课程作业2：在arduino上用PID算法控制电机仿真模拟系统

18机械工程4班 罗宏利 2018G0505025

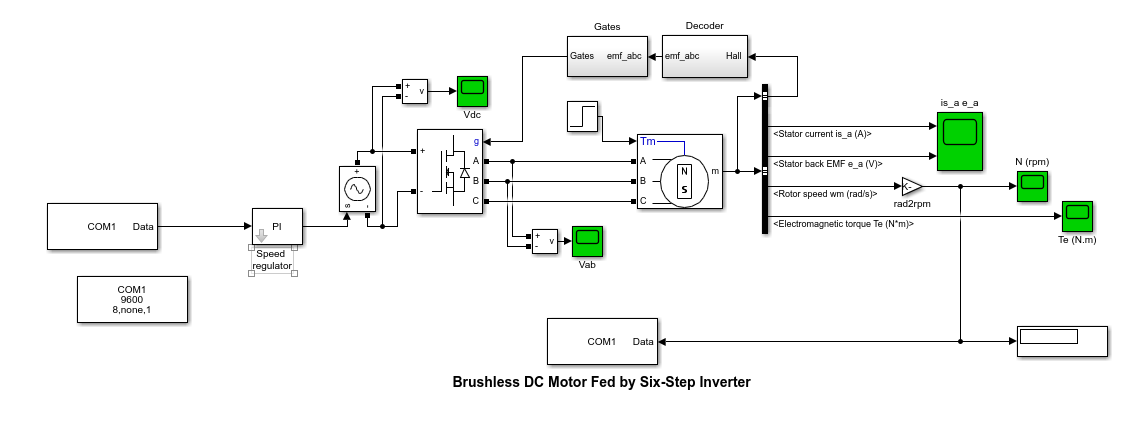
设计思路：

1. 、PC端仿真建模选择Matlab的simulink模块，串口通信及信号处理都比较方便，电机模拟系统也可减少编程的步骤。在simulink仿真中，建立电机模型，同时加入串口接收和串口发送模块，以便和Arduino连接。
2. 、串口数据读取通过Arduino的Serial函数读取，设置为当串口有数据进来时读取数据，数据处理结束后发送回串口。
3. 、电机速度图形利用示波器显示。
4. 、通过PID的误差计算，计算本次误差和上次误差，分析误差方向来改变控制量的大小。
5. 、设置仿真起止时间，设定总时间为2s，设定采样步长为0.1，则需要执行100步，设置Arduini串口数据读取时间为10ms，这样就有充足的时间读取串口数据并将数据返回至simulink，作为合理的同步手段。

实验材料：Arduino UNO板，Matlab/Simulink

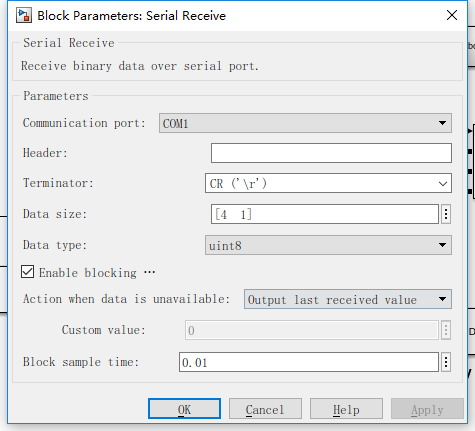
实验过程：

一、首先在Simulink中建立模型系统，本次报告中借用网上下载的完整三相交流电机仿真模型，并加入串口接收、串口发送模块和数据显示模块。仿真系统如图：



图表 1 电机仿真系统模型图

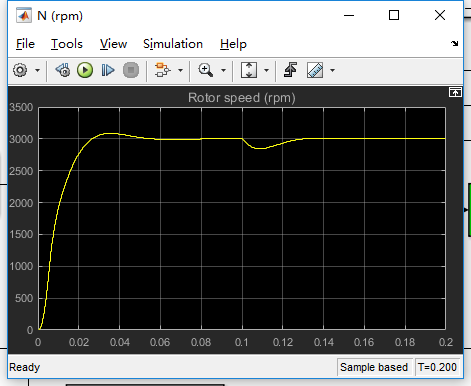
二、设置串口模块主要参数，因Arduino程序里面用的是Serial.prinln()，也就是每个字符串都以换行符结尾。所以选择CR('\r')，表示每组数据读到结尾控制符，但不包含该控制符，也就是在CR('\r')之前的制定位数的数据会被读取。处理数据可能为3位到4位，因此将矩阵设为[4,1],参数设置如下：



图表 2 串口参数设置

三、接下来在Arduino上编译PID算法，利用串口送过来的数据进行PID控制量计算，并计算上次误差和本次误差，计算误差的方向，根据误差计算的结果调整PID控制量的大小。

四、将计算后的结果发送到串口，在simulink中运行并显示，模拟后的电机转速重新通过串口发送给Arduino，循环以上步骤，得出最终图形。PID模拟图形如下图所示：



图表 3 PID模拟图形

反思与总结：

本次设计中，由于采用的是网上的模型，因此仿真系统较为复杂不好使用，事实上仿真模型可以大大简化，采用普通的交流电机对于本次设计可能更为合适，主要包含串口接发模块、电机模块、电源模块等即可。

PID算法中，计算PID控制量时实际上只使用了PI控制，本次设计对于系统反应速度要求较高，对精度要求不高，因此加入微分控制在误差出现前就消除反应会比较合适。

设计中最终的PID模拟图形与实际的PID控制系统图形有较大的偏差，可能是由于设计不完善及缺乏微分控制，应在后续阶段中改进。

主程序：  
#include <Arduino.h>

#include <avr/interrupt.h>

#define SETSpeed 3000 //设置速度

// 数据类型

typedef signed char sint08;

typedef unsigned char uint08;

typedef signed int sint16;

typedef unsigned int uint16;

typedef unsigned long uint32;

typedef signed long sint32;

//全局变量

uint16 u16RTICnt;

typedef struct { // PID

sint08 s08PIDDir; // 1:正向；-1:反向

sint16 s16InpMax, s16InpMin; // 输入量的最大/最小值，用于输入的归一化计算

sint16 s16OutMax, s16OutMin; // 控制量的最大/最小值，用于输出限幅

sint16 s16PIDCmd; // 归一化的命令值，范围为0 ~ 1023

sint16 s16PIDAcq; // 归一化的实际值，范围为0 ~ 1023

sint32 s32PIDCtr; // PID(的放大了256倍)的上一次控制量

sint16 s16PIDCtr; // PID的控制量

sint16 s16PID\_E0, s16PID\_E1; // 本次误差, 上次误差

uint16 u16PID\_Kp, u16PID\_Ki; // PID参数

} TParaPID;

//全局变量设置

TParaPID tPIDMotor;

const TParaPID CstParaPID = {

1, // 1:正向；-1:反向

512, 307, // 输入量的最大/最小值，用于输入的归一化计算

1023 ,0 , // 控制量的最大/最小值，用于输出限幅

0 , // 归一化的命令值，范围为0 ~ 1023

0 , // 归一化的实际值，范围为0 ~ 1023

0 , // PID(的放大了256倍)的上一次控制量

0 , // PID的控制量

0 , 0 , // 本次误差, 上次误差

4 ,40 , // PID参数

};

void InitPID(TParaPID \*tpPID01){ // 初始化PID参数

(void)memcpy(tpPID01, &CstParaPID, sizeof(TParaPID));

}

uint16 PID\_Sub1(TParaPID \*tpPID, sint16 s16V01){ // 归一化计算

uint32 u32Tmp1;

if(tpPID->s16InpMax <= tpPID->s16InpMin) return 0;

if(s16V01 > tpPID->s16InpMax){

s16V01 = tpPID->s16InpMax;

} else if(s16V01 < tpPID->s16InpMin){

s16V01 = tpPID->s16InpMin;

}

u32Tmp1 = s16V01 - tpPID->s16InpMin;

u32Tmp1 = u32Tmp1 \* 1023;

u32Tmp1 = u32Tmp1 / (tpPID->s16InpMax - tpPID->s16InpMin);

return (uint16)u32Tmp1;

}

void PID\_Cal(TParaPID \*tpPID){ // 计算PID控制量

sint32 s32Tmp1, s32Tmp2;

tpPID->s16PID\_E1 = tpPID->s16PID\_E0; // 上次误差

tpPID->s16PID\_E0 = (tpPID->s16PIDCmd - tpPID->s16PIDAcq) // 本次误差，命令值与实测值的差

\* tpPID->s08PIDDir; // 计算误差的方向

s32Tmp1 = ((sint32)tpPID->u16PID\_Kp)

\* ((sint32)(tpPID->s16PID\_E0 - tpPID->s16PID\_E1)); // 比例增量

s32Tmp2 = ((sint32)tpPID->u16PID\_Ki) \* ((sint32)tpPID->s16PID\_E0); // 积分增量

tpPID->s32PIDCtr = tpPID->s32PIDCtr + s32Tmp1 + s32Tmp2; // 控制量

s32Tmp1 = ((sint32)tpPID->s16OutMax) << 16; // 控制量的最大值

s32Tmp2 = ((sint32)tpPID->s16OutMin) << 16; // 控制量的最小值

if (tpPID->s32PIDCtr > s32Tmp1){ tpPID->s32PIDCtr = s32Tmp1;

} else if(tpPID->s32PIDCtr < s32Tmp2){ tpPID->s32PIDCtr = s32Tmp2;

}

tpPID->s16PIDCtr = (sint16)(tpPID->s32PIDCtr >> 16); // PID的控制量,只取整数

return (sint16)(tpPID->s32PIDCtr >> 16); // 只取整数

}

void MyPIDMotorSet(TParaPID \*ptPID03, sint16 s16P1){ // 电机的PID控制, 输入要求的值，不需要反复调用

ptPID03->s16PIDCmd = PID\_Sub1(ptPID03, s16P1);

}

uint08 MyPIDMotorCtr(TParaPID \*ptPID03, sint16 s16P2){ // 电机的PID控制, 输入实际的值，需要反复调用，这是控制过程

if(MyTimerQuery(&ptPID03->tTimer\_ms) == 0) return 0; // 时间到了返回1，否则返回0

ptPID03->s16PIDAcq = PID\_Sub1(ptPID03, s16P2);

PID\_Cal(ptPID03); // 计算PID控制量

return 1;

}

//设置PWM占空比

void Set\_PWM(uint16 Duty)

{

OCR1A=Duty;

}

//PID控制，查询形式，多次访问

void PIDCtl\_Scan(uint16 feedback)

{

TParaPID \*tPID=&tPIDMotor;

if(MyPIDMotorCtr(tPID, feedback) == 1)

// PID控制, 输入实际的值，需要反复调用，这是控制过程

Set\_PWM(tPID->s16PIDCtr); // PID的控制量,此处为占空比

}

uint16 u16val;

void setup() {

Serial.begin(9600); // 开启串口

while(Serial.read()>= 0){}

TParaPID \*tPID=&tPIDMotor;

pinMode(S0, INPUT); // S0 为串口输入

Serial.println(u16val); // 串口输出

InitPID(tPID); // 初始化PID参数

Init\_PWM(); // 初始化PWM

u16val=SETSpeed;

MyPIDMotorSet(tPID, u16val); // 电机的PID控制, 输入要求的值，不需要反复调用

}

void loop()

{

if (Serial.available() > 0) {

delay(10); // 等待数据传完

int numdata = Serial.available();

u16val=Serial.read(); //串口数据，作为速度反馈值

//Serial.println(u16val);

while(Serial.read()>=0){} //清空串口缓存

PIDCtl\_Scan(u16val);

}

}