**课程内容**

1、小车在起点接收到启动信号后走直线2米，到达2米处自动停下来。  
  
2、小车放在起点处时，OLED显示“准备中”，接收到启动信号时OLED显示“Car start”，然后在小车走直线过程中，把小车当前的位移值和速度值显示在OLED上(10ms更新一次显示屏)。小车跑到终点2米处，OLED显示小车“到达终点” 同时显示位移、速度值。

测量编码器10ms测量一次 10ms更新一次PWM占空比，舵机

*注意：小车轮子直径50mm，车轴齿轮齿数76，电机轴齿轮齿数18，编码器轴齿轮齿数26。*

/\*-------------------------------------------------------------------------------

函数名称：Speed\_Measurement

函数描述：读取编码器信息，计算编码器速度r/s、电机转速r/s、车轮速度mm/s、车子位移mm

输入参数：无

返回参数：无

备 注：无

---------------------------------------------------------------------------------\*/

void Speed\_Measurement()

{

Get\_Encoder(); // GetEncoder.V3(脉冲边沿增量) GetEncoder.CNT3（总边沿个数）

R\_encoder\_speed = (float)GetEncoder.V3/ENCODER\_RESOLUTION \*100;

// 量纲 r/s，因为这里的时间单位捕获时间（10ms/次），所以10ms/次 \* 100 = 1s/次

// ENCODER\_RESOLUTION=2000

R\_motor\_speed = (float)R\_encoder\_speed\* ENCODER\_GEAR /WHEEL\_GEAR;

// 量纲 r/s，求出车轮每秒的圈数 WHEEL\_GEAR=76, ENCODER\_GEAR=26

V\_motor\_speed = (float)R\_motor\_speed \* (PI \* DIAMETER\_WHEEL);

// 量纲 mm/s，求出车轮每秒所移动的距离值DIAMETER\_WHEEL=50mm，PI=3.14

// Car\_S= (float)V\_motor\_speed\*10 //量纲 mm

Car\_Position= (float)GetEncoder.CNT3/ENCODER\_RESOLUTION\*WHEEL\_GEAR/ENCODER\_GEAR\*PI\* DIAMETER\_WHEEL;

// 量纲 mm，求出车实际位移

}

**10ms循环函数**

{

Step0:声明变量

Step1:获取速度信息

Step2:计算当前位移与目标位移差值

Step3:判断是否发送PWM脉冲还是停止脉冲发出（固定占空比，调试时先用小一点的速度）

// MotorSpeedSetOne(MotorSpeed);

Step4:刷新OLED

// oled\_show();

}

**定时器初始化函数**

/\*-------------------------------------------------------------------------------

函数名称：TIM3\_Configuration

函数描述：编码器接口设置（TIM3） /PA6-A相 PA7-B相

输入参数：无

返回参数：无

备 注：无

---------------------------------------------------------------------------------\*/

void TIM3\_Configuration(void)

{

GPIO\_InitTypeDef GPIO\_InitStructure;

TIM\_TimeBaseInitTypeDef TIM\_TimeBaseStructure;

TIM\_ICInitTypeDef TIM\_ICInitStructure;

RCC\_APB2PeriphClockCmd(RCC\_APB2Periph\_GPIOA, ENABLE); // GPIOA外设时钟

RCC\_APB1PeriphClockCmd(RCC\_APB1Periph\_TIM3, ENABLE); // TIM3时钟使能

RCC\_APB2PeriphClockCmd(RCC\_APB2Periph\_AFIO, ENABLE);

GPIO\_InitStructure.GPIO\_Pin = GPIO\_Pin\_6 | GPIO\_Pin\_7;

GPIO\_InitStructure.GPIO\_Mode = GPIO\_Mode\_IPU; // 设置为上拉输入模式

GPIO\_InitStructure.GPIO\_Speed = GPIO\_Speed\_50MHz; // IO口的速率为50M

GPIO\_Init(GPIOA, &GPIO\_InitStructure); // IO口配置函数

GPIO\_WriteBit(GPIOA, GPIO\_Pin\_6, Bit\_SET);

GPIO\_WriteBit(GPIOA, GPIO\_Pin\_7, Bit\_SET);

TIM\_TimeBaseStructure.TIM\_Period = ENCODER\_RESOLUTION - 1;

// 设置在下一个更新事件装入活动的自动重装载寄存器周期的值

TIM\_TimeBaseStructure.TIM\_Prescaler = 0;

// 设置用来作为TIMx时钟频率除数的预分频值 不分频

TIM\_TimeBaseStructure.TIM\_ClockDivision = TIM\_CKD\_DIV1;

// 设置时钟分频，这里采用不分频，即可以简单设置为0

TIM\_TimeBaseStructure.TIM\_CounterMode = TIM\_CounterMode\_Up;

// TIM向上计数模式

TIM\_TimeBaseInit(TIM3, &TIM\_TimeBaseStructure); // TIM3初始化

/\* 设置定时器3为编码器模式 IT1 IT2为上升沿计数 \*/

TIM\_EncoderInterfaceConfig(TIM3, TIM\_EncoderMode\_TI12, TIM\_ICPolarity\_BothEdge, TIM\_ICPolarity\_BothEdge); // 编码器计数模式设置为四倍频(A、B相双边沿)以提高精度

TIM\_ICStructInit(&TIM\_ICInitStructure);

TIM\_ICInitStructure.TIM\_ICFilter = 6; // 输入滤波器 aHR0cHM6Ly9ibG9nLmNzZG4ubmV0L2d0a2tuZC9hcnRpY2xlL2RldGFpbHMvNTIxODczNzA=

TIM\_ICInit(TIM3, &TIM\_ICInitStructure);

TIM\_ClearFlag(TIM3, TIM\_FLAG\_Update); // 清除所有标志位

TIM\_ITConfig(TIM3, TIM\_IT\_Update, ENABLE); // 允许中断更新

TIM\_SetCounter(TIM3, 0); // TIM3->CNT = 0;

TIM\_Cmd(TIM3, ENABLE);

}

**中断服务函数**

/\*\*

\* @brief TIM3（编码器）溢出中断处理函数

\* @param None

\* @retval None

\*/

void TIM3\_IRQHandler(void)

{

if(TIM\_GetITStatus(TIM3, TIM\_IT\_Update)) // 定时器产生更新中断说明编码器旋转了一圈

{

/\*如果为向下计数模式圈数减一，否则为向上计数模式圈数加一\*/

if (TIM3 -> CR1 & 0x0010) //反转

{

GetEncoder.rcnt3 -= 1;

}

else GetEncoder.rcnt3 += 1; //正转

TIM\_ClearITPendingBit(TIM3, TIM\_IT\_Update); // 清除中断和捕获标记位

}

TIM\_ClearITPendingBit(TIM3, TIM\_IT\_Update); // 清除中断和捕获标记位

}

**编码器采集结构体：**

typedef struct

{

\_\_IO s16 V3; // 当前编码器的脉冲增量（本次编码器总脉冲数 - 上一次编码器总脉冲数）

\_\_IO s16 cnt3; // 当前编码器的脉冲计数值

\_\_IO s16 rcnt3; // 当前编码器的旋转圈数

\_\_IO s32 CNT3; // 当前编码器的总脉冲数

}EncoderType;

**编码器脉冲计数函数：**

void Get\_Encoder(void) //计算当前编码器的总脉冲个数

{

s32 CNT3\_temp, CNT3\_last;

GetEncoder.cnt3 = TIM3 -> CNT; // 获取编码器计数读数

CNT3\_last = GetEncoder.CNT3;

CNT3\_temp = GetEncoder.rcnt3 \* prd + GetEncoder.cnt3; // 计算总脉冲数 prd=EncoderPeriod

GetEncoder.V3 = CNT3\_temp - CNT3\_last; // 计算脉冲增量

GetEncoder.CNT3 = CNT3\_temp; // 保存本次编码器的总脉冲数

}