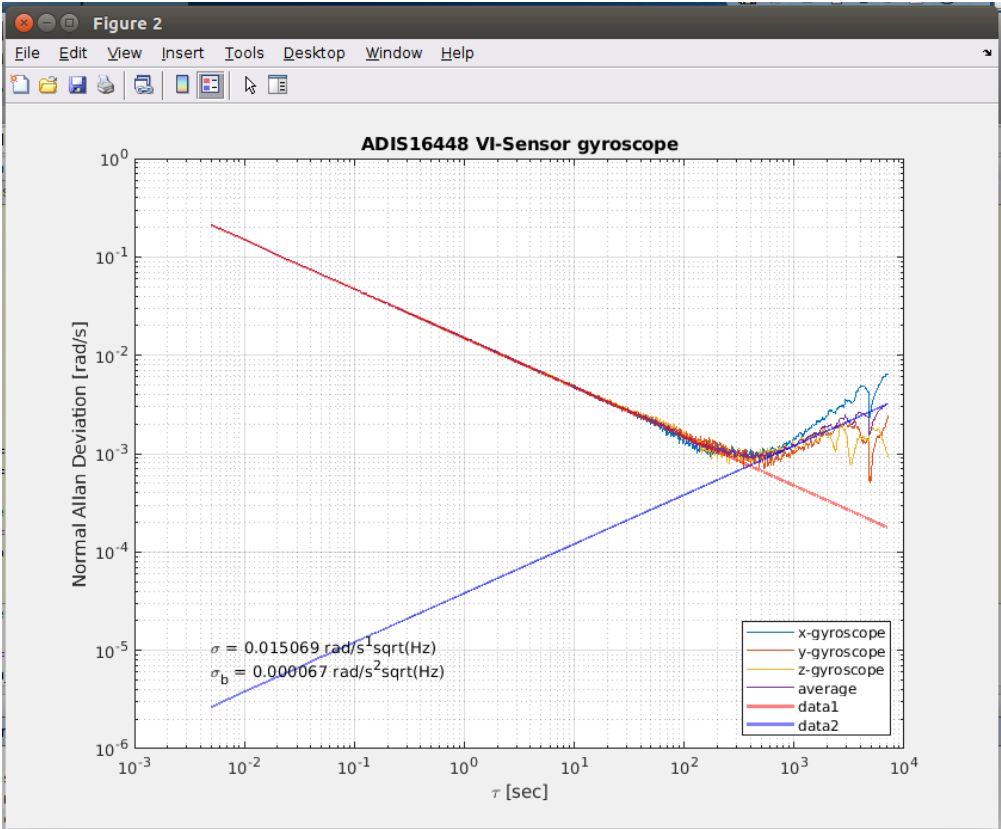
```
=> final results  
accelerometer_noise_density = 0.01906135  
accelerometer_random_walk   = 0.00046694  
gyroscope_noise_density      = 0.01506933  
gyroscope_random_walk        = 0.00006665
```

标定结果与程序设定值的比较如下:

| 程序项名称            | 程序设定值   | 单位      | 工具标定名称                      | 工具标定值      | 单位      |
|------------------|---------|---------|-----------------------------|------------|---------|
| gyro_bias_sigma  | 0.00005 |         | gyroscope_random_walk       | 0.00006665 |         |
| acc_bias_sigma   | 0.0005  |         | accelerometer_noise_density | 0.00046694 |         |
| gyro_noise_sigma | 0.015   | rad/s   | gyroscope_noise_density     | 0.01506933 | rad/s   |
| acc_noise_sigma  | 0.019   | m/(s^2) | accelerometer_noise_density | 0.01906135 | m/(s^2) |

结论: 通过上表可以得出: kalibr\_allan 工具标定值与程序原设定值非常接近.

曲线如下:



## 提升作业, 选做

- 阅读从已有轨迹生成imu数据的论文, 撰写总结推导:
  - 2013年 BMVC, Steven Lovegrove, Spline Fusion: A continuous-time representation for visual-inertial fusion with application to rolling shutter cameras.

## 回答:

- 已知多个离散的观测点, 通过B样条拟合曲线过这多个离散点
- 对光滑拟合曲线按照  $\Delta t$  为单位进行求导和二阶导, 可以得到速度与加速度
- 两个时刻之间的R的变化即为旋转角
- 具体总结推导如下:
  - 待进一步完善