目录

1. 实验背景简介

能源短缺问题是目前许多国家面临的最重要的问题，而太阳能作为一种清洁无污染的能源，有着巨大的开发前景。我国的太阳能资源比较丰富，从其分布来看，西部地区的太阳能年福射总量均在 5400MJ/(m.a)以上，西藏地区更是达到了 6700MJ/(m.a)，开发太阳能对于西部的发展有着重要的现实意义。利用太阳能的关键是提高太阳能电池的转换效率，目前一般情况下仍是采用太阳能电池板固定朝南安装的方式对太阳能进行采集，也有利用太阳运动规律采用定时的方法对太阳进行跟踪。但这些方法效果都不是特别理想，通过对光线进行感应从而依据光强实时调整太阳能电池板的方法才能使太阳能的利用率显著提升。

本次实验基于 K66 开发板为核心板进行二次扩展开发的 Cyber-Dorm 实验板，进行编程实践，实现对实验板下层板上的光敏三极管的信号进行检测，根据信号控制舵机运动实现对光线的追踪。Cyber-Dorm 实验板照片如图 1-1 所示。

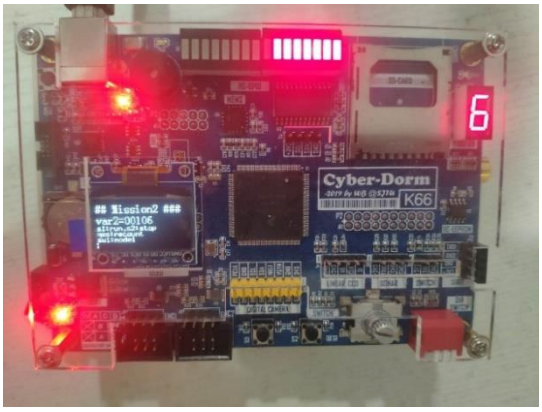


图 1-1 实验 Cyber-Dorm 开发板

2. 实验内容

本次实验需要完成的任务是使用实验板下层板的光敏三极管的信号进行采集并判断实验板当前所受光照的来源，进而根据计算的方向控制舵机运动，指示光线的方向并动态显示在16位光柱上。

实验具体内容如下：使用ADC采集光敏三极管信号，并对信号进行分析，找出信号特性，在光线适宜条件下识别光源相对于实验板的方位角，并控制舵机指向光源方向，实现“追光”效果，并将“方向角”动态的显示在16位光柱上

实验任务
配置接口，采集光敏三极管信号并显示
设定算法计算光源方向角
控制舵机指向光源方位
将方向角显示在 16 位光棒上

表格 2-1 实验任务表

3. 实验原理

3.1 舵机控制原理

R/C Servo 作为此次使用的舵机，基本工作原理是位置伺服系统。伺服电机接收到 1 个脉冲，就会旋转 1 个脉冲对应的角度，从而实现位移，因为，伺服电机本身具备发出脉冲的功能，所以伺服电机每旋转一个角度，都会发出对应数量的脉冲，这样，和伺服电机接受的脉冲形成了呼应，或者叫闭环，如此一来，系统就会知道发了多少脉冲给伺服电机，同时又收了多少脉冲回来，这样，就能够很精确的控制电机的转动，从而实现精确的定位。

舵机控制需要使用脉宽信号，信号范围为 0.52ms~2.52ms ，中位角对应 1.52ms。信号电平为 3.3~5.0V 兼容。

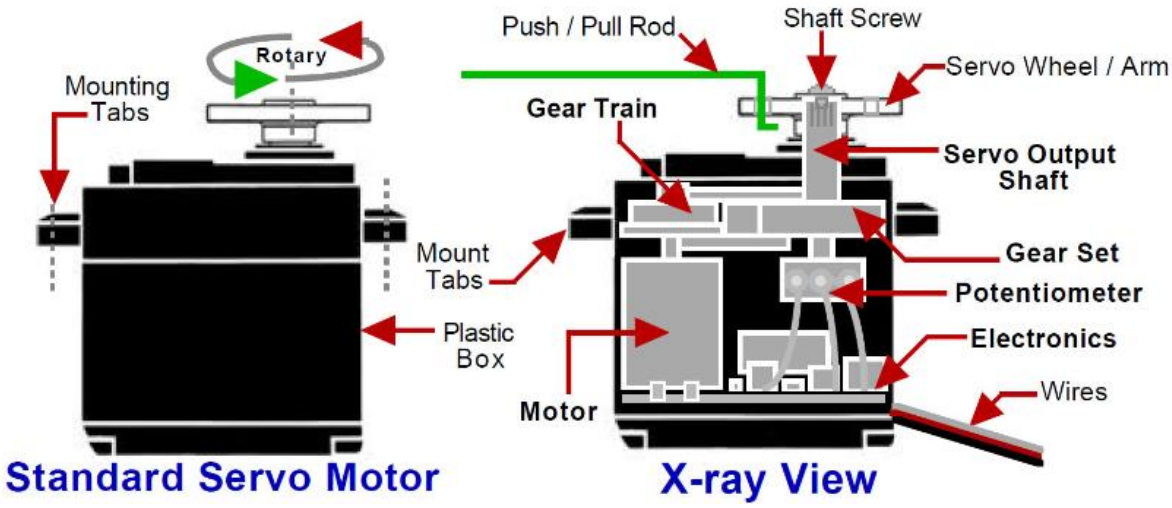


图 3-1 R/C Servo 结构图

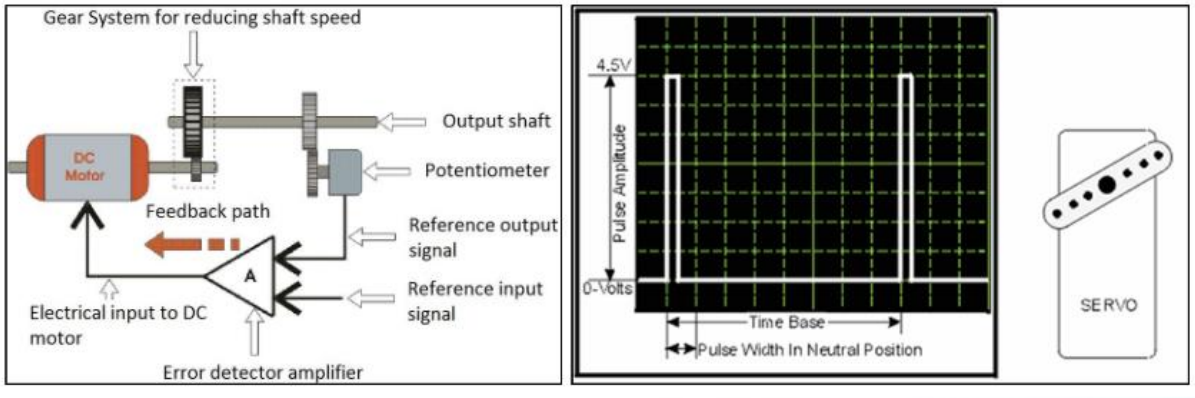


图 3-2 标准舵机控制方法

3.2 PID 控制原理

PID 控制器包括比例环节 P、微分环节 I、积分环节 D。PID 控制流程如图 3-3 所示

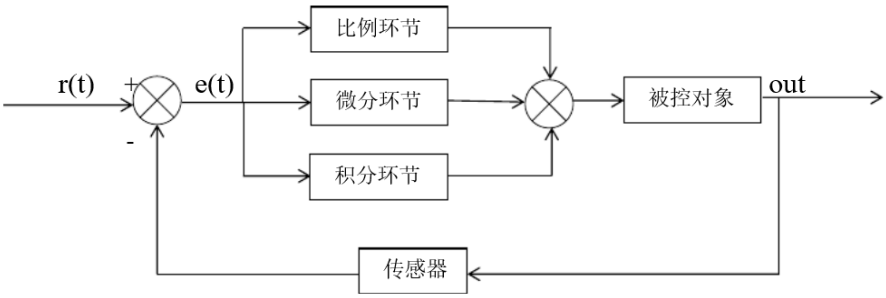


图 3-3 PID 控制流程图

PID 控制的基本方法是根据系统输入与预定输出的偏差的大小运用比例、积分、微分计算出一个控制量，将这个控制量输入系统，获得输出量，通过反馈回路再次检测该输出量的偏差，循环上述过程，以使输出达到预定值。

在 PID 算法中，比例环节 P 的作用是成比例地反映控制系统的偏差信号 e(t)，一旦产生偏差，比例控制环节立即产生控制作用以减小偏差。积分环节 I 的作用是消除静差，提高系统的无差度。微分环节 D 的作用是反映偏差信号的变化趋势，能够在偏差信号变化之前先引入一个有效的早期修正信号来提前修正偏差，加快系统的动作速度，减少调节时间。

在进行本次实验的舵机控制时，我们希望控制舵机转动的角度达到预期角度，因此采用位置式 PID 进行调节，使得舵机能够快速到达预定角度。

3.3 光敏三极管测量原理

光敏三极管是一种晶体管，它的电流受外部光照控制，是一种半导体光电器件。光敏三极管是一种相当于在三极管的基极和集电极之间接入一只光敏二极管的三极管，光敏二极管的电流相当于二极管的基极电流。因为具有电流放大作用，光敏三极管比光敏二极管灵敏得多，在集电极可以输出很大的光电流。

本实验采用的光敏三极管的型号为 LTR-546AD，对不同的光照产生不同的输出。该光敏三极管的工作条件如图 3-4 所示。通常在波长为 900nm、光强适当的光源下工作。

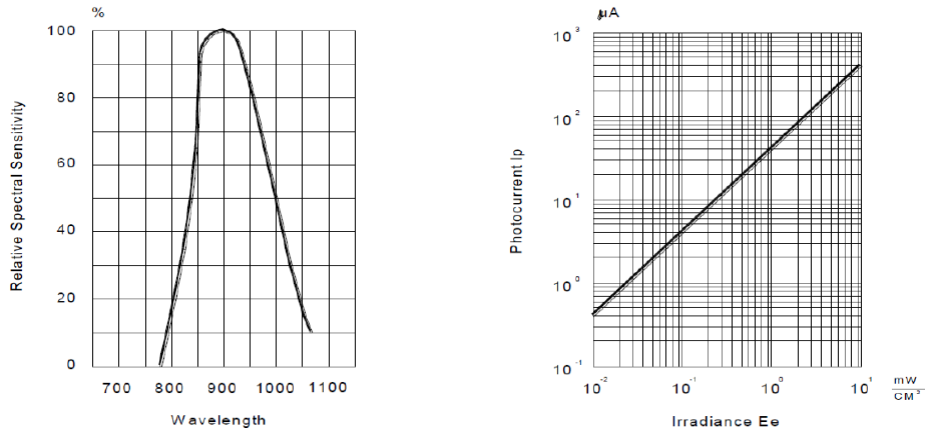


图 3-4 光敏三极管特性图

4. 实验方案

4.1 整体框架

本次实验的代码整体框架如下图 4-1 所示：

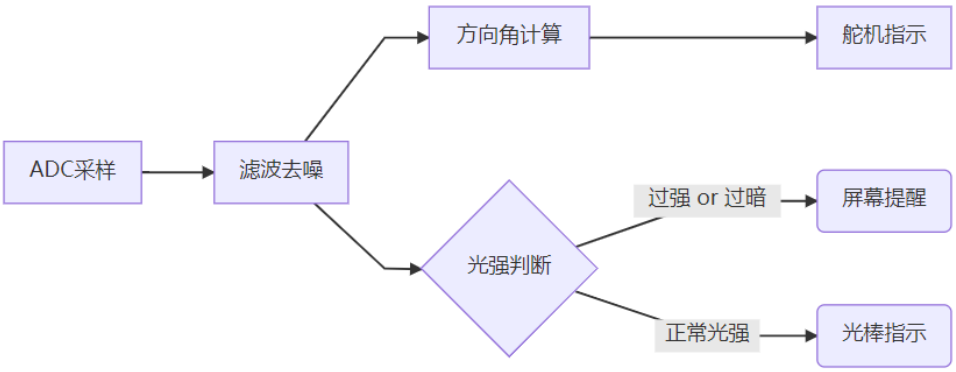


图 4-1 实验基本框图

我们小组利用基于 Ref3 示例代码文件进行开发拓展，完整实现了对光敏三极管的信号采样功能并对光源方向角及光强进行判别。如下文所述，我们将实验分为了 ADC 采样、舵机控制、方向角计算、光棒指示和屏幕提醒及电量显示几个部分。

4.2 ADC 信号采集

我们使用 PIT 中断实现 ADC 采样与舵机的定时控制，控制周期为 20ms。

4.2.1 ADC 信号采样

PIT 全称为 Programmable Interval Timer，是一个可编程的周期中断定时器。PIT 的时钟源只有一个，即总线时钟。在 PIT 定时器是一个减法定时器，内部存在许多个寄存器，有两个寄存器分别存储计数值与当前定时值，并在工作时会对存放的值以总线频率进行自减，而当计数值减为 0 时，PIT 就会被触发一次，并开启下一周期。对于 K66 核心板，可以开启的 PIT 中断共有四个通道（kPIT_Chnl_0-kPIT_Chnl_3），在实验中，我们采用 kPIT_Chnl_2 通道做为计时器中断对 ADC 进行定时采样，每 10ms 触发一次并进行采样。

PIT 定时器的中断服务部分程序代码及 ADC 采样的部分程序代码如下所示，完整代码可见文件 pit_timer.c 与 pit_timer.h，以及/CDK66/MEM.h 和/CDK66/MEM.c 。

```
1.  //-------------------------------------------------------------------------
2.  //  @brief      appTicks increasing at an interval of 20ms
3.  //-------------------------------------------------------------------------
4.  void ADC_PIT_HANDLER(void)
5.  {
6.      /* Clear interrupt flag.*/
7.      PIT_ClearStatusFlags(PIT, kPIT_Chnl_2, kPIT_TimerFlag);
```

```
    Light_get(&Light);
    cal_dir();
    servo_control(Light.error);

#if defined __CORTEX_M && (__CORTEX_M == 4U)
    __DSB();
#endif
}
```

```
//-------------------------------------------------------------------------
//  @brief          Get message from adc
//  @return         adc_value of MEMS
//  @data           2020/10/25
//  Sample usage:
//-------------------------------------------------------------------------
uint16_t ADC_convert(ADC_Type *adcn, uint16_t ch)
{
    adc16_channel_config_t adc16ChannelConfigStruct;
    adc16ChannelConfigStruct.channelNumber = ch;
    adc16ChannelConfigStruct.enableInterruptOnConversionCompleted = false;

    ADC16_SetChannelConfig(adcn, 0, &adc16ChannelConfigStruct);
    while (0U == (kADC16_ChannelConversionDoneFlag &
                        ADC16_GetChannelStatusFlags(adcn, 0U))) {}
    uint16_t adc_value=(ADC16_GetChannelConversionValue(adcn, 0) & 0xFFF);
    return  adc_value;
}
```

4.2.2 滤波去噪

直接采用 ADC 采样得到的结果噪声较大，数据难以直接使用，需使用滤波进行去噪处理。我们采取先中值滤波再进行中位值平均滤波的方法进行降噪处理。

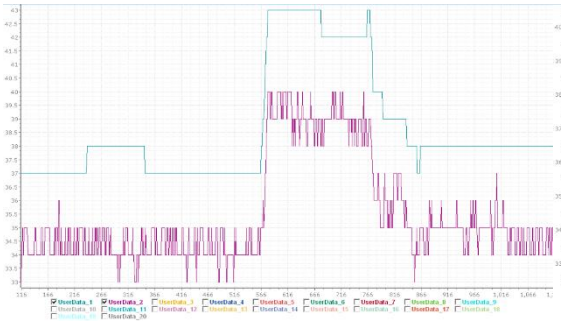


图 4-2 原始采样数据

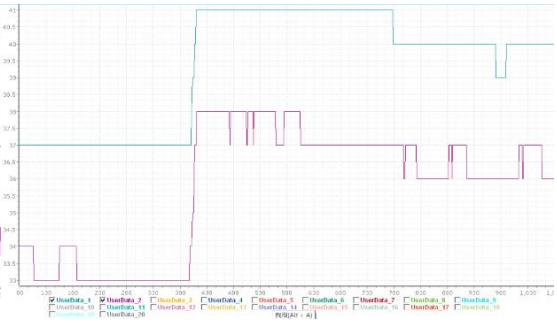


图 4-3 滤波后数据

4.2.2.1 中值滤波

中值滤波采用在一次中断时间连续采样三次并存入数据，经过快速排序后取中位数的方法进行去噪处理。

中值滤波的部分代码如下，完整代码见库文件/CDK66/MEM.h与/CDK66/MEM.c。

```
1.  //-------------------------------------------------------------------------
2.  //  @brief         mid filter
3.  //  @return        Normalize the sampled value
4.  //  @data          2020/10/25
5.  //  Sample usage:
6.  //-------------------------------------------------------------------------
7.  uint16_t ADC_mid(ADC_Type *adcn, uint16_t ch)
8.  {
9.      uint16_t tmp[3] = {0, 0, 0};
10.     for (int i = 0; i < 3; i++)
11.     {
12.         tmp[i] = ADC_convert(adcn,ch);
13.     }
14.     quicksort(MEM1_TEMP, 3, 0, 2);
15.
16.     return tmp[1];
17. }
18.
19. //-------------------------------------------------------------------------
20. //  @brief         Quick sort
21. //  @data          2020/10/25
22. //  Sample usage:
23. //-------------------------------------------------------------------------
24. void quicksort(volatile uint16_t array[], int maxlen, int begin, int end)
25. {
26.     int i, j;
27.
28.     if (begin < end)
29.     {
30.         i = begin + 1; // 将 array[begin]作为基准数，因此从 array[begin+1]开始与基准数比较!
31.         j = end;       // array[end]是数组的最后一位
32.
33.         while (i < j)
34.         {
35.             if (array[i] > array[begin]) // 如果比较的数组元素大于基准数，则交换位置。
```

```
36.             {
37.                 swap(&array[i], &array[j]); // 交换两个数
38.                 --j;
39.             }
40.             else
41.             {
42.                 ++i; // 将数组向后移一位，继续与基准数比较。
43.             }
44.         }
45.         if (array[i] >= array[begin]) // 这里必须要取等“>=”，否则数组元素由相同的值时，会出现错误！
46.         {
47.             --i;
48.         }
49.
50.         swap(&array[begin], &array[i]); // 交换 array[i]与 array[begin]
51.
52.         quicksort(array, maxlen, begin, i);
53.         quicksort(array, maxlen, j, end);
54.     }
55. }
56.
57. //------------------------------------------------------------------------------
58. //  @brief          swap
59. //  @data           2020/10/16
60. //  Sample usage:
61. //------------------------------------------------------------------------------
62. void swap(volatile uint16_t *a, volatile uint16_t *b)
63. {
64.     int temp;
65.
66.     temp = *a;
67.     *a = *b;
68.     *b = temp;
69.     return;
70. }
```

4.2.2.2 均值滤波

我们在中值滤波之后又增加了一层均值滤波，这样处理之后得到的数据结果比单独使用中值滤波得到的数据结果噪声更少。

```
1.  typedef struct _PHOTODIODE
```

```
2.  {
3.      int32_t    L1;
4.      int32_t    L2;
5.      int32_t    error;
6.  } ;
7.
8.
9.  //------------------------------------------------------------------------
10. //  @brief         Get adc value after fiter and normalizing
11. //  @return        Fianl adc_value of MEMS
12. //  @data          2020/10/25
13. //  Sample usage:
14. //------------------------------------------------------------------------
15. void Light_get(PHOTODIODE_t *AnalogInput)
16. {
17.     //Mean filtering
18.     for (int i = 0; i <3; i++)
19.     {
20.         Light1_TEMP[i] = ADC_mid(Light1_BASE,Light1_CHANNEL);
21.         Light2_TEMP[i] = ADC_mid(Light2_BASE,Light2_CHANNEL);
22.     }
23.     quicksort(Light1_TEMP, 4, 0, 3);
24.     quicksort(Light2_TEMP, 4, 0, 3);
25.     //the middle two
26.     AnalogInput->L1 =(Light1_TEMP[1] + Light1_TEMP[2])/2;
27.     AnalogInput->L2 =(Light2_TEMP[1] + Light2_TEMP[2])/2;
28. }
29.
```

4.3 舵机控制

在进行舵机控制时，为了让舵机转动的角度达到预期角度，我们采用位置式 PID 进行调节，使得舵机能够快速到达设定的角度。经过实测，我们小组标定了舵机的 PWM 的最小值、中值和最大值。舵机的控制程序代码见下所示。

```
1.  typedef struct pid_t{
2.      float kp;
3.      float ki;
4.      float kd;
5.      float _p;
6.      float _i;
7.      float _d;
```

```
    float _i_max;
}pid_t;


//舵机控制--位置 pid
float pid_solve(pid_t *pid, float target, float feedback)
{
    float error = target - feedback;
    pid->_d = error - pid->_p;
    pid->_i += error;
    pid->_i = MINMAX(pid->_i, -pid->_i_max, pid->_i_max);
    pid->_p = error;
    return pid->kp * pid->_p + pid->ki * pid->_i + pid->kd * pid->_d;
}
```

```
void servo_control(float anchor_point) //输入偏差
{
    servo_duty = SERVO_DUTY_MID - pid_solve(&servo_pid, anchor_point, 0);
//舵机 pid
    servo_duty = MIN(MAX(servo_duty, SERVO_DUTY_MIN), SERVO_DUTY_MAX);
    Update_ServoUS(kFTM_Chnl_0, servo_duty);
    Update_ServoUS(kFTM_Chnl_1, 3000-servo_duty);
}
```

```
void static inline Update_ServoUS(uint8_t ChanNo, uint32_t DutyCycleUS) {
    uint32_t cv, mod;

    if ((ChanNo<2) && (DutyCycleUS<2500) && (DutyCycleUS>500))
    {
        // mod value mapped to 10ms|10000us period (100Hz)
        mod = FTM3_PERIPHERAL->MOD;
        cv = (mod * DutyCycleUS) / 10000U;
        if (cv >= mod)
        {
            cv = mod + 1U;
        }
        FTM3_PERIPHERAL->CONTROLS[ChanNo].CnV = cv;
        FTM_SetSoftwareTrigger(FTM3_PERIPHERAL, true);
    }
```

```
16.     return;
17. }
```

4.4 方向角判断

假设点光源和传感器共面，则两个传感器的光强比，在平面上确定了光源所在的曲线；在这条曲线上，使传感器接收到相同光强的光源，可能是近处的弱光源，也可能是远处的强光源。因此，如果仅依靠两个传感器的光强来求解光源角度，在数学上有无数个解。

在这种情况下，我们提出了两种方向角判断算法：差比和算法和几何解算。差比和算法能大致指出光源的方向；几何解算能比较准确地指出光源方向，但需要在光源强度恒定，且光源和传感器大致共面的情况下使用。最后考虑泛化性我们采用了差比和算法判断方向角。

4.4.1 差比和算法

差比和算法既利用 ADC 采样得到的左右两个光敏三极管两组数据、通过两者之差除以两者之和来判断光源离哪个三极管更近一些，然后根据结果大致估测光源的方向，进而估算出光源的方向角。差比和算法程序代码见下。

```
1.  Light.error=250*(Light.L1-Light.L2)/(Light.L1+Light.L2);
2.  direction = 1.4*(Light.L1-Light.L2)/(Light.L1+Light.L2);
3.  direction= MIN(MAX(direction, -0.99), 0.99);
```

4.4.2 几何解算

下面介绍几何解算的原理。

在点光源和传感器共面的前提下，通过两个传感器采集到的测量值，求出点光源到两个传感器的距离。两个传感器和点光源构成一个三角形，已知三角形三边长度，可以求解点光源与两个传感器中点的连线和传感器的夹角，即求得光源相对实验板的方位角。

4.4.2.1 标定

对于同一个点光源，传感器的输出数值 y 和距离 x 的平方成反比，即 $y=k/x^2$。因此，可以通过测量不同距离处的传感器同一个点光源的响应，得到距离和传感器的函数关系式。

通过实验得知，以某手机的闪光灯作为点光源时，传感器的输出数值 y 和距离 x 的平方关系如下表。

距离 x/cm	输出数值 y
2.7	440
3.6	268
4.5	203
5.4	151
6.3	125
7.2	105
8.1	91

通过拟合得到 $y=2832.1/x^2+52.836$。

考虑到测量误差，在 2832 附近调节 k 值，使测角更准确。通过调试，最后选定 k=3000。即使用这个闪光灯作为光源时，传感器的输出数值 y 和距离 x 的平方关系为 $y=3000/x^2$ 。

4.4.2.2 解算

几何结算方法程序代码如下。

```
d1=sqrt(k/Light.L1);
d2=sqrt(k/Light.L2);
d0 = 1.7;//两个光敏三极管的距离
if(d1+d2>d3&&d2+d3>d1&&d3+d1>d2)//几何解算，估测的两个光敏三极管和光源的距离，和两个光敏三极管的距离，三边可以构成三角形
   {
  angle1 = acos((d1*d1 + d0 * d0 - d2 * d2) / 2 / d1 / d0);
     d3 = sqrt(d1*d1 + d0 * d0 / 4 - cos(angle1)*d1*d0);
     angle2 = asin(d0/2/d3*sin(angle1));
     angle = angle1 +angle2;
     error0=(2/3.1415926*angle-1);
     direction = error0;
     Light.error=(SERVO_DUTY_MAX-SERVO_DUTY_MIN)/2*error0;
   }
else//估测的两个光敏三极管和光源的距离，和两个光敏三极管的距离，三边不能构成三角形，不能进行几何解算，用差比和算法顶替
{
    Light.error=(SERVO_DUTY_MAX-SERVO_DUTY_MIN)/2*(Light.L1-Light.L2)/(Light.L1+Light.L2);
       direction = 1.6*(Light.L1-Light.L2)/(Light.L1+Light.L2);
       direction= MIN(MAX(direction, -0.99), 0.99);
}
```

4.5 显示模块

4.5.1 过亮过暗屏幕显示

在光源过亮或过暗的情况下光敏三极管的测量已经饱和，得到的数据并不能很好地反映真实的距离，同时舵机的指针可以自由转动的空间并不多，也为了舵机指针能够敏感的指示光源方向，我们在光源过亮或过暗的时候会在屏幕上输出显示警告“TOO LIGHT!”以及“TOO DARK”在此时舵机保持不动，不再进行更新。

具体运动状态判定程序代码如下。

```
if(Light.L1>=4096||Light.L2>=4096)
 {
   OLED_P8x16Str(0, 6, "TOO LIGHT!");
   BOARD_I2C_GPIO((1<<16)-1);
```

```
  }//光柱指示方向
else if(Light.L1<10 && Light.L2<10)
{
  OLED_P8x16Str(0, 6, "TOO DARK!");
  BOARD_I2C_GPIO((1<<16)-1);
}//光柱指示方向
```

4.5.2 光棒指示

在解算光源方向的时候，我们小组也设定了**direction**变量来作为光棒的方向指示，具体程序代码如下：

```
BOARD_I2C_GPIO(1<<(int)(8-direction*8));//光柱指示方向
```

在过亮或过暗的时候光棒会指向最左或最右，作为光强的一个指示。

4.5.3 电量显示

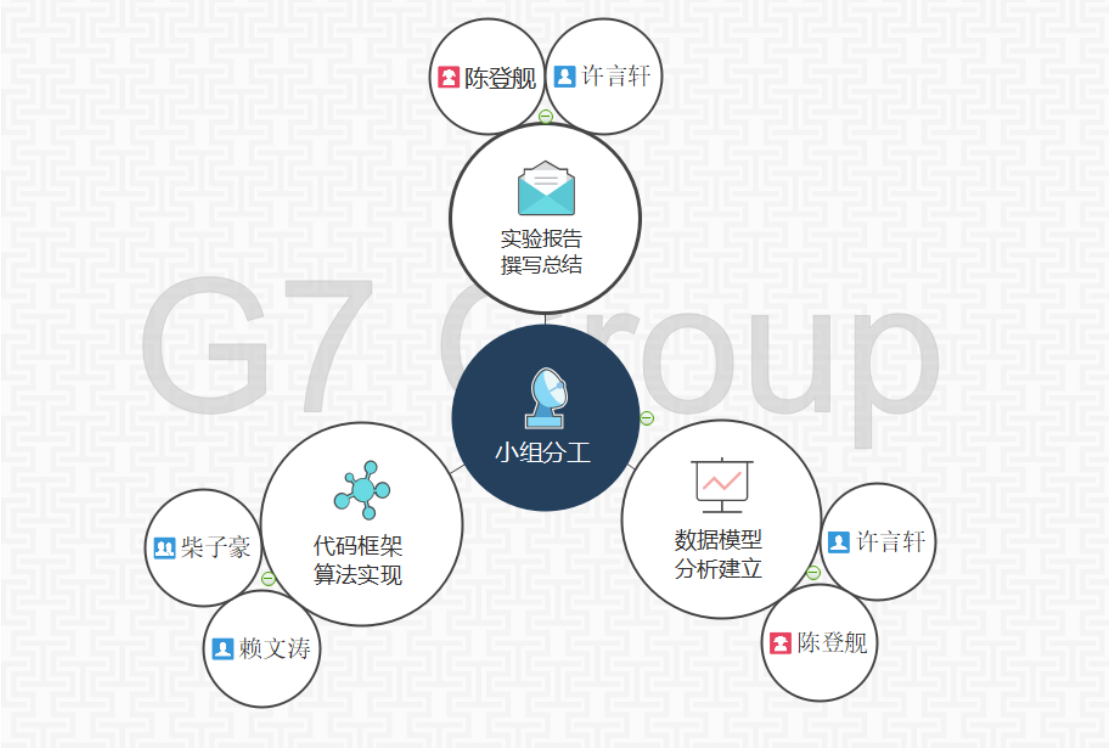
因为此次实验需要用到电池，而电池没电会影响程序使之不能正常运行，为了避免因为没电而得到错误的结果使得我们在毫无意义的debug，我们增加了一个电池电量的显示来告诉我们什么时候需要充电。因为屏幕大小问题，在最后的最终版程序中，将此部分代码注释，保证了屏幕输出的美观。程序代码见下。

```
  uint16_t volt=MEM_convertB(ADC0,7U)*1000*3.3*5/4096;
  OLED_uint16(0, 2, volt);
  OLED_P8x16Str(50, 2, "mV");
uint16_t MEM_convertB(ADC_Type *adcn, uint16_t ch)
{
    adc16_channel_config_t adc16ChannelConfigStruct;

    adc16_channel_mux_mode_t ADCMUX;
    ADCMUX = kADC16_ChannelMuxB;
    ADC16_SetChannelMuxMode(ADC0, ADCMUX);
    ADC16_SetChannelMuxMode(ADC1, ADCMUX);

    adc16ChannelConfigStruct.channelNumber = ch;
    adc16ChannelConfigStruct.enableInterruptOnConversionCompleted = false;

    ADC16_SetChannelConfig(adcn, 0, &adc16ChannelConfigStruct);
    while (0U == (kADC16_ChannelConversionDoneFlag &
                  ADC16_GetChannelStatusFlags(adcn, 0U))) {}
    uint16_t adc_value=(ADC16_GetChannelConversionValue(adcn, 0) & 0xFFF);
    return  adc_value;
}
```

5. 实验结果分析与结论

本次实验以光敏三极管LTR-546AD传感器信号的ADC采样与处理为基础，在实验板上接入舵机，在光照条件下，通过光敏三极管的输出信号来控制舵机的转向。

本次实验，我们小组基本分工如下图所示：



在本次实验的过程中，我们发现ADC采样光敏三极管的信号存在抖动噪声，为去噪以获得一个好的输出信号，我们采取两次滤波的方法，使采样信号值满足我们的实验需要。

为了在方位角识别可靠的情况下对舵机进行控制，我们使用差比和算法来判断方向角，实现“追光”效果，使舵机舵角始终指向光源方向。并将识别的方向角动态显示与16位光柱作直观显示。在实验中，我们小组首先尝试了几何解析法，对于同一个点光源，标定了光敏三极管传感器的输出数值y和距离x的值，并计算出两者之间的关系。但由于实验精度的局限与实验误差的不可避免性，我们在标定上述值时存在一定的人为误差，进而导致实验存在一定的误差。最终，我们依然选择了较为可靠的差比和算法进行最终测试。

经过本次实验，我们基本熟悉示例代码Ref3的基本思路与代码实现方法，并以此代码为基础进行调整与改进，进行基本库函数的完善与封装，为后续实验积累库函数与便于参照的工程代码，后续的实验内容我们也会在此次实验代码的基础上进行开发。同时，我们也成功的将ADC采样与滤波融入到了程序开发中，并对方向角进行了可视化光棒显示，来获得一个较为直观的实验效果。这为我们以后的程序开发与调试能力打下了良好的基础。

最后，感谢王冰老师为本门课程的精心准备与悉心讲解，深入浅出的为我们讲解嵌入式开发的诸多知识，将复杂的嵌入式知识网络按照对应模块分割开来，逐一细解，帮助我们拓宽视野，增长见识。