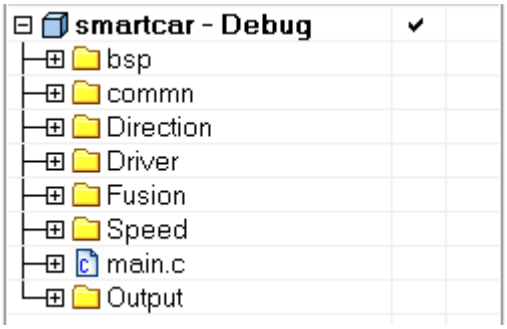
岱默直立车整车程序要点

本文是对武汉岱默科技有限公司的直立车教学套餐整车示例程序的详细注释，直立车教学套餐是整套的硬件方案，安装时由于机械安装的差异，程序需要做对应的调整，本文即使对相关要点的描述，请大家耐心阅读。

1. 工程组成

下图为 CCD 直立车模工程文件的组成：



bsp 文件夹主要是对所有模块的初始化；

commn 文件夹包含串口发送，中断以及系统文件；

Direction 文件夹包含 CCD 数据处理和方向控制文件；

Driver 文件夹包含芯片各模块的驱动文件；

Fusion 文件夹包含加速度计陀螺仪信号融合及控制算法；

Speed 文件夹包含速度处理及控制算法；

Main 函数里面只有一个发现车的程序段；

2. 使用要点

2.1. 直立部分：

Fusion.c 文件里面有六个参数需要调试，即：

```
void Data_Init(void)
{
  Angle_Offset =1360;//1385
  AngleVeloc_Offset = 1800;

  Tz = 1.0;          //时间常数
  Rgyro = 4.0;

  Kangle=4.1;
  Dangle=0.6;
  //5.5 0.8
}
```

零偏值的设定：

`Angle_Offset` 为加速度计的零偏值，获取方法为：将车模放置于平衡位置附近，采集到的加速度计的 AD 值。

`AngleVeloc_Offset` 为陀螺仪的零偏值，获取方法为：车模静止不动时采集到的陀螺仪 AD 值。

曲线融合：

打开主函数里面的发送函数

```
WirelessSerial(Angle,angle,AngleVeloc,AngleVelocAD);
```

将融合前后的角度 `angle` 和 `Angle` 发送到上位机，更改

```
Tz = 1.0;          //时间常数
Rgyro = 4.0;
```

这两个参数，使得 `Angle` 紧紧跟随 `angle` 的信号，其中 `Tz` 基本在 1 左右，因此调试重点放在 `Rgyro` 上。

直立调试：

直立控制的 PD 参数

```
Kangle=4.1;
Dangle=0.6;
```

在调试时满足在车模不抖动的条件下使得参数尽可能大，这样车模直立的状态会很好。先从 `Dangle` 开始调，然后再慢慢增大 `Kangle`。

2.2. 速度控制部分：

Speed.c 文件里面包含了速度控制的算法：

速度设定值：

```
SpeedSet = 800;//设定速度
```

更改这个参数能够修改车模设定的速度，我们速度采集的周期是 100ms 一次，因此这个设定速度即为 800 个脉冲/100ms。

速度控制参数：

```
//对于不同的速度，给予不同的速度控制参数
if(Speed<=200)
{
  Kspeed =0.3,Ispeed = 0.05;
}
else if((Speed>=200)&&(Speed<=600))
{
  Kspeed =0.5,Ispeed = 0.06;
}
else
{
  Kspeed =0.65,Ispeed = 0.1;
}
```

在不同的速度时给的控速参数有差异，低速时参数不能给得过大，否则会造成车模起步控制力度过大接触地面，起步失败。

起步操作：

在 main 函数里面

```
if(Button1==0)//如果拨码开关按下，启动发车计时
{
  if(Just_onetime)
  {
    Just_onetime=0;
    SpeedStartupTime=GetOffsetTime;
  }
}
```

该函数用来检测发车按键（主板上的拨码开关），按下去之后`SpeedStartupTime=`记录按键按下的时刻，`GetOffsetTime`在中断函数中一直保持 1ms 自加一次的状态，加到 60000 次即 1 分钟位置，因此上电之后要在 48 秒之内发车。

```
(GetOffsetTime>=(SpeedStartupTime+2000))&&(Just_onetime==0)
```

通过这个判断函数实现按键按下之后 2S 发车。

2.3.方向控制部分：

CCD 安装部分：

本程序 CCD 传感器的安装基本要求为：

1. CCD 的前瞻为 45cm 左右（即 CCD 打在赛道上的位置与车轮间的距离）
2. CCD 采集到的图像，赛道宽度要占到 128 个点的 1/3 到 1/2 左右

找寻阈值：

Direction_process.c 文件里面

```
unsigned char PixelAverage(unsigned char len, unsigned char *data)
```

该函数是用来找寻阈值的，采用平均阈值的方法，计算 128 个点的平均值作为阈值。

二值化及寻找左右边线：

Direction_process.c 文件里面

```
void CCDDataProcess(void)
```

该函数用来进行二值化和寻找左右边线，左右边线的寻找方法采用上升下降沿的方式来寻找，为了保证提取边线准确，最后从记录的几组跳变沿中选取最宽的部分作为赛道的左右边线。即：

```
for(tsss=1;tsss<10;tsss++)
{
     if((linmapr[tsss-1]-linmapl[tsss-1])>(linmapr[tsss]-linmapl[tsss]))
     {
        linmapl[tsss]=linmapl[tsss-1];//记录左右边沿之差最大的下标
        linmapr[tsss]=linmapr[tsss-1];
   }
}

LINL=linmapl[9];
LINR=linmapr[9];
```

求取偏差，转向控制：

Direction_process.c 文件里面

```
void DirectProcess()
```

```
LostLinelFlag1 =0;   //丢失左边线标志位

LostLinerFlag1 =0;   //丢失右边线标志位

CrossroadFlag  =0;   //交叉道标志位
```

赛道类型判断：

该函数里面三个标志位，根据需找的左右边线情况来判断是丢线，十字弯道还是普通的情况。以下为判断原则：

```
//左右丢线的一边的坐标是确定的，另外一边为刚开始丢线的时刻
if((LINR<=96)&&(LINL<=24))
{
    LostLinelFlag1 =1;
}

if((LINL>=49)&&(LINR>=115))
{
      LostLinerFlag1=1;
}

if((LINL<29)&&(LINR>112))
    CrossroadFlag=1;
```

计算中心线：

丢线和不丢线的情况下计算中心线的方法：

```
 LINCT=(LINL+LINR)/2;//计算赛道中心线投射到ccd像素点上的位置

 if(LostLinelFlag1)
   LINCT=LINR-(LastRoadwidth+1)/2;

 if(LostLinerFlag1)
    LINCT=LINL+(LastRoadwidth+1)/2;

  if(CrossroadFlag)
 {
       LINCT=61;
 }
```

其中LastRoadwidth为非丢线情况下记录的赛道宽度，不断更新：

```
 //记录8次赛道的宽度
if(!LostLinelFlag1 && !LostLinerFlag1 && !CrossroadFlag)
{
    LastRoadwidth=LINR-LINL;

    for(tsss=7;tsss>0;tsss--)
    {
        Roadwidth[tsss]=Roadwidth[tsss-1];
    }

    Roadwidth[0]=LastRoadwidth;

    LastRoadwidth=(uint)(0.3*Roadwidth[0]+0.2*Roadwidth[1]+0.2*Roadwidth[2]+

}
```

计算偏差：

```
DirectANew=(LINCT-61)/40.0;
```

其中 61 是根据将车模放置于赛道中心位置，并且车体正朝前方时计算出来的中心线。

偏差限幅：

```
//偏差限幅
if(DirectANew>=0.7)
   DirectANew=0.7;

if(DirectANew<=-0.7)
   DirectANew=-0.7;
```

转向参数的调试：

通过修改下面的 KdirA 和 DdirA 来进行调试，可以按照 DdirA 大致等于 8 倍的 KdirA 来进行。

```
//对速度控制参数进行赋值
KdirA = 100;
DdirA = 800;
```