第二课 底盘控制器和驱动源码学习

1. 底盘控制器（STM32F405）

意法半导体（STMicroelectronics）集团于 1987 年 6 月成立，是世界最大的半导体公司之一。从成立之初至今，ST 的增长速度超过了半导体工业的整体增长速度。据最新的工业统计数据，意法半导体（STMicroelectronics）是全球第五大半导体厂商，在很多市场居世界领先水平。例如，意法半导体是世界第一大专用模拟芯片和电源转换芯片制造商，世界第一大工业半导体和机顶盒芯片供应商，而且在分立器件、手机相机模块和车用集成电路领域居世界前列。

STM32 系列专为要求高性能、低成本、低功耗的嵌入式应用设计的 ARM Cortex®-M0，M0+，M3，M4 和 M7 内核。STM32 型号的说明：以 STM32F103RBT6 这个型号的芯片为例，该型号的组成为 7 个部分，其命名规则如下：

1	STM32	STM32代表ARM Cortex-M内核的32位微控制器。
2	F	F代表芯片子系列。
3	103	103代表增强型系列。
4	R	R这一项代表引脚数，其中T代表36脚，C代表48脚，R代表64脚，V代表100脚，Z代表144脚，I代表176脚。
5	B	B这一项代表内嵌Flash容量，其中6代表32K字节Flash，8代表64K字节Flash，B代表128K字节Flash，C代表256K字节Flash，D代表384K字节Flash，E代表512K字节Flash，G代表1M字节Flash。
6	T	T这一项代表封装，其中H代表BGA封装，T代表LQFP封装，U代表VFQFPN封装。
7	6	6这一项代表工作温度范围，其中6代表-40——85℃，7代表-40——105℃。

图 1-1-1 STM32 命名规则图例

ST（意法半导体）推出了以基于 ARM® Cortex™-M4 为内核的 STM32F4 系列高性能微控制器，其采用了 90 纳米的 NVM 工艺和 ART（自适应实时存储器加速器），ART 技术使得程序零等待执行，提升了程序执行的效率，将 Cortext-M4 的性能发挥到了极致，使得 STM32 F4 系列可达到 210DMIPS@168MHz。自适应实时加速器能够完全释放 Cortex-M4 内核的性能；当 CPU 工作于所有允许的频率(≤168MHz)时，在闪存中运行的程序，可以达到相当于零等待周期的性能。STM32F4 系列微控制器集成了单周期 DSP 指令和 FPU（浮点单元），提升了计算能力，可以进行一些复杂的计算和控制。

2.驱动代码介绍

HiBot 采用 STM32F405RGT6 作为底盘控制器的 MCU。我们所使用到的资源有：系统滴答定时器、通用 I/O、通用定时器（TIM7）、通用串口（USART1）、定时器正交解码（encoder）、定时器 PWM、IIC 总线、ADC 电压采样。对于以上功能的具体运用如下图所示：

功能	应用
系统滴答定时器	Delay 延时和系统时钟
通用 GPIO	输入输出 LED
通用定时器	系统处理逻辑时间调度
通用串口	和树莓派通信
定时器正交解码	编码器速度采样
PWM 输出	电机的速度控制实现

IIC 总线	IMU 的原始数据读取
ADC 电压采样	系统电流和电池电压采样

主函数初始化参数列表：

```
NVIC_PriorityGroupConfig(NVIC_PriorityGroup_2); // The system interrupts the packet
Huanyu_delay_init(168);                         // The tick timer  is initialized with a clock frequency of 168M
BaseBoard_TIM7_Init(999,8400);                  // PID cycle control, timer interrupt initial, the overflow time is 100ms(999)
Huanyu_Usart1_Init(115200);                     // Serial port 1 initialization, baud rate is 115200, PC communication interface
Huanyu_LED_init();
Huanyu_BEEP_init();
RightMoto_Encoder_Input_init();                 // collter encoder data calculate moto speed ,
LeftMoto_Encoder_Input_init();
Huanyu_Charge_configure();
Huanyu_IIC_Init();                              // IIC bus configure use to Mpu6050 commnication
Huanyu_MPU_Init();                              // MPU6050 register configure function
HuanyuMoto_PWM_Init(1000, 10);                  // Motor PWM output initialization, the frequency is 10.42Khz, PWM duty is 1-1000
Huanyu_IWDG_Init(4, 200);                       // IWDG feed init function timer=100ms
```

2.1 系统滴答定时器

Huanyu_delay_init(u8 SYSCLK)函数主要功能是初始化系统时钟为 168MHZ。

```
/*
 @ describetion: delay init
 @ param: prer: u8 SYSCLK
 @ return: none
 @ author: Xuewei Zhou
 @ date : 2019-4-17
 @ note: .
 @ function: void Huanyu_delay_init(u8 SYSCLK)
*/
void Huanyu_delay_init(u8 SYSCLK)
{
#if SYSTEM_SUPPORT_OS
    u32 reload;
#endif
    SysTick_CLKSourceConfig(SysTick_CLKSource_HCLK_Div8);
    fac_us=SYSCLK/8;
#if SYSTEM_SUPPORT_OS
    reload=SYSCLK/8;
    reload*=1000000/delay_ostickspersec;

    fac_ms=1000/delay_ostickspersec;
    SysTick->CTRL|=SysTick_CTRL_TICKINT_Msk;
    SysTick->LOAD=reload;
    SysTick->CTRL|=SysTick_CTRL_ENABLE_Msk;
#else
    fac_ms=(u16)fac_us*1000;
#endif
}
```

2.2 通用 GPIO

Huanyu_LED_init(void)函数主要初始化系统状态指示灯、充电电量指示灯的 GPIO 的输出功能。

```
/*
 @ describetion: led output gpio initializes configuration
 @ param: none
 @ return: none
 @ author: Xuewei Zhou
 @ date : 2019-4-17
 @ note:
 @ function: void Huanyu_LED_init(void);
*/
void Huanyu_LED_init(void)
{
    GPIO_InitTypeDef  GPIO_InitStructure;
    RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOC | RCC_AHB1Periph_GPIOA , ENABLE);

    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0 | GPIO_Pin_1 | GPIO_Pin_2;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_OUT;
    GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_100MHz;
    GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_UP;
    GPIO_Init(GPIOC, &GPIO_InitStructure);

    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_8;
    GPIO_Init(GPIOA, &GPIO_InitStructure);

    LED1 = LED2 = LED3 = 1;
}
```

2.3 通用定时器

BaseBoard_TIM7_Init(u16 arr,u16 psc)函数主要初始化系统运行逻辑的调度时间基

数。为系统的各个模块运行提供时间戳。初始化时间最小单位是 100ms（编码器的采样周期、系统运行指示灯周期）。

```
/*
 @ describetion: The timer overflow interrupt initializes the configuration
 @ param: arr:Automatic reload value
 @        psc:Clock Prescale Number
 @ return: none
 @ author: Xuewei Zhou
 @ date : 2019-4-17
 @ note: Tout=((arr+1)*(psc+1))/Ft us.
 @ function: void BaseBoard_TIM7_Init(u16 arr,u16 psc)
*/
void BaseBoard_TIM7_Init(u16 arr,u16 psc)
{
    TIM_TimeBaseInitTypeDef TIM_TimeBaseInitStructure;
    NVIC_InitTypeDef NVIC_InitStructure;

    RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM7,ENABLE);

    TIM_TimeBaseInitStructure.TIM_Period = arr;
    TIM_TimeBaseInitStructure.TIM_Prescaler=psc;
    TIM_TimeBaseInitStructure.TIM_CounterMode=TIM_CounterMode_Up;
    TIM_TimeBaseInitStructure.TIM_ClockDivision=TIM_CKD_DIV1;

    TIM_TimeBaseInit(TIM7,&TIM_TimeBaseInitStructure);

    TIM_ITConfig(TIM7,TIM_IT_Update,ENABLE);
    TIM_Cmd(TIM7,ENABLE);

    NVIC_InitStructure.NVIC_IRQChannel=TIM7_IRQn;
    NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority=0x01;
    NVIC_InitStructure.NVIC_IRQChannelSubPriority=0x03;
    NVIC_InitStructure.NVIC_IRQChannelCmd=ENABLE;
    NVIC_Init(&NVIC_InitStructure);
}
```

在 100ms 周期内进入定时器中断，处理相应的逻辑。

```
/*
 @ describetion: TIM7 interrupt function
 @ param: none
 @ return: none
 @ author: Xuewei Zhou
 @ date : 2019-4-17
 @ note:
 @ function void TIM7_IRQHandler(void)
*/
unsigned int Safeware_Count = 0;
void TIM7_IRQHandler(void)
{
    if(TIM_GetITStatus(TIM7,TIM_IT_Update)==SET)
    {
        //program runing time ccount add
        (Safeware_Count == 42949672) ?(Safeware_Count=0) : (Safeware_Count++);  //时间基数常数

        if(Safeware_Count%10 == 0){Huanyu_BAT_Show(Source_Valtage);}    //电量计采样周期 1000ms

        RUN_LED = ~RUN_LED;     //LED 指示周期 100ms

        Robot_Encoder_Get_CNT();    //编码器采样周期
    }
    TIM_ClearITPendingBit(TIM7,TIM_IT_Update);
}
```

2.4 通用串口

Huanyu_Usart1_Init(u32 bound)函数主要初始化串口配置(收发使能、中断使能、串口波特率)，HiBot 主要采用 usart1 和树莓派进行通信，波特率为 115200。

```
/*
 @ describetion: USART1 initialization function
 @ param: bound->115200
 @ return: none
 @ author: Xuewei Zhou
 @ date : 2019-4-17
 @ note:
 @ function: void Huanyu_Usart1_Init(u32 bound)
*/
void Huanyu_Usart1_Init(u32 bound)
{
    USART_InitTypeDef USART_InitStructure;

    USART_InitStructure.USART_BaudRate = bound;
    USART_InitStructure.USART_WordLength = USART_WordLength_8b;
    USART_InitStructure.USART_StopBits = USART_StopBits_1;
    USART_InitStructure.USART_Parity = USART_Parity_No;
    USART_InitStructure.USART_HardwareFlowControl = USART_HardwareFlowControl_None;
    USART_InitStructure.USART_Mode = USART_Mode_Rx | USART_Mode_Tx;
    USART_Init(USART1, &USART_InitStructure);
```

串口通信是 slam 机器人比较核心的内容,主要是将基础的传感器数据(自身位姿、运动状态、电源管理数据、障碍物信息、充电桩信息等等）和 ROS 系统之间搭建桥梁,让自己的机器人能够在 ROS 系统中建立运动模型。
同时接收 ROS 系统的控制命令,通过底层的模型控制机器人产生相应的运动。具体是在串口接收中断中等待上位机的控制数据。

```
/*
 @ describetion: USART1 interrupt process function
 @ param: void
 @ return: none
 @ author: Xuewei Zhou
 @ date : 2019-4-17
 @ note:
 @ function: void USART1_IRQHandler(void)
*/
unsigned char Rcount = 0;
void USART1_IRQHandler(void)
{
    if(USART_GetITStatus(USART1, USART_IT_RXNE) != RESET)
    {
        Recive_Data.buffer[Rcount] = USART_ReceiveData(USART1);
        (Recive_Data.buffer[0] == 0xFe)?(Rcount++):(Rcount = 0);
        if (Rcount == PROTOCL_DATA_SIZE)     //验证数据包的长度
        {
            if(Recive_Data.Sensor_Str.Header == PROTOCOL_HEADER)     //验证数据包的头部校验信息
            {
                if(Recive_Data.Sensor_Str.End_flag == PROTOCOL_END) //验证数据包的尾部校验信息
                {
                    //接收上位机控制命令，使机器人产生相应的运动
                    Kinematics_Positive(Recive_Data.Sensor_Str.X_speed, Recive_Data.Sensor_Str.Z_speed);
                    LED1 = ~LED1;    //接受状态的指示灯
                }
            }
            Rcount = 0;
        }
    }
}
```

2.5 定时器正交解码

定时器的正交解码功能主要是读取编码器在特定周期内的脉冲数据,然后通过计算得到相应的位姿里程单位，HiBot 里程计单位是：在 X 方向上线性速度为 m/s，航向角用 rad/s 表示。在已知 X、YAW 的线性速度和角速度后，通过机器人运动学模型可以计算机器人的位置信息,后面课程也有详细的讲解。下面代码主要表示了如何用 STM32 的定时器正交解码区读取 Encoder 的脉冲数，同时机器人轮子的速度信息。

```
/*
 @ describetion:left moto encoder input TIM4 configure
 @ param: float Dacvalue
 @ return: none
 @ author: Xuewei Zhou
 @ date : 2019-3-14
 @ function : void RightMoto_Encoder_Input_init(void)
*/
void RightMoto_Encoder_Input_init(void)
{
    GPIO_InitTypeDef gpio;

    RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA,ENABLE);
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM3 ,ENABLE);

    gpio.GPIO_Pin   = GPIO_Pin_6 | GPIO_Pin_7;
    gpio.GPIO_Mode  = GPIO_Mode_AF;
    gpio.GPIO_OType = GPIO_OType_PP;
    gpio.GPIO_PuPd  = GPIO_PuPd_UP;
    gpio.GPIO_Speed = GPIO_Speed_100MHz;
    GPIO_Init(GPIOA,&gpio);

    GPIO_PinAFConfig(GPIOA, GPIO_PinSource6,  GPIO_AF_TIM3);
    GPIO_PinAFConfig(GPIOA, GPIO_PinSource7,  GPIO_AF_TIM3);

    TIM_EncoderInterfaceConfig(TIM3, TIM_EncoderMode_TI12, TIM_ICPolarity_Rising, TIM_ICPolarity_Rising);

    TIM_Cmd(TIM3, ENABLE);
}
```

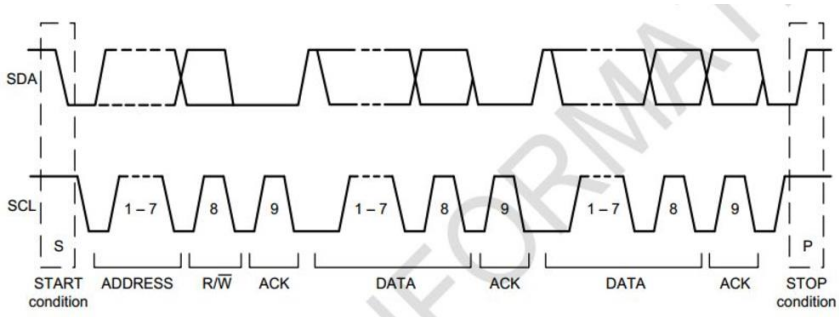
同时随着定时器的 10hz 的中断，在中断服务函数中读取和清除编码器的累积量，同时计算速度信息。函数 Robot_Encoder_Get_CNT()主要是用于以上逻辑处理。计算的公式也在程序中有相应的注释。

```
/*
 @ describetion: Get encoder value
 @ param:  NONE
 @ return: none
 @ author: Xuewei Zhou
 @ date : 2019-3-14
 @ function : void  Robot_Encoder_Get_CNT(void)
*/
void  Robot_Encoder_Get_CNT(void)
{
    Left_moto.Encoder_Value   = (TIM3->CNT)-0x7fff;     //读取左右轮子的脉冲累计数
    Right_moto.Encoder_Value  = -((TIM4->CNT)-0x7fff);

    //计算左右轮子的线性速度，速度 =（（轮子的直径 * 3.14 * （编码器脉冲数 / 轮子一圈积累的脉冲数））/ 采样周期）
    Left_moto.Current_Speed \
        = -((ROBOT_INITIATIVE_DIAMETER  *Pi_v * (Left_moto.Encoder_Value  / ENCODER_TTL_COUNT_VALUE))/CONTROL_TIMER_CYCLE);
    Right_moto.Current_Speed\
        = ((ROBOT_INITIATIVE_DIAMETER  *Pi_v * (Right_moto.Encoder_Value / ENCODER_TTL_COUNT_VALUE))/CONTROL_TIMER_CYCLE);

    TIM3->CNT = 0x7fff;     //清除左右轮子的脉冲数
    TIM4->CNT = 0x7fff;
}
```

2.6 I2C 通信协议介绍

I2C（Inter-Integrated Circuit）字面上的意思是集成电路之间，它其实是 I²C Bus 简称，所以中文应该叫集成电路总线，它是一种串行通信总线，使用多主从架构，由飞利浦公司在 1980 年代为了让主板、嵌入式系统或手机用以连接低速周边设备而发展。

I2C 串行总线一般有两根信号线，一根是双向的数据线 SDA，另一根是时钟线 SCL。所有接到 I2C 总线设备上的串行数据 SDA 都接到总线的 SDA 上，各设备的时钟线 SCL 接到总线的 SCL 上。设备上的串行数据线 SDA 接口电路应该是双向的，输出电路用于向总线上发送数据，输入电路用于接收总线上的数据。而串行时钟线也应是双向的，作为控制总线数据传送的主机，一方面要通过 SCL 输出电路发送时钟信号，另一方面还要检测总线上的 SCL 电平，以决定什么时候发送下一个时钟脉冲电平；作为接受主机命令的从机，要按总线上的 SCL 信号发出或接收 SDA 上的信号，也可以向 SCL 线发出低电平信号以延长总线时钟信号周期。以下图示表示了 I2C 的通信协议。同时通过代码模拟实现 I2C 的通信协议，在 HiBot 上我们主要用 I2C 总线去读取 IMU 的三轴加速度和三轴角速度数据。



Complete I²C Data Transfer

```
/*
 @ describetion: IIC start signal
 @ param: none
 @ return: none
 @ author: Xuewei Zhou
 @ date : 2019-4-17
 @ note:
 @ function void IIC_Start(void)
*/
void IIC_Start(void)
{
    SDA_OUT();
    IIC_SDA=1;
    IIC_SCL=1;
    delay_us(4);
    //START:when CLK is high,DATA change form high to low
    IIC_SDA=0;
    delay_us(4);
    IIC_SCL=0;
}
```

```
/*
 @ describetion: IIC stop signal
 @ param: none
 @ return: none
 @ author: Xuewei Zhou
 @ date : 2019-4-17
 @ note:
 @ function void IIC_Stop(void)
*/
void IIC_Stop(void)
{
    SDA_OUT();
    IIC_SCL=0;
    //STOP:when CLK is high DATA change form low to high
    IIC_SDA=0;
    delay_us(4);
    IIC_SCL=1;
    IIC_SDA=1;
    delay_us(4);
}
```

```
MPU_Get_Accelerometer(&Send_Data.Sensor_Str.Link_Accelerometer);    //通过IIC读取加速度信息
MPU_Get_Gyroscope(&Send_Data.Sensor_Str.Link_Gyroscope);    //通过IIC读取角速度信息
```

2.7 ADC 采样

ADC，Analog-to-Digital Converter 的缩写，指模/数转换器或者模数转换器 。是指将连续变化的模拟信号转换为离散的数字信号的器件。真实世界的模拟信号，例如温度、压力、声音或者图像等，需要转换成更容易储存、处理和发射的数字形式。模/数转换器可以实现这个功能，在各种不同的产品中都可以找到它的身影。

STM32F4 最多有 3 个 ADC，每个 ADC 有 12 位、10 位、8 位和 6 位可选，每个 ADC 有 16 个外部通道。另外还有两个内部 ADC 源和 VBAT 通道挂在 ADC1 上。ADC 具有独立模式、双重模式和三重模式，对于不同 AD 转换要求几乎都有合适的模式可选。ADC 功能非常强大，具体的我们在功能框图中分析每个部分的功能。

ADC 输入范围为：VREF- ≤ VIN ≤ VREF+。由 VREF-、VREF+ 、VDDA 、VSSA、这四个外部引脚决定。我们在设计原理图的时候一般把 VSSA 和 VREF-接地，把 VREF+和 VDDA 接 3V3，得到 ADC 的输入电压范围为：0~3.3V。如果我们想让输入的电压范围变宽，去到可以测试负电压或者更高的正电压，我们可以在外部加一个电压调理电路，把需要转换的电压抬升或者降压到 0~3.3V，这样 ADC 就可以测量了。

```
ADC_CommonInitStructure.ADC_Mode = ADC_Mode_Independent;
ADC_CommonInitStructure.ADC_TwoSamplingDelay = ADC_TwoSamplingDelay_5Cycles;
ADC_CommonInitStructure.ADC_DMAAccessMode = ADC_DMAAccessMode_Disabled;
ADC_CommonInitStructure.ADC_Prescaler = ADC_Prescaler_Div4;
ADC_CommonInit(&ADC_CommonInitStructure);

ADC_InitStructure.ADC_Resolution = ADC_Resolution_12b;
ADC_InitStructure.ADC_ScanConvMode = DISABLE;
ADC_InitStructure.ADC_ContinuousConvMode = DISABLE;
ADC_InitStructure.ADC_ExternalTrigConvEdge = ADC_ExternalTrigConvEdge_None;
ADC_InitStructure.ADC_DataAlign = ADC_DataAlign_Right;
ADC_InitStructure.ADC_NbrOfConversion = 1;
ADC_Init(ADC1, &ADC_InitStructure);


/*
 @ describetion: Get ADC value
 @ param: unsigned char ch
 @ return: none
 @ author: Xuewei Zhou
 @ date : 2019-4-14
 @ function : unsigned short Huanyu_Get_Adc(unsigned char ch)
*/
unsigned short Huanyu_Get_Adc(unsigned char ch)
{
    ADC_RegularChannelConfig(ADC1, ch, 1, ADC_SampleTime_480Cycles );
    ADC_SoftwareStartConv(ADC1);
    while(!ADC_GetFlagStatus(ADC1, ADC_FLAG_EOC ));
    return ADC_GetConversionValue(ADC1);
}
```

2.总结

通过本节课程的学习，我们了解了 STM32 系列 MCU 的基本用法，以及他的基本配置。同时在 HiBot 的驱动代码中，我们讲解了代码的基本组织结构和每个模块的初始化配置，HiBot 的驱动板携带了功能性 GPIO 的预留，我们也可以外接其他传感器和设备。