第十三课 系统工程的编程思维及技巧

学习了前几课有关 Arduino 对组件的基础控制后，在本课中利用 ROBOHERO 作为系统控制例子，讲解控制系统编写时的技巧和应有的工程思维。

在第 9 到 12 课，学习有关 Arduino 的基础原理和应用，包括指示灯和--舵机的控制、通过 wifi 来达到讯号传输、及组合多组件的控制系统建构。第 12 课中通过机器人配套的手机应用程序，学习对机械人的模块化编辑控制，了解基本的运作原理。

在第 13 课中，会带领学习者进入机器人的底层模块，也就是利用代码编辑，通过修正一些参数值来控制舵机、指示灯等组件，并且了解到一个系统工程编写代码时所需要注意的技巧和思维。

在多个舵机组合一起，生成一个系统时，事情就变得更为复杂。以下以 RoboHero 机器人的编码为例，讲述一个运动机械系统中，需要加入的安全机制、监控机制等。以 RoboHero 机器人为例，每当一条动作指令 A 发出，机器人接收后会把指令缓存并执行，在收到下一条指令 B 时，会先把指令 B 和指令 A 作出比对，求出前后指令各舵机的输出相差值，这一过程确保指令执行时，每个舵机的输出值都是相对于统一个参考值。以下详细介绍本例编码。

编码说明：

以下为 RoboHero 的部分代码

```
/*============================================================================*/
void Servo_PROGRAM_Zero()  清零指令
{
  //  清除备份目前马达数值
  for ( int Index = 0; Index < ALLMATRIX; Index++)
```

```
  {
     Running_Servo_POS[Index] = Servo_Act_0[Index] + (int8_t)EEPROM.read(Index);
  }


  // 重载马达预设数值
       for (int iServo = 0; iServo < ALLSERVOS; iServo++)
       {
          Set_PWM_to_Servo(iServo, Running_Servo_POS[iServo]);
          delay(10);
       }
}
```

以上代码为一段清零指令，目的是为了在使用过程中把 RoboHero 机器人的动作复位。这一类复位工能常见于测量工具上如：电子卡尺、天平、计算器等。在一个工具中，清零是十分重要，因为清零功能能让机器或仪器回到原设计状态，去除使用历史留下的影响，保正机器或仪器的可靠性。

```
/*================================================================================*/
void Servo_PROGRAM_Center()  指令执行结束后清零
{
  // 清除备份目前马达数值
  for ( int Index = 0; Index < ALLMATRIX; Index++)
  {
     Running_Servo_POS[Index] = Servo_Act_1[Index] + (int8_t)EEPROM.read(Index);
  }


  // 重载马达预设数值
  for (int iServo = 0; iServo < ALLSERVOS; iServo++)
  {
```

```
    Set_PWM_to_Servo(iServo, Running_Servo_POS[iServo]);
    delay(10);
  }
}
```

这一段代码同样是清零动作，却和上一段清零指令不一样，注意指令开始时是对 Servo_PROGRAM_Center() 来作声明函数。在原代码中，Servo_PROGRAM_Center()被定义为待机，所以这一段清零目的是为了给 RoboHero 机器人在完成每项指令后清零，来保证待机时接收每套动作指令都可以准确执行。

```
void Servo_PROGRAM_Run(int iMatrix[][ALLMATRIX],   int iSteps)
{
  int INT_TEMP_A, INT_TEMP_B, INT_TEMP_C;
  for ( int MainLoopIndex = 0; MainLoopIndex < iSteps; MainLoopIndex++)   // iSteps 步骤主循环
  {
    // InterTotalTime 此步骤总时间
    int InterTotalTime = iMatrix [ MainLoopIndex ] [ ALLMATRIX - 1 ] +
(int8_t)EEPROM.read(ALLMATRIX - 1);
    // InterDelayCounter 此步骤基本延迟次数
    int InterDelayCounter = InterTotalTime / BASEDELAYTIME;
    // 内差次数循环
    for ( int InterStepLoop = 0; InterStepLoop < InterDelayCounter; InterStepLoop++)
    {
      for (int ServoIndex = 0; ServoIndex < ALLSERVOS; ServoIndex++)   // 马达主循环
      {
        INT_TEMP_A=Running_Servo_POS[ServoIndex];   // 马达现在位置 INT_TEMP_A
        INT_TEMP_B=iMatrix[MainLoopIndex][ServoIndex]+(int8_t)EEPROM.read(ServoIndex);
```

```
        // 马达目标位置 INT_TEMP_B


        if (INT_TEMP_A == INT_TEMP_B)              // 马达数值不变
        {
          INT_TEMP_C = INT_TEMP_B;
        }


        else if (INT_TEMP_A > INT_TEMP_B)     // 马达数值减少
        {
          // PWM 内差值 = map( 执行次数时间累加，开始时间，结束时间，内差起始值，
内差最大值 )
          INT_TEMP_C =    map(BASEDELAYTIME * InterStepLoop, 0, InterTotalTime, 0,
INT_TEMP_A - INT_TEMP_B);


          if (INT_TEMP_A - INT_TEMP_C >= INT_TEMP_B)
          {
            Set_PWM_to_Servo(ServoIndex, INT_TEMP_A - INT_TEMP_C);
          }
        }
        else if (INT_TEMP_A < INT_TEMP_B)     // 马达数值增加
        {
          // PWM 内差值 = map( 执行次数时间累加, 开始时间, 结束时间, 内差起始值, 内
差最大值 )
          INT_TEMP_C =    map(BASEDELAYTIME * InterStepLoop, 0, InterTotalTime, 0,
INT_TEMP_B - INT_TEMP_A);


          if (INT_TEMP_A + INT_TEMP_C <= INT_TEMP_B)
```

```
            {
                Set_PWM_to_Servo(ServoIndex, INT_TEMP_A + INT_TEMP_C);
            }
          }
        }
        delay(BASEDELAYTIME);
    }
    // 备份目前马达数值
    for ( int Index = 0; Index < ALLMATRIX; Index++)
    {
        Running_Servo_POS[Index] = iMatrix[MainLoopIndex][Index] +
(int8_t)EEPROM.read(Index);
    }
  }
}
```

总结：

在学习完 Arduino 的一系列课程后，再加上本章节课程。基本上就能掌握小型系统工程的设计流程和思维。再经过长时间的练习和后续学习，保持工程思维，就可以培养出基础的工程习惯，为成长成工程师建立重要的一步，且养成良好的生活及行事风格。