Smooth Control工程说明

1.基础功能

smooth_control_node是一个ROS节点，用来处理所有未经处理的速度"/untreated_cmd_vel"，处理后将发出速度话题"/cmd_vel"给HIghLevel的控制节点。它有以下基础功能：

1）限制最大/小速度、打滑临界速度，且速度临界值可调

2）进行匀加速/匀减速，且加速度可调

3）内部提供计时器，监管运动规划节点是否有速度发出或其他异常

4）当运动规划节点关闭或出现其他异常导致一段时间内没有速度发出时，将机器人速度匀减速到0，目前设置为0.2S

5）降低最终速度发出的精度，目前为0.01

6）按照一定频率发布速度话题"/cmd_vel"目前频率是20HZ

2.技术原理

2.1 监管

订阅运动规划节点发出的未经处理的速度话题"/untreated_cmd_vel"，实时监听是否有速度消息发出。通过当前系统时间与上一次收到速度消息的时间差是否大于0.2S来判断是否有其他异常，若没有异常则正常进行速度平滑，否则静止（平滑减速到0）。

$$T_{now} - T_{last-time} > 0.2$$

2.2 速度平滑

1）速度约束

$$|V_{cmd-vel}| >= |V_{min}|$$
$$|V_{cmd-vel}| <= |V_{max}|$$

2）匀加速/匀减速运动

按照20HZ，即0.05s来进行匀加速/匀减速运动

$$V_{cmd-vel} = V_{last-vel} + a * \frac{1}{ROS - RATE - HZ}$$

3）平移速度约束（打滑）

机器人当前曲率小于等于机器人可执行的最大曲率

$$C_{robot} = |V_{linear}/V_{angular}|$$
$$C_{robot} <= C_{max}$$

3.代码说明

计时器 C++ 复制代码

```cpp
ros::Time twist_clock_; //计时器

// 进入速度回调函数
void CmdVelCallBack(const geometry_msgs::Twist::ConstPtr& twist)
{
    twist_clock_ = ros::Time::now(); // 在未经处理的速度回调中记录收到速度的系统时间
    target_vel_.linear = twist->linear; // target_vel_为目标速度
    target_vel_.angular = twist->angular;
}

inline bool isTwistPubContinuous(ros::Time last_sub_time)
{
    //计时器，判断距离上一次进入速度回调的时间是否大于0.2秒
    if ((ros::Time::now() - last_sub_time).toSec() > 0.2)
        return false;
    return true;
}
```

参数 Plain Text 复制代码

```
double max_linear_velocity_ = 0.3; //最大线速度
double min_linear_velocity_ = 0.05; // 最小线速度
double max_rotation_velocity_ = 1; // 最大角速度
double min_rotation_velocity_ = 0.01; // 最小角速度
double max_x_linear_acceleration_ = 0.025; // 最大x方向加速度
double max_rotation_acceleration_ = 0.1; // 最大角加速度
double max_translational_velocity_ = 1; // 最大平移速度
```

速度约束 C++ 复制代码

```
// 线速度不大于最大线速度
if(fabs(objective_vel.linear.x) > fabs(max_linear_velocity_))
    objective_vel.linear.x = ComputeSign(objective_vel.linear.x) * max_linear_velocity_;
// 线速度不小于最小线速度
if(fabs(objective_vel.linear.x) < fabs(min_linear_velocity_))
    objective_vel.linear.x = 0;
// 线速度不大于最大角速度
if(fabs(objective_vel.angular.z) > fabs(max_rotation_velocity_))
    objective_vel.angular.z = ComputeSign(objective_vel.angular.z) * max_rotation_velocity_;
// 线速度不小于最小角速度
if(fabs(objective_vel.angular.z) < min_rotation_velocity_)
    objective_vel.angular.z = 0;
```

速度平滑 C++ 复制代码

```cpp
// 加速
if(objective_vel.linear.x > last_vel_.linear.x)
{
    // 加速后不大于最大线速度
    if (last_vel_.linear.x + max_x_linear_acceleration_ <= objective_vel.linear.x)
    {
        // 加速后不小于最小线速度
        if (last_vel_.linear.x != 0 && ComputeSign(objective_vel.linear.x * last_vel_.linear.x) != ComputeSign(objective_vel.linear.x * (last_vel_.linear.x + max_x_linear_acceleration_)))
            objective_vel.linear.x = 0;
        else
            // 匀加速
            objective_vel.linear.x = last_vel_.linear.x + max_x_linear_acceleration_;
    }
}
// 减速
else if(objective_vel.linear.x < last_vel_.linear.x)
{
    // 加速（加速度<0）后不大于最大线速度
    if (last_vel_.linear.x - max_x_linear_acceleration_ >= objective_vel.linear.x)
    {
        // 加速（加速度<0）后不小于最小线速度
        if (last_vel_.linear.x != 0 && ComputeSign(objective_vel.linear.x * last_vel_.linear.x) != ComputeSign(objective_vel.linear.x * (last_vel_.linear.x - max_x_linear_acceleration_)))
            objective_vel.linear.x = 0;
        else
            // 匀减速
            objective_vel.linear.x = last_vel_.linear.x - max_x_linear_acceleration_;
    }
}
```

最大平移速度 C++ 复制代码

```cpp
// 机器人可执行的最大曲率
double velocity_rotation_ratio = max_translational_velocity_ / max_rotation_velocity_;
// 机器人可执行的最大曲率
double max_moving_rotation_velocity =
    /*max_rotation_velocity_ * (velocity_rotation_ratio - fabs(objective_vel.angular.z / max_rotation_velocity_));*/
    INT_MAX; // 目前不考虑打滑
// 平移速度约束
if(fabs(objective_vel.linear.x) > fabs(max_moving_rotation_velocity))
    objective_vel.linear.x = ComputeSign(objective_vel.linear.x) * max_moving_rotation_velocity;
```

4.测试

4.1 测试说明

打开一个新终端，输入：

流程 Plain Text 复制代码

```
$ roscore
$ cd catkin_ws/
$ catkin_make
$ source devel/setup.sh
$ rosrun smooth_spline smooth_spline
```

4.2 测试效果

正常进行速度平滑与监控。

目前调参需要重新编译代码，暂不支持动态调参。