



清华大学机械工程专业本科生课程

# 机电系统专题实验

## Mechatronic Systems Special Experiment

自主小车嵌入式软件框架与编程实现

任课教师：

尹文生 副教授

[yinws@tsinghua.edu.cn](mailto:yinws@tsinghua.edu.cn)

胡金春 副研究员

[hujinchun@tsinghua.edu.cn](mailto:hujinchun@tsinghua.edu.cn)

# 为什么嵌入式编程？

- 运行速度慢（时钟：72M?——对比：3G以上）
- 硬件资源少（存储：64K整型——对比：8G以上）
- 可用资源少（函数：专用库——对比：控件多）
- 编程工具少（语言：C——对比：傻瓜式语言多）
- 无操作系统（硬件：独占、自由——对比：共享，框架）
- ....

应用需求



求知方向

# 课程内容

- 认识嵌入式编程过程
- 一般嵌入式编程变量管理
- 车控系统软件建议框架
- 车轮转速控制实现
- PID控制律参数与调试

# 课程内容

- 认识嵌入式编程过程
- 一般嵌入式编程变量管理
- 车控系统软件建议框架
- 车轮转速控制实现
- PID控制律参数与调试



# 嵌入式编程过程：预处理、编译、汇编和链接

通过4个过程生成可执行代码，通过仿真器与集成开发环境帮助下下载并运行。

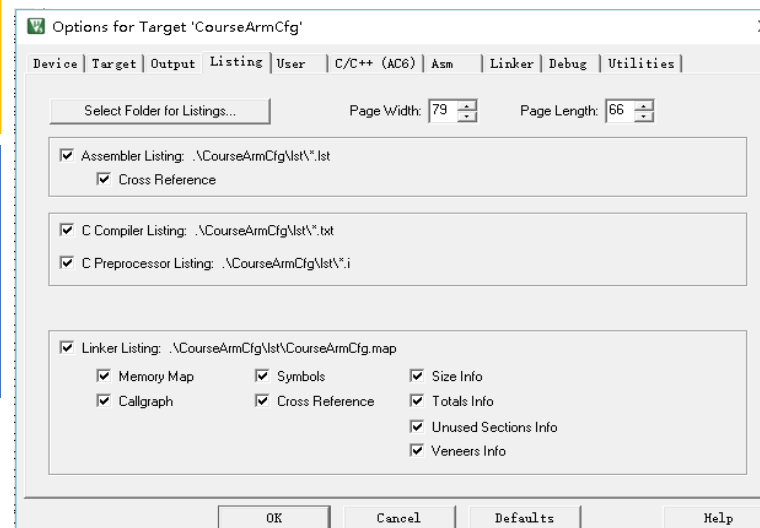


**预处理：**展开所有宏定义，删除所有“#define”；处理所有条件编译指令：“#if”、“#ifdef”、“#elif”、“#else”、“#endif”；处理“#include”预编译指令，将被包含文件插入到指令位置，删除所有注释“//”和“/\* \*/”；添加行号和文件名标识，为调试、编译错误或警告做好准备；保留所有编译器需要的#pragma 编译指令。

**编译：**利用编译程序从源语言编写的源程序产生目标程序的过程(机器语言)，包括词法分析、语法分析、语义检查、中间代码生成、代码优化、目标代码生成共5个阶段

**汇编：**用汇编器将汇编代码转换成机器执行的指令，每一个汇编语句几乎都对应一条机器指令。

**链接：**将各模块相互引用部分正确地衔接起来，包括了地址和空间分配、符号决议和重定向。最基本的链接是将目标文件和库（也是目标文件）一起链接形成最后的可执行文件





# 嵌入式编程过程：4过程中文件的变化



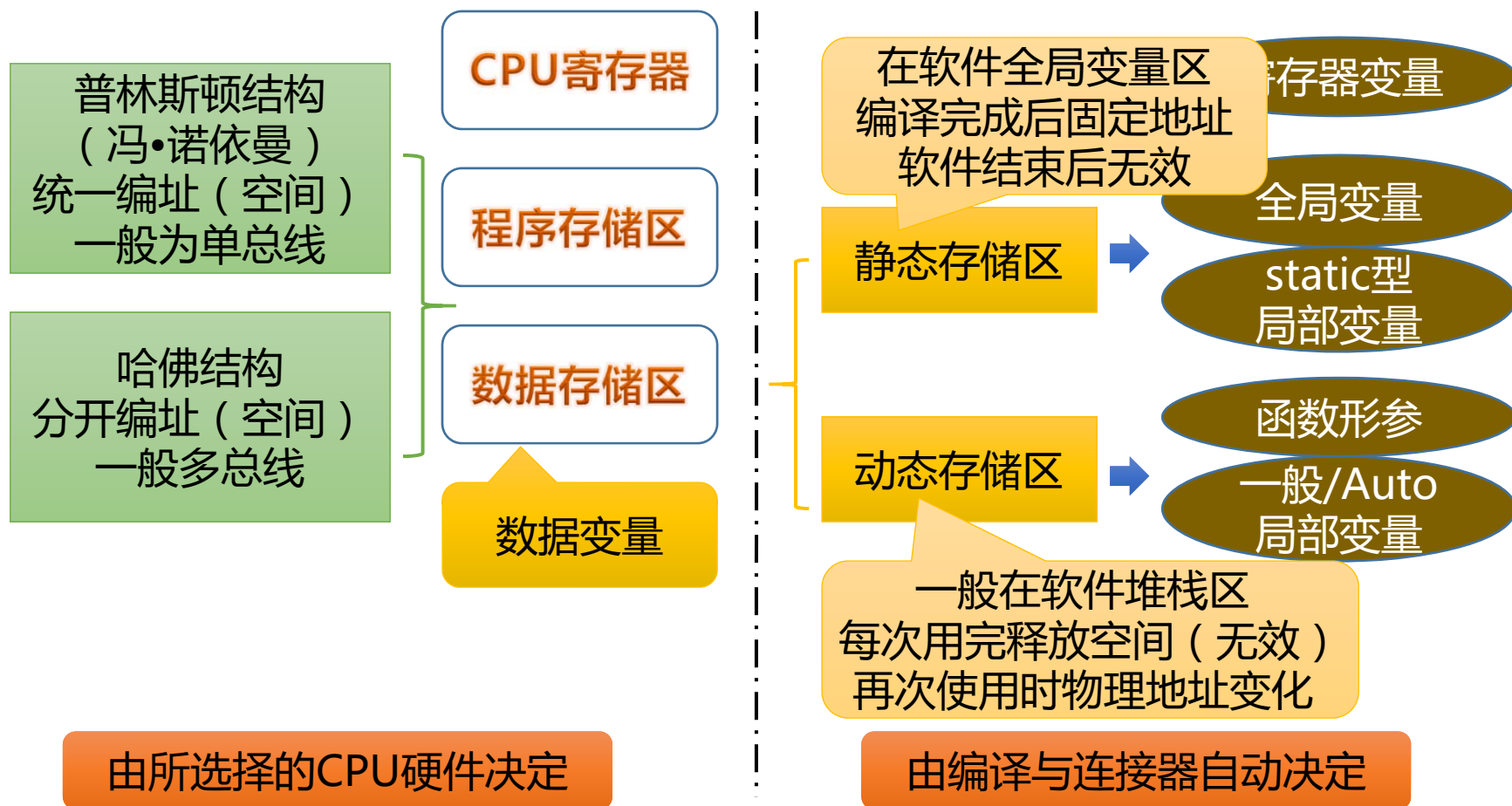
# 课程内容

- 认识嵌入式编程过程
- 认识嵌入式编程变量管理
- 车控系统软件建议框架
- 车轮转速控制实现
- PID控制律的参数与调试



# 嵌入式编程变量：C语言几种数据类型

## • 动态存储与静态存储







# 嵌入式编程变量：变量在内存中的占用

- 避免模块之间复制传递
  - 函数参数调用时被复制，增加存储负担与时间成本
- 变量范围与表示
  - 关注变量的表示范围、符号，避免表示溢出
  - 根据CPU实际支持数值类型与位数合理确定变量类型：8位51单片机尽可能使用char（有符号）与unsigned char(无符号)变量，float变量计算极其费CPU时间
- 连续运算可能导致编译器错误编译（并不提示）
  - 如将 $k=a*b*c*d$ 替换为 $k1=a*b$ ;  $k2=c*d$ ;  $k=k1*k2$
- 嵌入式CPU存储空间较小，避免越界带来的程序“乱窜”
  - 注意数组所定义空间大小
  - 合理分配堆栈空间大小
  - 注意实际使用内存空间大小

**尽可能定义较少全局与局部变量**：较少的变量，不会致空间不够

**尽可能使用全局变量**：全局变量区域较大，编译器会越界检查；堆栈区较小，编译器难以检查，越界后果难料

**合理组织全局变量与软件框架**：提高全局变量利用率同时避免耦合



# 嵌入式编程变量：表示范围与溢出

- 一般地：char: 8Bits, -128~+127, unsigned char: 8Bits, 0~+255
- 不同编译器变量表示字节数与范围不同，实际使用前应首先测试确认

*VarSize = sizeof(unsigned char)*

- 不是所有编译器支持所有常用变量类型：如在32位DSP上实际用16Bits表示char

*DSP28338: char(2Bytes), float(4Bytes), VC: char(1Byte) float(4Bytes)*

- 如变量超过表示范围，则溢出导致不可预测后果：如10\*100赋予char，结果可能是-8。

*解决方法：1) 设计时考虑变量表示范围；2) 使用前限制范围；3) 其他数据类型转换*



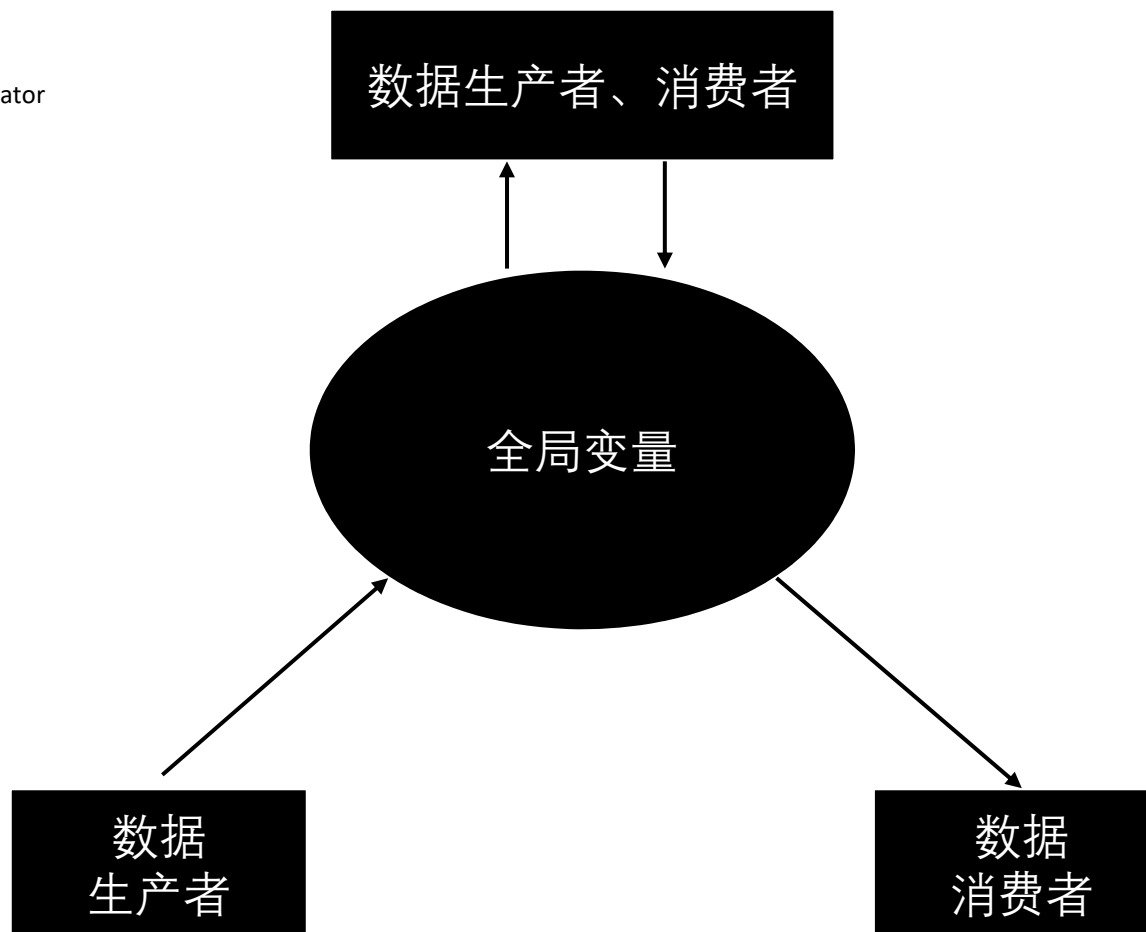
# 嵌入式编程变量：全局变量应用模型

```
typedef struct
{
    int iSetRSpd;//expect speed for speed code
    int iChkRSpd;//rotate speed from speed code
    int iPwmDTm;//time parameter for PWM generator
}STRN_MOTOR_IO;
```

```
typedef struct
{
    STRN_MOTOR_IO LH;
    STRN_MOTOR_IO LF;
    STRN_MOTOR_IO RH;
    STRN_MOTOR_IO RF;
}STRN_MOTOR;
```

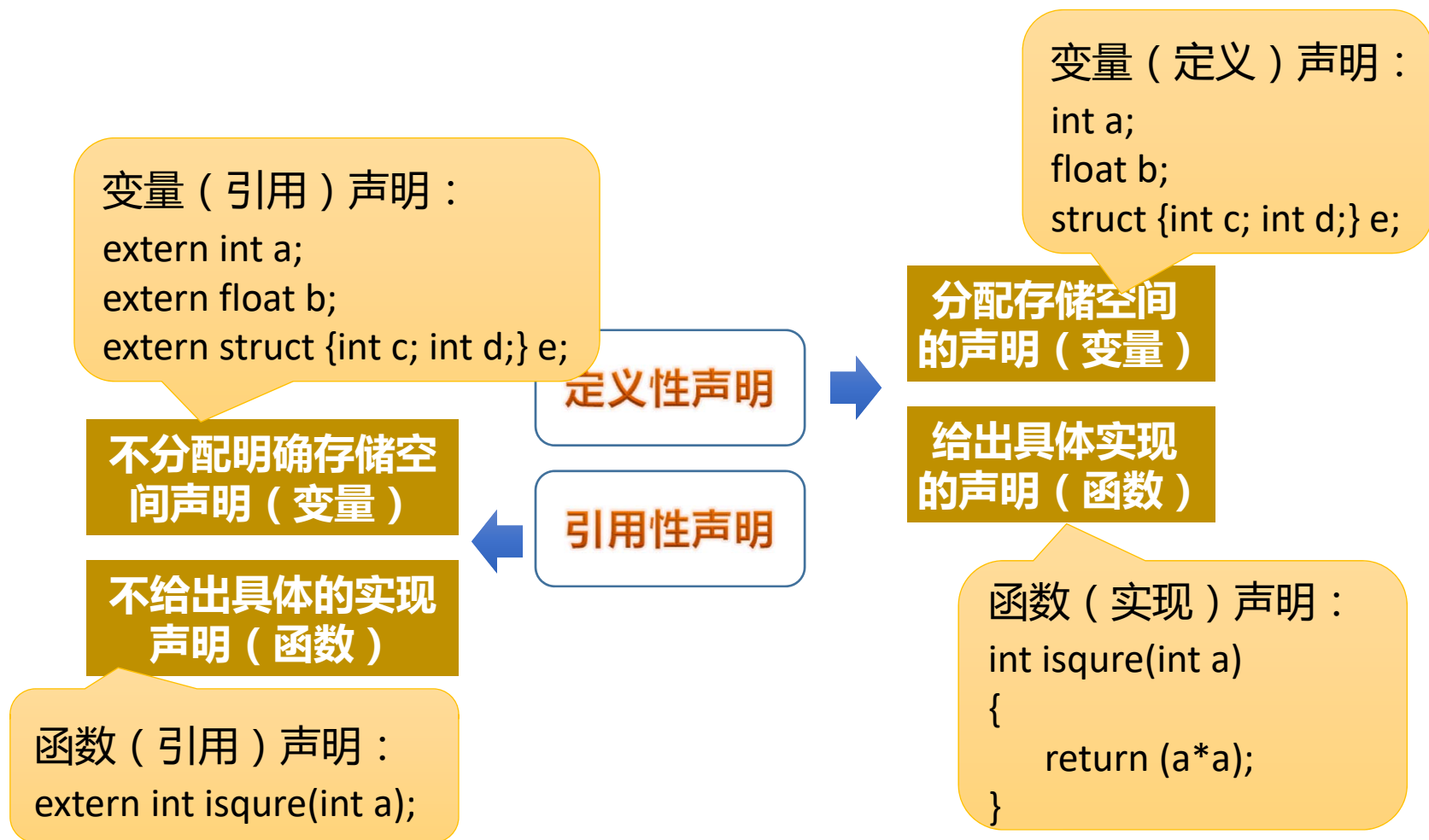
```
typedef struct
{
    uint8 cCtrMode;
    uint8 cCntLogic;
    uint8 cTimes;
    uint8 cNoUsed;
}STRN_SYS;
```

```
typedef struct
{
    unsigned char cGrayVal[128];
    int iOffset;
}STRN_LMCCD;
```





# 嵌入式编程变量：函数与变量定义与引用





# 嵌入式编程变量：两个C文件实例

全局变量 **iGlobalA** 定义 • 外部函数 **iFun** 引用声明

```
int iGlobalA = 0;  
extern void vFun(int *);
```

```
void main(void)
```

```
{
```

```
    int iLocalB;
```

```
    iLocalB = 3;
```

```
    vFun(&iLocalB);
```

```
    printf("%d", iLocalB);
```

```
}
```

局部变量

**iLocalB** 定义声明

外部函数 **iFun** 定义声明

全局变量

**iGlobalA** 引用声明

file2.c

```
extern int iGlobalA;
```

```
void vFun(int *b)
```

```
{
```

```
    int iLocalC;
```

```
    static int iTimes = 0;
```

局部静态变量 **iTimes** 定义声明

```
    iLocalC = 0 + iGlobalA;
```

```
    *a = iLocalC * iTimes;
```

```
}
```

局部变量 **iLocalC** 定义声明

函数形参 **b** 声明



# 嵌入式编程变量：结构体数据类型

## • 结构体

- 一组有机的数据整体
- 在内存中一般连续存放
- 没定义新数据，只是便于操作有机数据
- 定义：结构体名，成员类型与名称

```
struct  
{  
    int a;  
    float b;  
}Mystr1, *pMystr2;
```



```
typedef struct  
{  
    int a;  
    float b;  
}MY_STRUCT;  
MY_STRUCT Mystr1, *pMystr2;
```

定义？

```
Mystr1.a = 1;  
Mystr1.b = 2.54;  
pMystr2->a=2;  
pMystr2->b=3.78;
```



```
Mystr1=*pMystr2;
```



```
pMystr2=(MY_STRUCT *)alloc(sizeof(MY_STRUCT))
```





# 嵌入式编程变量：结构体类型数据传递

```
void vCopyMemAsUINT8(UINT8 *src, UINT8 *tgt, UINT8 intsize)
{
    UINT8 idx;
    for(idx=0; idx<intsize; idx++)
    {
        *tgt++ = *src++;
    }
}
```

Mystr1=\*pMystr2;



对DSP28335以2Bytes  
为单位; 对STM32F103、  
VC以1Byte为单位;

```
vCopyMemAsUINT8((UINT8 *)pMystr2, (UINT8 *)&Mystr1, sizeof(MY_STRUCT));
```

# 课程内容

- 认识嵌入式编程过程
- 认识嵌入式编程变量管理
- 车控系统软件建议框架
- 车轮转速控制实现
- PID控制律的参数与调试

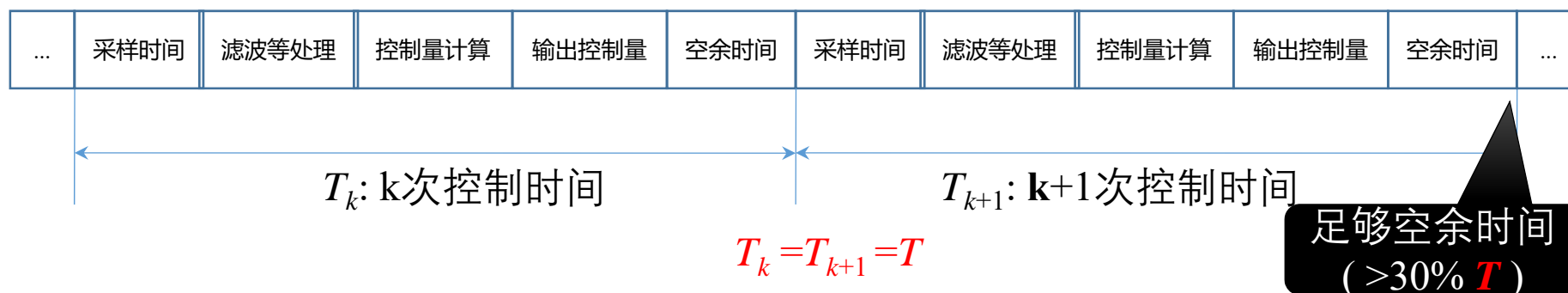




# 车控系统软件框架：实时控制的一般要求

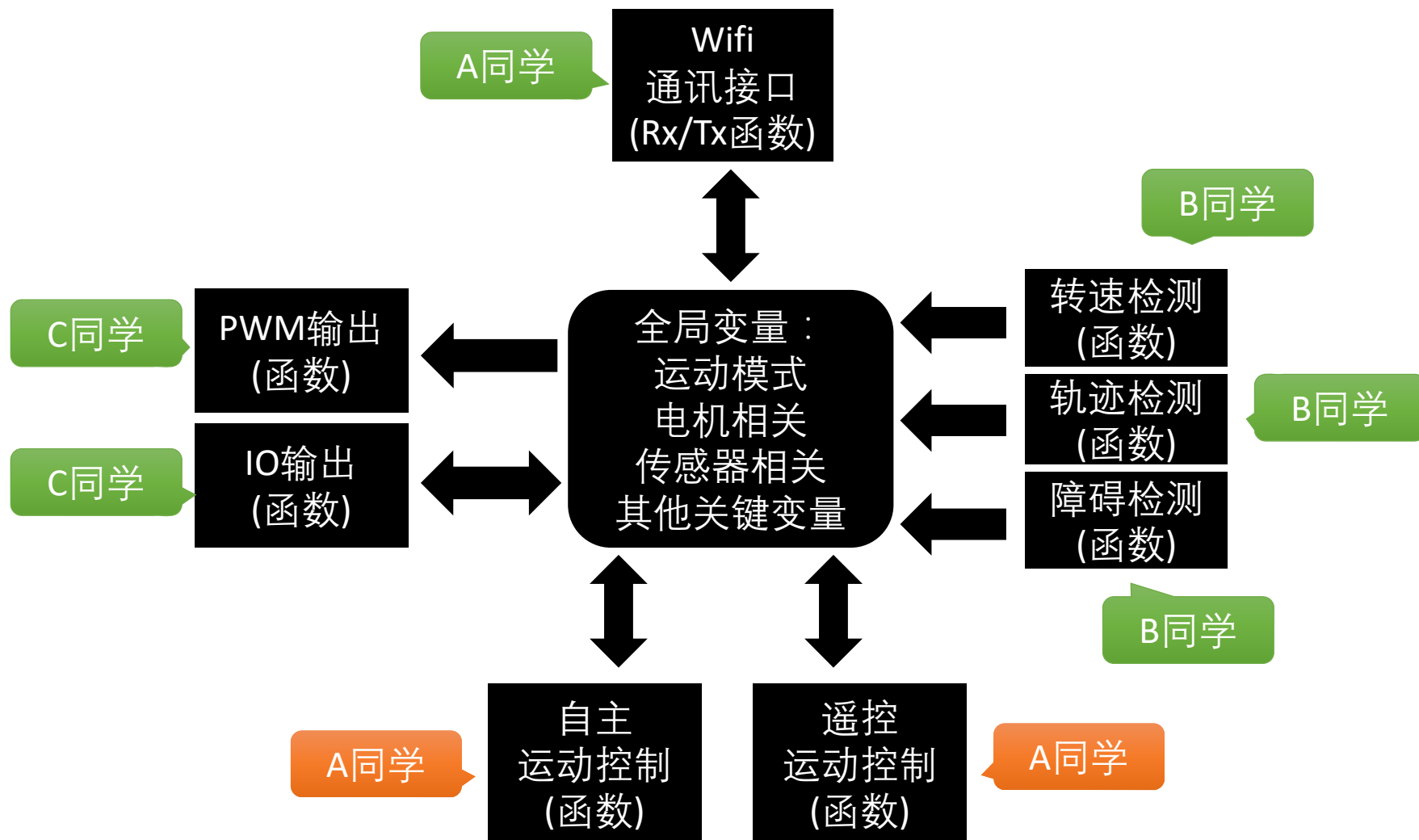
## 实时控制要求：

- 1) 准确的控制周期：每次采样、滤波、计算控制量、输出控制量这样完整的一个流程的执行时间及其间隔固定
- 2) 控制频率  $f$  (周期  $T$  倒数) 与控制系统带宽满足**香农采样定理**：  
控制频率**至少**是控制系统带宽的**2倍**，工程上建议**5倍以上**





# 车控系统软件框架：建议逻辑与分工方法





# 车控系统软件框架：建议变量定义举例

```
typedef struct
{
    int iSetRSpd;//expect speed for speed code
    int iChkRSpd;//rotate speed from speed code
    int iPwmDTm;//time parameter for PWM generator
}STRN_MOTOR_IO;
```

```
typedef struct
{
    STRN_MOTOR_IO LH;
    STRN_MOTOR_IO LF;
    STRN_MOTOR_IO RH;
    STRN_MOTOR_IO RF;
}STRN_MOTOR;
```

```
typedef struct
{
    uint8 cCtrMode;
    uint8 cCntLogic;
    uint8 cTimes;
    uint8 cNoUsed;
}STRN_SYS;
```

```
typedef struct
{
    unsigned char cGrayVal[128];
    int iOffset;
}STRN_LMCCD;
```

```
typedef struct
{
    int iGoSpd;//Speed for GoStraight: negative when go back
    int iTurnSpd;//Speed for Turn: negative when turn right
}STRN_PCCTRL;
```

```
typedef struct
{
    uint32 bObsExit0 : 1;
    uint32 bObsExit1 : 1;
    uint32 cNouse : 30;
    int iObsDis0;
    int iObsDis1;
}STRN_USONIC;
```

```
typedef struct
{
    STRN_SYS SYS;
    STRN_PCCTRL PCCTRL;
    STRN_MOTOR MOTOR;
    STRN_LMCCD LMCCD;
    STRN_USONIC USONIC;
}STRN_UGV;
```

```
#define CTRMODE_RC 00//remote control
#define CTRMODE_GT 01//go by trace
#define CTRMODE_AO 02//avoid obstacle
```

```
STRN_UGV strUGVVar;
```



# 车控系统软件框架：建议定时的实现H文件

```
#ifndef _MEETIMER_H_  
#define _MEETIMER_H_
```

```
typedef unsigned int  u16;  
typedef unsigned long u32;
```

```
#define TRUE 1  
#define FALSE 0
```

```
extern volatile STRN_SYS_TIMER strSysTimer;  
extern volatile u32 lSystemCnt;
```

```
#endif /* MeeTimer.h */
```

```
typedef struct
```

```
{
```

```
    u16 bTag10us : 1;  
    u16 bTag20us : 1;  
    u16 bTag50us : 1;  
    u16 bTag100us : 1;  
    u16 bTag1ms : 1;  
    u16 bTag5ms : 1;  
    u16 bTag20ms : 1;  
    u16 bTag25ms : 1;  
    u16 bTag50ms : 1;  
    u16 bTag100ms : 1;  
    u16 bTag200ms : 1;  
    u16 bTag500ms : 1;  
    u16 bTag1s : 1;  
    u16 bTag1D1s : 1;  
    u16 bTag2s : 1;  
    u16 bTag2D5s : 1;
```

```
}STRN_SYS_TIMER;
```

这里



# 车控系统软件框架：建议定时实现C文件

---

找到文件 stm32f1xx\_it.c, 在 SysTick\_Handler() 函数中添加定时代码

```
#include "MeeTimer.h"
volatile u32 ISystemCnt;
volatile STRN_SYS_TIMER strSysTimer;
void SysTick_Handler(void)
{
    ISystemCnt++; ISystemCnt %= 100000;
    strSysTimer.bTag1ms = TRUE;
    if( ISystemCnt%100 == 99 ) strSysTimer.bTag100ms = TRUE;
    if( ISystemCnt%200 == 199 ) strSysTimer.bTag200ms = TRUE;
    if( ISystemCnt%1000 == 999 ) strSysTimer.bTag1s = TRUE;

    HAL_IncTick();
}
```



# 车控系统软件框架：建议主程序框架伪码

```
void main()
{
    ....//初始化
    while(1)
    {
        vDetect_UartRx();//来自PC遥控：只改变 strUGVar.STRN_PCCTRL. 成员内容
        if(strSysTimer.bTag10ms==TRUE)
        {
            vDetect_LMCCD();//CCD函数：只改变 strUGVar.LMCCD. 成员内容
            vDetect_USONIC();//超声波函数：只改变 strUGVar.USONIC. 成员内容
            strSysTimer.bTag10ms = FALSE;
        }
        if(strSysTimer.bTag50ms==TRUE)
        {
            switch(strUGVar.SYS.cCtrMode)
            {
                case CTRMODE_GT:    vStrategy_GtMode();    break; //go by trace
                case CTRMODE_AO:    vStrategy_AoMode(); break; //avoid obstacle
                default:// CTRMODE_RC: vStrategy_RcMode(); break; //remote control
            }
            vControl_MotorPid();
            vHalExe_PwmGen();
            strSysTimer.bTag50ms = FALSE;
        }
    }
}
```



# 车控系统软件框架：建议函数模块-遥控

---

```
void vDetect_UartRx(void)
{
    static int bStartUp = 1;
    if(bStartUp) { strUGVar.SYS.cCtrMode = CTRMODE_RC; bStartUp = 0; }
    else
    {
        ...//处理自 PC 数据，只改变控制模式 及 strUGVar.STRN_PCCTRL. 成员内容
        ...//即：设定车体的直行与转弯速度
    }
}
```



# 车控系统软件框架：建议功能模块-遥控

---

```
void vStrategy_RcMode(void)
{
    //将车体的 设定车速 变换到电机的设定速度，如下面简单代码
    strUGVar.MOTOR.LH.iSetRSpd = strUGVar.PCCTRL.iGoSpd +
                                strUGVar.PCCTRL.iTurnSpd*2/3;

    //....
}
```





# 车控系统软件框架：建议功能模块-遥控

```
void vControl_MotorPid(void)
(
    switch(strUGVar.SYS.cCtrMode)
    {
        case CTRMODE_GT://go by trace
        case CTRMODE_AO://avoid obstacle
            //通过电机的设定转速与实际转速差(PID之比例闭环控制)，计算PWM参数
            strUGVar.MOTOR.LH.iPwmDTm
                = Kp * (strUGVar.MOTOR.LH.iSetRSpd - strUGVar.MOTOR.LH.iChkRSpd);
            ...
            break;

        default:// CTRMODE_RC: //remote control
            //由电机设定转速，直接转换到PWM（开环）；也可以用PID实现（闭环）
            strUGVar.MOTOR.LH.iPwmDTm = strUGVar.MOTOR.LH.iSetRSpd;
            ...
            break;
    }
)
```



# 车控系统软件框架：建议功能模块-PWM

---

```
void vHalExe_PwmGen(void)
{
    //将前述 PWM 数据转换为 定时器参数，以产生PWM 信号
    //..... = strUGVar.MOTOR.LH.iPwmDTm
}
```



# 车控系统软件框架：原程序的组织管理

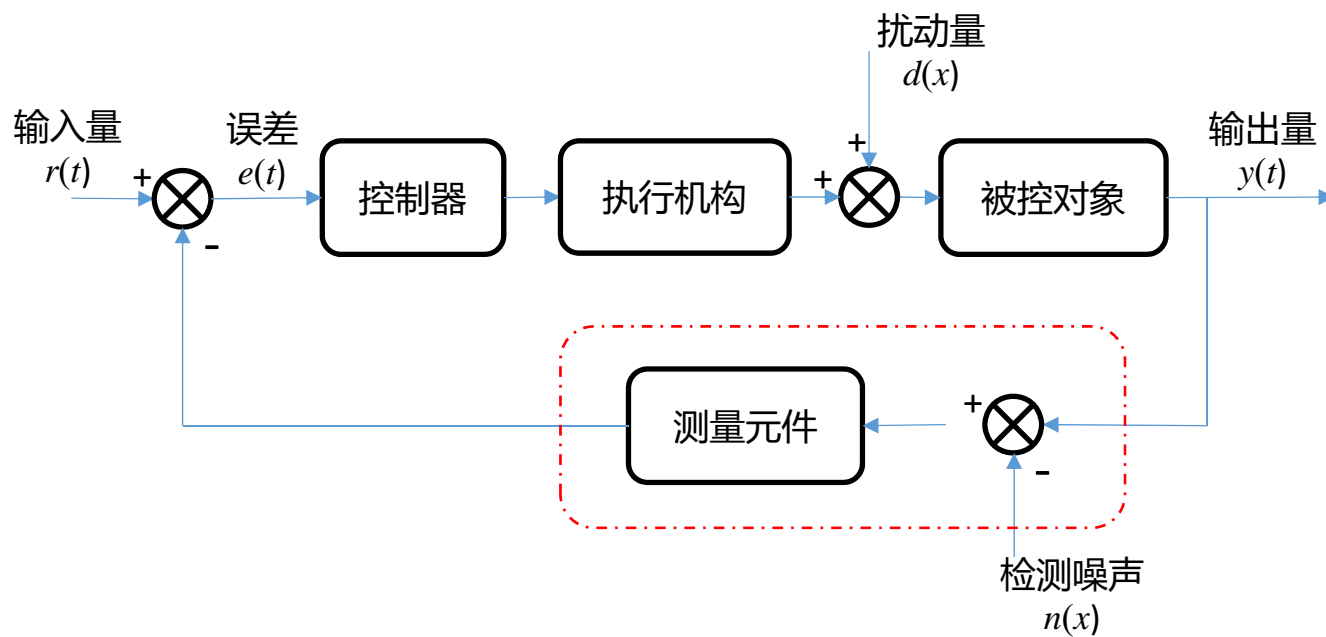
- 模块化原程序
  - 一组具有相对独立功能的程序构成一个原文件(C)
  - 一个源文件(C)内部**定义**（赋初值）全局变量、**定义**实现若干函数，**声明**（用关键字**extern**）本源文件使用的内部函数，**完成**相对独立的子功能
  - 一个源文件对应一个头文件(H)，**声明****对外接口**的函数与变量(用关键字**extern**)，以及宏定义
  - 一个源文件引用其他源文件的函数、变量时，**#include**对应头文件
- 若干原文件与有关头文件组合成工程(**工程+其他配置=project**)
- 对工程做相应设置以生成合适可执行文件(**HEX/OUT/EXE**)

# 课程内容

- 认识嵌入式编程过程
- 认识嵌入式编程变量管理
- 车控系统软件建议框架
- 车轮转速控制实现
- PID控制律的参数与调试



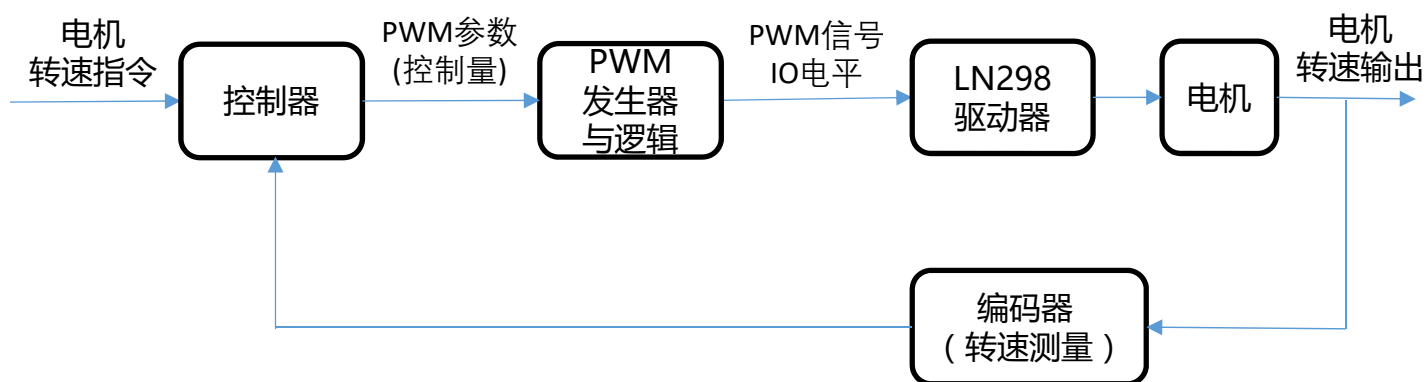
# 车轮转速控制实现：一般闭环控制系统



标准闭环控制系统示意图



# 车轮转速控制实现：本课程控制对象



车轮转速闭环控制框图



# 车轮转速控制实现：C编程实现PID算法

```
typedef struct
{
    int iSetRSpd;//expect speed for speed code
    int iChkRSpd;//rotate speed from speed code
    int iPwmDTm;//time parameter for PWM generator
}STRN_MOTOR_IO;

void vControl_MotorPid(void)
{
    int iCurErr;
    static long iErrSum = 0;
    iErrSum = iErrSum + iCurErr;

    iCurErr = strUGVar.MOTOR.LH.iSetRSpd - strUGVar.MOTOR.LH.iChkRSpd;//误差
    strUGVar.MOTOR.LH.PwmDTm = iCurErr * Kp;//P控制
    strUGVar.MOTOR.LH.PwmDTm += iErrSum * Ki;//I控制
    ...
}
```

通过调试，确定的PID  
控制器参数：Kp,Ki,Kd

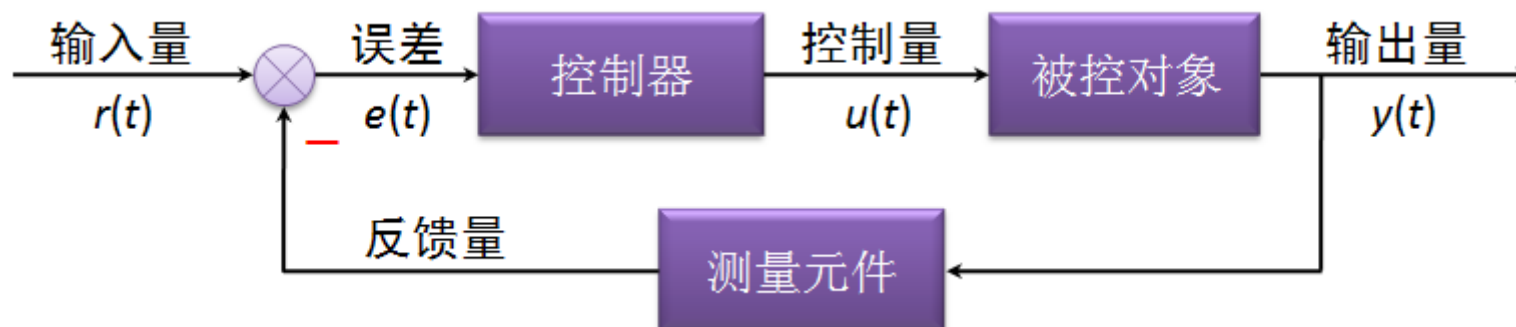
# 课程内容

- 认识嵌入式编程过程
- 认识嵌入式编程变量管理
- 车控系统软件建议框架
- 车轮转速控制实现
- **PID控制律的参数与调试**





# PID控制律基本形式



$$u(t) = K_p e(t) + K_i \int_0^t e(t) dt + K_d \frac{de(t)}{dt}$$



$$u(k) = K_p e(k) + K_d [e(k) - e(k-1)] + K_i [e(k) + e(k-1) + \dots]$$

便于计算机实现

比例(Proportional)

积分(Integral)

微分(Differential)

现在

过去

将来

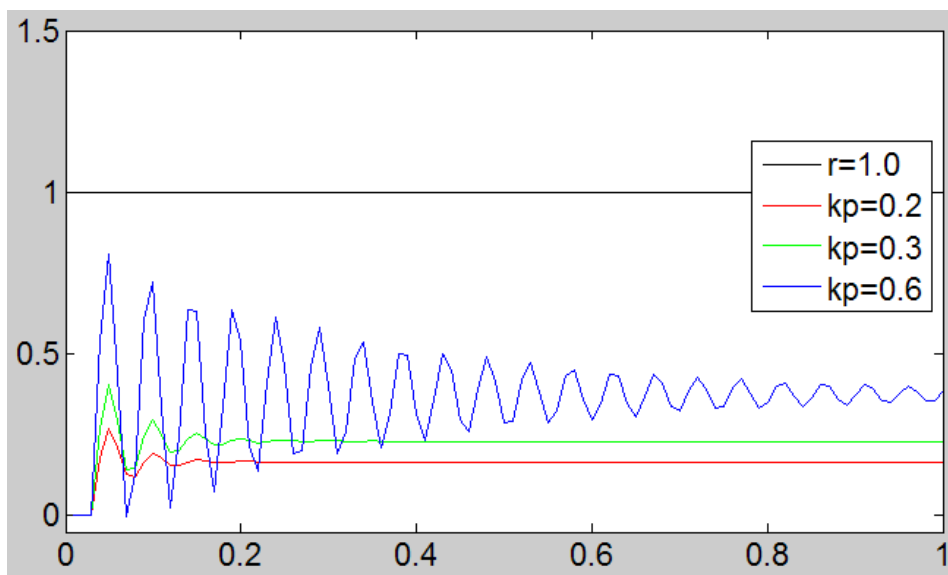
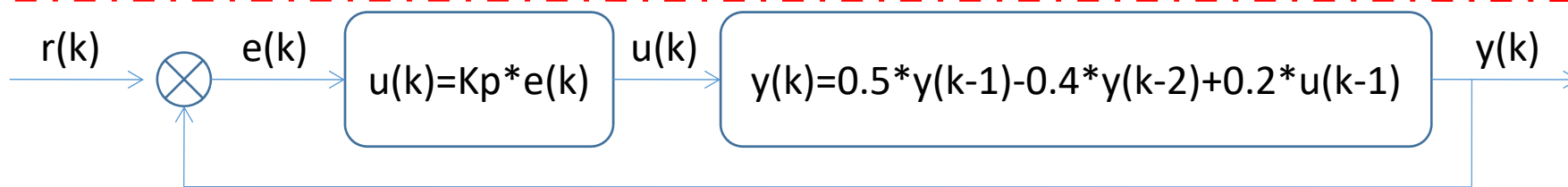


# PID控制律中比例增益作用

$$u(t) = K_p e + K_i \int e(t) dt + K_d \frac{de(t)}{dt}$$

$$e(t) = r(t) - y(t)$$

$e(t)$ 变化越大则 $u(t)$ 越大  
比例项放大误差，增加输出量跟踪输入量速度



$K_p$ 越大:  
响应速度越快  
稳态误差越小  
稳定性变越差  
过渡时间变长

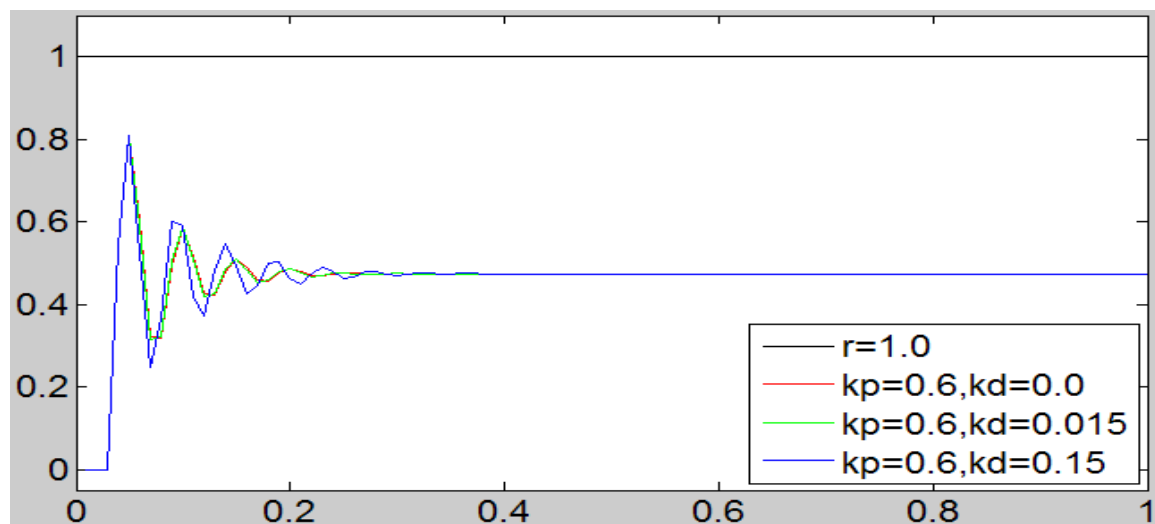
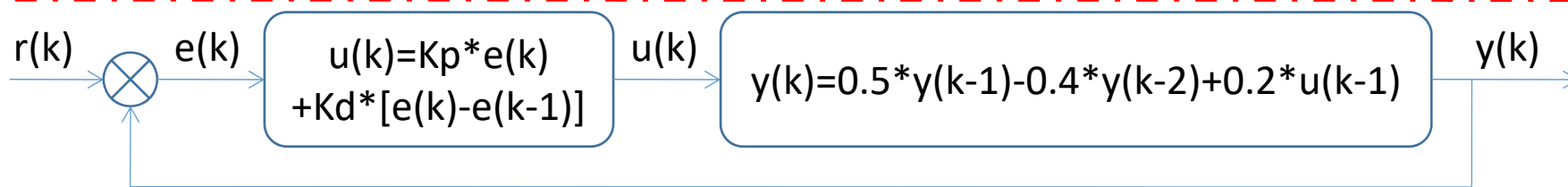


# 控制律中微分增益作用

$$u(t) = K_p e + K_i \int e(t) dt + K_d \frac{de(t)}{dt}$$

$y(t)$ 变化越大则 $u(t)$ 越小  
直观上微分项抑制输出变化

$$\frac{de(t)}{dt} = \frac{dr(t)}{dt} - \frac{dy(t)}{dt}$$



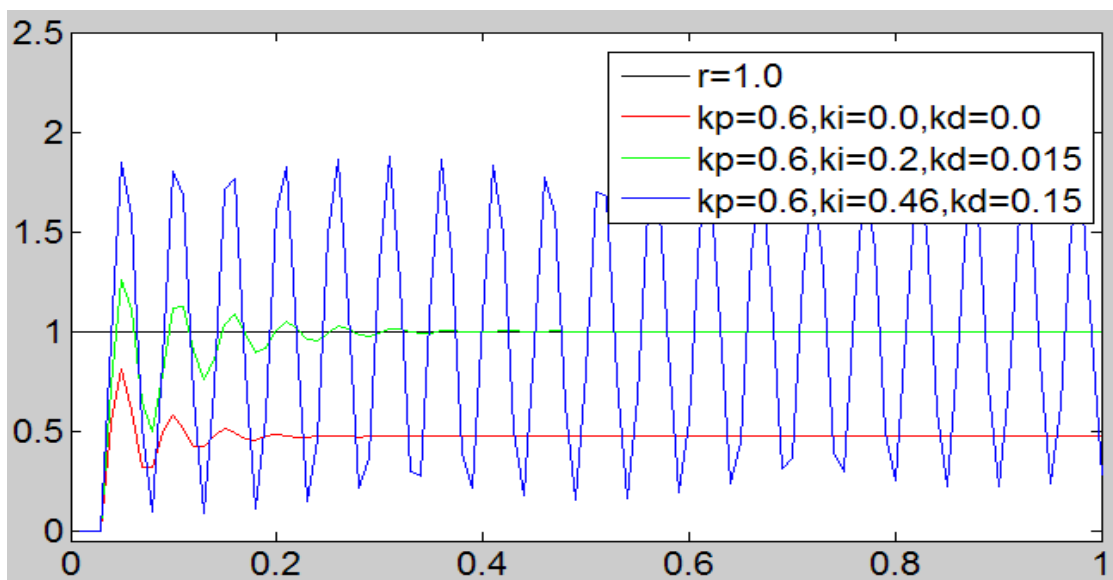
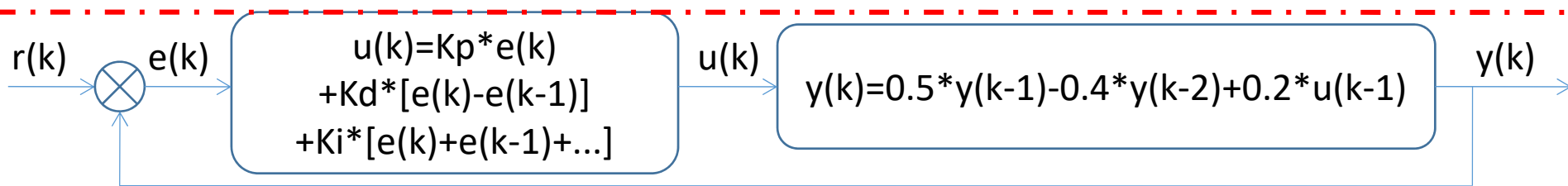
$K_d$ 越大:  
稳定性变好  
收敛速度加快  
过渡时间变短



# PID控制律中积分增益作用

$$u(t) = K_p e + K_i \int e(t) dt + K_d \frac{de(t)}{dt}$$

$e(t)$ 积分越大则 $u(t)$ 越大  
直观上积分项驱动系统减小误差



$K_i$ 越大:

稳态误差越小

响应速度加快

稳定性变差



# PID控制律三种增益手工调节方法

- 1 ) 所有增益清零，使用阶跃输入信号作为调节测试信号
- 2 ) 先调节比例增益： $K_p$ 从0开始，逐步增大至系统有少许超调及几个过渡震荡
- 3 ) 再调节微分增益：增加 $K_d$ ，使得减小超调以致没有震荡
- 4 ) 由3 ) 回到2 ) 再到3 )，直至输出性能基本满足要求，且电机无明显低沉嘟嘟声（内部有少许电流震荡，震荡越多系统功耗越大）
- 5 ) 少许增加积分增益：增加 $K_i$ 使得稳态误差接近零，且系统过渡震荡过程时间短至合适水平



# PID控制律C语言实现示例

(1) 计算原始误差： $fRErr\_c = fRef - fOut$ ;

(2) 计算误差微分： $fDErr\_c = fRErr\_c - fRErr\_p$ ;

(3) 为下一次计算做准备： $fRErr\_p = fRErr\_c$ ;

(4) 计算 (其中实现了防止积分饱和)

```
if( (fRErr_c > -fIntErrLmt) && (fRErr_c < +fIntErrLmt) ) fIntVol += fRErr_c;  
else fIntVol = 0; //大误差是切除积分项，防止积分饱和
```

(5) 计算比例项： $fCtrVol = Kp * fRErr\_c$ ;

(6) 计算并累加积分项： $fCtrVol += Ki * fIntVol$ ;

(7) 计算并累加微分项： $fCtrVol += Kd * fDErr\_c$ ;

(8) 限幅后的fCtrVol就是控制器的输出结果：

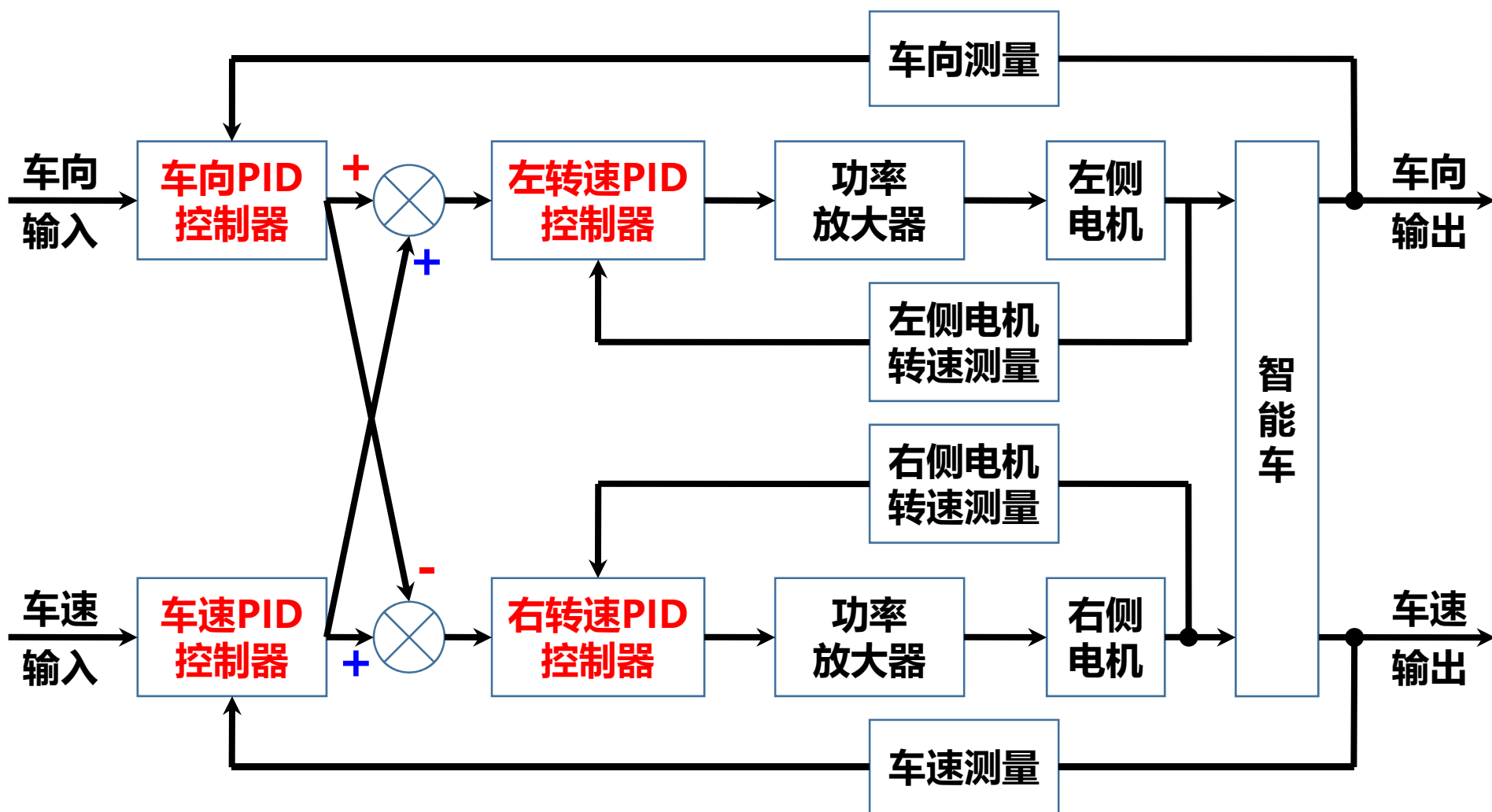
```
if(fCtrVol > V_fMaxVal) fCtrVol = V_fMaxVal;  
if(fCtrVol < V_fMinVal) fCtrVol = V_fMinVal;
```



逐条编写, 避免如下连续:  
 $fCtrVol = fRErr\_c * Kp + Ki * fIntVol + Kd * fDErr\_c$

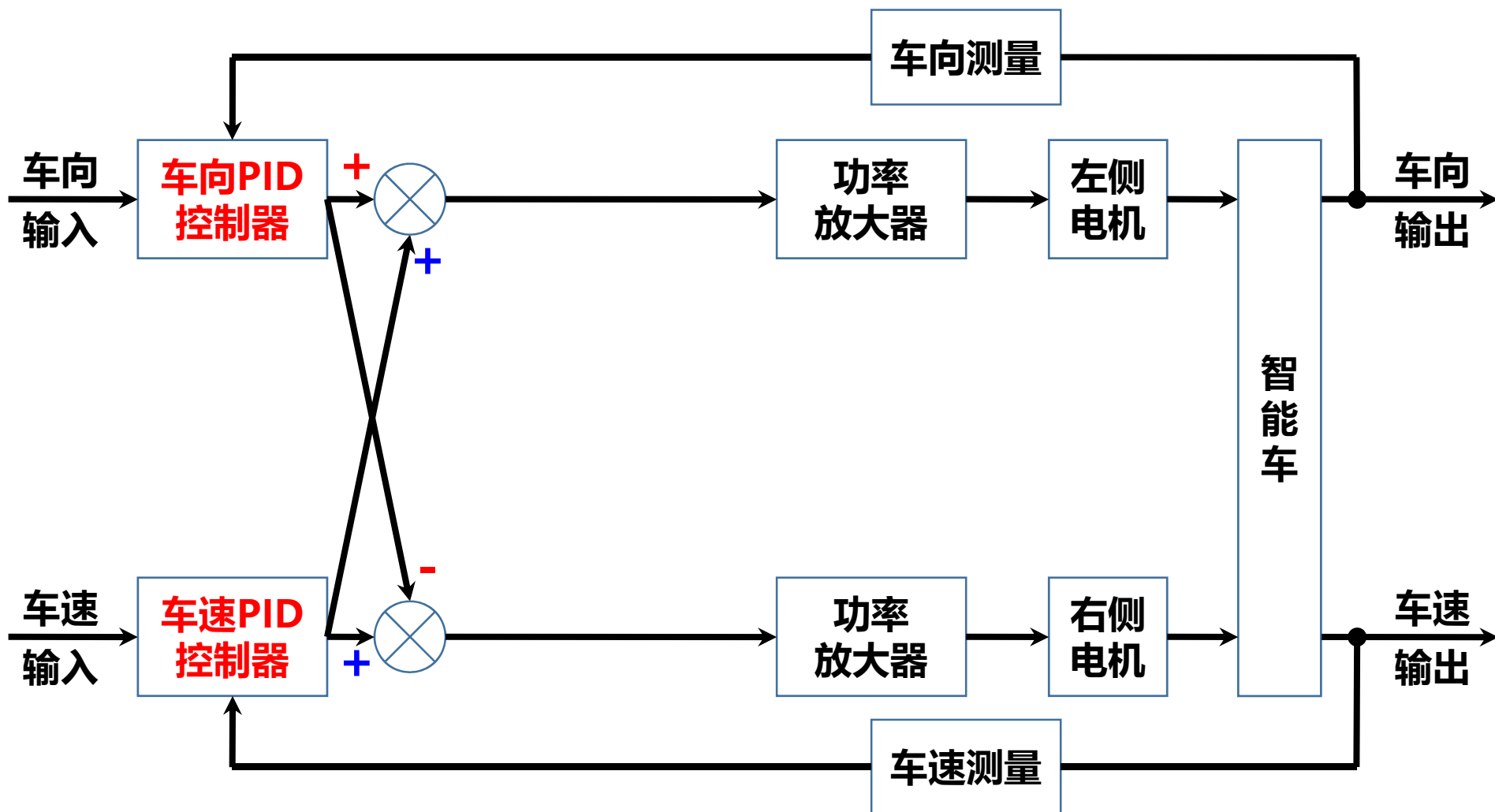


# PID应用：车向(速)闭环控制(双环)





# PID应用：车向(速)闭环控制(单环)







The End