【运动控制】Apollo6.0的leadlag_controller解析

隐匿潮水之中
其修远路漫，而上下求索。
+ 关注他

leadlag_controller解析

目录

对Apollo 6.0的超前滞后校正模块进行解析。

1 leadlag_controller.h

```
#pragma once

#include "modules/control/proto/leadlag_conf.pb.h"

namespace apollo {
namespace control {

class LeadlagController {
 public:

  // 初始化
  void Init(const LeadlagConf &leadlag_conf, const double dt);

  // 设置leadlag
  void SetLeadlag(const LeadlagConf &leadlag_conf);

  // 双线性变换离散化方法（T型积分：曲线包围面积用梯形估算）
  // 把连续的控制器系数离散化
  void TransformC2d(const double dt);

  void Reset();

  // 超前滞后也是误差输入的控制器
  virtual double Control(const double error, const double dt);

  // 内部状态是什么？

  int InnerstateSaturationStatus() const;

 protected:
  // Control coefficients in contiouous-time domain
  double alpha_ = 0.0;
  double beta_ = 0.0;
```



```
  // control coefficients in discrete-time domain
  double kn1_ = 0.0;
```

```
  double kn0_ = 0.0;
  double kd1_ = 0.0;
  double kd0_ = 0.0;
  // Inner (intermedia) state in discrete-time domain at Direct Form II
  double previous_output_ = 0.0;
  double previous_innerstate_ = 0.0;
  double innerstate_ = 0.0;
  double innerstate_saturation_high_ = 0.0;
  double innerstate_saturation_low_ = 0.0;
  int innerstate_saturation_status_ = 0;
  bool transfromc2d_enabled_ = false;
};

}  // namespace control
}  // namespace apollo
```

2 leadlag_controller.cc

2.1 Init

```
void LeadlagController::Init(const LeadlagConf &leadlag_conf, const dou
  previous_output_ = 0.0; // 前一输出
  previous_innerstate_ = 0.0; // 前一内部状态
  innerstate_ = 0.0; // 内部状态

  innerstate_saturation_high_ =
      std::fabs(leadlag_conf.innerstate_saturation_level());  // 内部状态
  innerstate_saturation_low_ =
      -std::fabs(leadlag_conf.innerstate_saturation_level()); // 内部状态
  innerstate_saturation_status_ = 0; // 内部状态饱和标志
  SetLeadlag(leadlag_conf); // 设置leadlag
  TransformC2d(dt); // 连续转离散
}
void LeadlagController::SetLeadlag(const LeadlagConf &leadlag_conf) {
  alpha_ = leadlag_conf.alpha();
  beta_ = leadlag_conf.beta();
  tau_ = leadlag_conf.tau();
}
```

`TransformC2d` 函数的作用是将连续形式的传递函数转换成离散形式的传递函数。

2.1.1 leadlag传递函数

超前-滞后补偿器的传递函数形式如下：

$$H(s)=\beta\frac{\tau s+1}{\tau\alpha s+1}$$

其中， α 为滞后系数； β 为超前系数； τ 为给定的时间系数。

采用双线性变换，T为采样周期

$$s=\frac{2}{T}\frac{z-1}{z+1}$$

可以得到

$$H(z) = \beta \frac{\tau \frac{2}{T} \frac{z-1}{z+1} + 1}{\tau\alpha \frac{2}{T} \frac{z-1}{z+1} + 1}$$

$$H(z) = \beta \frac{2\tau(z-1) + T(z+1)}{2\tau\alpha(z-1) + T(z+1)}$$

$$H(z) = \beta \frac{(T-2\tau) + (2\tau+T)z}{(T-2\alpha\tau) + (2\alpha\tau+T)z}$$

$$H(z) = \frac{(T\beta-2\beta\tau) + (2\beta\tau+T\beta)z}{(T-2\alpha\tau) + (2\alpha\tau+T)z}$$

$$H(z) = \frac{kn_0 + kn_1 * z}{kd_0 + kd_1 * z}$$

从上述公式来看，和代码里是一致的，如下

- $a1 = \alpha\tau$，$a0 = 1.00\$$，$b1 = \beta\tau$，$b0 = \beta$ ；
- $kn_1 = 2\beta\tau + T\beta$，$kn_0 = T\beta - 2\beta\tau$，$kd_1 = 2\alpha\tau + T$，$kd_0 = T - 2\alpha\tau$ ；

```
void LeadlagController::TransformC2d(const double dt) {
  if (dt <= 0.0) {
    AWARN << "dt <= 0, continuous-discrete transformation failed, dt: '
    transfromc2d_enabled_ = false;
  } else {
    // 公式?
    double a1 = alpha_ * tau_;
    double a0 = 1.00;
    double b1 = beta_ * tau_;
    double b0 = beta_;
    Ts_ = dt;
    // 带入默认参数数值后，dt，dt，dt，dt
    kn1_ = 2 * b1 + Ts_ * b0;
    kn0_ = Ts_ * b0 - 2 * b1;
    kd1_ = 2 * a1 + Ts_ * a0;
    kd0_ = Ts_ * a0 - 2 * a1;
    if (kd1_ <= 0.0) {
      AWARN << "kd1 <= 0, continuous-discrete transformation failed, kd
            << kd1_;
      transfromc2d_enabled_ = false;
    } else {
      transfromc2d_enabled_ = true;
    }
  }
}
```

2.1.2 相位角和幅值

超前-滞后补偿器的传递函数相角为

$$\begin{aligned}\phi(\omega) &= arctan(\tau\omega) - arctan(\alpha\tau\omega) \\ &= arctan\frac{(1-\alpha)\tau\omega}{1+\alpha\tau^2\omega^2}\end{aligned}$$

式中，计算公式参考和差化积

$$arctanA - arctanB = arctan\frac{A-B}{1+AB}$$

最大相角在极点 $\frac{1}{\alpha\tau}$ 和零点 $\frac{1}{\tau}$ 之间，其值大小取决于 α 的大小。

对 $\phi(\omega)$ 求导，得

$$\begin{aligned}\dot{\phi(\omega)} &= \frac{\tau}{1+(\tau\omega)^2} - \frac{\alpha\tau}{1+(\alpha\tau\omega)^2} \\ &= \frac{(\alpha-1)[\alpha(\tau\omega)^2-1]}{[1+(\tau\omega)^2][1+(\alpha\tau\omega)^2]}\end{aligned}$$

令 $\dot{\phi(\omega)}=0$，得

$$\omega_m = \frac{1}{\tau\sqrt{\alpha}}$$

最大相位角为

$$\phi(\omega_m) = arctan\frac{1-\alpha}{2\sqrt{\alpha}}$$

即

$$tan\phi(\omega_m) = \frac{1-\alpha}{2\sqrt{\alpha}}$$

为了便于分析（正切分母有根号项），将其转换为反正形式，有

$$\frac{sin\phi(\omega_m)}{cos\phi(\omega_m)} = \frac{1-\alpha}{2\sqrt{\alpha}} \quad sin^2\phi(\omega_m) + cos^2\phi(\omega_m) = 1$$

可得最大相位角为

$$\phi(\omega_m) = arcsin\frac{1-\alpha}{1+\alpha}$$

在 $\omega_m = \frac{1}{\tau\sqrt{\alpha}}$，其幅值增益为

$$\begin{aligned}|G(j\omega)|_m &= 20\beta lg\sqrt{1+(\tau\frac{1}{\tau\sqrt{\alpha}})^2} - 20\beta lg\sqrt{1+(\alpha\tau\frac{1}{\tau\sqrt{\alpha}})^2} \\ &= -10\beta lg\alpha\end{aligned}$$

这里可以得到，只要leadlag的 β 和 α 为正数，其幅值增益为负，即减小幅值。

2.1.3 参数分析

leadlag主要是超前或滞后(不能同时)，本质就是讨论极点 $\frac{1}{\alpha\tau}$ 和零点 $\frac{1}{\tau}$ 的前后位置，如下：

- 若零点 $\frac{1}{\tau}$ >极点 $\frac{1}{\alpha\tau}$，即 $\alpha<1$，此时为超前校正，作用是提高响应速度，避免引入高频震荡；
- 若零点 $\frac{1}{\tau}$ <极点 $\frac{1}{\alpha\tau}$，即 $\alpha>1$，此时为滞后校正，作用是提高稳态精度，但暂态响应将变慢。

对于增益系数 β，本质上就是讨论leadlag模块串入系统后对系统开环放大系数的影响，有如下情况：

- 若 $0<\beta<1$，会使得系统的开环放大系数下降，即幅值衰减；
- 若 $\beta>1$，会使得系统的开环放大系数上升，即幅值增加。

在工程实践中，可以通过提高系统其他环节的放大系数或增加比例放大器加以补偿，Apollo的leadlag采用后者的处理方法。但是β如果过大，容易引起饱和。

2.2 control

leadlag控制的流程如下：

1. 判断连续转离散(`transfromc2d_enabled_`)是否成功，失败则重新进行 `TransformC2d` ，如果再次失败，则发出警告 `C2d transform failed; will work as a unity compensator` ，并返回错误；
2. 检查步长 `dt` 是否小于等于零。如果小于等于零，则发出警告 `dt <= 0, will use the last output` ，返回上一时刻输出 `previous_output_` ；
3. 计算内部状态 `innerstate_` ，计算公式如下

   $s_{inner}(k) = \frac{e(k) - s_{inner}(k-1) * kd0}{kd1}$

   这个公式看起来很玄乎，进行移位后，如下

   $s_{inner}(k-1) * kd0 + s_{inner}(k) * kd1 = e(k)$

   上述公式不能直观理解，需要进行转换。

   观察离散传递函数，如下

   $\frac{y(z)}{u(z)} = \frac{kn_0 + kn_1 z}{kd_0 + kd_1 z}$ $y(z) = \frac{kn_0 + kn_1 z}{kd_0 + kd_1 z} u(z)$

   令

   $x(z) = \frac{1}{kd_0 + kd_1 z} u(z)$

   即

   $$kd_0 x(k) + kd_1 x(k+1) = u(k)$$
   $$x(k+1) = \frac{u(k) - kd_0 x(k)}{kd_1}$$

   其中， $x(k)$ 为 `innerstate` ， $u(k)$ 为 `error` (即系统输出与参考值的偏差)。

   传递函数变为

   $y(z) = (kn_0 + kn_1 z) x(z)$

   从而得到

   $y(k) = kn_0 x(k) + kn_1 x(k+1)$

   其中， $y(k)$ 为 `output` 。

   因此，这一步按照 $x(k+1) = \frac{u(k) - kd_0 x(k)}{kd_1}$ 计算 `innerstate` 的值。
4. 进行 `innerstate` 幅值判断：高于状态饱和上限，则等于状态饱和上限，并将状态饱和状态置1；低于状态饱和下限，则等于状态饱和下限，并将状态饱和状态置-1；其余情况状态饱和状态置0；
5. 按照 $y(k) = kn_0 x(k) + kn_1 x(k+1)$ ，计算 `output` ；

   注：这里的 $x(k+1)$ 是用 $\frac{u(k) - kd_0 x(k)}{kd_1}$ 计算的，
6. 保存 `innerstate` 和 `output` 变量，即为 `previous_innerstate` 和 `previous_output` 。

```
double LeadlagController::Control(const double error, const double dt)
  // check if the c2d transform passed during the initilization
  // 离散化还有不成功的？什么导致不能离散化的？
  if (!transfromc2d_enabled_) {
    TransformC2d(dt);
    if (!transfromc2d_enabled_) {
      AWARN << "C2d transform failed; will work as a unity compensator,
            << dt;
      return error;  // treat the Lead/Lag as a unity proportional cont
    }
  }
  // check if the current sampling time is valid
  if (dt <= 0.0) {
    AWARN << "dt <= 0, will use the last output, dt: " << dt;
    return previous_output_;
  }
  double output = 0.0;
```

```
  // 为什么这么定义内部状态?
  innerstate_ = (error - previous_innerstate_ * kd0_) / kd1_;  // calcu
  // the inner (intermedia) state under the Direct form II for the Lead
  // compensator factorization
  if (innerstate_ > innerstate_saturation_high_) {
    innerstate_ = innerstate_saturation_high_;
    innerstate_saturation_status_ = 1;
  } else if (innerstate_ < innerstate_saturation_low_) {
    innerstate_ = innerstate_saturation_low_;
    innerstate_saturation_status_ = -1;
  } else {
    innerstate_saturation_status_ = 0;
  }

  output = innerstate_ * kn1_ + previous_innerstate_ * kn0_;
  previous_innerstate_ = innerstate_;
  previous_output_ = output;
  return output;
}
```

3参考资料

3.1 传递函数转换差分方程

可参考传递函数转化为差分方程和 和 超前滞后补偿器的C++实现 。

3.2 传递函数转换状态方程

可参考线性离散系统状态空间表达式，大概涉及到的部分如下：

已知

$$\frac{Y(z)}{U(z)}=H(z)=\frac{kn_0+kn_1z}{kd_0+kd_1z}$$

转换成差分方程如下

$$kd_1y(k+1)+kd_0y(k)=kn_1u(k+1)+kn_0u(k)$$

为了将输出最高项系数置1，得到差分方程如下

$$y(k+1)+\frac{kd_0}{kd_1}*y(k)=\frac{kn_1}{kd_1}*u(k+1)+\frac{kn_0}{kd_1}*u(k)$$

$$y(k+1)+a_1y(k)=b_0u(k+1)+b_1u(k)$$
$$a_1=\frac{kd_0}{kd_1},b_0=\frac{kn_1}{kd_1},b_1=\frac{kn_0}{kd_1}$$

将两边作零初始条件下的Z变换，得到

$$zy(z)+a_1y(z)=b_0zu(z)+b_1u(z)$$

上式可以改写成

$$\frac{y(z)}{u(z)}=\frac{b_0z+b_1}{z+a_1}=b_0+\frac{b_1-b_0a_1}{z+a_1}=b_0+\frac{\beta_1}{z+a_1}$$
$$\beta_1=b_1-b_0a_1=\frac{kn_0}{kd_1}-\frac{kn_1}{kd_1}\frac{kd_0}{kd_1}$$

令 $x(z)=\frac{1}{z+a_1}u(z)$ ，并进行反变换得到

$$x(k+1)+a_1x(k)=u(k)$$

取状态变量

$$x_1(k)=x(k)$$
$$x_2(k)=x_1(k+1)=x(k+1)$$
$$y(k)=\beta_1x(k)+b_0u(k)$$
$$y(k)=\beta_1x_1(k)+b_0u(k)$$

关于$y(k)$的状态变量，需要额外作一些解释，推导如下

$$\frac{y(z)}{u(z)} = b_0 + \frac{\beta_1}{z + a_1}$$
$$y(z) = b_0 u(z) + \frac{\beta_1}{z + a_1} u(z)$$
$$= b_0 u(z) + \beta_1 \frac{u(z)}{z + a_1}$$
$$= b_0 u(z) + \beta_1 \frac{(z + a_1) x(z)}{z + a_1}$$
$$= b_0 u(z) + \beta_1 x(z)$$

故而得到

$$\begin{bmatrix} x_1(k+1) \\ x_2(k+1) \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & -a_1 \end{bmatrix} \begin{bmatrix} x_1(k) \\ x_2(k) \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u(k)$$

$$y(k) = \beta_1 x_1(k) + b_0 u(k)$$

这个系统应该是一阶的，再次转换如下

$$x(k+1) = -a_1 x(k) + u(k)$$

$$y(k) = \beta_1 x(k) + b_0 u(k)$$

其中， u 为补偿器输入(即系统输出与参考值的偏差)， x 为补偿器状态， y 为补偿器输出， k 为步数。

但这个与代码中的不同，所以代码中没有使用离散的状态空间方程。

编辑于 2021-09-15 16:35

自动驾驶 PID Controller PID控制 运动控制

【运动控制】Apollo6.0的lon_controller解析

lon_controller解析本文是对Apollo6.0的纵向控制模块解析。 【运动控制】Apollo6.0的lon_controller解析_yanghq13的博客-CSDN博客 纵向控制算法的步骤： 创建一个纵向控制器在文件control_c...

隐匿潮水之... 发表于自动驾驶

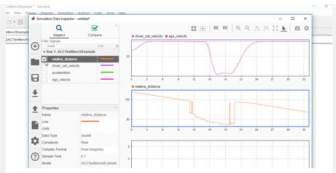
自动驾驶控制算法实例之LQR(以Apollo为例)

涅索斯衬衫

使用模型预测控制和PID实现自动驾驶的车道保持

https://club.leiphone.com/page/... (二维码自动识别) 本文为 AI 研习社编译的技术博客，原标题： Lane keeping in autonomous driving with Model Predictive Control & PID 作者 |...

AI研习社



如何在Simulink中实现基于模型预测控制的传感器融合自适

涅槃汽车

还没有评论

写下你的评论...