为底盘添加一阶滤波器，提升丝滑性

电控组 陈卓勋 2022年4月4日

1. 背景

假期后，最新版停滞在v30.14视觉联合测试附近，机器人底盘运动非常卡顿突兀。新建了一个gitee仓库存放最新的版本，用于队员上车调试PID、视觉组测试等。然而一直没来得及细细思考以及对底盘控制控制进行较大的改进。队友们在PID参数的调试上对于底盘运动丝滑性的改进效果不明显。2022年4月4日与电控组队员讨论后，突然想起官方步兵first\_order……之类的函数，会不会就是数字滤波算法。于是打开官方步兵代码，阅读底盘控制逻辑，明白了这就是一阶滤波的应用，于是应用到SAS\_Code中。

1. 编写过程

由于使用的是官方框架，添加滤波器相关的头文件后，新增底盘滤波函数和相关变量。以下增添均在naïve\_chassis\_task.c中

在总的速度控制变量中新增滤波器

//机器人控制量

struct RobotControl\_s{

    fp32 vx,vy,w;

    first\_order\_filter\_type\_t vx\_filter,vy\_filter,w\_filter;

    enum MovingAxis\_e axis;

};

增加滤波器初始化函数

static void initFilters(void)

{

    const static fp32 chassis\_x\_order\_filter[1] = {CHASSIS\_ACCEL\_X\_NUM};

    const static fp32 chassis\_y\_order\_filter[1] = {CHASSIS\_ACCEL\_Y\_NUM};

    const static fp32 chassis\_w\_order\_filter[1] = {CHASSIS\_ACCEL\_W\_NUM};

    first\_order\_filter\_init(&(robotTotalSpeedControl.vx\_filter), CHASSIS\_CONTROL\_TIME\_MS, chassis\_x\_order\_filter);

    first\_order\_filter\_init(&(robotTotalSpeedControl.vy\_filter), CHASSIS\_CONTROL\_TIME\_MS, chassis\_y\_order\_filter);

    first\_order\_filter\_init(&(robotTotalSpeedControl.w\_filter), CHASSIS\_CONTROL\_TIME\_MS, chassis\_w\_order\_filter);

}

增加每次使用的底盘滤波函数

static void firstOrderFilt()

{

    first\_order\_filter\_cali(&robotTotalSpeedControl.vx\_filter,robotTotalSpeedControl.vx);

    first\_order\_filter\_cali(&robotTotalSpeedControl.vy\_filter,robotTotalSpeedControl.vy);

    first\_order\_filter\_cali(&robotTotalSpeedControl.w\_filter,robotTotalSpeedControl.w);

    robotTotalSpeedControl.vx=robotTotalSpeedControl.vx\_filter.out;

    robotTotalSpeedControl.vy=robotTotalSpeedControl.vy\_filter.out;

    robotTotalSpeedControl.w=robotTotalSpeedControl.w\_filter.out;

}

所使用的常量

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*数字滤波所需常量\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*// 初始系数和控制方法来自官方步兵，系数已经根据实际步兵情况做了修改

#define CHASSIS\_ACCEL\_X\_NUM 300.1666f

#define CHASSIS\_ACCEL\_Y\_NUM 300.1666f

#define CHASSIS\_ACCEL\_W\_NUM 300.66f

1. 调试过程和教训

主要是调节系数。一开始的系数值为0.16，以为越小越丝滑，于是往下调，直到了0.0001，但结果却让底盘运动很突兀，像是没有用滤波器一样。不知道怎么回事，尝试自己编写别的滤波算法。

看了一些数学公式，回去把官方的滤波函数源码找出来，发现

void first\_order\_filter\_cali(first\_order\_filter\_type\_t \*first\_order\_filter\_type, fp32 input)

{

    first\_order\_filter\_type->input = input;

    first\_order\_filter\_type->out =

        first\_order\_filter\_type->num[0] / (first\_order\_filter\_type->num[0] + first\_order\_filter\_type->frame\_period) \* first\_order\_filter\_type->out + first\_order\_filter\_type->frame\_period / (first\_order\_filter\_type->num[0] + first\_order\_filter\_type->frame\_period) \* first\_order\_filter\_type->input;

}

写成更短的式子就是

其中，a是我们在宏中给的系数，在滤波器初始化时，被放到了num[0]中作为参数。

直观的物理图像应该是：a越大，输出值越偏向于原来的out，也就是越难加速。因此，a越大，对于加速度的限制越强，越丝滑。把a改成10000，按下小陀螺模式，机器人很久才慢慢动起来，于是验证了此物理图像。于是下调系数，到了300左右，机器人运动较为合适，丝滑，无较强振动。

所以在理论之外，需要一些定性的、偏于物理图像的思考和尝试。

另外，调节了一下云台PID系数，拨弹电机控制系数，将相关结果放在了v30.15\_2022年4月4日中。