Lab4 动态曝光实现、整体代码的解释以及结果展示

胡小雨

1. 整体代码：

根据这次要求我们需要使用线阵摄像头实现对于道路的检测进而控制舵机的转向，因而大体代码要实现的功能有判断道路（黑线）所在位置以及通过位置控制舵机的转动着两部分。为了实现我们的功能，定义了全局变量矩阵CCDDataGaus[128] 来接收并处理摄像头采集到的数据而由于OLED上面显示点的个数为128个，故数组规模为128，并通过曝光函数 LinearCameraOneShot()中的

CCDData[index] = ADC16\_GetChannelConversionValue(ADC0\_PERIPHERAL, ADC0\_CH2\_CONTROL\_GROUP) & 0xFFF； 语句来得到原始数据CCDData[]。

第一部分是寻找路线（中线）：

1. 通过观察实验我们发现，由于摄像头的凸性以及屏幕反光等问题，当屏幕全白时OLED屏幕上呈现出来的点线是一条高斯分布曲线（正态曲线），这一点对于正确可靠识别道路有很大影响，而由于是正态分布，所以我们可以通过将所得的128个CCDData []数据进行拉直，即

CCDData[i]=CCDData[i]/**exp**(-**pow**(i-64,2)/1800);

其实exp（）就是一个正态分布，方差1800是电脑屏幕数据，室内光数据应为3600左右。通过此函数可以很好的将摄像头由于凸性导致的图像不正确问题解决。

下面函数为镜头矫正函数

**void** **LDC**(){

**for**(**int** i = 0;i<128;i++){

CCDData[i]=CCDData[i]/**exp**(-**pow**(i-64,2)/1800);//正态分布方差需要调整，电脑屏幕1800左右，一般室内光3600左右

}

}

1. 同时为了得到平滑的数据曲线，我们对所得的CCDData []数据进行高斯滤波操作。数字图像用于后期应用，其噪声是最大的问题，由于误差会累计传递等原因，此时可用高斯滤波这种滤波器得到信噪比SNR较高的图像（反应真实信号）。简单的理解高斯滤波去噪就是对整幅图像像素值进行加权平均，针对每一个像素点的值，都由其本身值和邻域内的其他像素值经过加权平均后得到。对于我们的实验，我们采用对左右各三个点的范围内进行加权求和从而达到滤波效果：

下面函数为高斯滤波，减小噪声

//高斯滤波，减小噪声

**void** **CCDGaus**(){

uint16\_t CCDTemp[134]={0};

**for** (**int** i = 3;i < 131;i++){

CCDTemp[i] = CCDData[i-2];

}

**for** (**int** i = 0;i < 128;i++){

CCDDataGaus[i]=0.4\*CCDTemp[i+3];

CCDDataGaus[i]+=0.14\*(CCDTemp[i+2]+CCDTemp[i+4]);

CCDDataGaus[i]+=0.09\*(CCDTemp[i+1]+CCDTemp[i+5]);

CCDDataGaus[i]+=0.07\*(CCDTemp[i]+CCDTemp[i+6]);

}

}

通过使用一个中间过渡矩阵CCDTemp[]我们便能实现在某一像素点左右范围内求加权平均的效果，最终得到拥有良好平滑性的数组数据CCDDataGaus[]。

1. 在经过上述两个对数据进行预处理的函数后，我们已经可以得到一个可靠性高且利用性更好的数组数据CCDDataGaus[]。再利用我们上课时老师讲解过程中提到的二分法对道路进行识别分析：

求均值average

//求均值，认定小于0.4均值为道路线

**void** **Get\_01\_Value\_CCD**(uint16\_t\* Data){

**float** average=0;

**for** (**int** i = 0;i<128;i++){

average += Data[i]/128.0f;

}

**for** (**int** i = 0;i<128;i++){

**if** (Data[i]<0.4\*average)

Data[i]=1;

**else**

Data[i]=0;

}

my\_average=**abs**(average);

}

根据二分法，首先求得所有数据的平均值average，再由我们经验观察所得将所得到的CCDDataGaus[i]（在此函数中将实参CCDDataGaus[]赋给了形参Data[]）数据与经验所得的0.4倍平均值进行比较，小于后者的数据被重置为1，即被判断为是道路段中的一点，反之设为0，即不是道路段。此时我们得到的CCDDataGaus[i]数据即为判断道路后的数据了，在之后根据这一判断依据进行操作。

my\_average是动态曝光控制代码中需要使用的变量。

1. 为了方便观察调试，我们将所得的CCDDataGaus数据通过定义的全局数组变量CCD2PC[260]传送到上位机的串口，代码如下，具体实现原理不再多说：

下面的函数将数据Data送到上位机串口

**void** **SendCCDData**(uint16\_t\* Data){

//send 128 points CCD data (128\*2byte) to UART0, using seekfree protocol

**for**(**int** i=0; i<128; i++)

{

CCD2PC[i\*2+4] = Data[i] >> 8; // Upper byte

CCD2PC[i\*2+5] = Data[i] & 0XFF; // Lower byte

}

UART\_WriteBlocking(UART0\_PERIPHERAL, CCD2PC, 260U);

}

1. 由于实验中要求的场景是小车根据图像巡中线前进，则我们还需要对所判断出来的道路段进行中线寻找，方便通过中线位置来控制舵机旋转位置。

下面函数为找路线的中线

uint8\_t **FindMidLine**(){

uint8\_t left=0,right=128;

**for** (**int** i = 5; i<115;i++){//从左往右找第一个从0到1的分界点（左边界）

**if** (CCDDataGaus[i]^CCDDataGaus[i+1]){

left = i;

**break**;

}

}

**for**(**int** i=122;i>0;i--){//从右往左找

**if** (CCDDataGaus[i]^CCDDataGaus[i-1]){

right = i;

**break**;

}

}

**char** txt[16];

**sprintf**(txt,"l:%03d r:%03d", left, right);

OLED\_P6x8Str(10, 1, (uint8\_t \*)txt);

**return** (left+right)/2;//取均值返回道路中点

}

实现的原理其实很简单，因为我们已经在前面通过二分法将CCDDataGaus[]中的数据全部置为1（道路）或0（非道路），故只要从数组开头和结尾两侧分别向中心遍历像素点数据，若当前数据与相邻的之前数据不同，则说明当前数据是道路点，这样就能找到道路的左右边界left和right了。再取左右边界的中点，即（left+right）/2即得道路中线位置。

第二部分是通过中线位置来控制舵机转动从而使小车巡中线行驶

uint8\_t mid = FindMidLine();//找道路线中线

tempInt = mid\*10+850;

通过FindMidline（）函数找到中线mid，再将缩放后的mid赋值给tempInt，也就是用来控制舵机转动的变量，再用

Update\_ServoUS(*kFTM\_Chnl\_0*, tempInt);

Update\_ServoUS(*kFTM\_Chnl\_1*, 3000-tempInt);

这两个函数将变量tempInt传给舵机从而控制其转动。

最终实现的代码如下（除了上述函数以及定义的全局变量外，其余部分均在main（）函数中的case 1U中实现并调用）：

uint16\_t CCDDataGaus[128];

uint8\_t CCD2PC[260]; // data to be sent to PC

**volatile** **static** **float** my\_average=0;

**case** 1U: // using TSL1401

delay1ms(10);

//CollectCCD();

LinearCameraOneShot();

LDC();//镜头矫正

CCDGaus();//高斯滤波

Get\_01\_Value\_CCD(CCDDataGaus);//将括号中波形按均值二分成0和1

// draw linear CCD 128 pixel on OLED

Draw\_LinearView(CCDDataGaus);

**if** (!KEY1()) // Key S1 pressed down?

{

SendCCDData(CCDDataGaus);//向串口发送滤波后的波形

}

**if** (!KEY2()){

SendCCDData(CCDData);//发送原始波形

}

uint8\_t mid = FindMidLine();//找道路线中线

tempInt = mid\*10+850;

**char** txt[16];

**sprintf**(txt,"m:%03d", mid);

OLED\_P6x8Str(80, 1, (uint8\_t \*)txt);//将mid位置显示到OLED上

Update\_ServoUS(*kFTM\_Chnl\_0*, tempInt);

Update\_ServoUS(*kFTM\_Chnl\_1*, 3000-tempInt);

LED1\_TOGGLE(); //LED灯闪烁以示摄像头工作

**break**;

1. 动态曝光控制的代码

首先我们知道，曝光时间越长，进入光圈的光越多，则得到的图像更亮，反之图像更暗，则为了实现动态曝光控制以得到最好的图像，我们需要在环境（摄像头前）较亮时减小曝光时间，而在环境较暗时增大曝光时间。

在线阵摄像头实验中，我们需要通过程序来对曝光时间进行控制，具体时利用LinearCamerOneShot（）中的delay1ms(exposureTime)来控制曝光时间，而exposureTime是我们需要根据经验调试所得到的数据，环境越亮，exposureTime越小，反之越大。那么如何判断环境是较亮还是较暗呢？这一点可以根据我们之前所设置的全局变量average来进行控制，由二分法的函数求CCDDataGaus[]所有值的平均值average可知，average代表的是整体环境亮度数值的大小，average越大代表环境越亮，反之说明环境月暗，故可以通过这一变量的缩放去控制曝光时间的大小。

下面的函数中通过delay1ms（）和average的缩放来控制曝光时间。

**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 delay

uint16\_t exposureTime=10;

exposureTime=-my\_average/150+15;

delay1ms(exposureTime);

……

}

而exposureTime的计算是根据与average相关的my\_average变量进行缩放得到的，具体公式是根据多次调试后得出的，经测试，此动态曝光控制可以较好的适应电脑屏幕亮度变化的舵机控制。

1. 结果展示

（1）通过线阵摄像头进行识别路线从而控制舵机转动：

当路线（黑直线）在摄像头右侧时，舵机顺时针转动：

（图）

当路线（黑直线）在摄像头左侧时，舵机逆时针转动：

（图）

且转动角度与摄像头与直线的水平距离有关，距离越远，舵机转动角度越大。

（2）动态曝光控制：

当电脑屏幕亮度从低到高变化时，路线识别功能较好，通过OLED屏幕上的mid可以看出，大概范围均为20-100.

（两张图）

（3）存在的不足：

最明显的不足是，当摄像头离路线距离较远时，路线寻找会出现错误，即mid会变为常数64，从而使舵机不动，在实际巡线中可能将导致小车偏离路线后直接向更远的方向前进，从而失败。这一点可能是由于摄像头所拍摄的内容有限，当其捕捉到的像素点不存在路线时，则会使mid变为默认值64，这一点在后续实验中有待解决。

另外一点是实验的防抖功能不是很好，由于舵机转动需要根据mid来决定，而在OLED屏幕上我们可以明显看到mid的值会有较明显的抖动，尤其是摄像头角度、位置有改变（人手持时有抖动）时，其跳变可能会导致舵机转动的不稳定，这一点由于时间问题并不能很好的完善代码，但在后续中有很大希望可以解决。