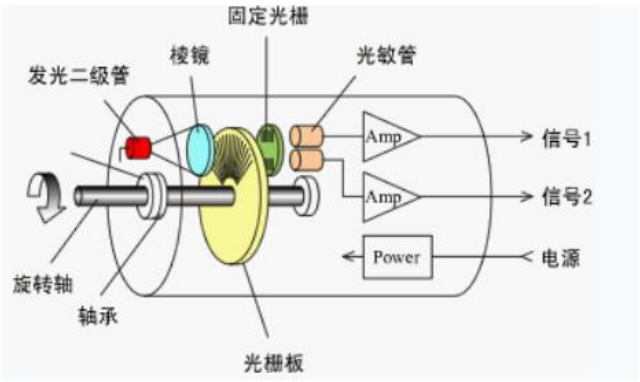
轻舟机器人驱动电机编码器数据读取

AI 航 团队

1. 编码器简介

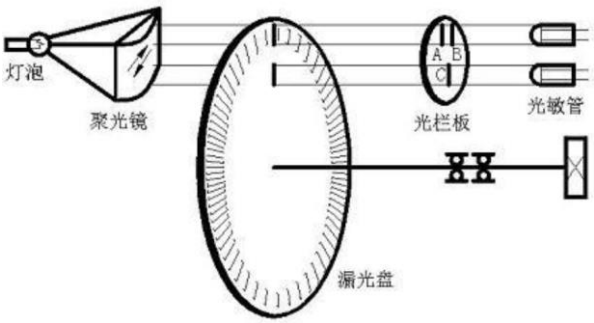
编码器在工程应用中十分广泛，是常用的电机定位设备，可以精确的测试电机的角位移和旋转位置，如下图所示，编码器将信号或数据进行编制、转换为可用以通讯、传输和存储的信号形式的设备。编码器把角位移或者直线位移转化成电信号，前者称为码盘，后者称为码尺。光电编码器是集光、机、电技术于一体的数字化传感器，可以高精度测量被测物体的转角或直线位移量。

编码器按照工作原理可以分为增量式和绝对式两类。增量式编码器是将位移转换成周期性的电信号，再把这个电信号转变成计数脉冲，用脉冲的个数表示位移的大小。绝对式编码器的第一个位置对应一个确定的数字码，因此它的示值只与测量的起始和终止位置有关，而与测量的中间过程无关。



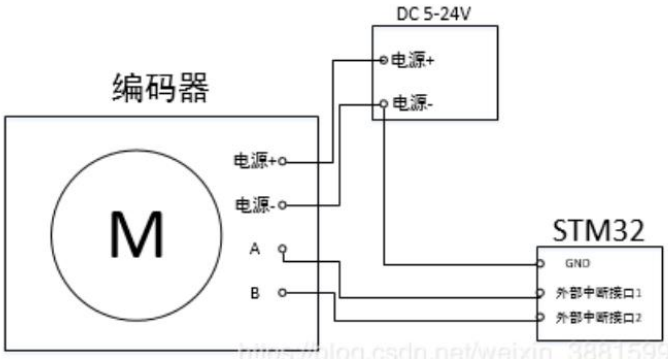
2. 增量式编码器

轻舟机器人驱动电机自带增量式编码器，所以接下来重点介绍增量式编码器的相关知识。增量式编码器是直接利用光电转换原理输出三组方波脉冲 A、B 和 Z 相；A、B 两组脉冲相位差 90，从而可方便的判断出旋转方向，而 Z 相为每转一个脉冲，用于基准点定位。它的优点是原理构造简单，机械平均寿命可在几万小时以上，抗干扰能力强，可靠性高，适合于长距离传输。其缺点是无法输出轴转动的绝对位置信息。



一个中心有轴的光电码盘，其上有环形通、暗的刻线，有光电发射和接收器件读取,获得 A 相、B 相和 Z 相（C 相），A 相与 B 相差 90 度相位差（相对于一个周波为 360 度）；另每转输出一个 Z 相脉冲以代表零位参考位。

由于 A、B 两相相差 90 度，可通过比较 A 相在前还是 B 相在前，以判别编码器的正转与反转，通过零位脉冲，可获得编码器的零位参考位。



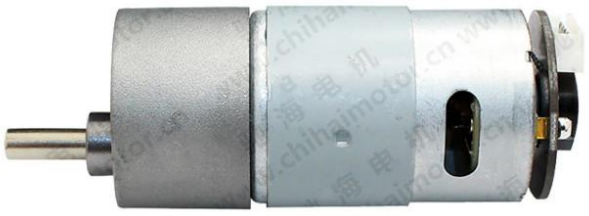
编码器码盘的材料有玻璃、金属、塑料，玻璃码盘是在玻璃上沉积很薄的刻线，其热稳定性好，精度高，金属码盘直接以通和不通刻线，不易碎，但由于金属有一定的厚度，精度就有限制，其热稳定性就要比玻璃的差一个数量级，塑料码盘是经济型的，其成本低，但精度、热稳定性、寿命均要差一些。

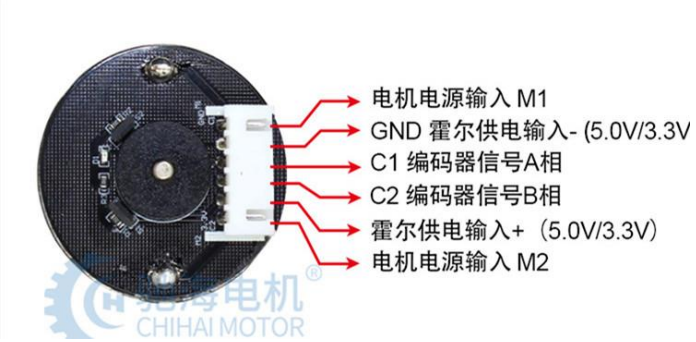
分辨率—编码器以每旋转 360 度提供多少的通或暗刻线称为分辨率，也称解析分度、或直接称多少线，一般在每转分度 5~10000 线。

信号输出有正弦波（电流或电压）,方波（TTL、HTL）,集电极开路（PNP、NPN）,推拉式多种形式，其中 TTL 为长线差分驱动（对称 A,A-;B,B-;Z,Z-）,HTL 也称推拉式、推挽式输出，编码器的信号接收设备接口应与编码器对应。

3. 轻舟机器人减速电机编码器接口

轻舟机器人后轮驱动电机如下所示，电机出厂已经集成了减速机和光电编码器，具体参数前边已经介绍过，在电机转动控制中，只需要连接电源输入 M1 和电源输入 M2 两个引脚即可。如果要采集编码器的信号，则需要连接其余四个信号。





【外径尺寸】∮37mm长度尺寸依据参数不同长度不同

【轴长】21mmD长12mm

【轴径】直径6mmD字型轴

【电压】6-24V

【重量】290g左右不同减速比不同参数重量不同

【接线规格】XH2.54-6PIN端子连接头

【编码器规格】AB双相编码器11线基本信号电压3.3V或5.0V

【基础脉冲数】绿色板**17 PPR** 黑色板 **11PPR** 已收到实物为准；

如果只是简单地测速可以只使用 A 相或 B 相，外加 3.3V 电源和地。编码器的测速方法有以下几种：

（1）使用定时器一路输入捕获功能，记录任意一相的上升沿或下降沿次数，或者上下边沿都计数。

（2）使用两路输入捕获功能，记录两相的上升沿或下降沿次数，或者上下边沿都计数。

（3）使用 STM32 定时器的编码器接口功能，记录任意一相上升沿或下降沿，或者两相的上下边沿都计数。

（4）使用 stm32 定时器输入异或功能，将 3 个通道的引脚异或，这样每个通道的电平变化都可以出发中断，进行测速。

编码器的信号检测有多重方法，其中最简单的就是使用 stm32 的定时器输入捕获功能，把编码器 A 相或 B 相引脚接入到 stm32 定时器通道引脚。在中断响应中不断的计数脉冲数，这种方法只能计数，不能判断方向，如果需要判断方向，则需要在计数的同时判断另一相位的电平，这就跟第 3 种方法一样了。第 4 种方法是用于霍尔传感器输入，如果可以灵活的运用，也可以作为增量式编码器的输入。Stm32 的高级定时器（TIM1 或 TIM8）和部分通用定时器有一种专门用于读取编码器信号的工作模式，允许同时接入 A 相和 B 相引线。如果同时使用 A 相和 B 相，那么不仅可以测出编码器的速度，还可以根据 A 相和 B 相之间的延迟，判断方向。这也是比较好的方法，相比较之下，使用编码器接口功能可以减少中断次数，测量结果可以直接读取计数器值，实际上，编码器接口功能也是属于输入捕获，不过这个是由 STM32 的硬件处理响应，直接使用计数器技术编码器脉冲，减少程序的开支。而 STM32 的编码器接口功能是在任意一相脉冲的上升沿或下降沿判断另一相的电平，从而判断方向，以计数器递增计数或递减计数代表编码器的旋转方向。

4. Stm32 程序实现

```
//函数功能：把 TIM2 初始化为编码器接口模式，对应左电机编码器
void Encoder_Init_TIM2(void)
{
```

```
    TIM_TimeBaseInitTypeDef TIM_TimeBaseStructure;
    TIM_ICInitTypeDef TIM_ICInitStructure;
    GPIO_InitTypeDef GPIO_InitStructure;
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_AFIO,ENABLE);      // 需要使能 AFIO 时钟
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM2, ENABLE);     //使能定时器 2 的时钟
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA, ENABLE);    //使能 PA 端口时钟
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB, ENABLE);    //使能 PB 端口时钟
    GPIO_PinRemapConfig(GPIO_PartialRemap1_TIM2, ENABLE);    //引脚重映射

    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_15;                //端口配置
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING;     //浮空输入
    GPIO_Init(GPIOA, &GPIO_InitStructure);                    //根据设定参数初始化 GPIOA
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_3;                 //端口配置
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING;     //浮空输入
    GPIO_Init(GPIOB, &GPIO_InitStructure);                    //根据设定参数初始化 GPIOB
    //定时器初始化
    TIM_TimeBaseStructInit(&TIM_TimeBaseStructure);
    TIM_TimeBaseStructure.TIM_Prescaler = 0x0;                // 预分频器
    TIM_TimeBaseStructure.TIM_Period = ENCODER_TIM_PERIOD; //设定计数器自动重装值
    TIM_TimeBaseStructure.TIM_ClockDivision = TIM_CKD_DIV1;   //选择时钟分频：不分频
    TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up; //TIM 向上计数
    TIM_TimeBaseInit(TIM2, &TIM_TimeBaseStructure);

    TIM_EncoderInterfaceConfig(TIM2,TIM_EncoderMode_TI12,TIM_ICPolarity_Rising,TIM_ICPolarity_Rising); //使
     用编码器模式 3
    TIM_ICStructInit(&TIM_ICInitStructure);
    TIM_ICInitStructure.TIM_ICFilter = 10;
    TIM_ICInit(TIM2, &TIM_ICInitStructure);
    TIM_ClearFlag(TIM2, TIM_FLAG_Update);    //清除 TIM 的更新标志位
    TIM_ITConfig(TIM2, TIM_IT_Update, ENABLE);
    //Reset counter
    TIM_SetCounter(TIM2,0);
    TIM_Cmd(TIM2, ENABLE);                    //使能 TIM
}

//函数功能：把 TIM3 初始化为编码器接口模式，对应右电机编码器
void Encoder_Init_TIM3(void)
{
    TIM_TimeBaseInitTypeDef TIM_TimeBaseStructure;
    TIM_ICInitTypeDef TIM_ICInitStructure;
    GPIO_InitTypeDef GPIO_InitStructure;
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM3, ENABLE);     //使能定时器 3 的时钟
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA, ENABLE);    //使能 PB 端口时钟

    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_6|GPIO_Pin_7;      //端口配置
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING;     //浮空输入
    GPIO_Init(GPIOA, &GPIO_InitStructure);                    //根据设定参数初始化 GPIOA

    TIM_TimeBaseStructInit(&TIM_TimeBaseStructure);
    TIM_TimeBaseStructure.TIM_Prescaler = 0x0;                // 预分频器
    TIM_TimeBaseStructure.TIM_Period = ENCODER_TIM_PERIOD;    //设定计数器自动重装值
    TIM_TimeBaseStructure.TIM_ClockDivision = TIM_CKD_DIV1;   //选择时钟分频：不分频
    TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up; //TIM 向上计数
    TIM_TimeBaseInit(TIM3, &TIM_TimeBaseStructure);

    TIM_EncoderInterfaceConfig(TIM3, TIM_EncoderMode_TI12, TIM_ICPolarity_Rising, TIM_ICPolarity_Rising);//
使用编码器模式 3
    TIM_ICStructInit(&TIM_ICInitStructure);
    TIM_ICInitStructure.TIM_ICFilter = 10;
    TIM_ICInit(TIM3, &TIM_ICInitStructure);
    TIM_ClearFlag(TIM3, TIM_FLAG_Update);                     //清除 TIM 的更新标志位
    TIM_ITConfig(TIM3, TIM_IT_Update, ENABLE);
    //Reset counter
```

```
    TIM_SetCounter(TIM3,0);
    TIM_Cmd(TIM3, ENABLE);    //使能 TIM
}

//函数功能：单位时间读取编码器计数，入口参数：定时器，返回值：速度值
int Read_Encoder(u8 TIMX)
{
      int Encoder_TIM;
     switch(TIMX)
        {
           case 2:    Encoder_TIM= (short)TIM2 -> CNT;    TIM2 -> CNT=0;break;
           case 3:    Encoder_TIM= (short)TIM3 -> CNT;    TIM3 -> CNT=0;break;
           default:    Encoder_TIM=0;
        }
        return Encoder_TIM;
}

//TIM3 中断服务函数
void TIM3_IRQHandler(void)
{
       if(TIM3->SR&0X0001)    //溢出中断
       {
       }
       TIM3->SR&=~(1<<0);      //清除中断标志位
}

//TIM2 中断服务函数
void TIM2_IRQHandler(void)
{
       if(TIM2->SR&0X0001)    //溢出中断
       {
       }
       TIM2->SR&=~(1<<0);    //清除中断标志位
}
```