Hiwonder | 深圳市幻尔科技有限公司
Shenzhen Hiwonder Technology Co., Ltd.

# 幻尔科技

# STM32 版本开发教程

## V1.0

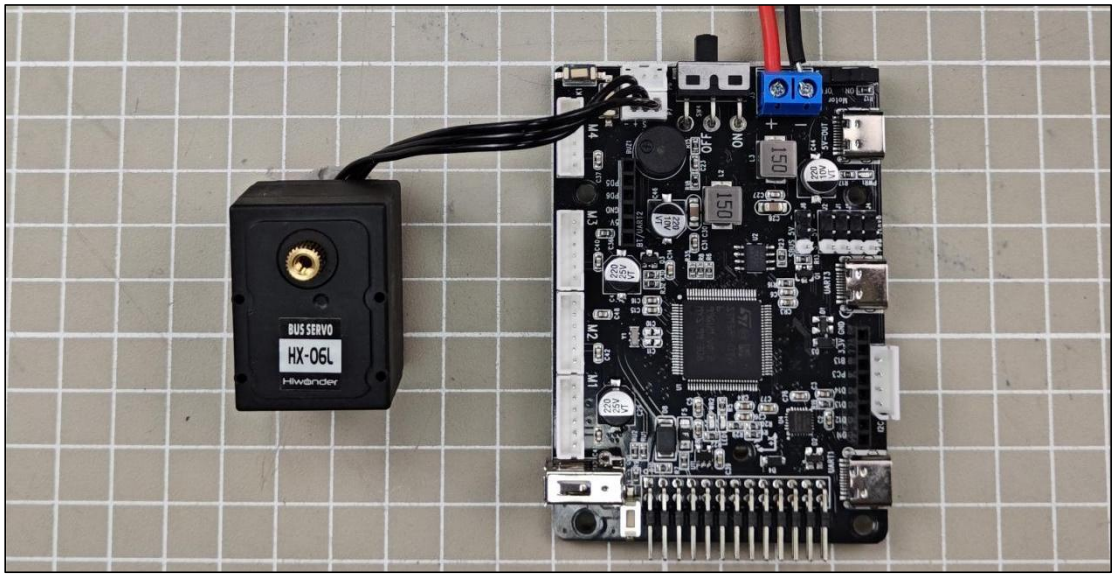

**Hiwonder 官方网站**

| 版本号 | 修改日期 | 修改摘要 |
| --- | --- | --- |
| V1.0 | 20230923 | 初次发布 |

# 1.准备工作

## 1.1 接线说明

本节示例使用的是基于 STM32F407 的 RRC 控制器和总线舵机进行开发，通过 7.4V 6000mAh 锂电池来供电。将总线舵机连接至 RRC 控制器的任意一个总线接口。




**注意：**

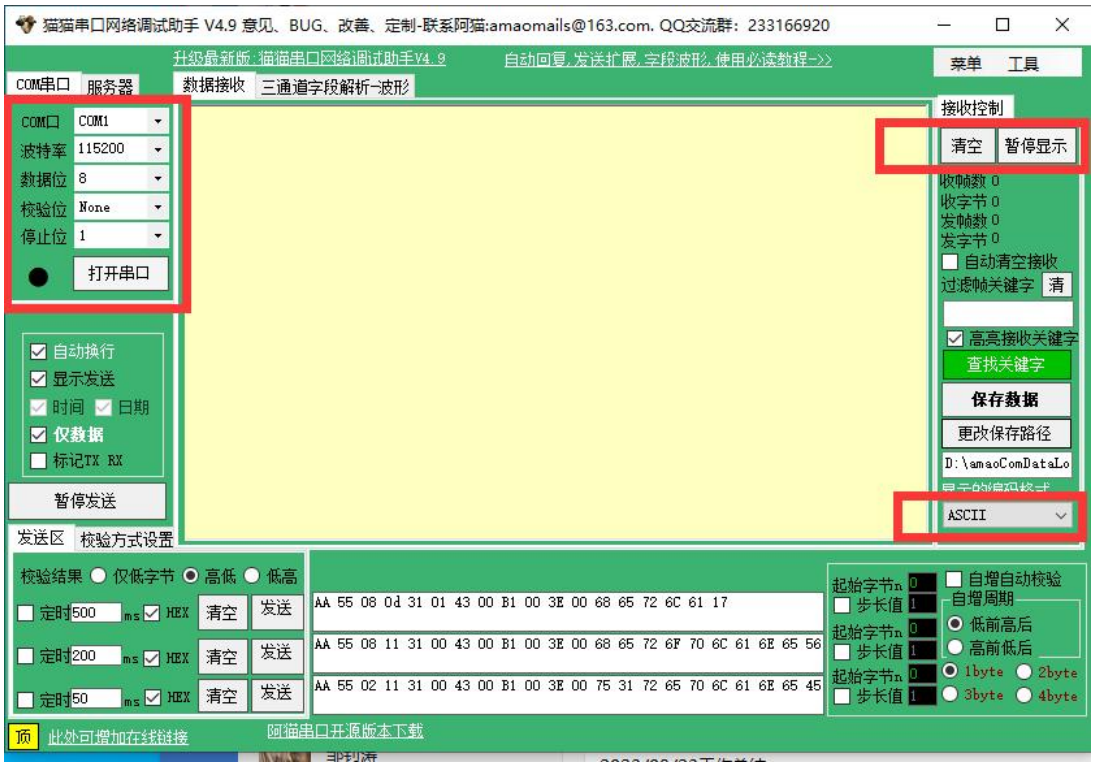
**① 如使用我司锂电池，连接锂电池对接线请以红接＋，黑接-接到 DC 接口。**

**② 如对接线未连接锂电池，请勿与电池对接线直接对接，避免正负极发生短路从而造成短路。**

## 1.2 环境配置

在电脑端安装 Keil5，软件包位于“05 软件工具”下。关于 Keil5 的详细使用，可在对应目录下进行学习。

串口助手也位于“05 软件工具”下，配置如下：




串口 1 一般为通信接口，用户要根据实际的串口选择。

## 1.3 程序下载 USB 下载

编译好工程之后，就可以将生成好的 hex 文件下载到 stm32 主控板上面，需要准备以下软硬件材料。

### 1.3.1 软硬件准备

**软件：**mcuisp（在“**05** 软件工具”中带有）


**硬件：**Type-C 数据线，stm32 主控板

### 1.3.2 下载步骤

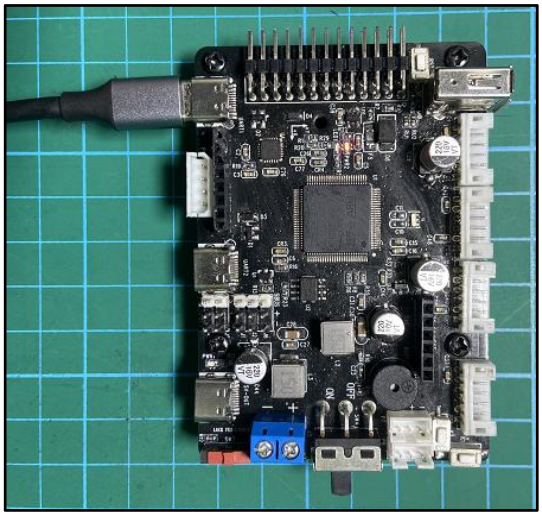
1、Type-C 数据线转 USB 连接线用于连接电脑和 stm32 主控板（后面简称 Type-C 线）。

2、stm32 控制板

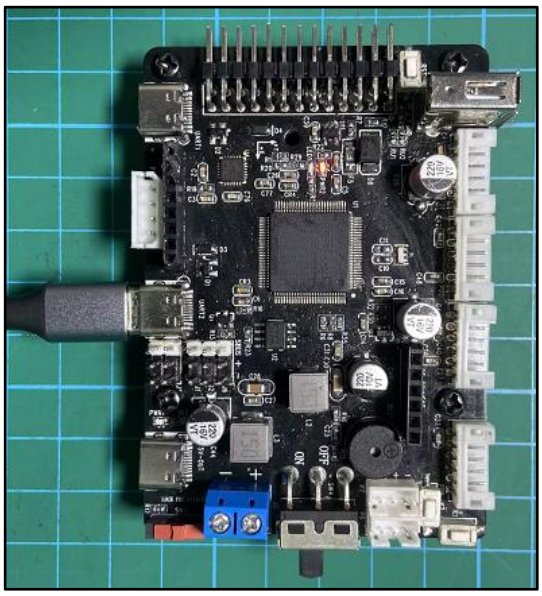
具体的操作步骤如下所示：

（1）硬件连接：

使用 Type-C 线插入 stm32 主控板的 type-c 口(UART1)，并且 与电脑的 USB 口连接：



[UART1]

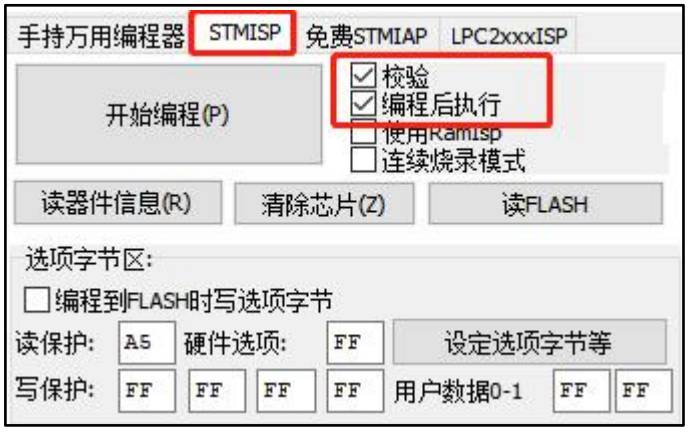


[UART2]

（2）基础设置：

打开 mcuisp 软件，在软件上方的菜单栏里面点击“搜索串口”，然后设置波特率(bps)为 115200：


在软件界面中点击“STMISP”选项




在底部选择“DTR 低电平复位，RTS 高电平进 BootLoader”：

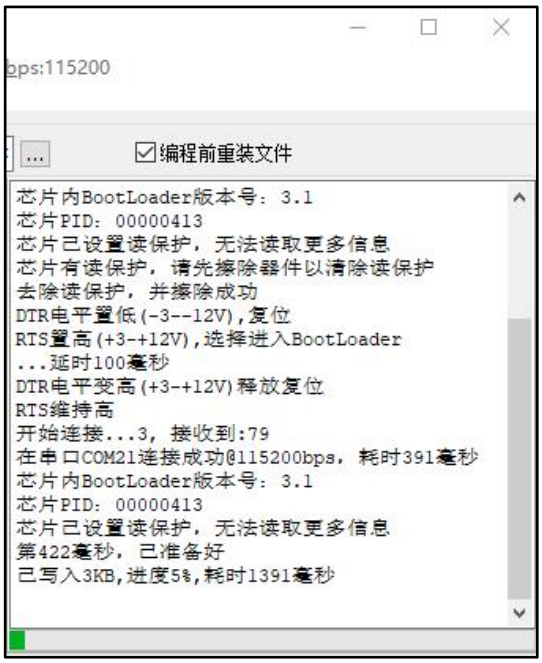

（3）软件烧录：

在 mcuisp 软件界面点击下图红框内的按钮，选择需要烧写的 hex 文件。

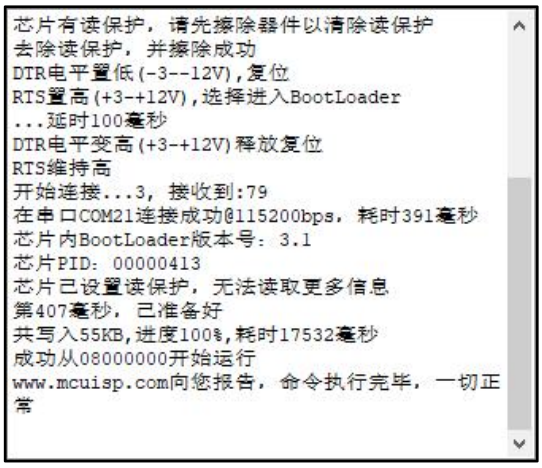


回到上一级界面点击“开始编程”按钮，将生成好的 hex 文件烧录到 stm32 主控板上：


烧录正在进行：




若侧边栏出现以下图说明烧录完成。




**注意：为了避免烧录过程会出现的异常现象，请用户自行按照步骤顺序进行操作！！！**

# 2.案例开发

## 2.1 案例 1 总线舵机信息读取

本案例通过终端窗口显示出总线舵机 ID、位置、温度等相关信息。

```
Start...
read id fail, again
□read id fail, again
□read id fail, again
□read id fail, again
□servo id is 1
servo deviation is -125
read angle_limit fail, again
servo angle_limit is 0 ~ 1000
read vin_limit fail, again
servo vin_limit is 4500 ~ 14000
servo vin is 7977
servo temp_limit is 85
servo temp is 49
servo load_unload is 1
servo position is 521
************************************
read servo id is 1
servo deviation is -125
servo angle_limit is 0 ~ 1000
servo vin_limit is 4500 ~ 14000
servo vin is 7977
servo temp_limit is 85
servo temp is 49
servo load_unload is 1
servo position is 521
************************************
```

### 2.1.1 运行程序

“**STM32 案例 1 总线舵机信息读取\RosRobotControllerM4_serial_servo_read**”路径下打开该工程并下载 hex 文件到 RRC 控制器。

### 2.1.2 实现效果

程序运行后，打开“**05 软件工具**”下的 

串口助手，打印画面会滚屏打印舵机各项状态信息。

```
**************************************
read servo id is 1
servo deviation is -125
servo angle_limit is 0 ~ 1000
servo vin_limit is 4500 ~ 14000
servo vin is 7977
servo temp_limit is 85
servo temp is 49
servo load_unload is 1
servo position is 521
**************************************
```

id:舵机 ID。在这里示例为 1。

deviation：舵机偏差。在这里示例为-125。

angle_limit：舵机限位范围。在这里示例为 0-1000。

vin_limit：舵机电压范围。在这里示例为 4.5~14V。

vin：舵机当前电压值。在这里示例为 7.977V。

temp_limit：舵机过温报警阈值。在这里示例为 85℃。

temperature：舵机当前温度。在这里示例为 49℃。

Load_unload：舵机是否为上电状态。在这里示例为 1，代表为上电状态。当为 0，即掉电状态。

position：舵机当前所在的位置。在这里示例为 521。

### 2.1.3 案例程序简要分析

- **导入必要头文件**

```
#include "serial_servo.h"
```

该头文件主要封装用于总线舵机通信的各功能模块，我们可以使用其中定义的变量和函数来控制舵机。

- **串口初始化操作**

```
/* 串口舵机控制初始化 */
serial_servo_init();
```

- **获取舵机状态信息并打印**

```
int serial_servo_read_id(SerialServoControllerTypeDef *self, uint32_t servo_id, uint8_t *ret_servo_id);
int serial_servo_read_position(SerialServoControllerTypeDef *self, uint32_t servo_id, int16_t *position);
int serial_servo_read_deviation(SerialServoControllerTypeDef *self, uint32_t servo_id, int8_t *deviation);
int serial_servo_read_angle_limit(SerialServoControllerTypeDef *self, uint32_t servo_id, uint16_t limit[2]);
int serial_servo_read_temp_limit(SerialServoControllerTypeDef *self, uint32_t servo_id, uint8_t *limit);
int serial_servo_read_temp(SerialServoControllerTypeDef *self, uint32_t servo_id, uint8_t *temp);
int serial_servo_read_vin_limit(SerialServoControllerTypeDef *self, uint32_t servo_id, uint16_t limit[2]);
int serial_servo_read_vin(SerialServoControllerTypeDef *self, uint32_t servo_id, uint16_t *vin);
int serial_servo_read_load_unload(SerialServoControllerTypeDef *self, uint32_t servo_id, uint8_t* load_unload);
```

通过调用上述函数，获取舵机的各种状态信息。这些状态信息包括舵机 ID、位置、偏差、角度范围、电压范围、过温报警阈值、当前温度、电压以及舵机上电状态。

返回值为 0 则为读取成功，否则则为读取失败。

```
printf("**************************************\r\n");
printf("read servo id is %d\r\n",ret_servo_id);
printf("servo deviation is %d\r\n",deviation);
printf("servo angle_limit is %d ~ %d\r\n",angle_limit[0],angle_limit[1]);
printf("servo vin_limit is %d ~ %d\r\n",vin_limit[0],vin_limit[1]);
printf("servo vin is %d\r\n",vin);
printf("servo temp_limit is %d\r\n",temp_limit);
printf("servo temp is %d\r\n",temp);
printf("servo load_unload is %d\r\n",load_unload);
printf("servo position is %d\r\n",position);
printf("**************************************\r\n");
```

在获取后，可将上面的这些参数打印出来。

## 2.2 案例 2 总线舵机 ID 设置

本案例修改舵机 ID，并通过终端窗口显示出总线舵机新的 ID。

```
Start...
read old id fail, again
□read old id fail, again
□read old id fail, again
□servo old id is 1
set servo new id is 5
read new id fail, again
□read new id fail, again
□read new id fail, again
□read new id fail, again
□servo new id is 5
set servo id is success
```

### 2.2.1 运行程序

在“**03 STM32\案例 1 总线舵机信息读取\RosRobotControllerM4_serial_servo_change_id**”路径下打开该工程并下载 hex 文件到 RRC 控制器。

### 2.2.2 实现效果

程序运行后，打开“05 软件工具”下的 

串口助手，打印画面会滚屏打印舵机各项状态信息。

```
Start...
read old id fail, again
□read old id fail, again
□read old id fail, again
□servo old id is 1
set servo new id is 5
read new id fail, again
□read new id fail, again
□read new id fail, again
□read new id fail, again
□servo new id is 5
set servo id is success
```

各项信息的具体含义如下：

old id ：修改前的舵机 ID。在这里示例为 1。

new id ：修改后的舵机 ID。在这里示例为 5。

### 2.2.3 案例程序简要分析

- **导入必要头文件**

```
#include "serial_servo.h"
```

该头文件主要封装用于总线舵机通信的各功能模块，我们可以使用其中定义的变量和函数来控制舵机。

- **串口初始化操作**

```
/* 串口舵机控制初始化 */
serial_servo_init();
```

- **读取连接的总线舵机的 ID**

```
//读取舵机ID，只能连接一个舵机，否则会有总线冲突
ret = serial_servo_read_id(&serial_servo_controller , broadcast_id , &ret_servo_id);
```

此处 broadcast_id 变量的值为 254（0xFE），在总线舵机通信协议中表示为广播 ID，可用于对未知 ID 的舵机进行读取 ID 值。

- **修改舵机 ID 信息**

```
//设置舵机ID
serial_servo_set_id(&serial_servo_controller , ret_servo_id , servo_id_new);
```

通过调用 serial_servo_set_id()函数，将连接的舵机 ID 值修改为 5。

在获取后，将新的 ID 打印出来。

```
printf("servo new id is %d\r\n",ret_servo_id);
```

通过读取舵机 ID，将刚刚修改的舵机 ID 打印出来。

```
//判断是否设置成功
if(ret_servo_id == servo_id_new)
{
    printf("set servo id is success\r\n");
    //若设置成功，则可以用新的ID号控制舵机运动
    serial_servo_set_position(&serial_servo_controller , servo_id_new , 750 , 1000);
    osDelay(1000); //等待运动完成
}else{
    printf("set servo id is fail\r\n");
}
```

判断设置 ID 是否成功，成功则用新的 ID 号控制舵机运动。

## 2.3 案例 3 控制总线舵机转动

本案例使用 RRC 控制器控制舵机运动到位置 1000，用时 1s；再运动到位置 0，用时 1s；运动结束。

### 2.3.1 运行程序

在“**03 STM32\案例 1 总线舵机信息读取\RosRobotControllerM4_serial_servo_run**”路径下打开该工程并下载 hex 文件到 RRC 控制器。

### 2.3.2 实现效果

程序运行后，舵机会从运动到位置 1000，用时 1s；再运动到位置 0，用时 1s；运动结束。

### 2.3.3 案例程序简要分析

- **导入必要头文件**

```
#include "serial_servo.h"
```

该头文件主要封装用于总线舵机通信的各功能模块，我们可以使用其中定义的变量和函数来控制舵机。

- **串口初始化操作**

```
/* 串口舵机控制初始化 */
serial_servo_init();
```

- **控制舵机转动**

```
// 设置位置范围值为 0~1000 ，具体请看《总线舵机通讯协议》
//将舵机转到位置 1000，并运动时间为2000毫秒
serial_servo_set_position(&serial_servo_controller , servo_id, 1000, 2000);

osDelay(1000); //延时1000ms

//将舵机转到位置0，并运动时间为2000毫秒
serial_servo_set_position(&serial_servo_controller , servo_id, 0, 2000);

osDelay(1500);//延时1500ms
```

通过调用 serial_servo_set_position()函数，控制舵机转动。上述过程主要实现：舵机会从运动到位置 1000，用时 1s；等待 1.5s，再运动到位置 0，用时 1s；等待 1.5s。

舵机转动范围为：0-1000，对应角度为：0°-240°。

## 2.4 案例 4 调节总线舵机速度

本案例通过 RRC 控制器控制舵机以不同的速度进行转动。

### 2.4.1 运行程序

在“**03 STM32**

**\案例 1 总线舵机信息读取\RosRobotControllerM4_serial_servo_speed**”路径下打开该工程并下载 hex 文件到 RRC 控制器。

```
servo new id is 5
servo run duration 4000 ms
over
servo run duration 3000 ms
over
servo run duration 2000 ms
over
```

上传成功后，将总线舵机接到 RRC 控制上，舵机开始以不同的速度来回转动三遍。

### 2.1.2 实现效果

程序运行后舵机现象如下：

舵机从位置 1000 开始

以 4 秒的时间转动到位置 0

以 4 秒的时间转动到位置 1000

等待 5 秒

以 3 秒的时间转动到位置 0

以 3 秒的时间转动到位置 1000

等待 5 秒

以 2 秒的时间转动到位置 0

以 2 秒的时间转动到位置 1000

等待 5 秒

上述现象只执行一遍。

### 2.1.3 案例程序简要分析

- **导入必要头文件**

```
#include "serial_servo.h"
```

该头文件主要封装用于总线舵机通信的各功能模块，我们可以使用其中定义的变量和函数来控制舵机。

- **串口初始化操作**

```
/* 串口舵机控制初始化 */
serial_servo_init();
```

- **以不同的速度控制舵机转动**

```
/* 通过控制舵机的运动时间，来控制舵机的运动速度 */
for( run_time = 4000 ; run_time >= 2000 ; run_time -= 1000)
{
    printf("servo run duration %d ms\r\n",run_time);
    //控制舵机运动到 0 位置
    serial_servo_set_position(&serial_servo_controller , ret_servo_id , 0 , run_time);
    osDelay(run_time); //等待运动完成

    //控制舵机运动到 1000 位置
    serial_servo_set_position(&serial_servo_controller , ret_servo_id , 1000 , run_time);
    osDelay(5000); //等待运动完成
    printf("over\r\n");
}
```

通过调用 serial_servo_set_position()函数，控制舵机转动。

舵机转动范围为：0-1000，对应角度为：0°-240°。

最后一个参数为转动到指定位置所需时间，参数值越小速度越快。

## 2.5 案例 5 示教记录操作

本案例使用 RRC 控制器与按键控制舵机，通过存储位置，让舵机转动到指定角度。

### 2.1.1 运行程序

在“**03 STM32**

**案例 1 总线舵机信息读取\RosRobotControllerM4_serial_servo_teach**”路径下

打开该工程并下载 hex 文件到 RRC 控制器。

### 2.1.2 实现效果

程序运行后，第一次按下按键 2，舵机掉电，此时掰动舵盘至任意角度，再按下按键 2 记录当前位置（最多可记录 4 个位置，如果超过 4 个将会从第一个位置开始覆盖）。当记录好位置后，按下按键 1，此时舵机上电，舵机会按照记录好的角度从第一个开始依次进行转动。

### 2.1.3 案例程序简要分析

- **导入必要头文件**

```
#include "serial_servo.h"
```

该头文件主要封装用于总线舵机通信的各功能模块，我们可以使用其中定义的变量和函数来控制舵机。

- **串口初始化操作**

```
/* 串口舵机控制初始化 */
serial_servo_init();
```

在 for 循环里进行模式的选择与运行

```
for(;;)
{
    switch(mode)
    {
        case 0:
            mode_run();
            break;
        case 1:
            mode_read();
            break;
        default:
            mode = 0;
            break;
    }
}
```

按下按键 1 时，mode 为 0，舵机根据位置点运动；按下按键 2 时，mode 为 1，根据按键 2 读取运动位置点。

- **控制舵机转动运动到不同的位置**

```
static void mode_run(void)
{
    static uint32_t run = 0;
    static uint32_t posi_old = 0;
    static uint32_t duration = 1000;

    if(position[run] > posi_old)
    {
        duration = position[run] - posi_old;
    }else{
        duration = posi_old - position[run];
    }
    posi_old = position[run];
    printf("duration:%d\r\n",duration);
    serial_servo_set_position(&serial_servo_controller , servo_id , position[run] , duration*3);
    osDelay(duration*3 + 500);
    printf("Reach %d position\r\n",position[run]);

    run++;
    if(run >= 4)
    {
        run = 0;
        osDelay(5000);
    }
}
```

当按键 1 按下时，mode=0，进入运行模式，舵机根据不同位置点进行运动。

position[]用来存储 4 个位置，方便在控制舵机转动的时候设置舵机位置；

通过 run 变量来判断运行哪一个位置。

- **读取舵机的位置并存储**

当按键 2 第一次按下时，mode=1，更改当前模式为输入模式，并使舵机卸载电压。

掰动舵机角度，再次按下按键 2 时，会读取舵机的角度并存储在 position[]里。

最多可记录 4 个位置，如果超过 4 个将会从第一个位置开始覆盖。

```
static void mode_read(void)
{
    static uint32_t run = 0;
    static uint32_t posi_old = 0;

    serial_servo_load_unload(&serial_servo_controller , servo_id , 0);

    if( 1 == key2_ispress)
    {
        key2_ispress = 0;
        //读取舵机位置
        serial_servo_read_position(&serial_servo_controller , servo_id , &position[run]);
        printf("position[%d]:%d\r\n",run ,position[run]);
        run++;
        if(run >= 4)
        {
            run = 0;
        }
    }
    osDelay(200);
}
```

按键 1 按下后，将舵机上电，并控制舵机根据 position[]里存储的位置进行运动。