目录

1．实验背景简介 1
2. 实验内容 1
3. 实验原理 2
3.1 舵机控制原理 2
3.2 PID 控制原理 3
3.3 线阵 CCD 基本工作原理 3
4. 实验方案 4
4.1 整体框架 4
4.2 ADC 信号采集 4
4.3 舵机控制与光棒指示 6
4.3.1 舵机控制 6
4.3.2 光棒显示 7
4.4 畸变处理 7
4.4.1 去除暗电流 7
4.4.2 自适应增益 8
4.4.3 代码实现 9
4.5 大津法动态阈值 10
4.6 打擂计分寻线 12
5. 实验结果分析与结论 13

1. 实验背景简介

CCD（ChargeCoupledDevice）——电荷耦合器件，越来越广泛被应用于工业、军事、民用等行业，因其具有高解析度、低杂讯、动态范围广、良好的线性特性曲线等优点，在非接触测量及成像系统中应用极为广泛。特别是线性 CCD 因其具有体积小、重量轻、接口简单、采样率高等优点，适合于运动物体的数据采集，如智能小车运动过程中对赛道信息的采集等。采用性能优良的 CCD 作为光电接收器件，感光效果好，响应时延短，实时性能和精度优秀，满足多种移动性强的现场应用要求。

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，在参考路线识别可靠的情况下对舵机进行控制，实现随动控制的效果，使舵机舵角始终参考路线方向。Cyber-Dorm 实验板照片如图 1-1 所示。

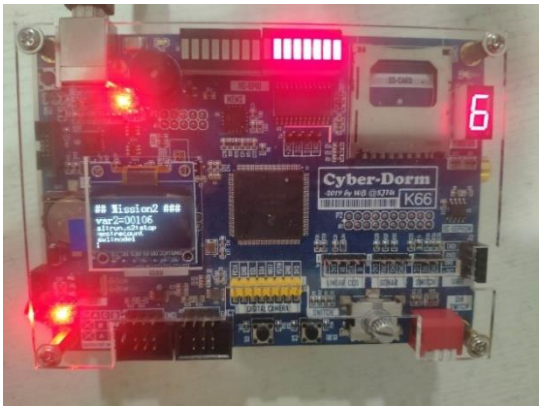

图 1-1 实验 Cyber-Dorm 开发板

2. 实验内容

本次实验需要完成的任务是使用线阵CCD传感器在参考路线可以可靠识别的情况下对舵机进行控制，实现随动控制的效果，即使舵机的角度始终参考路线的方向。

实验具体内容如下：使用ADC采集线阵CCD传感器信号，并对信号进行分析，找出信号特性并进行标定。在光线适宜条件下识别参考路线，并控制舵机随着路线变化而变化，实现舵机的随动控制，并将路线识别结果动态的显示在16位光柱上。

实验任务
配置接口，采集线阵 CCD 传感器信号并显示
设定算法处理畸变并提取信号特征
控制舵机沿路线随动
将路线识别结果显示在 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。

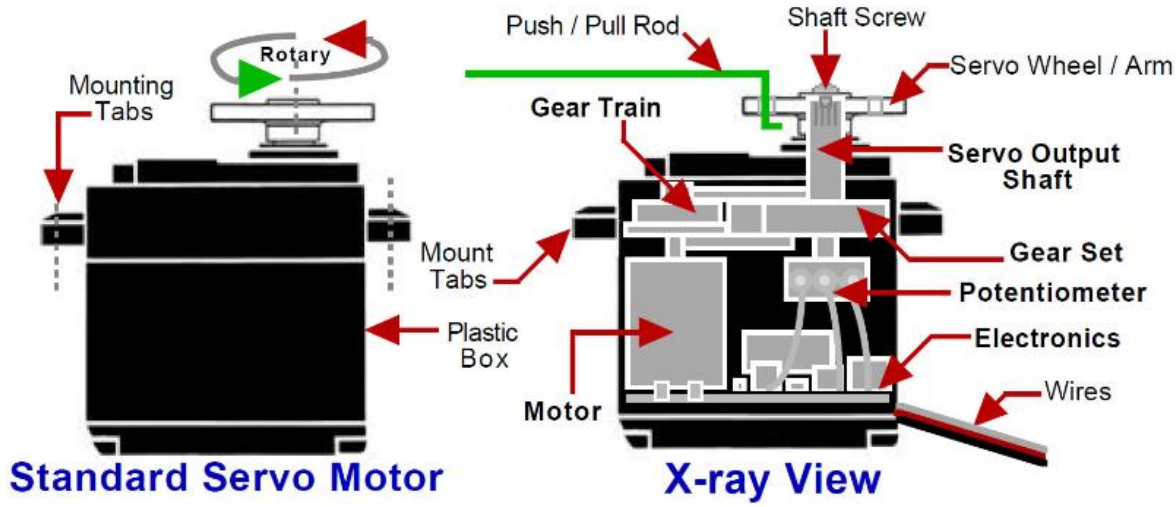

图 3-1 R/C Servo 结构图

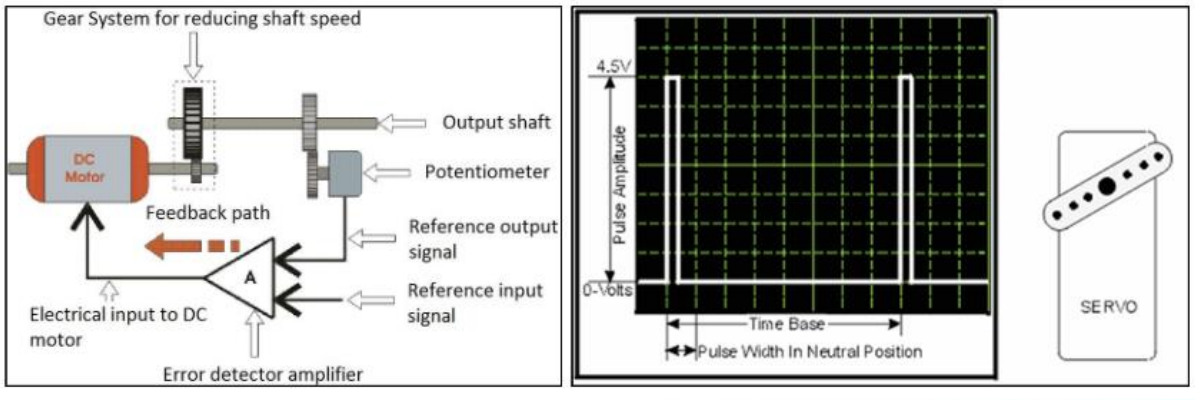

图 3-2 标准舵机控制方法

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

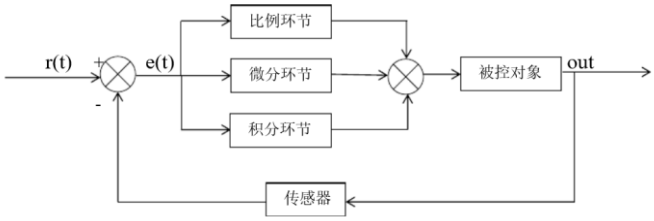

图 3-3 PID 控制流程图

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 线阵 CCD 基本工作原理

线阵 CCD 是将光的强弱转换为电信号的线性光/电器件。

线阵 CCD 的构成为：在 N 型或 P 型硅衬底上生长一层二氧化硅薄层，再在二氧化硅层上淀积并光刻腐蚀出金属电极，这些规则排列的金属-氧化物-半导体电容器阵列和适当的输入、输出电路就构成基本的 CCD 移位寄存器。

对金属栅电极施加时钟脉冲，在对应栅电极下的半导体内就形成可储存少数载流子的势阱。可用光注入或电注入的方法将信号电荷输入势阱。然后周期性地改变时钟脉冲的相位和幅度，势阱深度则随时间相应地变化，从而使注入的信号电荷在半导体内作定向传输。CCD 输出是通过偏置 PN 结收集电荷，然后放大、复位，以离散信号输出。

线阵 CCD 的具体工作原理图图见下

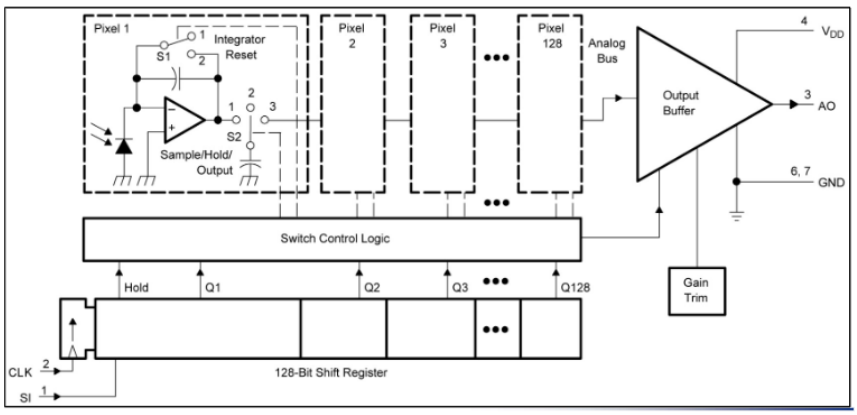

图 3-4 光敏三极管特性图

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：

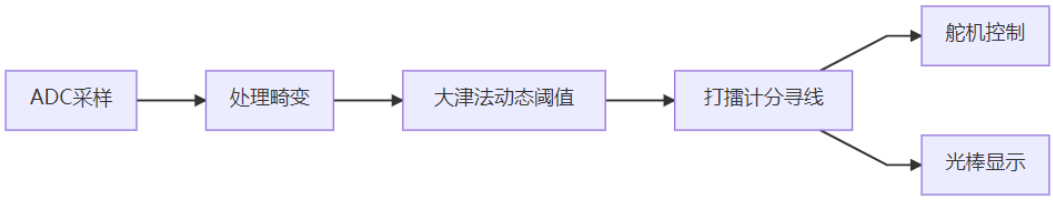

图 4-1 实验基本框图

我们小组利用基于 Ref5 示例代码文件进行开发拓展，完整实现了对线阵 CCD 的信号采样功能并实现了对舵机的随动控制。如下文所述，我们将实验分为了 ADC 采样、畸变处理、动态阈值、打擂计分法寻线、光棒指示和舵机控制几个部分，考虑到本次实验的重点在于信号数据处理，所以在下文的叙述中将 ADC 采样和舵机控制与光棒指示提前，在此部分之后再介绍畸变处理、大津法和打擂计分寻线的方法。

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。

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/CCD.h 和/CDK66/CCD.c 。

```
1.  void ADC_PIT_HANDLER(void)
2.  {
3.      /* Clear interrupt flag.*/
4.      PIT_ClearStatusFlags(PIT, kPIT_Chnl_2, kPIT_TimerFlag);
5.
6.      LinearCameraOneShot();
7.      Distortion_correction();
8.      threshold=autoThreshold();
9.      threshold=MINMAX(60,threshold,120);
```

```
    line_find();
    CCD_cal();
    servo_control(CCD_error);

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

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

```
void LinearCameraOneShot(void)
{
    uint8_t index=0;

    // flush previously integrated frame before capturing new frame
    LinearCameraFlush();
    // wait for TSL1401 to integrate new frame, exposure time control by del
ay
    systick_delay_ms(20);

    CCD_SI_HIGH;
    CCD_delay();
    CCD_CLK_HIGH;
    CCD_delay();
    CCD_SI_LOW;
    CCD_delay();
    CCD_CLK_LOW;

    // ADC mux to CCD channel
    ADC16_SetChannelMuxMode(ADC0, kADC16_ChannelMuxA);

    for(index=0; index<128; index++)
    {
        // read AO value using ADC
        ADC16_SetChannelConfig(ADC0_PERIPHERAL, ADC0_CH2_CONTROL_GROUP, &ADC
0_channelsConfig[2]);
        while (0U == (kADC16_ChannelConversionDoneFlag &
```

```
                    ADC16_GetChannelStatusFlags(ADC0_PERIPHERAL, ADC0_CH2_CONTROL_GROUP))) {}
        CCDData[index] = ADC16_GetChannelConversionValue(ADC0_PERIPHERAL, ADC0_CH2_CONTROL_GROUP) & 0xFFF;
        // clock pulse to read next pixel output
        CCD_CLK_HIGH;
        CCD_delay();
        CCD_CLK_LOW;
    }

    // 129th pulse to terminate output of 128th pixel
    CCD_CLK_HIGH;
    CCD_delay();
    CCD_CLK_LOW;
}
```

4.3 舵机控制与光棒指示

对于舵机的控制本次实验采用 PID 控制算法，在计算方向角误差的时候采用低通滤波的方法，可以去除高频噪声的干扰，增强项目工程的普适性。

4.3.1 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。

```
typedef struct pid_t{
    float kp;
    float ki;
    float kd;
    float _p;
    float _i;
    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 CCD_cal(void)
{
    CCD_error=-(CCD_position-64)*140/64* 0.3 + CCD_error*0.7;
    direction = -1.4*(CCD_position-64)/64* 0.3+direction*0.7;
    direction= MIN(MAX(direction, -0.99), 0.99);
}

```

4.3.2 光棒显示

光棒显示与之前的方法相同，但在方向的计算上参考上一次的方向，将上一次的朝向考虑进去，使得方向变化连续，去除高频噪声干扰。

```
//////////////光棒显示
BOARD_I2C_GPIO(1<<(int)(8-direction*8));
```

4.4 畸变处理

影响线性 CCD 图像信号的因素可分内部因素和外部因素。内部因素包活暗电流、镜头畸变、温漂等。畸变主要是由镜头引起的，即存在中间亮两边暗的情况，且信号的放大倍数越大该畸变就越严重，绘利用阈区分黑白赛道带来了问题。

为了取得较好的效果，我们需要对 CCD 信号进行处理，其中主要是处理畸变。

4.4.1 去除暗电流

即使将 CCD 捂住，在一定的曝光周期下也会存在输出信号。信号的可用范围变窄并且

通过实验验证不同像素点的暗电流大小是不一样的，经过放大器后不同像素点暗电流的差异会给最终的信号带来毛刺，所以我们需要先去除暗电流。

去除暗电流方法如下：

首先将线性 CCD 捂住，此时 CCD 的输出值就是暗电流，对于放大倍数固定的情况，该暗电流的值可以固化成表。在光线均匀的情况下，将一张干净均匀的白色纸张放在线性 CCD 前，采集并记录此时 CCD 的输出，减去暗电流并记录。

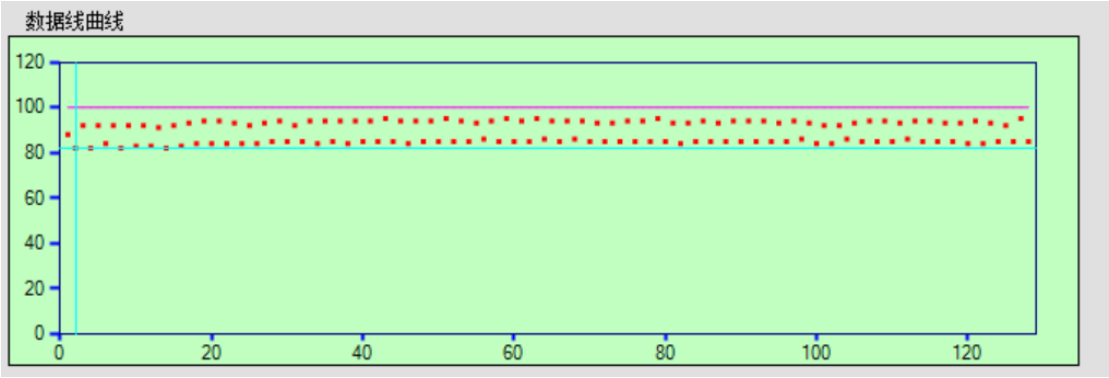

图 4-2 暗电流标定(81,93）

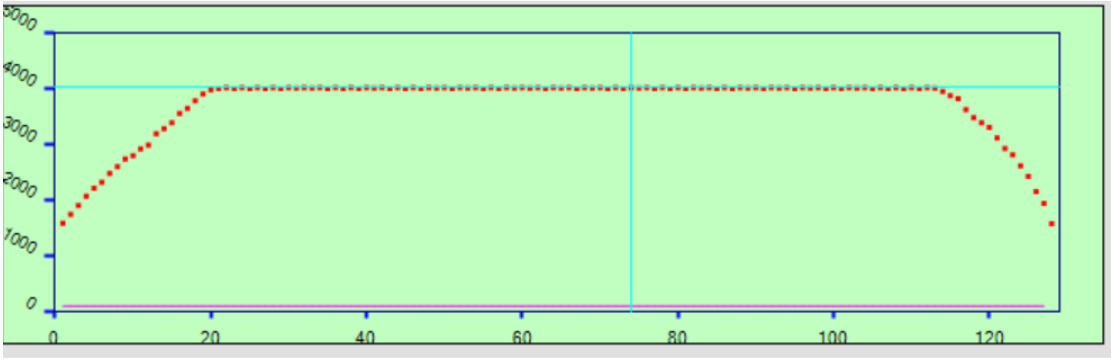

图 4-3 纯白数据标定 (1634,4009)

4.4.2 自适应增益

在将一张干净均匀的白色纸张放在线性 CCD 前，理想的 CCD 的输出应该是一条平直的直线，但是实际上是一条上凸的弧线。为了得到理想的曲线，我们标定了每个像素点的自适应增益来去除畸变。

在 4.4.1 节里已经获得了一张减去暗电流后的数值表，得到了一个上凸曲线，取次上凸曲线的最大值做一条直线，计算每个点到此最大值的的增益。并将此增益固化成表。

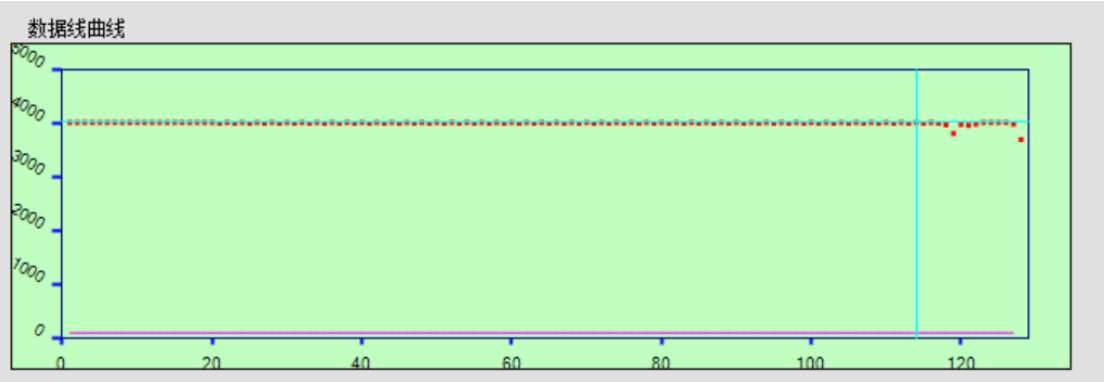

图 4-4 标定处理后最终结果图

4.4.3 代码实现

我们在得到数据的图像后将之近似成一个梯形，在两侧为与数据位的一次函数，中间的数据近似为直线。在两侧的数据在减去暗电流的值后通过一次函数进行调整，中间部分的数据在减去暗电流的值后不做较多处理。

去畸变代码程序如下，在项目工程中路径为/CDK66/CCD.c

```
1.  void Distortion_correction(void)
2.  {
3.      int32_t tempInt=0;
4.      for(tempInt=0; tempInt<128; tempInt++)
5.      {
6.
7.          if(tempInt%2==0)
8.          {
9.                CCDData[tempInt] = MAX(0,CCDData[tempInt]-96); // Upper byte
10.         }
11.         else
12.         {
13.               CCDData[tempInt] = MAX(0,CCDData[tempInt]-83); // Lower byte
14.         }
15.         if(tempInt<21)
16.         {
17.               int32_t tep=125*(20-tempInt); //118.75
18.               CCDData[tempInt] =  MIN(CCDData[tempInt]+tep,4009);    // Upper
    byte
19.         }
20.         else if(tempInt>128-20)
21.         {
22.               int32_t tep=250/2*(tempInt+20-128);
```

```
            CCDData[tempInt] = MIN(CCDData[tempInt]+tep,4009); // Upper byte
        }
    }
    for(tempInt=0; tempInt<128; tempInt++)
    {
        uint16_t data=MIN(255,CCDData[tempInt]*255/4009);
        NormalCCDData[tempInt]=data;
    }

}
```

4.5 大津法动态阈值

大津法是一种自适应的阈值确定的方法。它是按图像的灰度特性，将图像分成背景和目标 2 部分。背景和目标之间的类间方差越大，说明构成图像的 2 部分的差别越大，当部分目标错分为背景或部分背景错分为目标都会导致 2 部分差别变小。因此，使类间方差最大的分割意味着错分概率最小。大津法对噪音和目标大小十分敏感，它仅对类间方差为单峰的图像产生较好的分割效果。

在线性 CCD 传感器的使用中，对于 CCD 采集到的一维图像，将车道两侧的黑线作为前景，白色 KT 板作为背景，分割阈值记为 T。前景像素点的比例为 a_0，平均值为 b_0，背景像素点的比例为 a_1，平均值为 b_1。图像的总平均值为 b，类间方差为 g。可以得到

$$g = a_0 * a_1 \ (b_0 - b_1)^2$$

从最小值到最大值遍历 g，当 g 取得最大值时，可以认为此时前景和背景的差异最大，此时的灰度值 b 是最佳阈值。

由于线性 CCD 检测的灵敏度较高，外界光线环境的微小变化都会使线性 CCD 采集的像素值发生波动，而这种波动会导致动态阈值发生频繁的随机波动，从而容易引起震荡。为此，为了避免动态阈值发生频繁波动，同时又不降低线性 CCD 对光线环境的适应能力，我们利用周期动态阈值法，每隔几个周期就利用大津法进行动态阈值的计算。在得到最佳阈值之后进行二值化。

```
int autoThreshold()
{
  int pixelCount[GrayScale] = {0}; //每个灰度值所占像素个数
  float pixelPro[GrayScale] = {0}; //每个灰度值所占总像素比例
  int tmp_threshold = 0;

  //统计灰度级中每个像素在整幅图像中的个数
  for (int i = 0; i < 128; i++)
  {
      pixelCount[NormalCCDData[i]]++;
```

```
  }

  for(int i = 0; i < GrayScale; i++)
  {
     pixelPro[i] = (float)pixelCount[i] / 128; //计算每个像素在整幅图像中的比例
  }
  float w0, w1, u0tmp, u1tmp, u0, u1, u, deltaTmp, deltaMax = 0;
  for (int i = 0; i < GrayScale; i++) // i 作为阈值
  {
    w0 = w1 = u0tmp = u1tmp = u0 = u1 = u = deltaTmp = 0;
    for (int j = 0; j < GrayScale; j++)
    {
      if (j <= i) //背景部分
      {
        w0 += pixelPro[j];                          //背景像素点占整个图像的比例
        u0tmp += j * pixelPro[j];
      }
      else //前景部分
      {
        w1 += pixelPro[j];                          //前景像素点占整个图像的比例
        u1tmp += j * pixelPro[j];
      }
    }
    u0 = u0tmp / w0;                                //背景平均灰度 μ0
    u1 = u1tmp / w1;                                //前景平均灰度 μ1
    deltaTmp = (float)(w0 * w1 * pow((u0 - u1), 2)); //类间方差公式 g = w1 * w2 * (u1 - u2) ^ 2
    if (deltaTmp > deltaMax)
    {
      deltaMax = deltaTmp;
      tmp_threshold = i;
    }
  }
  return tmp_threshold;
}
for(uint32_t i=0; i<128; i++)
{
     if(NormalCCDData[i]>threshold) CCDBinary[i]=255;
     else CCDBinary[i]=0;
}
```

4.6 打擂计分寻线

在得到 CCD 的二值化信号之后，需要根据二值化的数据找到路线。此时可以采取记分打擂法来找到路线。

首先先遍历一遍数据，找到路线可能存在的位置，然后对满足条件的位置的权重赋予初值。

在权重初值赋予之后，再进行遍历并调整权重分值。对于靠近上一个路线位置的像素点对应的权重值进行增加，对于靠近中间的像素点对应的权重值也增加。最后选取得分最高的像素点的位置做为路线对应的位置。

打擂计分寻线法的程序代码如下：

```
1.  void line_find(void)
2.  {
3.      for(uint32_t i=0; i<128; i++)
4.      {
5.           if(NormalCCDData[i]>threshold) CCDBinary[i]=255;
6.           else CCDBinary[i]=0;
7.      }
8.      for(uint32_t i= 3; i<125; i++)
9.      {
10.        if(CCDBinary[i-
    1]==0 && CCDBinary[i]==0 && CCDBinary[i+1]==0)  {CCDScore[i]=1;}
11.        else  { CCDScore[i]=0;}
12.     }
13.     uint32_t max_score=0;
14.     for(int32_t i=3; i<125; i++)
15.     {
16.        if(i<24)   { CCDScore[i]=i*CCDScore[i];}
17.        else if(i>104)  {CCDScore[i]=(128-i)*CCDScore[i];}
18.        else            {CCDScore[i]=25*CCDScore[i];}
19.        if(abs(i-last_CCD_position)<20)
20.        {
21.             CCDScore[i]=(abs(i-last_CCD_position)+50)*CCDScore[i]/50;
22.        }
23.
24.        if(max_score<CCDScore[i])
25.        {
26.            max_score=CCDScore[i];
27.            CCD_position=i;
28.        }
29.     }
30.     last_CCD_position=CCD_position;
31. }
```

5. 实验结果分析与结论

本次实验以线阵CCD传感器信号的ADC采样与处理为基础，在实验板上接入舵机，在事宜光照条件下，通过线阵CCD传感器的输出信号来控制舵机的转向。

本次实验大致分工如下图所示：

在本次实验的过程中，为了取得较好的实验效果，我们查阅了一些与智能车竞赛相关的文献，从他人的文献中获得启发，经过实验，线CCD畸变处理、大津法、PID等算法对舵机控制有着不错的效果。

通过使用大津法设置动态阈值，程序在不同的相对正常的光照条件下可取得较佳的实验效果。

经过本次实验，我们基本熟悉示例代码Ref5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，我们也成功的将大津法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，同时我们还将方向角以及确定的位置误差通过串口发送了出来并做了可视化。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。