

## 【运动控制】Apollo6.0的pid\_controller解析




隐匿潮水之中

其修远路漫，而上下求索。

已关注

5 人赞同了该文章

### pid\_controller解析

 目录 寸Apollo 6.0的纵向PID控制模块解析。

【运动控制】Apollo6.0的pid\_controller  
解析\_yanghq13的博客-CSDN博客

[blog.csdn.net/weixin\\_44041199/article/details/104404119](http://blog.csdn.net/weixin_44041199/article/details/104404119)



### 1 pid\_controller.h

```
#pragma once
#include "modules/control/proto/pid_conf.pb.h"

namespace apollo {
namespace control {

class PIDController {
public:

    void Init(const PidConf &pid_conf); // 初始化

    void SetPID(const PidConf &pid_conf); // 设置PID

    void Reset(); // 重设

    virtual double Control(const double error, const double dt); // 暂时未

    virtual ~PIDController() = default;

    int IntegratorSaturationStatus() const; // 积分饱和状态

    bool IntegratorHold() const; // 积分保持?

    void SetIntegratorHold(bool hold); // 设置积分保持?

protected:
    double kp_ = 0.0;
    double ki_ = 0.0;
    double kd_ = 0.0;
    double kaw_ = 0.0;
    double previous_error_ = 0.0;
    double previous_output_ = 0.0;
    double integral_ = 0.0;
```

▲ 赞同 5 ▼

● 2 条评论

➦ 分享

♥ 喜欢

★ 收藏

📄 申请转载

...

```
bool first_init_ = false;
bool integrator_enabled_ = false;
```

```

bool integrator_hold_ = false;
int integrator_saturation_status_ = 0;
// Only used for pid_BC_controller and pid_IC_controller
double output_saturation_high_ = 0.0;
double output_saturation_low_ = 0.0;
int output_saturation_status_ = 0;
};

} // namespace control
} // namespace apollo

```

## 2 pid\_controller.cc

### 2.1 Init

```

void PIDController::Init(const PidConf &pid_conf) {
    previous_error_ = 0.0; // 前一时刻误差
    previous_output_ = 0.0; // 前一时刻输出
    integral_ = 0.0; // 积分
    first_hit_ = true; // 第一次脉冲, 即第一次计算
    integrator_enabled_ = pid_conf.integrator_enable(); // 积分使能
    integrator_saturation_high_ =
        std::fabs(pid_conf.integrator_saturation_level()); // 积分饱和上限
    integrator_saturation_low_ =
        -std::fabs(pid_conf.integrator_saturation_level()); // 积分饱和下限
    integrator_saturation_status_ = 0; // 积分饱和状态
    integrator_hold_ = false; // 积分保持
    output_saturation_high_ = std::fabs(pid_conf.output_saturation_level());
    output_saturation_low_ = -std::fabs(pid_conf.output_saturation_level());
    output_saturation_status_ = 0; // 输出饱和状态
    SetPID(pid_conf); // 设置PID
}x(k)

```

### 2.2 SetPID

```

void PIDController::SetPID(const PidConf &pid_conf) {
    kp_ = pid_conf.kp(); // Kp
    ki_ = pid_conf.ki(); // Ki
    kd_ = pid_conf.kd(); // Kd
    kaw_ = pid_conf.kaw(); // 这个参数保有疑问
}

```

### 2.3 Reset

```

void PIDController::Reset() {
    previous_error_ = 0.0;
    previous_output_ = 0.0;
    integral_ = 0.0;
    first_hit_ = true;
    integrator_saturation_status_ = 0;
    output_saturation_status_ = 0;
}

```

## 2.4 Control

pid控制的主程序control流程如下：

1. 判断步长是否小于零，若小于零则警告，并返回上一输出 `previous_output_` ；
2. 判断是否是 `first_hit_`，若是则将 `first_hist_` 置为false，否则计算
$$diff = (error - error_{previous}) / dt$$
3. 判断积分器使能 `integrator_enabled_`，假则将 `integral_` 置零，真则计算`integral_`，在积分前应用 `Ki` 以避免在稳态时改变 `Ki` 时出现阶跃；
$$integral_ += error * dt * ki$$
4. 积分幅值判断：高于积分饱和上限，则等于积分饱和上限，并将积分饱和状态置1；低于积分饱和下限，则等于积分饱和下限，并将积分饱和状态置-1；其余情况积分饱和状态置0；
5. 保留这一时刻误差，`previous_error_ = error`；输出u幅值判断：高于输出饱和上限，则等于输出饱和上限，并将输出饱和状态置1；低于输出饱和下限，则等于输出饱和下限，并将输出饱和状态置-1；其余情况输出饱和状态置0；
6. 计算输出值，如下
$$output = error * kp + integral + diff * kd$$
7. 保留这一时刻输出，`previous_output_ = output`。

```
double PIDController::Control(const double error, const double dt) {
    if (dt <= 0) {
        AWARN << "dt <= 0, will use the last output, dt: " << dt;
        return previous_output_;
    }
    double diff = 0;
    double output = 0;

    if (first_hit_) {
        first_hit_ = false;
    } else {
        diff = (error - previous_error_) / dt;
    }
    // integral hold
    if (!integrator_enabled_) {
        integral_ = 0;
    } else if (!integrator_hold_) {
        integral_ += error * dt * ki_;
        // apply Ki before integrating to avoid steps when change Ki at step
        if (integral_ > integrator_saturation_high_) {
            integral_ = integrator_saturation_high_;
            integrator_saturation_status_ = 1;
        } else if (integral_ < integrator_saturation_low_) {
            integral_ = integrator_saturation_low_;
            integrator_saturation_status_ = -1;
        } else {
            integrator_saturation_status_ = 0;
        }
    }
    // 输出u幅值判断：高于输出饱和上限，则等于输出饱和上限，并将输出饱和状态置1；低于
    previous_error_ = error;
    output = error * kp_ + integral_ + diff * kd_; // Ki already applied
    previous_output_ = output;
    return output;
}
```

### 3 pid\_BC\_controller.cc

这里BC指的是 backward calculation 后向计算。流程如下

1. 判断步长是否小于零，若小于零则警告，并返回上一输出 `previous_output_` ；
2. 判断是否是 `first_hit_`，若是则将 `first_hist_` 置为false，否则计算
$$diff = (error - error_{previous}) / dt$$
3. 判断积分器使能 `integrator_enabled_`，假则将 `integral_` 置零，真则计算
$$u = error * kp + integral + error * dt * ki + diff * kd$$
4. Clamp函数判断(u, `output_saturation_high_`, `output_saturation_low_`)-u
5. awterm判断，如果大于1e-6，说明awterm>u，在Clamp函数中输出的是u\_min，小于饱和状态下限，输出饱和状态为-1；如果小于-1e-6，说明awterm<u，在Clamp函数中输出的是u\_max，大于饱和状态上限，输出饱和状态为1；（源代码正确，谢谢 @一大群蜗牛 的指正）
6. 计算积分值
$$integral += kawa_{term} + error dt$$
7. a保留这一时刻误差， `previous_error_ = error`；计算输出值，如下
$$output = error * kp + integral + diff * kd$$
8. 保留这一时刻输出， `previous_output_ = output`。

反馈抑制抗饱和 (back-calculation)

参考: <https://zhuanlan.zhihu.com/p/49572763>

基本思想：当饱和时，对积分项加入负反馈，使其尽快退出饱和。

其中Kb设置的范围为0.3~3\*Ki/Kp, 默认的为Ki/Kp即可。

[illegible]

```

        u;
    if (aw_term > 1e-6) {
        output_saturation_status_ = -1;
    } else if (aw_term < -1e-6) {
        output_saturation_status_ = 1;
    } else {
        output_saturation_status_ = 0;
    }
    integral_ += kaw_ * aw_term + error * dt;
}

previous_error_ = error;
output = common::math::Clamp(error * kp_ + integral_ + diff * kd_,
                             output_saturation_high_,
                             output_saturation_low_); // Ki already
previous_output_ = output;
return output;
}

```

#### 4 pic\_IC\_controller.cc

这里的IC是指 integral clamping，是指积分限定（夹紧）。

流程如下

1. 判断步长是否小于零，若小于零则警告，并返回上一输出 `previous_output_`；
2. 判断是否是 `first_hit_`，若是则将 `first_hist_` 置为false，否则计算  

$$diff = (error - error_{previous}) / dt$$
3. 判断积分器使能 `integrator_enabled_`，假则将 `integral_` 置零，真则计算  

$$u = error * kp + integral + error * dt * ki + diff * kd$$
4. 判断：如果 $error * u$ 大于零，且 $u$ 大于输出饱和上限或者 $u$ 小于输出饱和下限，应该是维持积分值不变；反之计算  

$$integral += error * dt * ki$$
5. 计算积分值  

$$integral += kaw_{term} + error dt$$
6. a保留这一时刻误差，`previous_error_ = error`；计算输出值，如下  

$$output = error * kp + integral + diff * kd$$
7. 输出 $u$ 幅值判断：高于输出饱和上限，则等于输出饱和上限，并将输出饱和状态置1；低于输出饱和下限，则等于输出饱和下限，并将输出饱和状态置-1；其余情况输出饱和状态置0；
8. `Clamp(error * kp_ + integral_ + diff * kd_, output_saturation_high_, output_saturation_low_)`；
9. 保留这一时刻输出，`previous_output_ = output`。

积分遇限削弱法(clamping)

参考：<https://zhuanlan.zhihu.com/p/49572763>

基本思想：当执行器处于饱和、且误差信号与控制信号同方向（同号）时，积分器停止更新（其值保持不变），除此之外，积分器正常工作。即，在饱和情况下，只进行有助于削弱饱和程度的积分运算。

积分环节为

$$u_i(k) = u_i(k-1) + T \times k_i \times e(k) \times f(k)$$

其中

$$f(k) = \begin{cases} 0 & (|u(k-1)| \geq u_{\max}) \wedge (u(k-1) \times e(k) \geq 0) \\ 1 & \text{else} \end{cases}$$

```
double PIDICController::Control(const double error, const double dt) {
    if (dt <= 0) {
        AWARN << "dt <= 0, will use the last output";
        return previous_output_;
    }
    double diff = 0;
    double output = 0;

    if (first_hit_) {
        first_hit_ = false;
    } else {
        diff = (error - previous_error_) / dt;
    }
    // integral clamping
    if (!integrator_enabled_) {
        integral_ = 0;
    } else {
        double u = error * kp_ + integral_ + error * dt * ki_ + diff * kd_;
        if (((error * u) > 0) &&
            ((u > output_saturation_high_) || (u < output_saturation_low_)))
        {
            // Only update integral then
            integral_ += error * dt * ki_;
        }
    }

    previous_error_ = error;
    output = error * kp_ + integral_ + diff * kd_;
```

```
if (output >= output_saturation_high_) {
    output_saturation_status_ = 1;
} else if (output <= output_saturation_low_) {
    output_saturation_status_ = -1;
} else {
    output_saturation_status_ = 0;
}

output = common::math::Clamp(error * kp_ + integral_ + diff * kd_,
                             output_saturation_high_,
                             output_saturation_low_); // Ki already
previous_output_ = output;
return output;
}
```

编辑于 2021-09-15 16:36

「真诚赞赏，手留余香」

赞赏

还没有人赞赏，快来当第一个赞赏的人吧！

PID Controller PID控制

控制算法

控制系统

文章被以下专栏收录



自动驾驶

主要为学习自动驾驶技术的朋友们分享一些见解

推荐阅读

### 位置型PID代码实现

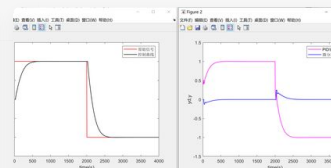
//第一步：定义PID变量结构体，代码如下： struct \_pid{ float SetSpeed; //定义设定值 float ActualSpeed; //定义实际值 float err; //定义偏差值 float err\_last; //定义上一个偏差值 flo...

艳阳高照

### PID控制器调参工具——DR-PID Tuning(Matlab GUI)

介绍调参工具前，简单讨论一下PID控制器。抗扰PID控制研讨QQ群：878392613 一、PID困境受益于PID控制器的简单，我们在控制工程中往往首选PID。受限于PID调参的困难，我们常常质疑PID控制...

Sukung实验室



### PID库与PID基本优化（四）

韭菜的菜

### 控制器设计怎么做

最近看到一个不错的网站，讲解了一些控制教程，感觉很好，文章翻译自此网站， Control Tutorials For Matlab & Simulink .>>PID 选项卡目录一、PID概述二、比例项P，微分项D，积...

刘扬扬

2 条评论

⇌ 切换为时间排序

写下你的评论...



一大群蜗牛

2021-08-11

同学你好，看了你的文章写的很好，有个地方有一点不同见解，也不是很重要：

3 `pid_BC_controller.cc`中：

5.`aw_term`判断，如果大于 $1e-6$ ，输出饱和状态为-1；如果小于 $1e-6$ ，输出饱和状态为1；其余为0；(这里应该写错了，应该是上限状态为1，下限状态为-1)

应该没有错，可以自己手写一下试试，`u`超过上限，`aw_term`=上限-`u` <  $1e-6$  达到上限饱和状态

👍 赞



隐匿潮水之中 (作者) 回复 一大群蜗牛

2021-08-11

谢谢指正，已在原文修改🙏

👍 赞