日志分析工具使用说明 V_1_0

版本	更新日期	编制者	变更说明
V_1_0	2019-12-05	刘会良	初次设计

一、日志参数列表及用途

1.1 运动控制参数

1.1.1 位置数据

```
//参考位置           单位： 位置m，角度°，速度m/s，角速度°/s
float ref_x;
float ref_y;
float ref_theta;
//当前位置
int loca_index;     //记录location更新信息，每次调用callback函数后加一，0-49循环一次，用于分析定位状态
float cur_x;
float cur_y;
float cur_theta;
```

以上数据是topic发布的原始数据，同时后期处理可产生如下数据：

```
float loca_dotx;
float loca_doty;
float loca_dottheta;
```

loca_dot由两次定位数据的偏差量与控制周期的商得到，可以与如下odom速度数据做对比分析，评价定位数据的稳定性；

1.1.2 速度数据

```
//轨迹规划速度
float ref_dotx;
float ref_doty;
float ref_dottheta;
//控制指令输出速度
float set_dotx;
float set_doty;
float set_dottheta;
//里程计反馈速度
int odom_index;     //记录odom更新信息，每次调用callback函数后加一，0-49循环一次，用于反馈状态
```

```
float odom_dotx;
float odom_doty;
float odom_dottheta;
float gyro_omg;      //预留给陀螺仪的，用于判断是否打滑
```

如上规划速度与控制指令速度对比分析，可评价控制参数的合理性；控制指令速度与odom对比，可分析驱动响应特性；

1.2 状态数据

1.2.1 NTT监控与LPP的接口

```
int16 traj_index             //标识轨迹更新状态，每收到一次轨迹加一，且与下述轨迹类型结合分析
int16 recieve_traj_type      //轨迹类型  贝赛尔、停止、旋转
```

1.2.2 自身状态信息

```
int16 tracker_ctrl_status    //控制器模式，可以清楚的看到当前控制器在状态机中的状态
int16 tracker_action_mode    //底盘的轨迹或伺服控制状态
int16 ntt_error_code         //错误码
```

1.2.3 NTT向LPP的输出状态

```
int64 index                  //NTT执行的当前轨迹的Index
int64 current_traj_seq       //接收到的当前正在执行的轨迹的sequence%10（%10是为了便于显示分析）
int16 current_traj_status    //反馈当时执行轨迹的状态 :-1：初始状态 0：执行中 1：执行结束
```

二、日志分析工具功能需求

2.1 图形化显示

选中每个数据后可以时间轴绘制图形；可同时选中多个数据绘制在一个时间轴上对比分析；可框选放大局部区域；可选中曲线上的点，显示曲线上的点的详细信息，可多选显示；

三、日志分析工具使用方法

3.1 工具安装

工具安装见Feature #4041

3.2 工具操作方法介绍

运行后，界面如下图：

操作方法教程见菜单 help -> cheatsheet

四、使用工具进行分析

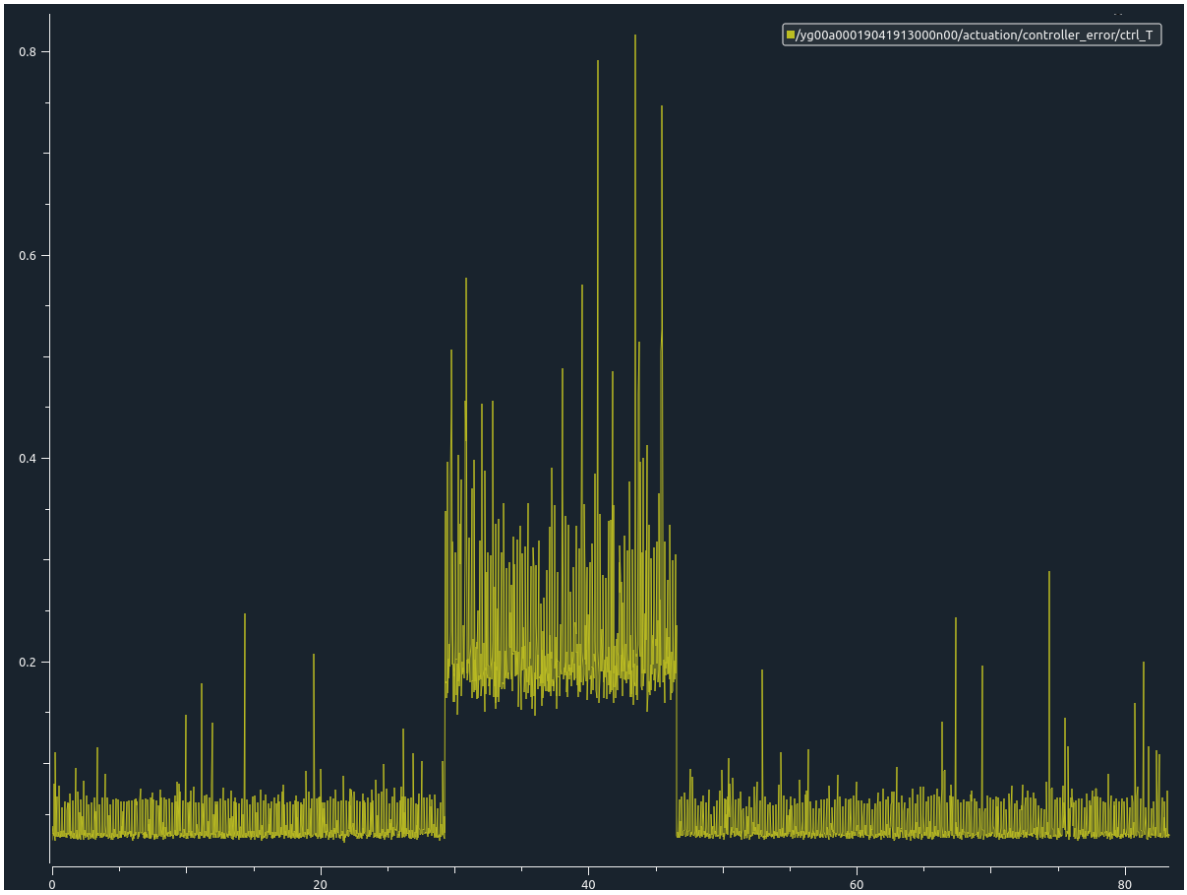
4.1 NTT分析

NTT节点主要功能是提供不同控制器进行轨迹跟踪控制（目前只进行轨迹跟踪控制，后期可能会加入伺服控制）。下叙分析通过录制 /yg***/actuation/controller_error 加载到分析工具进行分析。

4.1.1 性能分析

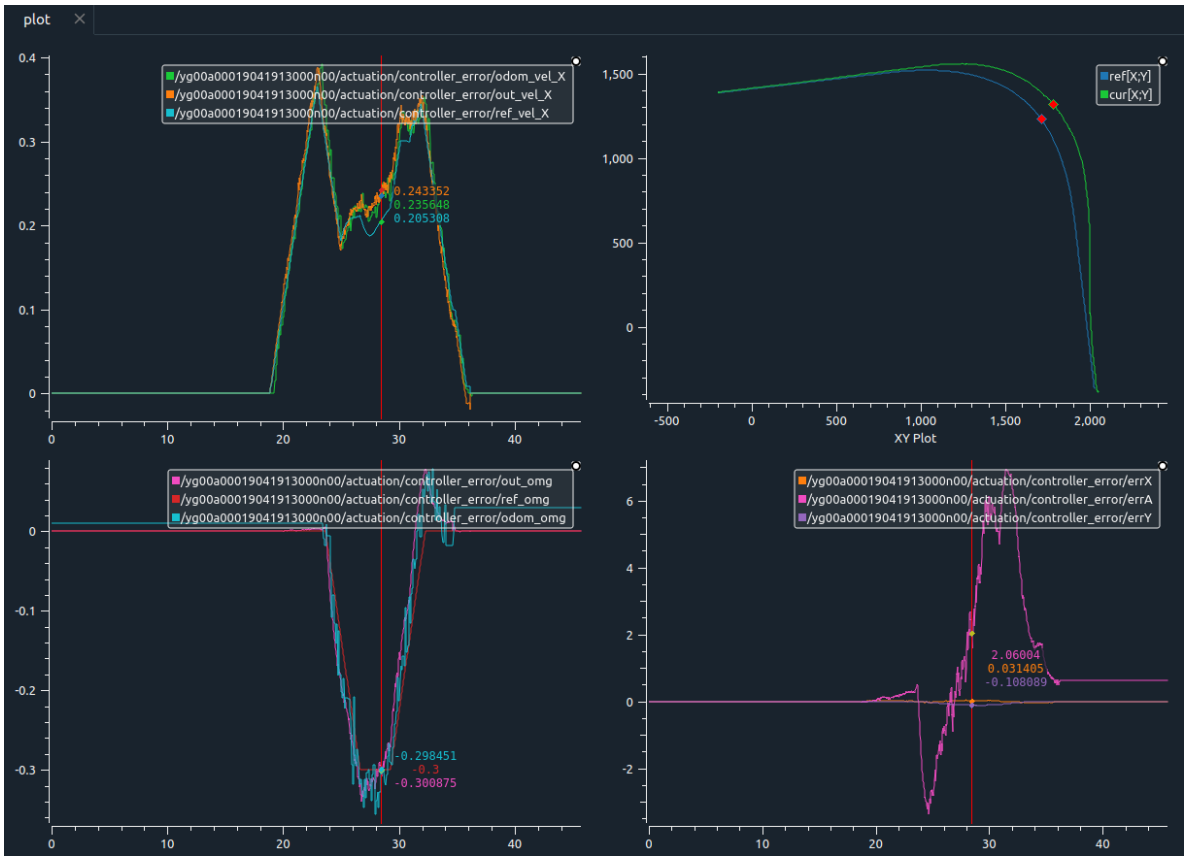
4.1.1.1 控制器运算时间分析

ctrl_T （单位ms）：控制频率通常设置为35hz,允许控制器单周期运行的最长时间对应未1/35 s =28.57ms 。使用PID控制器的运算时间如下图，(最长的运算时间为0.82ms)：



使用其他的控制器可能计算时间超时，造成控制器误差较大。

4.1.1.2 速度跟踪效果分析



如上图左侧两图分别为速度和角速度的规划值ref_xx、当前输出指令值out_xx、反馈值odom_xx。主要用于控制算法的分析、驱动响应规律分析、调参分析。

4.1.1.3 控制误差分析

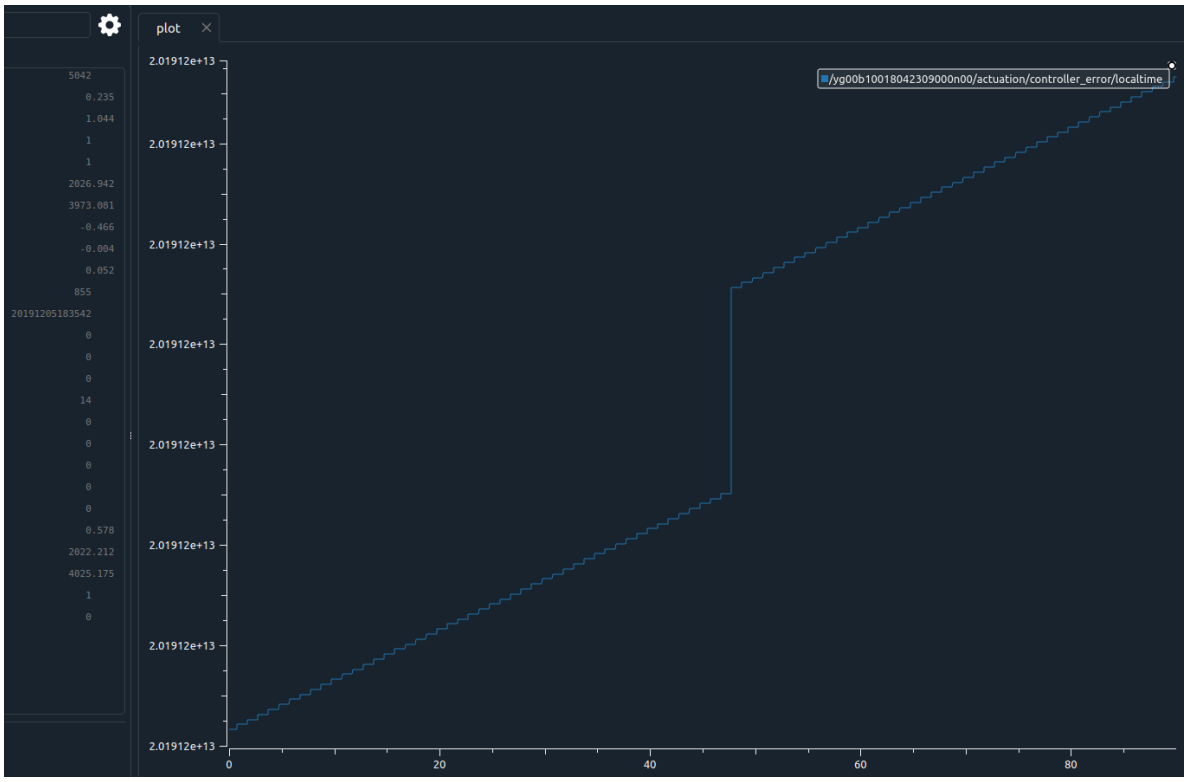
如上图右侧两图分别为轨迹和轨迹跟踪误差分析。ref[X,Y]为规划轨迹，cur[X,Y]为实际odom轨迹；在非全向底盘的模型下，err_X、err_Y和err_A分别为车身坐标系下的X轴、Y轴和角度误差；在全向底盘的模型下，err_X、err_Y和err_A分别为绝对坐标系下的误差。主要用于分析跟踪的控制误差。

4.1.2 状态分析

状态分析作为研发或者FAE排查现场故障很有用，可以快速定位是哪个模块的问题。本小节以NTT为中心进行故障排查，分为输入，NTT自身状态，NTT反馈给LPP的状态分析三部分。

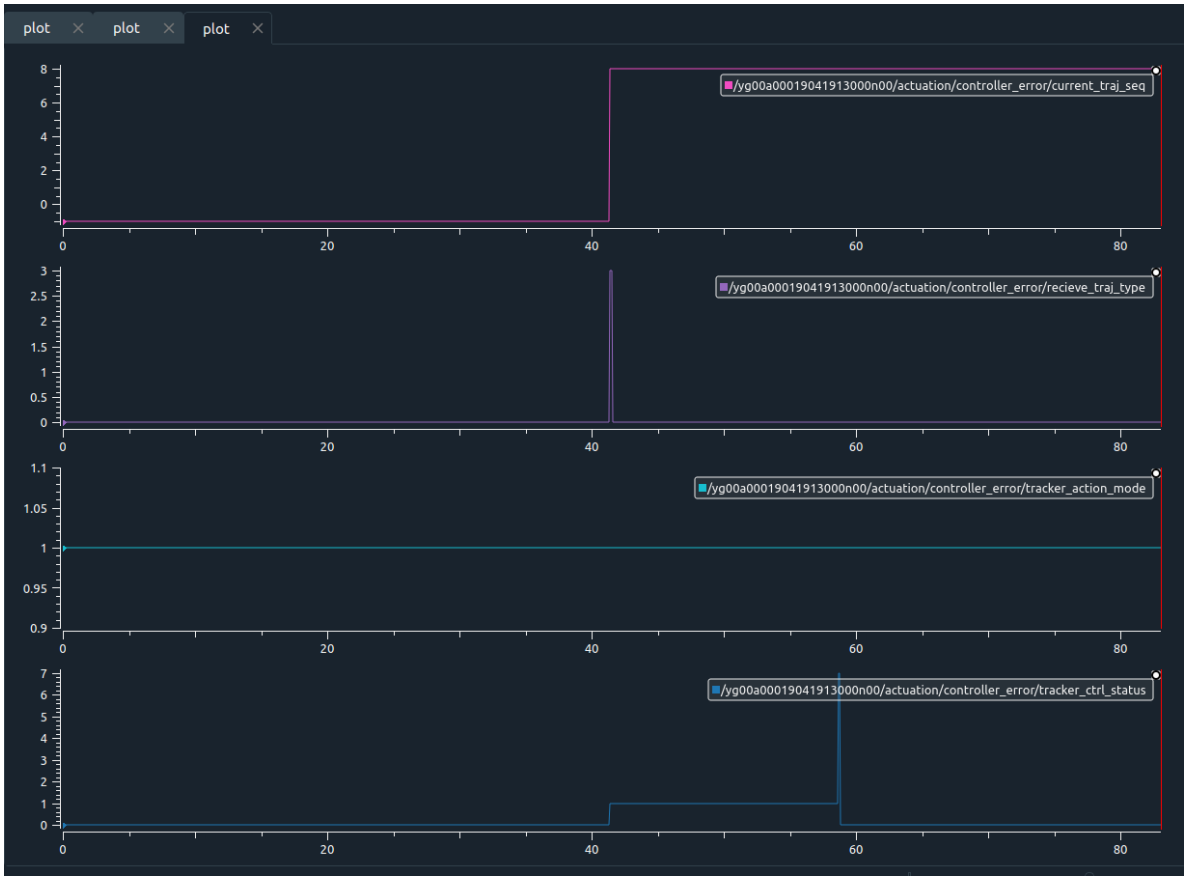
4.1.2.1 时间戳确定

进行状态分析的时候，我们通常是在进行故障分析。故障分析的第一步首先是确定故障发生的时间，一方面，明确故障发生的时刻后，可缩小分析的数据量，以快速分析问题；另一方面方便不同模块的协同分析。



4.1.2.2 LPP 下发轨迹命令分析

出故障时，需要分析故障是否因LPP引起，主要按照下叙方法排查：



如上图，

1）**current_traj_seq** ：为LPP下发的sequence%10，正常情况下，LPP以sequence数据来标识是否是同一个轨迹的重规划，还是下一条轨迹，新的轨迹会加1。总之，NTT用此数据来表示是否收到了LPP新下发了轨迹。

2）**recieve_traj_type** ：表示新下发的轨迹类型，具体如下：

```
enum NEW_TRAJECT_TYPE{
    IDLE_TRAJ=0,
    SAME_ID_TRAJ=1,
    EMPTY_TRAJ=2,
    NORMAL_TRAJ=3,
    TURN_TRAJ=4
  };//注意此类型与接口的定义不相关，NTT自身分析使用
```

如上图可见在时刻41左右，收到了一个sequence为8结尾的新轨迹，同时轨迹类型是3（NORMAL_TRAJ，表示正常的贝赛尔曲线轨迹）。

说明：recieve_traj_type在收到一个新类型的轨迹，5个周期过后会恢复为IDLE_TRAJ类型，可以用此脉冲直接看到接收到一个新的轨迹，和轨迹的类型。

3）**tracker_action_mode** ：表示收到的是轨迹任务或者伺服任务（目前只支持轨迹任务TAM_TRACKER）

```
enum TRACKERACTIONMODE{
    TAM_IDLE=0,
    TAM_TRACKER=1,
    TAM_ACTION=2
  };
```

4.1.2.3 NTT控制器状态分析

1）**tracker_ctrl_status** ：表示控制器NTT当时执行的状态，可用于对输入的关联分析。

```
enum CONTROLLER_STATUS_Enum {
    NTTCS_IDLE = 0,
    NTTCS_TRACKING = 1,
    NTTCS_ADJUSTERROR = 3,
    NTTCS_TURNING= 4,
    NTTCS_PAUSE= 5,
    NTTCS_ACTION= 6,            //保留
    NTTCS_STOP = 7,
    NTTCS_ERROR = 8
  };
```

1.1）正常执行一条NORMAL_TRAJ类型的轨迹时，状态切换如下：

NTTCS_IDLE —> NTTCS_TRACKING —> NTTCS_STOP —> NTTCS_IDLE

a）执行NORMAL_TRAJ类型的轨迹时，若行走过程中发现偏差过大，会在NTTCS_TRACKING 状态下停止下发速度，LPP收到轨迹执行为错误的状态，进行重新规划。（与此同时，此过程中NTT会一直返回）

b）执行NORMAL_TRAJ类型的轨迹时，轨迹的起始或者结束的过程中速度比较小的时候会进行角度检查，偏差超过0.2 rad的时候，NTTCS_TRACKING —> NTTCS_TURNING —> NTTCS_PAUSE —> NTTCS_IDLE；若角度偏差大于0.2 rad，且位置误差大于0.1m时，NTTCS_TRACKING —> NTTCT_ADJUSTERROE —> NTTCS_STOP —> NTTCS_IDLE。

1.2）正常执行一条TURN_TRAJ类型的轨迹时，状态切换如下：

NTTCS_IDLE —> NTTCS_TURNING —> NTTCS_PAUSE —> NTTCS_IDLE

2）**ntt_error_code** ：

故障代码，目前来说暂时没有加入，后期会对非正常的轨迹跟踪都加入故障代码

4.1.2.4 NTT控制器反馈分析

如上图，

1）**current_traj_status**：

```
enum TRAJECTORY_STATUS{
  TRAJ_STATUS_IDLE=-1,
  TRAJ_STATUS_DOING=0,
  TRAJ_STATUS_DONE=1,
  TRAJ_STATUS_ERROR=2       //lpp receive this ,should replan
};
```

刚上电的时候current_traj_status为TRAJ_STATUS_IDLE；收到LPP下发的轨迹后状态变为轨迹执行中TRAJ_STATUS_DOING；轨迹执行正常结束后状态变为TRAJ_STATUS_DONE；执行过程中，有任何非正常的结束会将状态置为TRAJ_STATUS_ERROR，LPP收到TRAJ_STATUS_ERROR状态需要进行重规划。

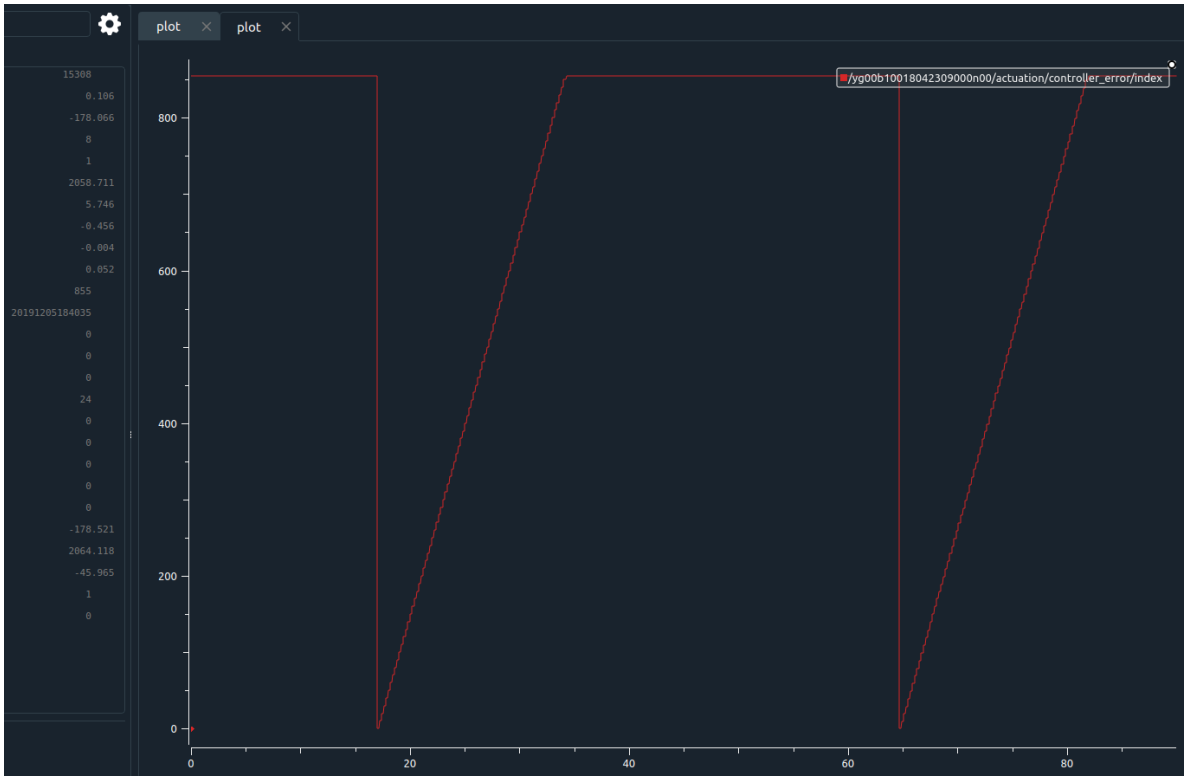
正常：TRAJ_STATUS_IDLE —> TRAJ_STATUS_DOING —> TRAJ_STATUS_DONE —> TRAJ_STATUS_DOING —> TRAJ_STATUS_DONE

异常：TRAJ_STATUS_IDLE —> TRAJ_STATUS_DOING —> TRAJ_STATUS_ERROR —> LPP replan —> send a new trajectory to NTT —> TRAJ_STATUS_DOING —> TRAJ_STATUS_DONE

说明：4.1.2.1 小节中的**current_traj_seq**也是给LPP的一个反馈，LPP用此来判断是哪条轨迹结束。

2）**index**：

反馈当前周期NTT发布给LPP的，NTT控制器里面执行的轨迹点的index信息。此Index在LPP中也用于判断当前轨迹是否运行结束。


plot
plot
15308
0.106
-178.066
8
1
2058.711
5.746
-0.456
-0.004
0.052
855
20191205184035
0
0
0
24
0
0
0
0
0
-178.521
2064.118
-45.965
1
0
/yg00b10018042309000n00/actuation/controller_error/index
800
600
400
200
0
0
20
40
60
80