

LShang001: 帮我写一个用类封装的PID函数，用于arduino框架，包含积分限幅和输出限幅，变量和参数定义清晰

GPT-4: 在编写一个用于Arduino框架的封装好的PID类之前，让我们首先了解一下PID控制器的基本原理。PID控制器是一种常见的反馈回路控制器，广泛应用于工业控制系统中，它通过调整一个或多个控制输入来控制一个过程变量，通常包括比例（Proportional）、积分（Integral）、微分（Derivative）三个参数。

PID控制器的目标是减少系统当前值（PV）与期望值（SP）之间的差值，称为偏差（error）。通过调整比例（P）、积分（I）和微分（D）三个参数，PID控制器能够以最快的速度并且最小的超调来达到或维持在设定点。

下面是一个简单的实现PID控制器的Arduino类，包含了积分限幅和输出限幅的功能。

```
class PIDController {
public:
    // 构造函数
    PIDController(double Kp, double Ki, double Kd, double outputMin, double outputMax)
    : Kp(Kp), Ki(Ki), Kd(Kd), outputMin(outputMin), outputMax(outputMax) {
        integral = 0;
        pre_error = 0;
    }

    // 计算PID值的函数
    double compute(double setpoint, double pv) {
        // 计算偏差
        double error = setpoint - pv;
        // 计算积分项，同时进行积分限幅
        integral += error;
        if (integral < -integralMax) integral = -integralMax;
        else if (integral > integralMax) integral = integralMax;

        // 计算微分项
        double derivative = error - pre_error;

        // 计算输出，同时进行输出限幅
        double output = Kp * error + Ki * integral + Kd * derivative;
        if (output > outputMax) output = outputMax;
        else if (output < outputMin) output = outputMin;

        // 更新错误
        pre_error = error;
    }
};
```

```

        return output;
    }

    // 设置积分限幅值
    void setIntegralLimit(double limit) {
        integralMax = limit;
    }

private:
    double Kp; // 比例系数
    double Ki; // 积分系数
    double Kd; // 微分系数
    double outputMin, outputMax; // 输出限幅值
    double integral; // 积分项
    double pre_error; // 上一次的误差值
    double integralMax = 1000; // 积分限幅的默认值
}

```

在这个类中，我们定义了几个关键的成员变量：

- ``Kp``、``Ki``、``Kd`` 分别表示PID控制器的比例、积分、微分系数。
- ``outputMin``、``outputMax`` 用于限制PID输出值的范围，防止输出值过大。
- ``integral`` 用于累积误差，实现积分作用。
- ``pre_error`` 用于存储上一次的误差，用于计算微分项。
- ``integralMax`` 用于限制积分项的最大值，防止积分饱和现象。

通过调用 ``compute`` 方法，传入当前的设定点（setpoint）和过程变量（pv），即可计算出当前的PID输出值。在实际应用中，这个输出值可以用来调整系统的控制输入，以达到控制目标。

这个类还提供了一个 ``setIntegralLimit`` 方法，允许用户自定义积分限幅值，以适应不同的控制需求。

希望这个简单的PID控制器类能够帮助你更好地理解 and 实现PID控制逻辑。在实际应用中，你可能还需要根据具体情况调整PID参数，以达到最佳的控制效果。

LShang001: 结合示例代码说明如何使用

GPT-4: 当然！我来详细说明如何使用上面提供的 ``PIDController`` 类在Arduino项目中实现PID控制。

首先，你需要在Arduino项目中包含这个类的定义。通常，你会将类定义放在项目的顶部或者一个单独的头文件中。

```
// PIDController.h
#ifndef PIDController_h
#define PIDController_h

class PIDController {
    // ... 类定义 ...
};

#endif
```

然后，在你的Arduino主程序文件中包含这个头文件，并且创建一个 `PIDController` 对象。

```
#include "PIDController.h"

// 设置PID参数
double Kp = 2.0; // 比例系数
double Ki = 0.5; // 积分系数
double Kd = 1.0; // 微分系数

// 设置输出限幅
double outputMin = -255; // 最小输出值
double outputMax = 255; // 最大输出值

// 创建PID控制器对象
PIDController pid(Kp, Ki, Kd, outputMin, outputMax);
```

在你的 `setup()` 函数中，你可能需要初始化一些变量或者硬件设置。

```
void setup() {
    // 初始化代码
}
```

然后，在 `loop()` 函数中，你可以读取传感器数据或者其他输入，计算PID输出，并将这个输出用于控制你的系统。

```
void loop() {
    // 读取系统当前值，例如温度、位置等
    double pv = analogRead(A0); // 假设使用模拟输入A0

    // 定义期望的设定点
    double setpoint = 100; // 待控制的目标值
```

```
// 计算PID输出
double output = pid.compute(setpoint, pv);

// 使用PID输出来控制系统，例如调整电机速度等
analogWrite(9, output); // 假设使用PWM输出到引脚9

// 可能需要一定的延迟来稳定读数
delay(100);
}
```

在上面的 `loop()` 函数中，我们首先读取了一个模拟输入作为过程变量（pv），这个值代表了我们希望控制的系统当前的状态。然后，我们定义了一个设定点（setpoint），这是我们希望系统达到的目标值。

调用 `pid.compute(setpoint, pv)` 函数会计算出PID控制器的输出值。这个输出值考虑了比例、积分和微分三个部分，通过PID算法尝试减少偏差。

最后，我们将这个PID输出值用于控制系统，例如通过PWM控制一个电机的速度。这里假设我们使用的是Arduino的PWM输出引脚9。

如果你需要设置积分限幅，可以在创建 `PIDController` 对象后调用 `setIntegralLimit()` 方法：

```
// 在setup()中设置积分限幅
void setup() {
    // 其他初始化代码...

    // 设置积分限幅值
    pid.setIntegralLimit(100);
}
```

这样，你就可以在Arduino项目中使用封装好的PID控制器类来实现复杂的控制逻辑了。记住，为了达到最佳效果，你可能需要根据实际情况调整Kp、Ki、Kd参数。这通常通过实验和调整来完成，直到系统响应按照你的期望来进行。

LShang001： 创建PID控制器对象分别用于roll pitch yaw

GPT-4： 在飞行器控制系统中，通常需要控制其在三个轴向上的角度：横滚（Roll）、俯仰（Pitch）和偏航（Yaw）。每个轴向都可以使用一个PID控制器来实现稳定的控制。下面，我们将创建三个不同的 `PIDController` 对象，分别对应横滚、俯仰和偏航的控制。

首先，确保我们有一个适当的 `PIDController` 类定义。然后，在Arduino代码中，我们分别为Roll, Pitch, Yaw创建三个 `PIDController` 实例。每个实例可以有不同的PID参数，因为不同的控制轴可能需要不同的控制响应。

```
#include "PIDController.h"

// 设置Roll, Pitch, Yaw的PID参数
double Kp_roll = 2.0, Ki_roll = 0.5, Kd_roll = 1.0;
double Kp_pitch = 2.0, Ki_pitch = 0.5, Kd_pitch = 1.0;
double Kp_yaw = 2.0, Ki_yaw = 0.5, Kd_yaw = 1.0;

// 设置输出限幅
double outputMin = -255;
double outputMax = 255;

// 创建Roll, Pitch, Yaw的PID控制器对象
PIDController pidRoll(Kp_roll, Ki_roll, Kd_roll, outputMin, outputMax);
PIDController pidPitch(Kp_pitch, Ki_pitch, Kd_pitch, outputMin, outputMax);
PIDController pidYaw(Kp_yaw, Ki_yaw, Kd_yaw, outputMin, outputMax);

void setup() {
    // 初始化代码, 可能包括传感器校准、串口通信设置等
}

void loop() {
    // 读取飞行器的当前Roll, Pitch, Yaw值
    double current_roll = readRollSensor(); // 读取Roll传感器数据的函数
    double current_pitch = readPitchSensor(); // 读取Pitch传感器数据的函数
    double current_yaw = readYawSensor(); // 读取Yaw传感器数据的函数

    // 定义Roll, Pitch, Yaw的设定点
    double setpoint_roll = 0; // 飞行器横滚角度的目标值
    double setpoint_pitch = 0; // 飞行器俯仰角度的目标值
    double setpoint_yaw = 0; // 飞行器偏航角度的目标值

    // 计算Roll, Pitch, Yaw的PID输出
    double output_roll = pidRoll.compute(setpoint_roll, current_roll);
    double output_pitch = pidPitch.compute(setpoint_pitch, current_pitch);
    double output_yaw = pidYaw.compute(setpoint_yaw, current_yaw);

    // 使用PID输出来控制飞行器的Roll, Pitch, Yaw
    controlRoll(output_roll); // 控制Roll的函数
    controlPitch(output_pitch); // 控制Pitch的函数
    controlYaw(output_yaw); // 控制Yaw的函数
}
```

```

// 延迟一段时间后再次循环，以保持合适的控制频率
delay(20);
}

// 示例函数定义，你需要根据实际情况来实现这些函数
double readRollSensor() {
    // 实现读取Roll角度的逻辑
    return 0.0;
}

double readPitchSensor() {
    // 实现读取Pitch角度的逻辑
    return 0.0;
}

double readYawSensor() {
    // 实现读取Yaw角度的逻辑
    return 0.0;
}

void controlRoll(double output) {
    // 实现控制Roll的逻辑
}

void controlPitch(double output) {
    // 实现控制Pitch的逻辑
}

void controlYaw(double output) {
    // 实现控制Yaw的逻辑
}

```

在上面的代码中，我们首先定义了Roll、Pitch和Yaw的PID参数。在实际应用中，这些参数需要通过实验调整得到最佳的控制效果。然后，我们创建了三个 `PIDController` 对象，每个对象分别用于控制飞行器的Roll、Pitch和Yaw轴。

在 `loop()` 函数中，我们循环读取当前的Roll、Pitch和Yaw角度值，计算出PID的输出，然后将这些输出值用于控制飞行器的相应轴。

请注意，这里的 `readRollSensor`、`readPitchSensor`、`readYawSensor`、`controlRoll`、`controlPitch` 和 `controlYaw` 函数需要你根据你的硬件和需求来具体实现。这些函数应该负责读取传感器数据和控制飞行器的运动。

最后，我们在每个循环结束时加入一个小的延迟，以保持PID控制循环的稳定性和适当的更新频率。

下载 [uTools](#)，从插件市场搜索 “[ChatGPT.好友](#)” 可生成此对话页面！