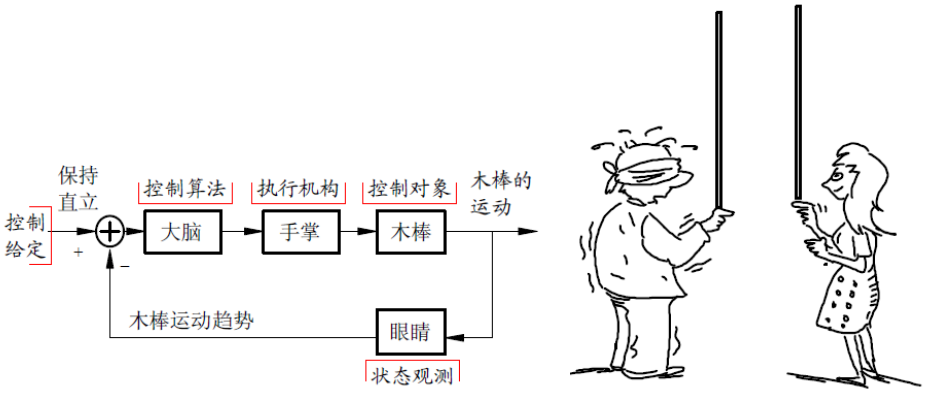
平衡原理

一、平衡小车原理

平衡小车是通过两个电机运动下实现小车不倒下直立行走的多功能智能小车，在外力的推拉下，小车依然保持不倒下。这么一说可能还没有很直观的了解究竟什么是平衡小车，不过这个平衡小车实现的原理其实是在人们生活中的经验得来的。如果通过简单的练习，一般人可以通过自己的手指把木棒直立而不倒的放在指尖上，所以练习的时候，需要学会的两个条件：一是放在指尖上可以移动，二是通过眼睛观察木棒的倾斜角度和倾斜趋势（角速度）。通过手指的移动去抵消木棒倾斜的角度和趋势，使得木棒能直立不倒。这样的条件是不可以缺一的，实际上加入这两个条件，控制过程中就是负反馈机制。

而世界上没有任何一个人可以蒙眼不看，就可以直立木棒的，因为没有眼睛的负反馈，就不知道笔的倾斜角度和趋势。这整个过程可以用一个执行式表达：



平衡小车也是这样的过程，通过负反馈实现平衡。与上面保持木棒直立比较则相对简单，因为小车有两个轮子着地，车体只会在轮子滚动的方向上发生倾斜。控制轮子转动，抵消在一个维度上倾斜的趋势便可以保持车体平衡了。

所以根据上述的原理，通过测量小车的倾角和倾角速度控制小车车轮的加速度来消除小车的倾角。因此，小车倾角以及倾角速度的测量成为控制小车直立的关键。我们的亚博智能平衡小车使用了测量倾角和倾角速度的集成传感器陀螺仪-MPU6050（模块详细介绍在亚博智能平衡小车光盘资料3.硬件资料中）。

二、角度（物理分析 PD 算法）

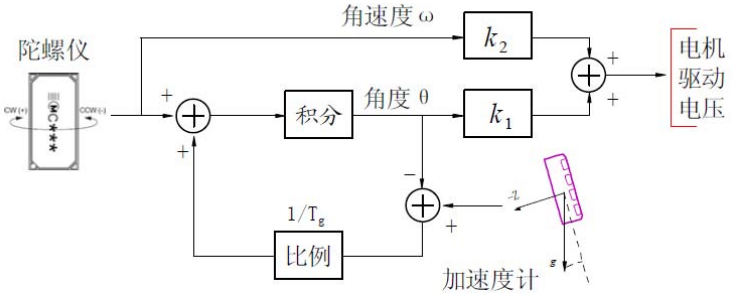


图 1

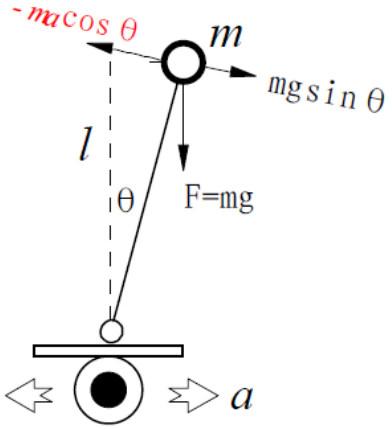


图2

控制平衡小车，使得它作加速运动。这样站在小车上（非惯性系，以车轮作为坐标原点）分析倒立摆受力，它就会受到额外的惯性力，该力与车轮的加速度方向相反，大小成正比。这样倒立摆(如图 2)所受到的回复力为：公式 1

$F = mg\sin\theta - ma\cos\theta \approx mg\theta - mk_1\theta$ 式 1 中，由于 θ 很小，所以进行了线性化。假设负反馈控制是车轮加速度 a 与偏角 θ 成正比，比例为 k_1。如果比例

k1>g，（g 是重力加速度）那么回复力的方向便于位移方向相反了。

而为了让倒立摆能够尽快回到垂直位置稳定下来，还需要增加阻尼力。增加的阻尼力与偏角的速度成正比，方向相反，因此公式1可改为：

F = mg θ-mk1 θ -mk2 θ`

按照上述倒立摆的模型，可得出控制小车车轮加速度的算法：

a =k1 θ+k2 θ` 式中 θ 为小车角度，θ`为角速度。k1 k2都是比例系数

根据上述内容，建立速度的比例微分负反馈控制，根据基本控制理论讨论小车通过闭环控制保持稳定的条件（这里需要对控制理论有基本了解）。假设外力干扰引起车模产生角加速度x（t）。沿着垂直于车模地盘方向进行受力分析，可以得到车模倾角与车轮运动加速度以及外力干扰加速度a（t）x（t）之间的运动方程。如图3所示。

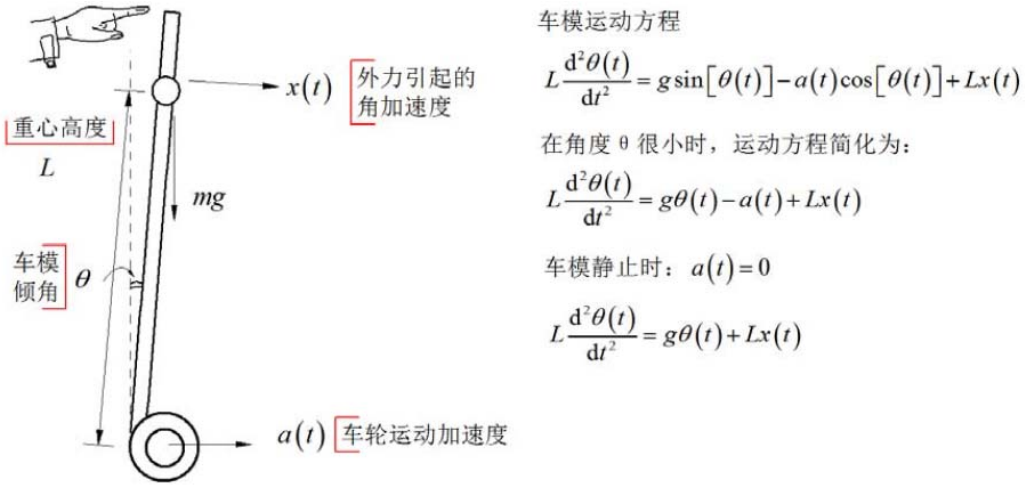


图3

在角度反馈控制中，与角度成比例的控制量是称为比例控制；与角速度成比例的控制量称为微分控制（角速度是角度的微分）。因此上面系数 k1,k2 分别称为比例和微分控制参数。其中微分参数相当于阻尼力，可以有效抑制车模震荡。通过微分抑制控制震荡的思想在后面的速度和方向控制中也同样适用。

总结控制车模直立稳定的条件如下：

（1）能够精确测量车模倾角 θ 的大小和角速度 θ '的大小；

（2）可以控制车轮的加速度。

上述控制实际结果是小车与地面不是严格垂直，而是存在一个对应的倾角。在重力的作用下，小车会朝着一个方面加速前进。为了保持小车的静止或者匀速运动需要消除这个安装误差。在实际小车制作过程中需要进行机械调整和软件参数设置。另外需要通过软件中的速度控制来实现速度的稳定性。在小车角度控制中出现的小车倾角偏差，使得小车在倾斜的方向上产生加速。这个结果可以用来进行小车的速度控制。下面将利用这个原理来调节小车的速度。

三、测速（物理模型 建立数学模型 传递函数 PD 算法）

假设小车在上面直立控制调节下已经能够保持平衡了，但是由于安装误差，传感器实际测量的角度与车模角度有偏差，因此小车实际不是保持与地面垂直，而是存在一个倾角。在重力的作用下，小车就会朝倾斜的方向加速前进。控制速度只要通过控制小车的倾角就可以实现了。具体实现需要解决三个问题：

（1）如何测量小车速度？

（2）如何通过小车直立控制实现小车倾角的改变？

（3）如何根据速度误差控制小车倾角？

第一个问题可以通过安装在电机输出轴上的光码盘来测量得到小车的车轮速度。如图 4 所示。利用控制单片机的计数器测量在固定时间间隔内速度脉冲信号的个数可以反映电机的转速。

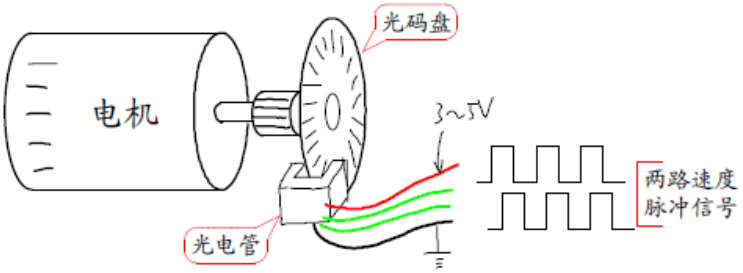


图4

第二个问题可以通过角度控制给定值来解决。给定小车直立控制的设定值，在角度控制调节下，小车将会自动维持在一个角度。通过前面小车直立控制算法可以知道，小车倾角最终是跟踪重力加速度Z轴的角度。因此小车的倾角给定值与重力加速度Z轴角度相减，便可以最终决定小车的倾角

第三个问题分析起来相对比较困难，远比直观进行速度负反馈分析复杂。首先对一个简单例子进行分析。假设小车开始保持静止，然后增加给定速度，为此需要小车往前倾斜以便获得加速度。在小车直立控制下，为了能够有一个往前的倾斜角度，车轮需要往后运动，这样会引起车轮速度下降（因为车轮往负方向运动了）。由于负反馈，使得小车往前倾角需要更大。如此循环，小车很快就会倾倒。原本利用负反馈进行速度控制反而成了“正”反馈。

为什么负反馈控制在这儿失灵了呢？原来在直立控制下的小车速度与小车倾角之间传递函数具有非最小相位特性（在此省略了分析），在反馈控制下容易造成系统的不稳定性。

为了保证系统稳定，往往取的小车倾角控制时间常数T_Z很大。这样便会引起系统产生两个共轭极点，而且极点的实部变得很小，使得

系统的速度控制会产生的震荡现象。这个现象在实际参数整定的时候可以观察到。那么如何消除速度控制过程中的震荡呢？

要解决控制震荡问题，在前面的小车角度控制中已经有了经验，那就是在控制反馈中增加速度微分控制。但由于车轮的速度反馈信号中往往存在着噪声，对速度进行微分运算会进一步加大噪声的影响。为此需要对上面控制方法进行改进。原系统中倾角调整过程时间常数往往很大，因此可以将该系统近似为一个积分环节。将原来的微分环节和这个积分环节合并，形成一个比例控制环节。这样可以保持系统控制传递函数不变，同时避免了微分计算。

但在控制反馈中，只是使用反馈信号的比例和微分，没有利误差积分，所以最终这个速度控制是有残差的控制。但是直接引入误差积分控制环节，会增加系统的复杂度，为此就不再增加积分控制，而是通过与角度控制相结合后在进行改进。

要求小车在原地停止，速度为0。但是由于采用的是比例控制，如果此时陀螺仪有漂移，或者加速度传感器安装有误差，最终小车倾角不会最终调整到0，小车会朝着倾斜的方向恒速运行下去。注意此时车模不会像没有速度控制那样加速运行了，但是速度不会最终为0。为了消除这个误差，可以将小车倾角设定量直接积分补偿在角度控制输出中，这样就会彻底消除速度控制误差。第二点，由于加入了速度控制，它可以补偿陀螺仪和重力加速度的漂移和误差。所以此时重力加速度传感器实际上没有必要了。

此时小车在控制启动的时候，需要保持小车的垂直状态。此时

陀螺仪的积分角度也初始化为0。当然如果电路中已经包括了重力加速度传感器，也可以保留这部分，从而提高小车的稳定性。在后面的最终给定的控制方案中，保留了这部分的控制回路。

四、转向控制（PD 算法）

通过左右电机速度差驱动小车转向消除小车距离道路中心的偏差。通过调整小车的方向，再加上车前行运动，可以逐步消除小车距离中心线的距离差别。这个过程是一个积分过程，因此小车差动控制一般只需要进行简单的比例控制就可以完成小车方向控制。但是由于小车本身安装有电池等比较重的物体，具有很大的转动惯量，在调整过程中会出现小车转向过冲现象，如果不加以抑制，会使得小车过度转向而倒下。根据前面角度和速度控制的经验，为了消除小车方向控制中的过冲，需要增加微分控制。

五、全方案整合

通过上面介绍，将车模直立行走主要的控制算法集中起来，如图

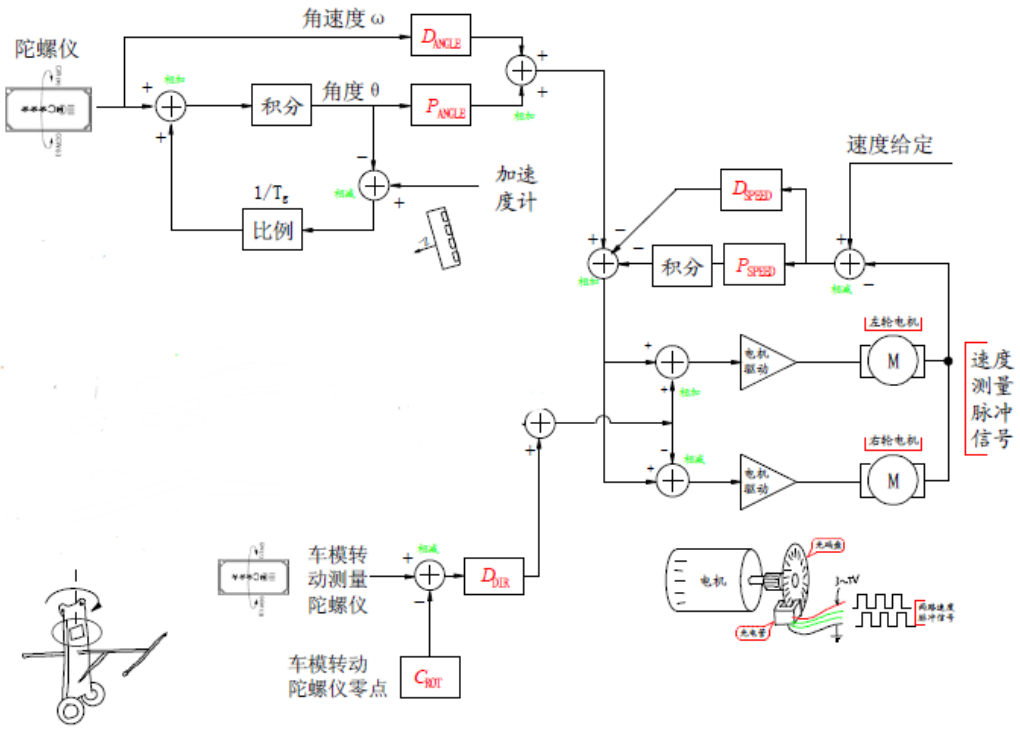


图5

为了实现小车直立行走，需要采集如下信号：

（1）小车倾角速度陀螺仪信号，获得小车的倾角和角速度。

（2） 重力加速度信号（z轴信号），补偿陀螺仪的漂移。该信号可以省略，有速度控制替代。

（3） 小车电机转速脉冲信号，获得小车运动速度，进行速度控制。

（4） 小车转动速度陀螺仪信号，获得小车转向角速度，进行方向控制。

在小车控制中的直立、速度和方向控制三个环节中，都使用了比例微分（PD）控制，这三种控制算法的输出量最终通过叠加通过电机运动来完成。

（1）小车直立控制：使用小车倾角的PD（比例、微分）控制；

```
g_fAngleControlOut =  g_fCarAngle * g_fCarAngle_P + \
 gyro[0] * g_fCarAngle_D ;
```

（2）小车速度控制：使用PD（比例、微分）控制；

```
g_fSpeedControlOutNew = (CAR_SPEED_SET - g_fCarSpeed) *
g_fCarSpeed_P +\
   (CAR_POSITION_SET - g_fCarPosition) * g_fCarSpeed_I;
```

（3）小车方向控制：使用PD（比例、微分）控制。

```
speednow=-speedtarget*3.4 -gyro[2]*0.0015 ;
```

可通过单片机软件实现上述控制算法。

在上面控制过程中，车模的角度控制和方向控制都是直接将输出电压叠加后控制电机的转速实现的。而车模的速度控制本质上是通过调节车模的倾角实现的，由于车模是一个非最小相位系统，因此该反馈控制如果比例和速度过大，很容易形成正反馈，使得车模失控，造成系统的不稳定性。因此速度的调节过程需要非常缓慢和平滑。

六、PID 算法

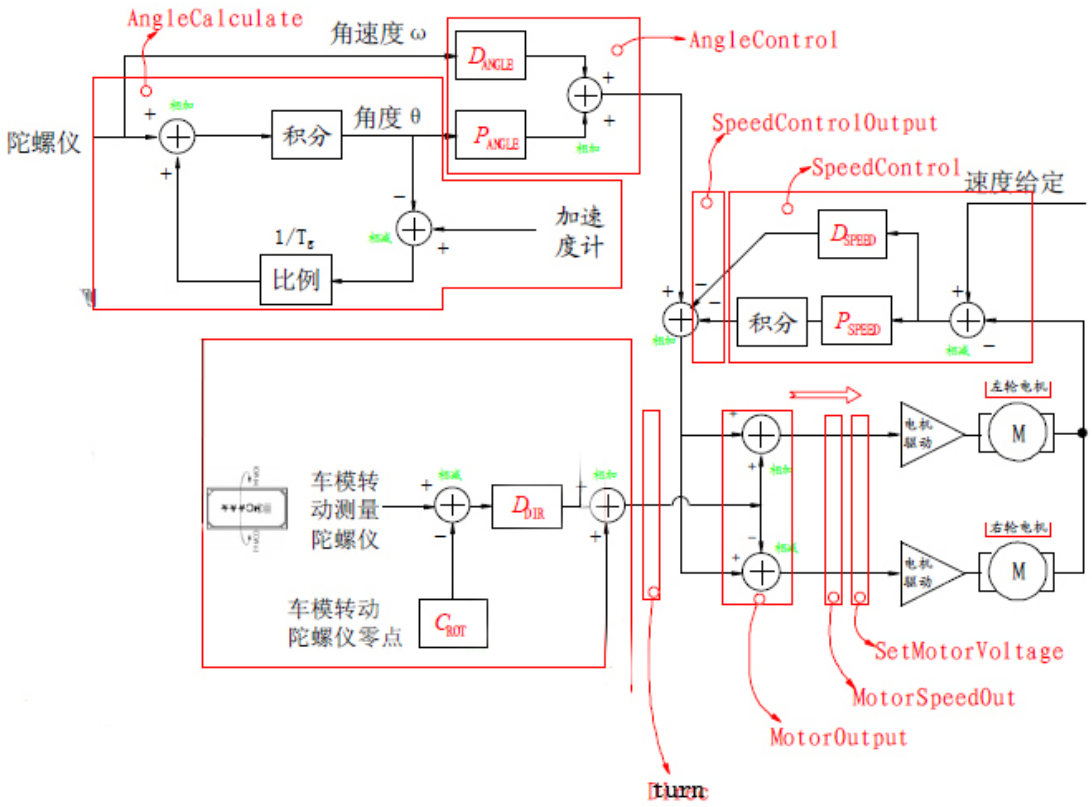


图 6

控制相关的软件函数包括：

1.

AngleCalculate：小车倾角计算函数。根据采集到的陀螺仪和重力加速度传感器的数值计算小车角度和角速度。如果这部分的算法由外部一个运放实现，那么采集得到的直接是小车的角度和角速度，这部分算法可以省略。该函数是每 5 毫秒调用一次。

2.

AngelControl：小车直立控制函数。根据小车角度和角速度计算小车电机的控制量。直立控制是 5 毫秒调用一次。

3.

SpeedControl：小车速度控制函数。根据小车采集到的电机转速和速度设定值，计算电机的控制量。该函数是 100 毫秒调用一次。

4.

SpeedControlOutput：速度输出平滑函数。由于速度是每 100 毫秒进行一次计算。为了使得速度控制更加平滑，该函数将速度输出变化量平均分配到 20 步 5 毫秒的控制周期中。

5.

DirectionControlOutput：方向控制函数输出平滑函数。将方向控制的输出变化量平均分配到 2 步 5 毫秒的控制周期中。

6.

MotorOutput：电机输出量汇集函数。根据前面的直立控制、速度控制和方向控制所得到的控制量进行叠加，分别得到左右两个电极的输出电压控制量。对叠加后的输出量进行饱和处理。函数调用周期 5 毫秒。在此请大家注意速度控制量叠加的极性是负。

7.

MotorSpeedOut：电机 PWM 输出计算函数。根据左右两个电极的输出控制量的正负极性，叠加上一个小的死区数值，克服车模机械静态摩擦力。函数调用周期 5 毫秒。

8.

SetMotorVoltage：PWM 输出函数：根据两个电机的输出量，计算出 PWM 控制寄存器的数值，设置四个 PWM 控制寄存器的数值。函数调用周期 1 毫秒。

以上 9 个函数都是在 1 毫秒中断服务中进行被相互调用的。下图显示了这些函数之间的调用与参数传递关系。在个函数附近也表明了调用的周期。

9.

Chaoshengbo： 加入超声波壁障模块：根据前方障碍物的距离检测，一旦检测到后，通过直接 PWM 值输出（ g_fchaoshengbooutput），相障碍物反方上运动，无需算法实现。每 30 毫秒调用一次。

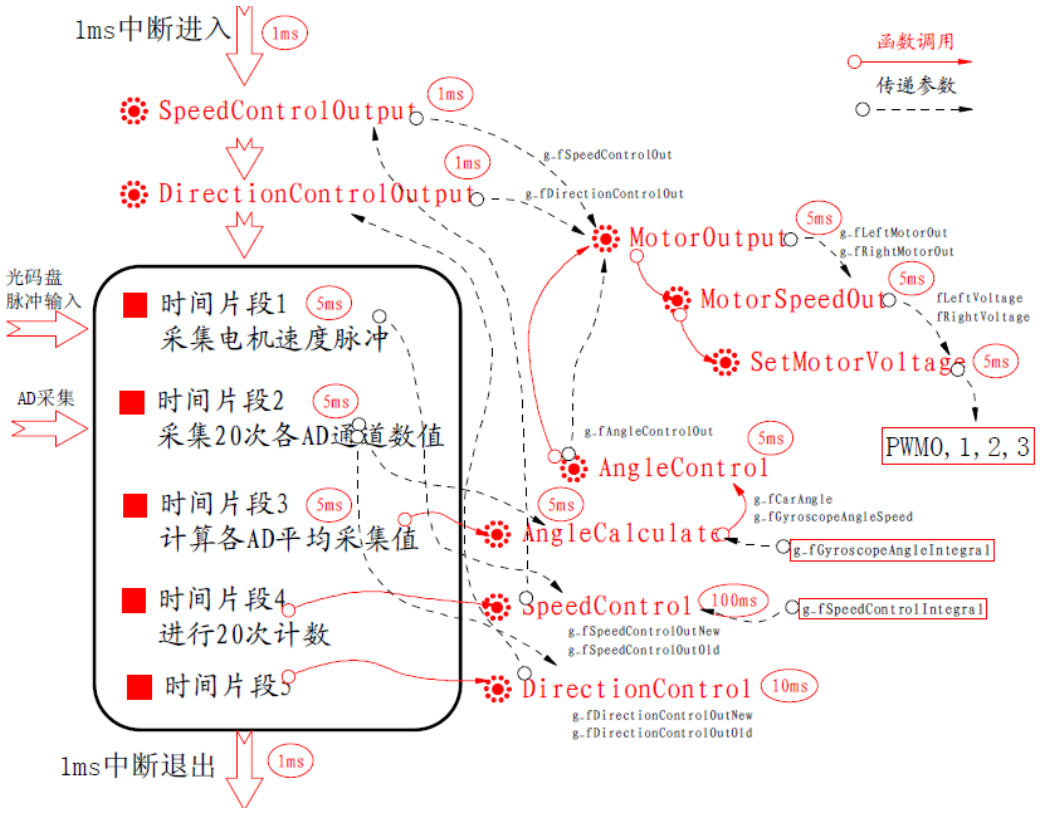


图 7

（注：以上函数框图为未添加旋转及超声波，添加方法与融合速度方式一样）

七、程序（只给出一部分内容）

（1） 时序总算法

```
void SysTick_Handler(void)                    //1ms 定时器
{

    BST_u8MainEventCount++;                   //总循环计数值

    BST_u8SpeedControlPeriod++;               //速度平滑输出计算比例值
    SpeedControlOutput();                     //速度 PWM 输出函数


    BST_u8DirectionControlPeriod++;           //转向平滑输出计算比例值
    DirectionControlOutput();                 //转向 PWM 输出函数

    BST_u8turnPeriod++;                       //旋转平滑输出计算比例值
    turnfliteroutput();                       //旋转 PWM 输出函数

    BST_u8trig++;                             //超声波循环计数值

    if(BST_u8MainEventCount>=5)               //若总计数值=5，即总循环时间为
5ms 时，调用脉冲计算函数
    {
        BST_u8MainEventCount=0;               //每 5ms，总循环体计数值清零
        GetMotorPulse();                      //脉冲计算函数

        BST_u8trig++;                         //超声波循环计数值

        if(BST_u8trig>=12)                    //6*5=30ms  每 30ms 做一次超声波
检测
      {


        UltrasonicWave_StartMeasure();        //调用超声波发送程序  给 Trig 脚  <10us  高电
平
        chaoshengbo();                        //计算超声波测距距离    juli
        BST_u8trig=0;                         //循环清零
```

```
        }

    }
    else if(BST_u8MainEventCount==1)          //若总计数值=1，即总循环时间为 5ms 时，
获取陀螺仪的角度状态
    {
        MPU6050_Pose();                       //获取 MPU6050 角度状态

    }
    else if(BST_u8MainEventCount==2)        //若总计数值=2，即总循环时间为 5ms 时，角
度 PD 控制 PWM 输出（每 5ms 调用一次，保持小车平衡），小车总 PWM 输出
    {

        AngleControl();                       //角度 PD 控制 PWNM 输出

        MotorOutput();                        //小车总 PWM 输出



    }
    else if(BST_u8MainEventCount==3)          //若总计数值=3，即总循环时间为 5ms 时，
车模车速融入判断
    {
        BST_u8SpeedControlCount++;              //小车速度控制调用计数值
        if(BST_u8SpeedControlCount>=10)         //当计数值为 10 时，即总系统运行 50ms
时候(每 10 个角度 PWM 输出中融入 1 个速度 PWM 输出，这样能保持速度 PID 输出不干扰
角度 PID 输出，从而影响小车平衡)
        {

          SpeedControl();                       //车模速度控制函数     每 50ms 调用一次
          BST_u8SpeedControlCount=0;          //小车速度控制调用计数值清零
          BST_u8SpeedControlPeriod=0;         //平滑输出比例值清零
        }
    }
    else if(BST_u8MainEventCount==4)            //若总计数值=3，即总循环时间为 5ms 时，
调用转向和旋转函数
    {
            BST_u8DirectionControlCount++;          //转向函数调用计数值
            BST_u8turnCount++;                      //旋转函数调用计数值

              if(BST_u8turnCount>=3)              //每 15ms 调用旋转函数
              {
                BST_u8turnCount=0;                //计数值清零
                BST_u8turnPeriod=0;               //平滑输出比例值清零
```

```
                turn();                                 //旋转函数
            }

            if(BST_u8DirectionControlCount>=5)//每 25ms 调用转向函数
            {
              BST_u8DirectionControlCount=0;     //计数值清零
              BST_u8DirectionControlPeriod=0;   //平滑输出比例值清零
              DirectionControl();                       //转向函数

            }
    }

}
```

（2） 平衡程序

```
/*************************************************************
**  函数名称: SpeedControl
**  功能描述: 速度环控制函数
*************************************************************/

void SpeedControl(void)
{

    BST_fCarSpeed = (BST_s32LeftMotorPulseSigma    + BST_s32RightMotorPulseSigma ) * 0.5 ;          //左右电机脉冲数平均值作为小车当前车速
    BST_s32LeftMotorPulseSigma =BST_s32RightMotorPulseSigma = 0;         //全局变量 注意及时清零

    BST_fCarSpeed = 0.7 * BST_fCarSpeedOld + 0.3 * BST_fCarSpeed ;            //速度一阶滤波
    BST_fCarSpeedOld = BST_fCarSpeed;

    BST_fCarSpeed *= CAR_SPEED_CONSTANT;  //单位：转/秒

    BST_fCarPosition += BST_fCarSpeed;                //路程   即速度积分

    BST_fCarPosition += BST_fBluetoothSpeed;     //融合蓝牙给定速度

    BST_fCarPosition +=     fchaoshengbo;             //融合超声波给定速度
    //积分上限设限//
    if((s32)BST_fCarPosition > CAR_POSITION_MAX)       BST_fCarPosition = CAR_POSITION_MAX;
    if((s32)BST_fCarPosition < CAR_POSITION_MIN)       BST_fCarPosition =
```

```
CAR_POSITION_MIN;

    BST_fSpeedControlOutOld = BST_fSpeedControlOutNew;

                        //速度 PI 算法  速度*P +位移*I=速度 PWM 输出
    BST_fSpeedControlOutNew = (CAR_SPEED_SET - BST_fCarSpeed) * BST_fCarSpeed_P
+\
        (CAR_POSITION_SET - BST_fCarPosition) * BST_fCarSpeed_I;

}
void SpeedControlOutput(void)
                    //速度平滑输出函数
{
  float fValue;
  fValue = BST_fSpeedControlOutNew - BST_fSpeedControlOutOld ;
  BST_fSpeedControlOut = fValue * (BST_u8SpeedControlPeriod + 1) /
SPEED_CONTROL_PERIOD + BST_fSpeedControlOutOld;   //转向平滑输出计算公式

}
```

（3） 蓝牙以及超声波

```
/**************************************************************
** 函数名称: BluetoothControl
** 功能描述: 蓝牙控制函数
            手机发送内容
            前：0x01     后：0x02
            左：0x04     右：0x03
            停止：0x07
            功能键：（旋转）
            左旋转:0x05       右旋转：0x06
            停转：0x07
** 输   入:
** 输   出:
**************************************************************/
void BluetoothControl(void)                                    //蓝牙控制
{
    u8 u8BluetoothValue;

    while(USART_GetFlagStatus(USART3,USART_FLAG_RXNE)!=SET);  //等待 USART3 的
接受区不为空
```

```
    //{
        u8BluetoothValue =USART3->DR;
        //ucBluetoothValue = USART_ReceiveData(USART3);
        USART_ClearFlag(USART3,USART_FLAG_RXNE);
        //USART1_Send_Byte(u8BluetoothValue);

    //}

    switch (u8BluetoothValue)
    {
      case 0x01 : BST_fBluetoothSpeed =     250 ; break;     //前进数字大时速度快  70
        300
      case 0x02 : BST_fBluetoothSpeed = (-250);   break;          //后退                -300
      case 0x04 : BST_fBluetoothDirectionSL = 1; break;//左转
      case 0x03 : BST_fBluetoothDirectionSR = 1; break;//右转
      case 0x05 : BST_fBluetoothDirectionL = 1; break ;//500    //左旋
      case 0x06 : BST_fBluetoothDirectionR = 1; break ;//-500       //右旋转
      case 0x07 : BST_fBluetoothDirectionL   =0; BST_fBluetoothDirectionR = 0;   break; //停
      case 0x08 : BST_fBluetoothDirectionSL =0; BST_fBluetoothDirectionSR = 0; break;
//停旋转
      case 0x09 :   BST_fBluetoothSpeed =     0 ;   break;
      default : BST_fBluetoothSpeed = 0;
BST_fBluetoothDirectionL=BST_fBluetoothDirectionR =
0;BST_fBluetoothDirectionSR=BST_fBluetoothDirectionSL=0;break;
    }

}


/**********************蓝牙 PWM 输出  转向*************************/
void DirectionControl(void)                                          //转向函数
{


static float temp ,turnconvert,speedtarget,temp2;
static   int turncount,speedlimit=100;

    BST_fBluetoothDirectionNew=BST_fBluetoothDirectionOld ;

   if(gyro[2]>32768) gyro[2]-=65536;                          //强制转数据类型转换  2G 转换
成 1G
    chose=0;

   if(BST_fBluetoothDirectionSL==1|BST_fBluetoothDirectionSR==1)     //蓝牙遥控是否有
```

```
指令判断
{
    chose=1;
     if(++turncount==1)
     {
     temp   = BST_s16LeftMotorPulse+ BST_s16RightMotorPulse;       //初始速度 在转向
前，根据初始速度，再给出输出速度。
     if(temp<0) temp2=-temp;                                          //当前小车运动方
向上的测速正负
     turnconvert=400/temp2;                                          //在实际调试过程中确
定。（转向时车的稳定性决定）
     }

     if(turnconvert<3)turnconvert=5;
     if(turnconvert>5)turnconvert=10;

}
  else
{

     turnconvert=1;
     BST_speedout=0;
     turncount=0;
     speedtarget=0;
     temp=0;
}

 if(BST_fBluetoothDirectionSL==1) speedtarget+=turnconvert;


 else if(BST_fBluetoothDirectionSR==1) speedtarget-=turnconvert;

        else speedtarget=0;

if(speedtarget>speedlimit) speedtarget=speedlimit;                    //转向速度幅值
限制
if(speedtarget<-speedlimit) speedtarget=-speedlimit;

     BST_fBluetoothDirectionNew=-speedtarget*19-gyro[2]*0.4;          //转向 PD 控
制。


}
```

```
void DirectionControlOutput(void)                                    //转向平滑输出
{
   float fValue;


   fValue = BST_fBluetoothDirectionNew - BST_fBluetoothDirectionOld;
   BST_fBluetoothDirectionOut = fValue *(BST_u8DirectionControlPeriod + 1) /25 + BST_fBluetoothDirectionOld; //平滑输出计算公式


}
/*****************************旋转操作************************************/



void turn(void)                                                      //  旋转函数
{
    static float temp ,turnconvert,speedtarget,temp1,temp2,temp3;
    int turncount,speedlimit=100;

    btcount=0;
    BST_speedold=BST_speednow;
    if(gyro[2]>32768) gyro[2]-=65536;

    if(BST_fBluetoothDirectionL==1|BST_fBluetoothDirectionR==1)
{
     btcount=1;
     if(++turncount==1)                                              //初始速度  在旋转前，根据初始速度，再给出输出速度。
     {
     temp    = BST_s16LeftMotorPulse+ BST_s16RightMotorPulse;
     temp1 = BST_s16LeftMotorPulse-BST_s16RightMotorPulse;
     if(temp<0) temp2=-temp;
     if(temp1<0) temp3=-temp1;

     turnconvert=temp3/temp2;                                        //在实际调试过程中确定。（转向时车的稳定性决定）
//     turnconvert=100/temp1;

     }

     if(turnconvert<1)turnconvert=1;
     if(turnconvert>5)turnconvert=5;
```

```
}
   else
{
      turnconvert=1;
      BST_speedout=0;
      turncount=0;
      speedtarget=0;
      temp=0;
}

 if(BST_fBluetoothDirectionL==1) speedtarget+=turnconvert;


 else if(BST_fBluetoothDirectionR==1) speedtarget-=turnconvert;

         else speedtarget=0;

if(speedtarget>speedlimit) speedtarget=speedlimit;
if(speedtarget<-speedlimit) speedtarget=-speedlimit;

      BST_speednow=-speedtarget*19-gyro[2]*0.4;                    //旋转 PD 控制算法


}



void turnfliteroutput(void)                                                  //旋转 PWM 值
平滑输出
{
float turnvalue;
turnvalue=BST_speednow-BST_speedold;
BST_speedout=turnvalue*(BST_u8turnPeriod + 1) / 15    +BST_speedold;          //旋转平
滑输出计算公式


}
```

```
/**********************超声波距离计算**************************/
void chaoshengbo(void)
{


        if(TIM_GetCounter(TIM1)>0)                    //如果检测到超声波模块被插入，或者超声波检测到距离后调用测距
        {
    juli=TIM_GetCounter(TIM1)*5*34/200.0;          //初始化后发送 8 个脉冲信号后，开启 TIM1 定时计算器，当有回波时候，停止计算器，距离 l=T *SPEED/2
    if(juli<=8.00)                                 //判断若距离小于 8cm，小车输出向后 PWM 值。
     {
       fchaoshengbo= (-100);
    }
    else fchaoshengbo=0;                           //距离大于 8cm ，超声波 PWM 输出为 0


    }


}
```