西安交通大學

项目设计报告

姓名：石恩升

班级：自动化 53

学号：2150504072

指导老师：刘晓勇，姚向华

目录

0、项目设计概述

- 开发资料百度云网址：http://yun.baidu.com/
- 开发平台：win10
- Freescale Kinetis K60 芯片
- IAR EWARM 开发环境

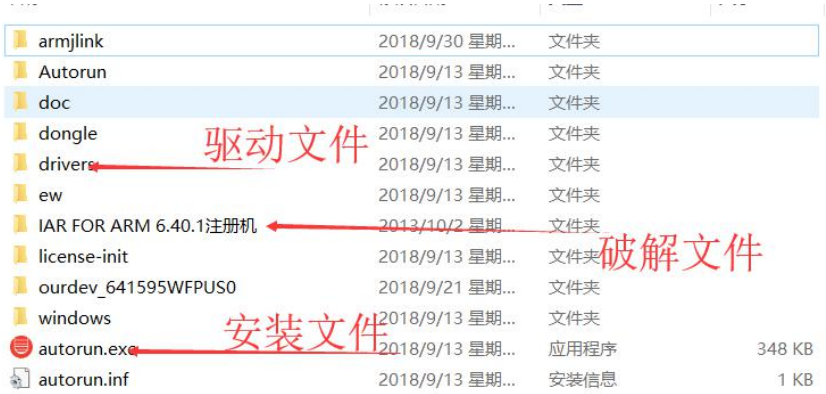
一、编程软件的安装与使用

1.1 下载安装软件

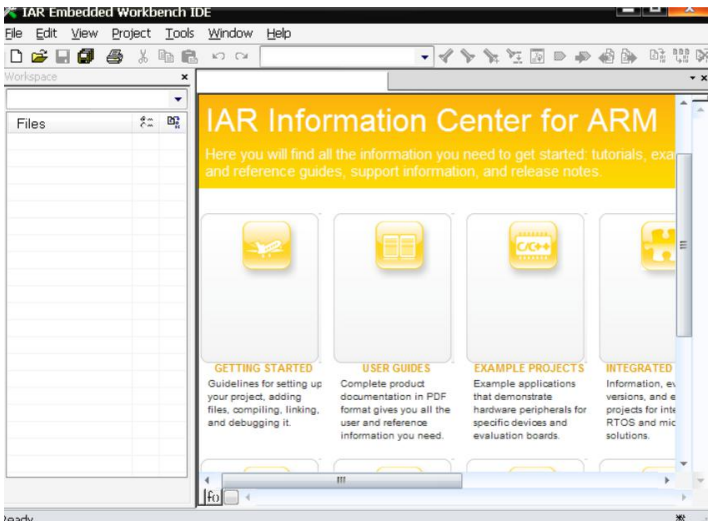
目前最新的 IAR for ARM 为 v6.30，支持更多的 Kinetics 系列芯片，因此我们安装最新版，避免因为版本太低而出现不兼容，甚至出现异常错误的情况

1.2 安装 IAR 详细过程

1.下载后解压文件，打开目录，运行安装文件：



2. IAR 的编程界面

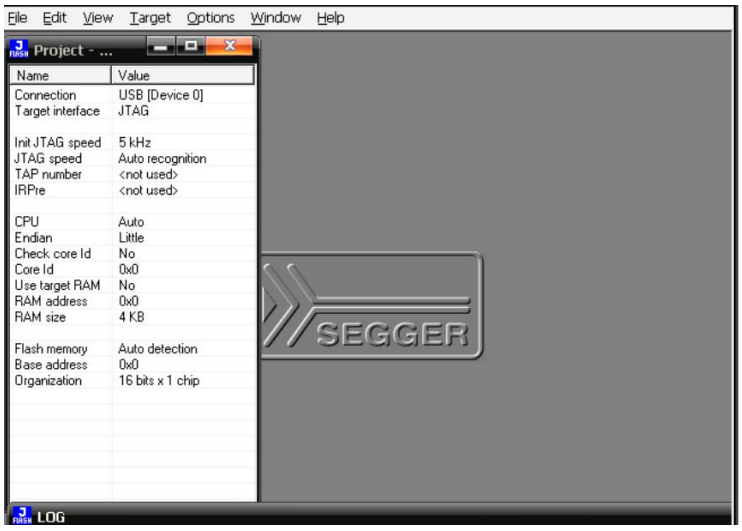


相对于 Keil For ARM、CodeWarrior 而言，IAR for ARM 的编程界面是最简

单的，编译效率高，在嵌入式系统的调试方面提供了可供调试的插件。

3. 安装仿真器

又换会原来的安装导航界面：选择 Install driver，选择了用 jlink 作为仿真调试器，因此这里选择 jlink，运行安装，运行后，会自动安装驱动，不会有其他提示。另外，为了可以使用擦除芯片等功能，我们可以选择用 segger 公司的 j-link。



二、模块检测与调试

在车辆组装过程中，因为人为原因，可能出现各个问题，需要在利用一些电气设备和程序的情况下，进行模块检测。在模块检测时，各个模块间的线，先不插入插座中，待各个模块检测完后，在按照说明连接线路。

2.1 检测的电源模块。

检测方法:给电源模块正常供电后，用万用表测量各个端子的电压是否正常。具体电压,虽然电池上写着是 7.2v,但是满充状态下电压是 8.3v 左右,低于 3.5v 后，整车不工作。

2.2 检测核心板

核心板的检测，需要打开代码检测程序。，用 jlink 给系统板供电，下载该程序代码，程序下载后，核心板上指示灯会闪烁。如果接上串口，波特率设置为 115200，会不断的发数据出来。以证明系统版是能够正常工作的。把系统板插到主板上。这一步上一届学长学姐替我们做过啦，所以这步我们只是简单的点灯看系统版是否正常工作。主板上的外设通过万用表可以检测是否和外设连接正确。

2.3 检测电机驱动

2.3.1 程序验证

下载电机检测程序[E:\BaiduYunDownload\项目设计\蓝宙飞思卡尔资料\蓝宙电子智能车摄像头套件 k60 平台资料-更新至 20130601\03 蓝宙电子智能车竞赛单电机驱动 BTS7960 资料\蓝宙电子智能车竞赛单电机驱动 BTS7960 资料\landzoK60 单电机检测程序]，连接好电源线和驱动线。查看电机是否正反。可以看到此程序包含电机模块的初始化。

```
 ************************************/
/* */

//gpio_init (PORTC , 4, GPO,HIGH);                          //电机使能  初始化PORTx portx端口号, u8 n引脚, GPIO_CFG cfg0
// gpio_init (PORTC , 2, GPO,LOW);                          //电机方向使能
 FTM_PWM_init(FTM0 , CH1, 80000,0);                        //功能定时器模块 通道号 频率 占空比
 FTM_PWM_init(FTM0 , CH2, 80000,0);                        //电机占空比设置初始化     占空比 = duty /(MOD +1 ) ;FTM_CnV_REG

// uart_init (UART0 , 115200);                        //初始化UART0，输出脚PTA15，输入脚PTA14，串口频率 9600
 gpio_init (PORTA , 17, GPO,HIGH);
 gpio_init (PORTB , 17, GPI,HIGH);
 pit_init_ms(PIT0, 5);                                  //定时中断 初始化PIT0，定时时间为：  5ms
 pit_init_ms(PIT1, 1000);                               //初始化PIT1，定时时间为：  1000ms
 EnableInterrupts;                                      //开总中断

 /***************************************
  执行程序
  ***************************************/
 while(1)
```

电机模块的初始化

与设置占空比的不同来进行正反转

```
 {
    count = 2 ;
    FTM_PWM_Duty(FTM0 , CH1,0);
    FTM_PWM_Duty(FTM0 , CH2,50);
 } else if(count == 2)
 {

    count = 1 ;
    FTM_PWM_Duty(FTM0 , CH1,0);
    FTM_PWM_Duty(FTM0 , CH2,0);

 }else if(count == 1)
 {

    count = 0 ;

    FTM_PWM_Duty(FTM0 , CH1,50);
    FTM_PWM_Duty(FTM0 , CH2,0);

 } else if(count == 0)
 {

    count = 3 ;
    FTM_PWM_Duty(FTM0 , CH1,0);
    FTM_PWM_Duty(FTM0 , CH2,0);

 }

}
```

//设置通道占空比

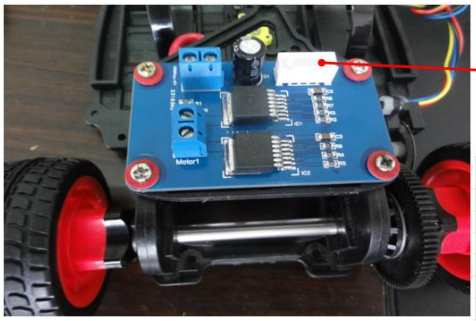
01 正转

00转动

10反转

00制动

我们可以查看电机模块说明书[E:\BaiduYunDownload\项目设计\前三节 PPT\2 姚向华]电机使用隔离芯片 5V 电源（可以与单片机共用 5V）；单片机到驱动模块用 4 跟线即可（GND、5V、PWM1、PWM2）；最大频率支持 15KHz，超过会造成芯片发烫、电机运转不正常。



控制端口说明		
引脚	标示	信号
1	CF	电流反馈
2	IN1	PWM
3	IN2	PWM
4	INH	IO
5	GND	地

真值表			
INH	IN1	IN2	方向
1	1	0	正
1	0	1	反
1	0	0	能耗制动
1	1	1	能耗制动
0	X	X	休眠

从真值表我们可以看到 INH 是使能标志，IN1 与 IN2 的组合代表着正反转和制动。我们知道 FTM 模块可以提供定时、计数、输出 PWM 波等功能。查看程序中 FTM 模块中相关变量。

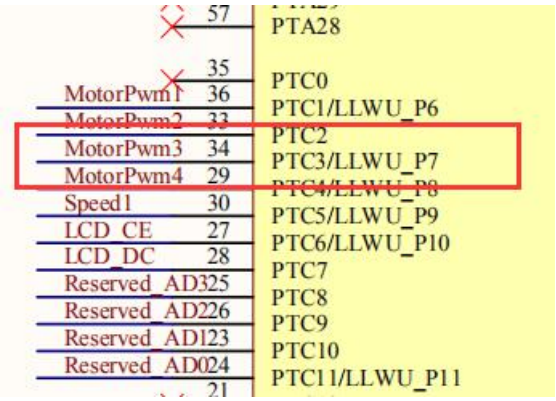
```
typedef enum FTMn
{
    FTM0,
    FTM1,
    FTM2
} FTMn;

typedef enum CHn
{
    //            --FTM0--   --FTM1--   --FTM2--
    CH0,    //     PTC1       PTA8       PTA10
    CH1,    //     PTC2       PTA9       PTA11
    CH2,    //     PTC3         ×          ×
    CH3,    //     PTC4         ×          ×
    CH4,    //     PTD4         ×          ×
    CH5,    //     PTD5         ×          ×
    CH6,    //     PTD6         ×          ×
    CH7     //     PTD7         ×          ×
    // ×表示不存在
} CHn;
```

可以看到 FIMn 与 CHn 都是枚举变量，两者的组合对应一个引脚。此次程序是：

```
1 FTM_PWM_init(FTM0 , CH1, 80000,0);  //功能定时器模块 通道号 频率 占空比
2 FTM_PWM_init(FTM0 , CH2, 80000,0);
```

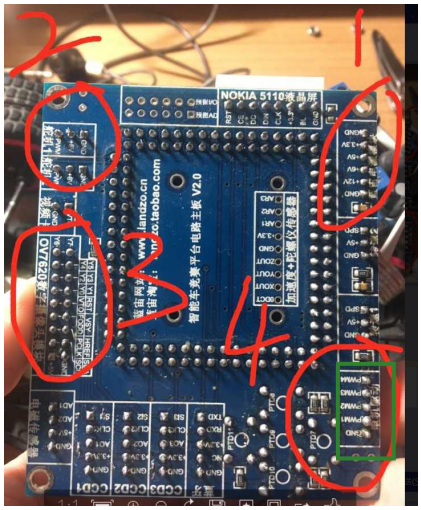
使用的是 FIM0 与 CH1,CH2 对应的引脚号为 PTC2 与 PTC3。

打开主板的原理图 [E:\BaiduYunDownload\项目设计\蓝宙 飞思卡尔资料\蓝宙电子智能车摄像头套件 k60 平台资料-更新至 20130601\01 蓝宙电子智能车 MK60 主板资料\蓝宙电子智能车 MK60 主板资料] 里的主板原理图，看到 PTC2 与 PTC3 正好对应着 MotorPwm2 和 MotorPwm4，也就是电机驱动的 PWM 波。



2.3.2 电路连接

如果认为这个猜想不正确，那么可以继续验证一下，我们可以看到主板的背面，



4 这里的 5 个引脚分别于电机的控制电路的 5 个引脚相连，而 PWM2、PWM3 与电机模块的 IN1 与 IN2 相连，这也就是解释了为什么

```
1 FTM_PWM_Duty(FTM0 , CH1,0);
2 FTM_PWM_Duty(FTM0 , CH2,50);
```

01 是正传，10 是反转，00 与 11 都是制动，有人也许会说这个正反转和上面的真值表有所不同，是这样，有些车就是 01 正传，10 反转，试一下就知道。

```
{
    count = 2 ;
    FTM_PWM_Duty(FTM0 , CH1,0);          //设置通道占空比
    FTM_PWM_Duty(FTM0 , CH2,50)
} else if(count == 2)
{

    count = 1 ;
    FTM_PWM_Duty(FTM0 , CH1,0);
    FTM_PWM_Duty(FTM0 , CH2,0);

}else if(count == 1)
{

    count = 0 ;

    FTM_PWM_Duty(FTM0 , CH1,50);
    FTM_PWM_Duty(FTM0 , CH2,0);

} else if(count == 0)
{

    count = 3 ;
    FTM_PWM_Duty(FTM0 , CH1,0);
    FTM_PWM_Duty(FTM0 , CH2,0);

}

}
```

01 正转

00转动

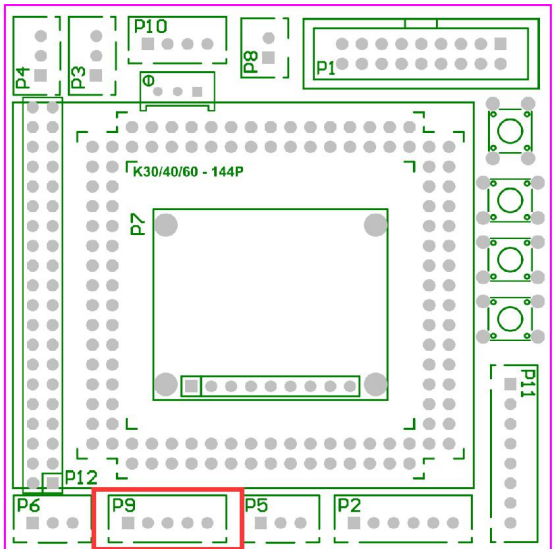
10反转

00制动

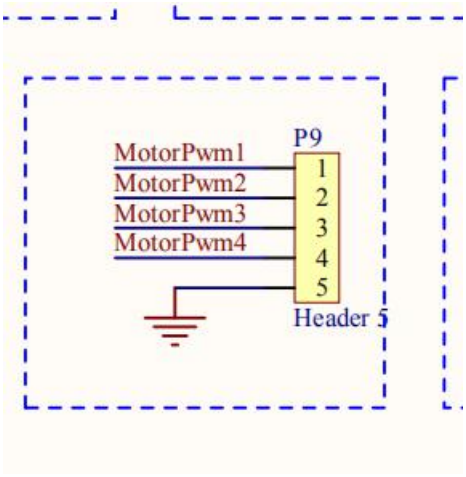
改变占空比，占空比越大，转速越快。在我们的最终程序中，占空比在 25%左右。

2.3.3 PCB 与对应引脚

打开主板的原理图［E:\BaiduYunDownload\项目设计\蓝宙 飞思卡尔资料\蓝宙电子智能车摄像头套件 k60 平台资料-更新至 20130601\01 蓝宙电子智能车 MK60 主板资料\蓝宙电子智能车 MK60 主板资料］里的主板原理图，看到 PCB 电路图中



P9 对应着电机模块，怎么看的，就是根据引脚数量和大致位置就可以知道啦，5 个引脚的就 p9 这一个，然后向上翻找到 P9 模块，



可以看到四路 PWM 波中 PWM2、PWM3 对应着 MotorPwm2 和 MotorPwm3，而 MotorPwm2 和 MotorPwm3 连接着电机模块的 IN1 与 IN2，这样一切就解释都很合理了。

2.4 检测舵机和舵机对正

机械安装好后，先把舵机上面的黑色小圆盘从舵机上拔出来。改装舵接的连接线，舵连接线的改装方法参考主板原理图。把舵机连接线插入主板上，下载舵机测试程序，舵机测试程序会让舵机左右转动。如果舵机能够正常工作，下载舵机对中程序，让舵机对中，安装上黑色小圆，说完这些是不是不知道我在说什么，好我们回到电机调试的思路来。

2.4.1 舵机概述

首先看一下舵机的材料 [E:\BaiduYunDownload\项目设计\前三节 PPT\2 姚向华]：

一组减速齿轮;

电源线：一般为 4.8v 或 6v。一般为红色;

地线：一般为黑色

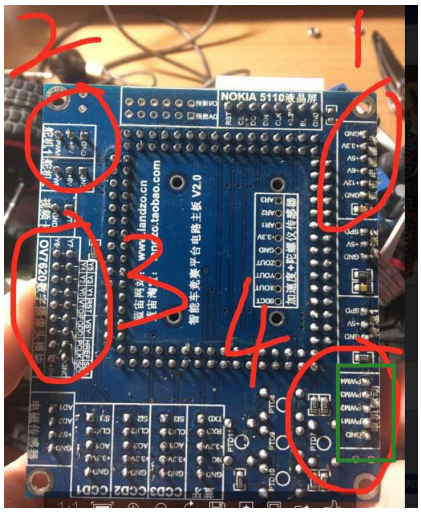
控制线：周期是 20ms（也就是 50HZ）的脉宽调制（PWM）信号 （FUTABA 为白色）

脉冲宽度从 0.5ms-2.5ms，相对应舵盘的位置为 0-180 度，呈线性变化。从这里可以得到占空比从 0.5/50-2.5/50,也就是 2.5%到 12.5%。

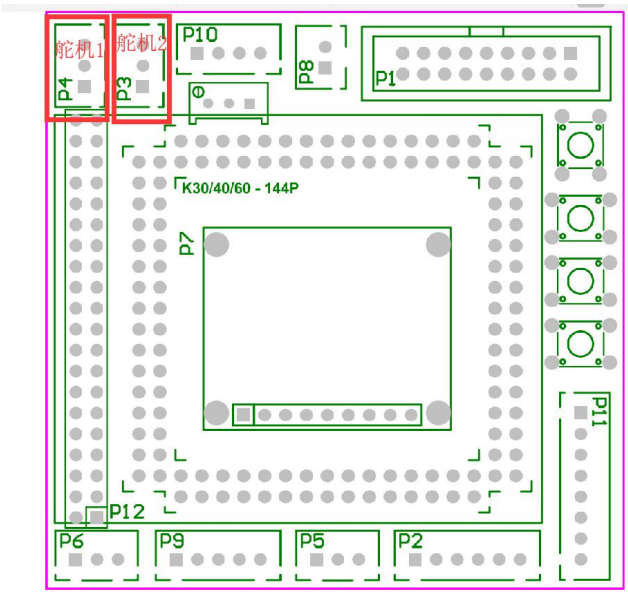
重点是，控制线提供一定的脉宽，它的输出轴就会保持在一个相对应的角度上，直到给它提供一个另外宽度的脉冲信号，它才会改变输出角度到新的对应的位置上。不是说各个占空比他就一直转，而是一个占空比对应一个角度的。

2.4.2 舵机的电路连接

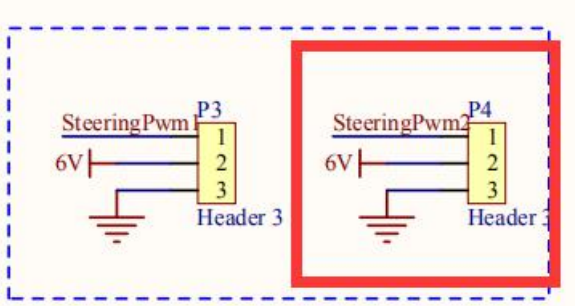
依旧看主板，可以看到主板的后面有舵机 1 与舵机 2，我们任意选一个，以舵机 1 为例。



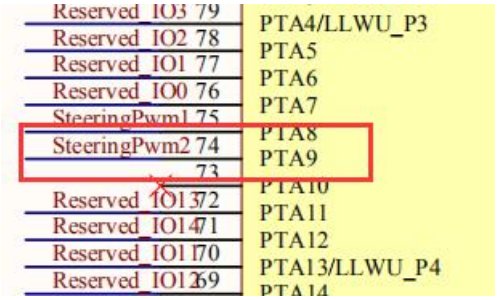
我们选择舵机 1 模块，在 PCB 电路图中为：



可以看到舵机 1 是 P4 模块，怎么看出来，根据相对位置猜的，因为舵机 1 靠外面一点。我们向上翻可以看到 P4 模块：



它标的 SteeringPwm2，不过不用管他，可能就是想把 P3 标成 SteeringPwm1，把 P4 标成 SteeringPwm2，这时候也该反应出来这应该和电机一个套路，用的是 FTM 模块，我们具体看一下 SteeringPwm2 是哪个引脚？



可以看到 SteeringPwm2 对应的是 PTA9。

2.4.3 舵机调试

我们回到电机的调试程序中，看到 FTM2 与 CH1 正好对应着 PTA9，

```
typedef enum FTMn
{
    FTM0,
    FTM1,
    FTM2
} FTMn;

typedef enum CHn
{
    //          --FTM0--   --FTM1--   --FTM2--
    CH0,    //      PTC1       PTA8       PTA10
    CH1,    //      PTC2       PTA9       PTA11
    CH2,    //      PTC3       ×          ×
    CH3,    //      PTC4       ×          ×
    CH4,    //      PTD4       ×          ×
    CH5,    //      PTD5       ×          ×
    CH6,    //      PTD6       ×          ×
    CH7     //      PTD7       ×          ×
    // ×表示不存在
} CHn;
```

那么我们只需要在电机的调试程序里加入舵机的初始化与占空比的改变就完成了对舵机的调试。怎么加呢？如下图：
在电机模块初始化完成后，进行舵机模块的初始化，注意频率为 50HZ。

```
  /* */

  //gpio_init (PORTC , 4, GPO,HIGH);
 // gpio_init (PORTC , 2, GPO,LOW);
   FTM_PWM_init(FTM0 , CH1, 80000,0);
   FTM_PWM_init(FTM0 , CH2, 80000,0);

   FTM_PWM_init(FTM1 , CH1, 50,0);


 // uart_init (UART0 , 115200);                    //
  gpio_init (PORTA , 17, GPO,HIGH);
  gpio_init (PORTB , 17, GPI,HIGH);
  pit_init_ms(PIT0, 5);
```

在 80ms 中断里，改变舵机的占空比因此使得舵机向左与向右，由于舵机的占空比为 2.5%-12.5%，5 和 7 是试出来的，一个向左偏，另一个向右偏。

```
if(TIME0flag_80ms == 1)
{
  TIME0flag_80ms = 0 ;
if( count == 3 )
 {
    count = 2 ;
    FTM_PWM_Duty(FTM0 , CH1,0);    //设置通道占空比
    FTM_PWM_Duty(FTM0 , CH2,50);
 } else if(count == 2)
 {

    count = 1 ;
    FTM_PWM_Duty(FTM0 , CH1,0);
    FTM_PWM_Duty(FTM0 , CH2,0);
    FTM_PWM_Duty(FTM1 , CH1,5);
 }else if(count == 1)
 {

    count = 0 ;

    FTM_PWM_Duty(FTM0 , CH1,50);
    FTM_PWM_Duty(FTM0 , CH2,0);

 } else if(count == 0)
 {

    count = 3 ;
    FTM_PWM_Duty(FTM0 , CH1,0);
    FTM_PWM_Duty(FTM0 , CH2,0);
    FTM_PWM_Duty(FTM1 , CH1,7);
 }
```

2.4.4 舵机对正

上面的调试，5%向左偏，7%向右偏。这也告诉我们一个道理，用占空比调试比较麻烦，我们可以进入这个函数

```
/*************************************************************************/
void FTM_PWM_Duty(FTMn ftmn, CHn ch, u32 duty)
{
    u32 cv;
    u32 mod;

    ASSERT( (ftmn == FTM0) || ( (ftmn == FTM1 || ftmn == FTM2 ) && (ch <= CH1)) ); //检查
    ASSERT(duty <= FTM_PRECISON);      //用断言检测 占空比是否合理

    //占空比 = (CnV-CNTIN)/(MOD-CNTIN+1)
    mod = FTM_MOD_REG(FTMx[ftmn]);          //读取 MOD 的值

    cv = (duty * (mod - 0 + 1)) / FTM_PRECISON;

    // 配置FTM通道值
    FTM_CnV_REG(FTMx[ftmn], ch) = cv;

}
```

可以看到，利用公式 占空比 = (CnV-CNTIN)/(MOD-CNTIN+1) ，我们将 duty 转化为 cv 值，然后给了寄存器，因为 FTM_PRECISON = 0.1%，这样的 cv 值的范围很大，我们根据 cv 值和

```
FTM_CnV_REG(FTMx[ftmn], ch) = cv
```

来调试舵机中值。我们把

```
FTM_PWM_Duty(FTM1 , CH1, duty)
```

改成

```
FTM_CnV_REG(FTMx[FTM1], CH1) = cv
```

根据 2 分法来调试舵机中值，cv = 3000 时舵机向左转，cv=4000 时舵机向右转，舵机等于 3500 时在中间位置，有时候其实 3300，3700 都可能是中值，因为我们改变的是寄存器的值，不是很敏感，方便调试舵机。

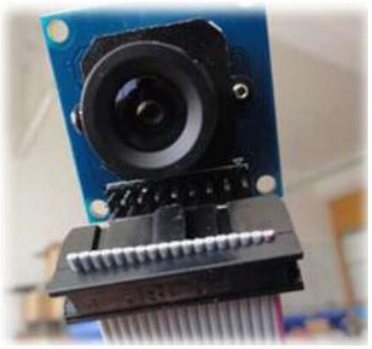
2.5 检测摄像头

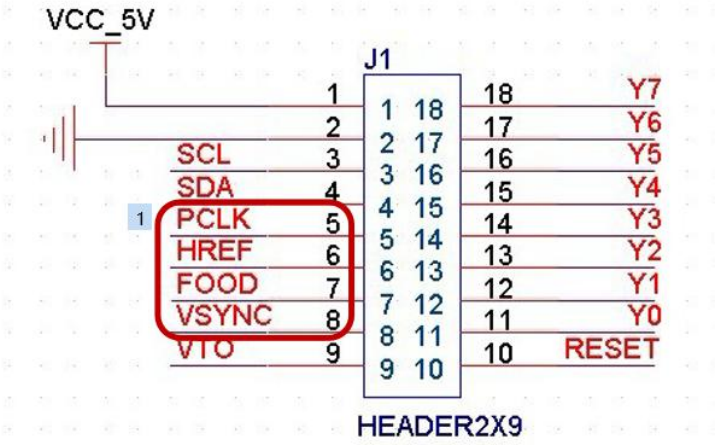
连接电源线到主板上，查看核心板的 power 灯是否亮着。把摄像头排线接到主板上，用万用表测试一下供电电压是不是 5V。如果供电正常，把排线插到摄像头上，看摄像头的指示灯是否亮。用视频卡查看，摄像头的图像。下载摄像头

测试程序，看串口是否有采集的 AD 值输出。当然我并没有这样做，看着灯亮，就觉得就开始调试啦。毕竟上一届替我们做过了这些工作。顺着以前的思路，我们先看一下，摄像头的一些资料

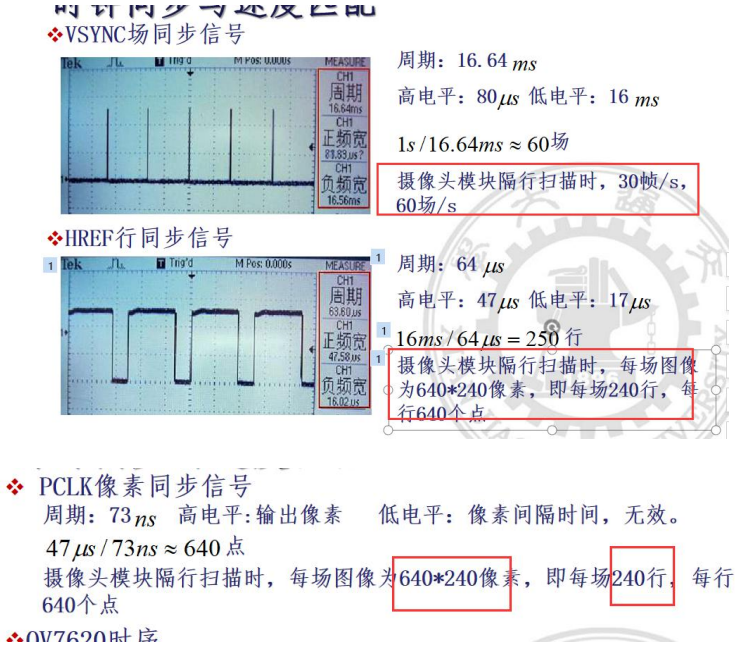
2.5.1 摄像头概述

先看一下摄像头的基本资料 [E:\BaiduYunDownload\项目设计\前三节 PPT\5 王莹]





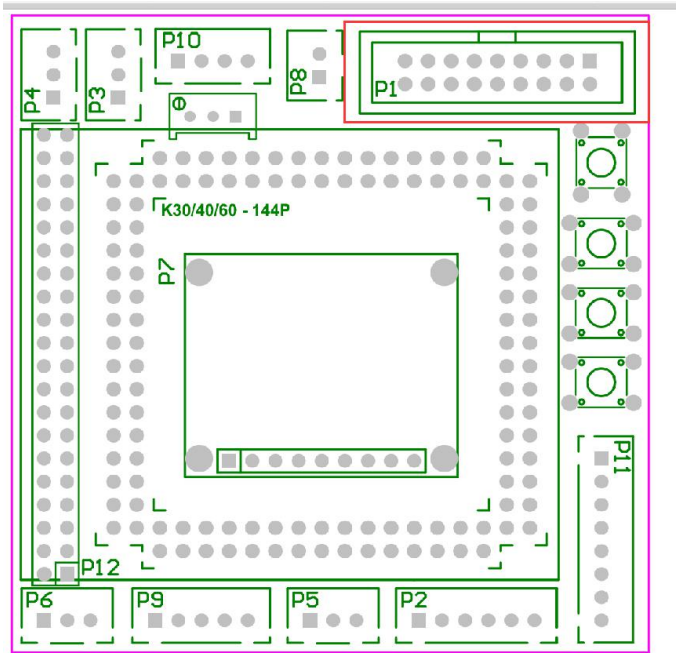
可以看到摄像头主要用到 PCLK 像素同步信号，HREF 水平同步信号，VSYNC 垂直同步信号，FOOD 奇偶场同步信号没有用到。Y0-Y7 是传输一个 8bit 的像素的，在 73ns 的 PCLK 像素同步信号下，一次传一个像素，继续往下看：



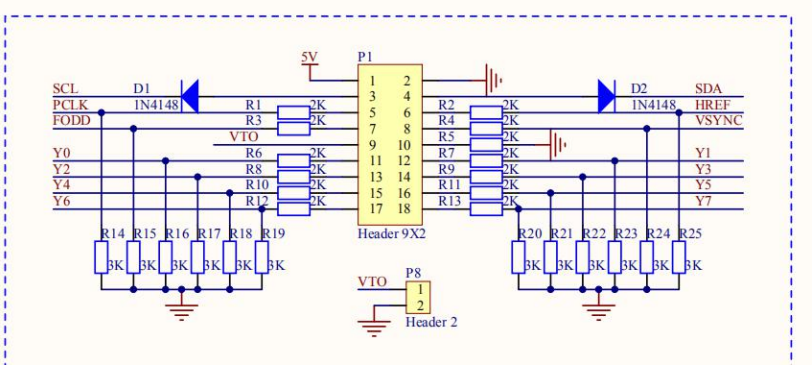
可以看到摄像头模块隔行扫描时，30 帧/s，60 场/s，采出来的图像时 240*640，它的那种表示方法和我们数字图像处理的不太一样。后期我们每个 24 行取一行，取 10 行进行处理。

2.5.2 摄像头的连接

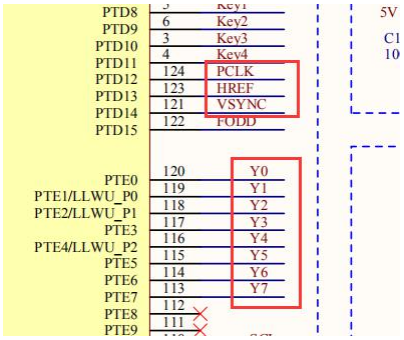
这里我就不细说了，他看 PCB 图，



是 P1 模块的电路图为



在主板上找到相应的引脚号



调试摄像头就是对这些引脚进行操作。

2.5.3 摄像头例程

1. 0v7620 采集初始化

```
/*************************************************************
函数名称：CMOS_init
函数功能：
入口参数：
出口参数：无
备 注：
*************************************************************/
extern u8  BUFF[500] ;
void CMOS_INIT (void){
    gpio_Interrupt_init(PORTD,14, GPI_UP,FALLING) ;          //场中断
    gpio_Interrupt_init(PORTD,13, GPI_DOWN,RISING) ;          //行中断
    DMA_PORTx2BUFF_Init (DMA_CH4, (void *)&PTE_BYTE0_IN, BUFF, PTD12, DMA_BYTE1, DATACOUNT, DMA_rising_down);
    //DMA通道4初始化，PTD12上升沿触发DMA传输，源地址为PTE0~PTE7，目的地址为：BUFF
    //每次传输1Byte，传输DATACOUNT（400）次后停止传输，恢复目的地址
    }


/*************************************************************
```

2. PORTD 中断服务程序

```
/*************************
摄像头采用变量
*************************/
u8 LinCout =0;
u8 LinADCout =0;
u8 AcqFlg =0 ;
u8 *linarrycot = 0;
u8 AcqAryy[] = {0X69 ,0X6A ,0X6B ,0XFF};

void PORTD_IRQHandler(){
    if(PORTD_ISFR & 0x4000)         //PTD14触发中断,采集的场中断。
    {
        PORTD_ISFR  |= 0x4000;        //写1清中断标志位
        linarrycot = &AcqAryy[0] ;
        LinCout = 0 ;

    }
    if(PORTD_ISFR & 0x2000)            //PTD13触发中断，采集的行中断
    {
        PORTD_ISFR  |= 0x2000;         //写1清中断标志位
        LinCout ++ ;
        if(*linarrycot == LinCout )     //判断是否是需要采集的行
        {
           AcqFlg = 1 ;
          if(*linarrycot != 0xff )
          {
            linarrycot ++ ;
          }
        }
        if(AcqFlg ==1  )               //需要采集行的标志号
        {
          LinADCout ++ ;
          AcqFlg = 0;
          DMA_IRQ_CLEAN(DMA_CH4);                           //清除通道传输中断标志位     (这样才能再次进入中断)
          DMA_IRQ_EN(DMA_CH4) ;                             //允许DMA通道传输
          DMA_EN(DMA_CH4);                                  //使能通道CH4 硬件请求       (这样才能继续触发DMA传输)
        }
    }
```

3. DMA 中断服务程序

```
/*!
 *  函数名称：DMA_CH4_Handler
 *  功能说明：DMA通道4的中断服务函数
 *  参数说明：设置标志位能够及时搬移。
 *  函数返回：无
 */
u8  DMA_Over_Flg = 0 ;      //行采集完成标志位
void DMA_CH4_Handler(void)
{
  //DMA通道4
    DMA_IRQ_CLEAN(DMA_CH4) ;    //清除通道传输中断标志位
    DMA_IRQ_DIS(DMA_CH4);       //禁止DMA通道传输
    DMA_DIS(DMA_CH4);           //禁止通道硬件DMA请求
    DMA_Over_Flg = 1 ;          //设置行采集完成标志位

}
```

4. 数据搬移

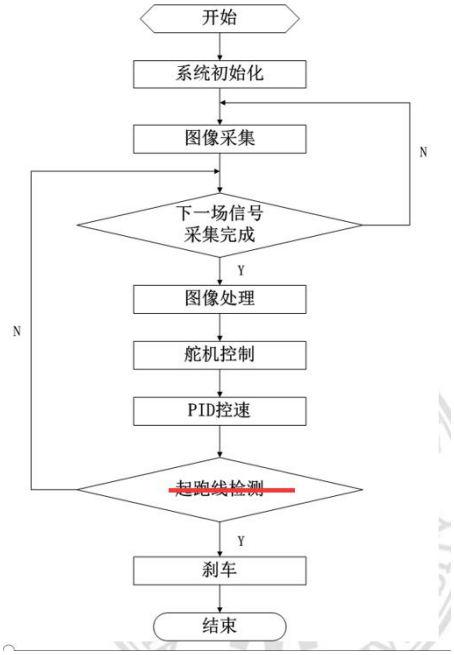
main.c | calculation.c | isr.c * | gpio.c | MK60DZ10.h | gpio.h | nokia.c | common.h | calculation.h

```
142       if(DMA_Over_Flg == 1)
143       {
144         DMA_Over_Flg = 0 ;
145         ALineOverCout ++ ;
146         if(LinADCout >= DATALINE )
147             LinADCout = 0 ;
148         Atemp0 = LinADCout  * DATACOUNT ;
149         finger = &ADdata[Atemp0] ;
150         finger1 = &BUFF[0] ;
151         for(Atemp0 = 0 ;Atemp0 < DATACOUNT;Atemp0++ ){
152             *finger = *finger1 ;
153             finger1 ++ ;
154             finger ++ ;
155         }
```

三、项目设计实现

3.1 整体框架

到此我们就可以实现我们自己程序里。我们的流程类似



1. 在摄像头程序采集中，只用了 10 行数据。
2. 找黑线，本历程的找黑线程序，是先二值划，再根据双线特征，进行黑线查找
3. 。在用 PID 算法，算出转向 PWM。计算转向值程序，是需要根据道路的实际情况结合摄像头前瞻和舵机响应时间，计算出来的值。
4. 计算转向值，根据视觉中心与赛道中心，给舵机一个适当的占空比，改变舵机角度。
5. 速度算法，需要根据自己赛道是直道和转弯半径给出合适的目标车速。还需要对车速响应做出调整，这里主要是看需要不需要刹车和加速候的 PID 值。

起跑线的检测我们就不需要了。

3.2 初始化

起初我们进行电机，舵机与摄像头的初始化，程序为：

```
    /*****************************
    初始化
    *****************************/
   FTM_PWM_init(FTM0, CH1 , 8000,0);  //初始化电机
   FTM_PWM_init(FTM0, CH2 , 8000,0);

   FTM_PWM_init(FTM1, CH1, 50, 5);    //初始化舵机

   CCD_INC ();      //初始化摄像头

   gpio_init (PORTA , 17, GPO,LOW);  //点灯

   EnableInterrupts;                                    //开总中断

   /*****************************************
    执行程序
    *****************************************/
```

初始化之前必须要先关闭中断，之后再打开，以防在程序中跑飞。初始化失败。对于摄像头的初始化正如前面例程所说：

```
*******************************************************/
extern u8  BUFF[500] ;
extern u8  ADdata[DATALINE][DATACOUNT];

void CCD_INC (void){
     gpio_Interrupt_init(PORTD,14, GPI_UP,FALLING) ;          //场中断
     gpio_Interrupt_init(PORTD,13, GPI_UP, RING) ;          //行中断
     DMA_PORTx2BUFF_Init (DMA_CH4, (void *)&PTE_BYTE0_IN, ADdata[0], PTD12, DMA_BYTE1, DATACOUNT, DMA_rising_keepon);
      //DMA通道4 ，PTE0-PTE7 数据采集，ADdata数据存储位置，PTD12触发通道，DMA_BYTE1  数据存储长度，DATACOUNT  数据总个数，DMA_rising_keepon 触发方式
     set_irq_priority(90,0) ;    //设置优先级


     }
```

我们用全局变量数组 AData[10][400]来存放从摄像头采集来的 10 行数据，

```
DMA_PORTx2BUFF_Init (DMA_CH4, (void *)&PTE_BYTE0_IN, ADdata[0], PTD12,
DMA_BYTE1, DATACOUNT, DMA_rising_keepon);
```

DMA_PORTx2BUFF_Init()采用 DMA 通道 4 ，PTE0-PTE7 数据采集，ADdata 数据存储位置，PTD12 触发通道，DMA_BYTE1 数据存储长度 1，DATACOUNT 数据总个数 400，DMA_rising_keepon 触发方式上升沿。其中 DMA_CH4 与 DMA_BYTE1 都是枚举变量。

```
typedef enum DMA_BYTEn       //DMA每次传输字节数
{
    DMA_BYTE1 = 0,
    DMA_BYTE2 = 1,
    DMA_BYTE4 = 2,
    DMA_BYTE16 = 4
} DMA_BYTEn;


typedef enum DMA_CHn
{
    DMA_CH0,
    DMA_CH1,
    DMA_CH2,
    DMA_CH3,
    DMA_CH4,
    DMA_CH5,
    DMA_CH6,
    DMA_CH7,
    DMA_CH8,
    DMA_CH9,
    DMA_CH10,
    DMA_CH11,
    DMA_CH12,
    DMA_CH13,
    DMA_CH14,
    DMA_CH15
} DMA_CHn;
```

代表 16 个 DMA 通道和 DMA 每次传输字节数。

3.3 摄像头数据的采集

PTD12 触发通道对应着 PCLK 像素同步信号，之前我们说过，每个 24 行采集一次，总共采集 10 行结束，就是对应着这个程序。

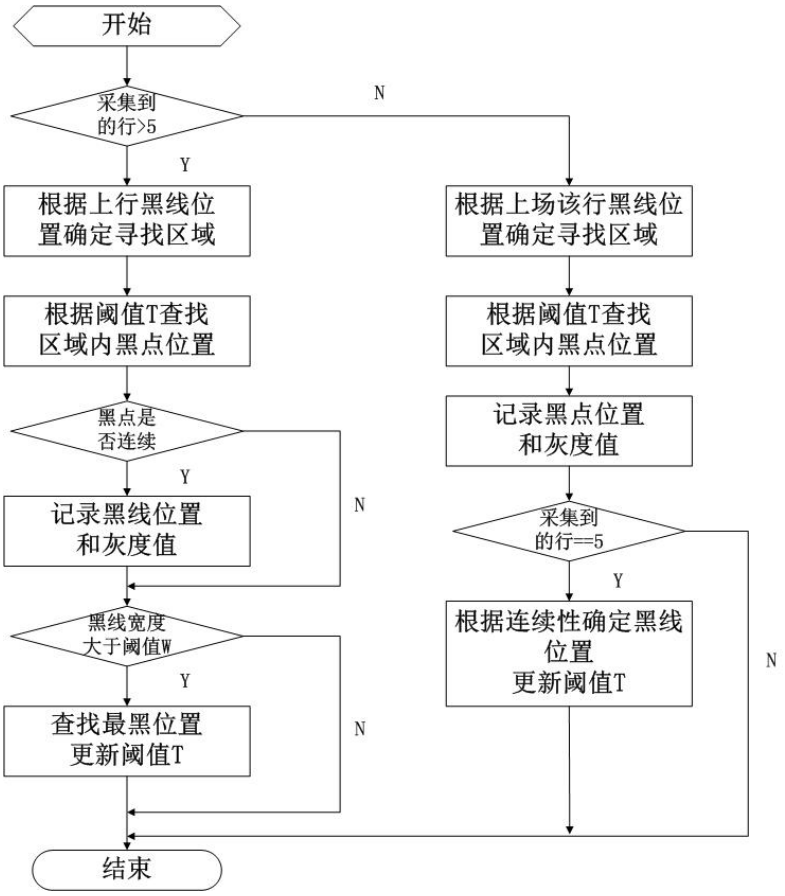
```
void PORTD_IRQHandler(){

    if(PORTD_ISFR & 0x2000)           //PTD13触发中断，采集的行中断
    {
        PORTD_ISFR  |= 0x2000;        //写1清中断标志位

        if(Sample_Flag==0)
      {
        return;
      }

        if((LinCout%(240/DATALINE)==0)&&(LinADCout<DATALINE))
        {
         // dlay() ;      //摄像头行中断改为下降沿触发，有效点前面有一段行消隐区，故需要一段延时消除消耗掉，该处的时间，程序里面的需要根据单片机的主频调整。
          DMA_DADDR(DMA_CH4) = (uint32_t)ADdata[LinADCout] ;   //数据存储地址变化
          DMA_EN(DMA_CH4);                                    //使能通道CHn 硬件请求       (这样才能继续触发DMA传输)
          DMA_IRQ_EN(DMA_CH4) ;                              //允许DMA通道传输
        //  PTA16_OUT = ~PTA16_OUT ;
         LinADCout ++ ;
        }
        LinCout++;

         if(LinADCout==DATALINE)
         {
           LinADCout=0;
//          DMA_DIS(DMA_CH4);
//           DisableInterrupts;
          DMA_Over_Flg = 1 ;
         }
    }

    if(PORTD_ISFR & 0x4000)         //PTD14触发中断,采集的场中断。0100 00000 00000  0000
    {
        PORTD_ISFR  |= 0x4000;        //写1清中断标志位


        LinCout = 0 ;
        LinADCout=0;
        Sample_Flag=1;
        gpio_Interrupt_init(PORTD,13, GPI_UP, RING) ;          //开启行中断
        //PORTD_ISFR  |= 0x2000;        //写1清中断标志位
    }

}
```

采集到的 10*400 的数据存放在全局变量数组 AData[][]中。

3.4 摄像头数据的处理

摄像头的处理包括了图像二值化与寻找黑线，我们的思路与下面的流程图相

符合，不过我们每次都采集 10 行，因此第一个条件就”采集到的黑线行数>5”不用判断啦。



3.4.1 图像二值化

我们可以设置一个全局阈值对图像进行简单的二值化，程序为：

```
}
//图像二值化
void BinaData()
{
  int i=0,j=0;
  unsigned char *p;
  for(i=0;i<V;i++)
  {
    p=ADdata[i];
    for(j=0;j<H;j++)
    {
      if(*(p+j)>=threshold)
      {
        Bina_data[i][j]=255;
      }
      else
        Bina_data[i][j]=0;
    }
  }
}
```

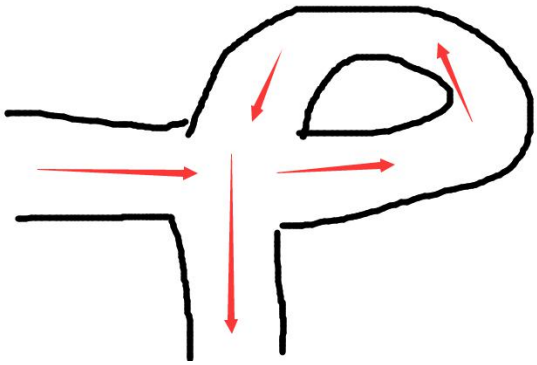
用两个 for 循环遍历一个二维数组，然后 if 条件做判断，如果此像素大于阈值 128 则设置为 255，否则为 0。

3.4.2 寻找黑线与赛道中心

我从图像的中心相两边找，如果找到三个连续的点则为黑线。程序为

```
//提取中心线
void get_center()
{
  unsigned char *p;
  int last_center=linecenter;                              //#define   linecenter        H/2
  for(CCD_V=V-1;CCD_V>=0;CCD_V--)                         //V=10
  {
    p=Bina_data[CCD_V];
    for(CCD_H=last_center;CCD_H>1;CCD_H--)//中间往右
    {
      if((*(p+CCD_H)==0) && (*(p+CCD_H-1)==0) && (*(p+CCD_H-2)==0)) //右边黑线检测
      {
        RightBorder[CCD_V]=CCD_H;
        break;
      }
      else     RightBorder[CCD_V]=0;       //如果右边没有检测到有黑线

    }

   for(CCD_H=last_center;CCD_H<H-2;CCD_H++)   //中间往左
   {
      if((*(p+CCD_H)==0) && (*(p+CCD_H+1)==0) && (*(p+CCD_H+2)==0))     //左边黑线检测
      {
        LeftBorder[CCD_V]=CCD_H;
        break;
       }
      else    LeftBorder[CCD_V]=H-1;     //如果左边没有检测到有黑线

   }
    CenterLine[CCD_V]=(LeftBorder[CCD_V]+RightBorder[CCD_V])/2; //计算中心线
    last_center=CenterLine[CCD_V];

  }
}
```

两边黑线的 1/2 出为赛道中心，保存在 CenterLine[]数组里，这对于下图的弯道，



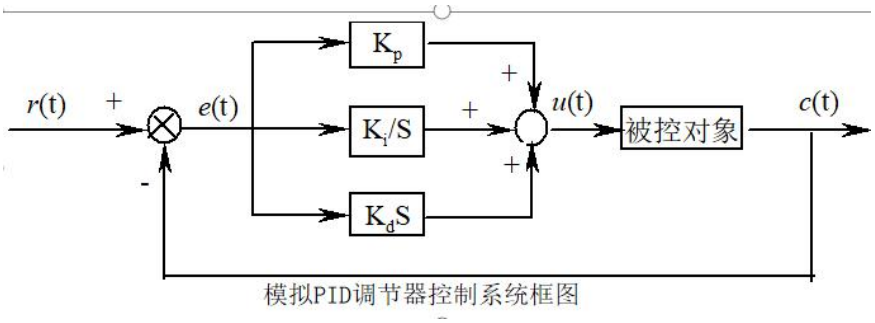
当我们检测不到黑线的位置的时候，我们应该就取图像的中心，也就是视觉中心与赛道中心重合，可以保持前行。

3.5 PID 算法

在工业控制中，比例+积分+微分的控制规律是一种常用的方法，可适用于多种被控对象。PID 调节器是一种线性调节器，它的实质是将设定值 r（t）与输出值 c（t）进行比较构成控制偏差

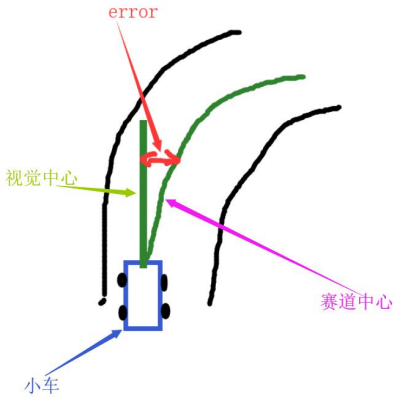
$$e(t)=r(t)-c(t)$$

将其按比例、积分、微分运算后，并通过线性组合构成控制量，如图所示，所以简称为 P（比例）、I（积分）、D（微分）调节器。



模拟PID调节器控制系统框图

这个误差我们以视觉中心与赛中中心的距离作为误差



由此我们便可以计算得到误差，将中间 5 个 error 取平均，然后用 PID 得到 u(t)也就是 result，程序如图：

```
//计算中心线
err =0 ;
for(CCD_V=ROW;CCD_V<ROW+5;CCD_V++)    err = err + (CCCline - CenterLine[CCD_V]);
 err = err/5;



pgain = SD_PID->fKp;
igain = SD_PID->fKi;
dgain = SD_PID->fKd;

pterm = pgain * err;  //P项
iterm += igain * err;
dterm = ((err - olderr)) * dgain;

 /*if(fabsf(err) < 10) {dterm = 0;pterm=0;iterm = 145;}*/
//  if((fabsf(CenterLine[top_line] - CenterLine[top_line/2]) < 4) && (top_line >=45))
//    {dterm = 0;iterm=145;}

result = -pterm - dterm;
```

3.6 舵机控制

根据视觉中心与赛道中心的误差来进行舵机的控制，我们很自然的想到，将 error 放大一定倍数加到舵机中值上，如果每次转弯的方向是相反的，我们就用舵机中值减去得到的误差。如下图所示：

```
result = -pterm - dterm;

if(result>=1000)
  result=1000;
if(result<=-1000)
  result=-1000;


olderr = err;
//oldolderr = olderr;
ATurnPWM=MIDSTRING-result;
FTM_CnV_REG(FTMx[FTM1], CH1) = ATurnPWM;
```

可能会注意到，为什么要对 result 进行判断，那是因为舵机的占空比取值范围不大，如果大于一定程度，角度超过一定极限容易损伤舵机，而且不一定左右都要在 1000 范围内，根据具体情况，具体调整。

3.7 电机控制

此次项目设计，我们的电机占空比在 25%左右，占空比不是很大，程序如图

```
steerPIDA(&SD5_PID,3);

if(result<0){
  speedppp=(u32)-result;}
else
{speedppp=(u32)result;}
speedddd=21+speedppp/160;
FTM_PWM_Duty(FTM0, CH2,  0);
FTM_PWM_Duty(FTM0, CH1, speedddd);


}
```

但是在转弯的时候，由于舵机在极限位置，小车可能会卡着慢慢移动，这时候我们应该适当的增加速度，以保证在转弯的时候，小车正常行驶，不会出现停下来，缓慢转弯的情况。

四、总结

此次项目设计到这里也算告一段落，其中学到了很多，比如熟悉了一下基本环境的搭建，由于例程比较详细，这里显得轻松了许多。对于整车的调试，需要各模块进行，这样很轻松。在一个就是在海量的信息中找到我们有用的信息，这个很重要，信息爆炸的时代，如果对信息过滤的好，那必然会赢得时代，做一个弄潮儿，可我这种能力有待加强。因为首先是 Thomas 调试了程序，告诉了我有用的文档，我在开始开发的。在开发过程中，我也是表现出来极大的陌生感，因为对电路以及硬件调试的不熟，自己又好高骛远，不是很情愿动手动脑，导致最后开发速度慢，但在其中也学到了，边看开发文档边写程序，边读例程，找对应引脚，理清逻辑，边修改程序，实现自己想要的功能。未来的路还很长，希望自己静下心来，认认真真做每一件事，要想成功，兴趣与坚持，一直要动手，天生聪明，不如每天动手。

五、附录

main.c

```
#include "include.h"
#include "calculation.h"
#include "camera.h"
#include "common.h"
#include "steerPID.h"

/************************
设置系统的全局变量
************************

uint16 ATurnPWM=MIDSTRING ;
extern  u8  LPT_INT_count ;
extern  u8  DMA_Over_Flg ;            //采集完成标志位
extern  u8  LinADCout ;
extern float result;
u32 speedddd;
u32 speedppp;
u16 LinCout =0;
u8  BUFF[500] ;

u8 ADdata[DATALINE][DATACOUNT] ={ 0  } ;        //黑线 AD 数组存储

void main()
{

    DisableInterrupts;                              //禁止总中断

  /*********************************************************
  初始化程序
  *********************************************************/
   //自行添加代码

   uart_init (UART0 , 115200);                          //初始化 UART0，输出脚
PTA15，输入脚 PTA14，串口频率 9600

    /*************************
    初始化摄像头采样
    *************************/
   FTM_PWM_init(FTM0, CH1 , 8000,0);  //初始化电机
   FTM_PWM_init(FTM0, CH2 , 8000,0);
```

```
  FTM_PWM_init(FTM1, CH1, 50, 5);   //初始化舵机

  CCD_INC ();     //初始化摄像头
  gpio_init (PORTA , 17, GPO,LOW);  //点灯
  EnableInterrupts;                                 //开总中断

  /*****************************************
   执行程序
   *****************************************/
while(1)
   {
 //FTM_CnV_REG(FTMx[FTM1], CH1) = 3500;
       if(DMA_Over_Flg == 1)
     {
       DMA_Over_Flg = 0 ;
       BinaData();
        //二值化
       get_center();
        //检测中心线
       STEER_PID_init(&SD5_PID,5.8,0.0,0.6);
       steerPIDA(&SD5_PID,3);
       if(result<0){
         speedppp=(u32)-result;}
       else
       {speedppp=(u32)result;}
       speeddddd=21+speedppp/160;
       FTM_PWM_Duty(FTM0, CH2,  0);
       FTM_PWM_Duty(FTM0, CH1, speeddddd);
     }
   }
}
```