

电子系统专题设计与制作
课程设计报告
“基于 74HC595 寄存器的全彩 RGB 灯
条控制及 pov 图案实现”

无 04
何佳齐、邵晨扬、唐钰凯

目录

电子系统专题设计与制作	1
课程设计报告	1
“基于 74HC595 寄存器的全彩 RGB 灯条控制及 pov 图案实现”	1
第一部分：设计构思和想法来源	3
1.1 启发&引子	3
1.2 寄存器与扫描	3
1.3 项目的难点所在	4
第二部分：电路设计和底层实现	4
2.1 项目所使用的原材料	4
2.2 电路设计方案	5
2.3 底层实现：	7
第三部分：74HC595 寄存器相关知识的学习与使用	7
第四部分：图像处理与转化（任意彩色图像转换成可识别编码）	8
4.1 图像颜色增强。	8
4.2 矩阵→极坐标展开。	10
4.3 极坐标展开后图像颜色的提取。	12
4.4 二进制编码转换	17
4.5 小结：	21
第五部分：卡尔曼滤波学习应用	22
第六部分：具体困难及解决方案	23
6.1 焊接困难：	23
6.2 调试困难	25
6.3 运行时间控制问题	27
6.4 Uno 板内存不足	27
第七部分：成品展示与实测效果	28
7.1 单个板的亮灯展示：	28
7.2 骑行时图片显示：	28
7.3 骑行时动图 gif 显示：	32
7.4 骑行时车速显示：	33
第八部分：调试代码与优化过程	34
8.1 单个板的亮灯代码：	34
8.2 图片显示代码：	42
8.3 动图 gif 显示代码：	46
8.4 车速显示代码：	48
8.5 利用缓冲器的时间优化-代码：	51
第九部分：参考资料	63

第一部分：设计构思和想法来源

1.1 启发&引子

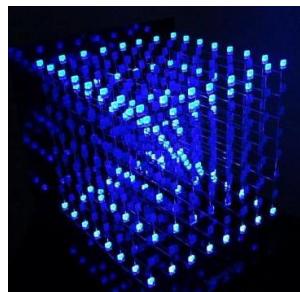
最初的启发是看到了视频网站上有人在自行车上安装了一种 RGB 灯条，可以在自行车以较高速行驶时显示图案，形成一种线动成面的效果。当时就觉得这个很有意思。这其实也就是我们所说的 POV (Persistence of vision)，以视觉暂留效应为基础实现的图像显示。理论上而言，利用 POV 可以实现图片的播放、gif 动图的显示，甚至小视频的播放。但是，通常的 POV 都是用 RGB 灯条来实现的。这种 RGB 灯条都是封装好的，在网上可以直接购买，使用起来比较方便，但是一个很大的缺点是自由度不高，不能够任意精准控制某个灯的 RGB 颜色，所以还是不太完善的，也不满足我们的要求。

于是我们就想，能不能自己制作一种 RGB 灯条，要能够同时精准控制这一个灯条上所有灯的 RGB 颜色，可以制作两个，或者三个，等角度安装在车轮上，在车轮以较高的速度旋转时，就可以实现在车轮上显示图片的效果。而当我们能够显示图片以后，我们就可以进一步来显示视频、gif 等动态的效果。只要在合适的时间更换需要显示的图片即可，就像视频的一帧一帧显示，连起来就有动感。

这就是我们的最初始的想法的来源。随着我们对其中机制了解的越发深入，接触到了很多限制和难点之后，我们这个项目也相应做出了调整。

1.2 寄存器与扫描

在我们调查、学习如何控制这一排 RGB 灯的时候，我们又看到了另外一种利用灯光来实现的作品，那就是 Light-Cube，也可以称为“光立方”。它大概的样子是这样的：



这是一个 $8 \times 8 \times 8$ 的正方体灯光点阵，512 个灯，但是最后连接到单片机上只有 8 根线，也就是说可以用 8 根线来控制这五百多个灯。我们察觉到这背后一定有我们能够利用上的原理。

经过资料的查询和探索，我们知道这背后的因由所在。它涉及到了一个非常重要的电路元件——移位寄存器 74HC595。以及两种非常重要的思想——扫描和 pwm。



寄存器的图片如上所示。利用这个寄存器就可以实现我们预期中的，对很多灯的精准控制。所以在我们的项目中，寄存器是非常重要的一环。我们后来的整个项目都是在寄存器的基础上来实现的。这一块我们将在后面的**环节 3** 中详细阐释。这里就简单提一下移位寄存器在我们整个项目中的重要性。

而扫描则是基于寄存器的一种传达信息的思想。在我们后来设计的 5 个共阴极中，一个共阴极上有 8 个灯，我们通过芯片去扫描这 5 个共阴极，这 5 个共阴极一次亮一个，当扫描足够块的时候，就像同时亮一样。这和电视机的显示的原理有异曲同工之妙。

而 **pwm** 的思想是这样的：我们在之前，都是想去控制输出给每个灯的电压大小，来控制灯的亮灭的程度。但是，我们这里采用 **pwm** 的思想。通俗地解释一下就是：我们在 0.1 秒中对所有灯轮流控制 10 次，每次只有亮或不亮两种情况，对应 0 或 1 的状态。那么如果灯 A 亮度应该很高，那么我们就在 10 次中每次都让它亮。如果灯 B 亮度不高，那么我们就在 10 次中，只让它亮 3 次。这样的话，当时间拉长，我们肉眼见到的情况就是灯的亮度有区别。所以我们只要这样去快速控制所有 RGB 灯的亮灭，就可以完成对所有灯亮度的控制了。

所以，关于寄存器的理解与使用是我们这个项目的核心所在。

1.3 项目的难点所在

首先，最大的困难就在于，我们没有任何现成的代码，在硬件这方面也没有可供参考的完备案例。这意味着我们需要从头开始学习所有的知识，自己去摸索和探索所有的功能，自己去解决所有的 bug。

其次，我们之前对于寄存器的知识了解甚少，专门讲解单片机与寄存器的课程也不多。学习的渠道很有限，难度也很大。

再者，我们没有任何现成的模块可以买来安装调试，我们需要从最底层的电路、布线开始设计，利用 PCB 板、RGB 灯、电阻、寄存器芯片，这些最基础的元件来组成电路，来完成我们自己的封装，以实现我们想要的所有的功能。

所以，如果用一个词语来描述我们这个项目的历程，那就是：从零开始。

第二部分：电路设计和底层实现

2.1 项目所使用的原材料

我们这个项目使用的原材料如下：

98×9（指的是板子上的焊孔数目）的空 PCB 板，板子上只有可供焊接的铜圈，没有任何事先的布线。



3 色全彩 RGB 灯。就是最普通的那种 4 管脚的那种。



移位寄存器 74HC595 芯片：这个是核心元件，它的管脚图是右边这样的：

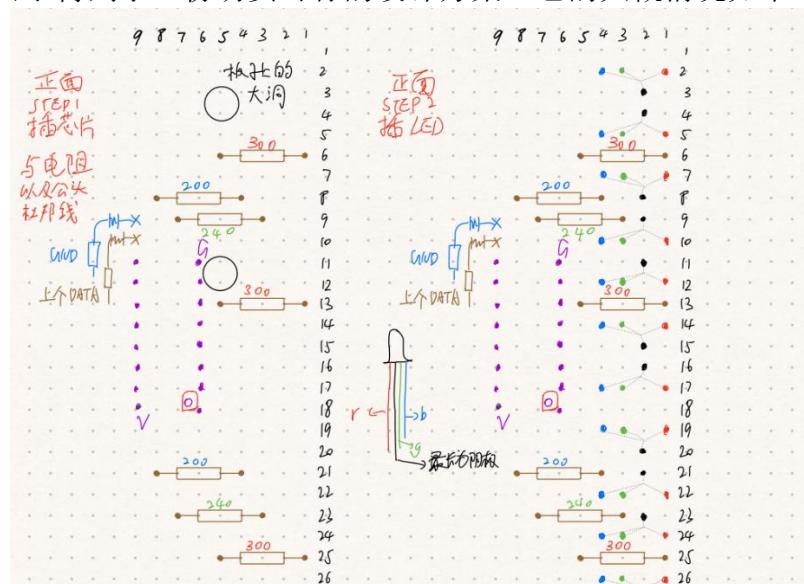


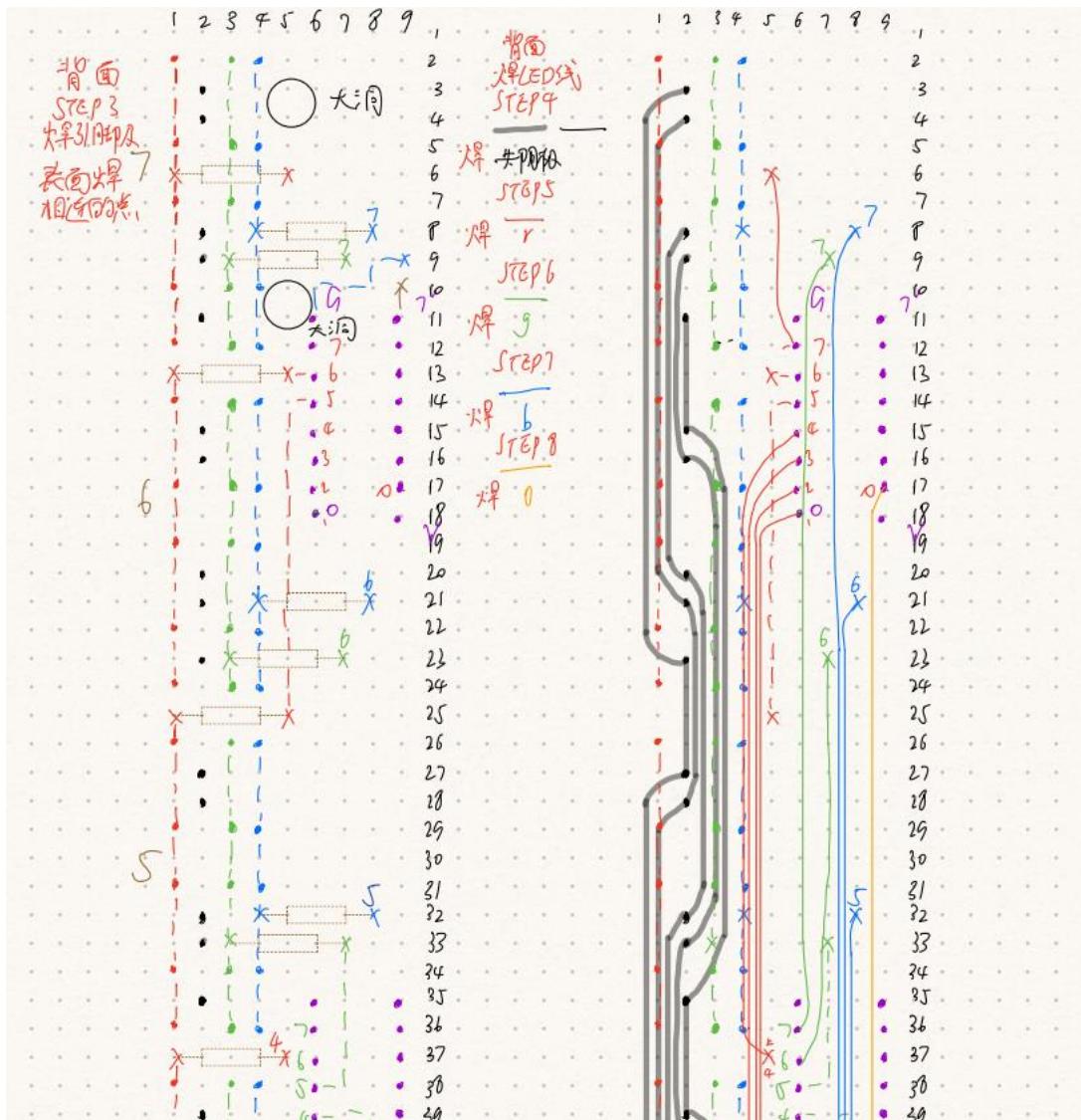
同时我们还利用了 MPU6050，这也是十分重要的元件。因为我们需要获取某一时刻某块 PCB 板子在车轮上的具体的角度的位置。

这些就是最重要的元件了，其他诸如杜邦线、不同阻值的电阻、焊台等我们使用的也很多。

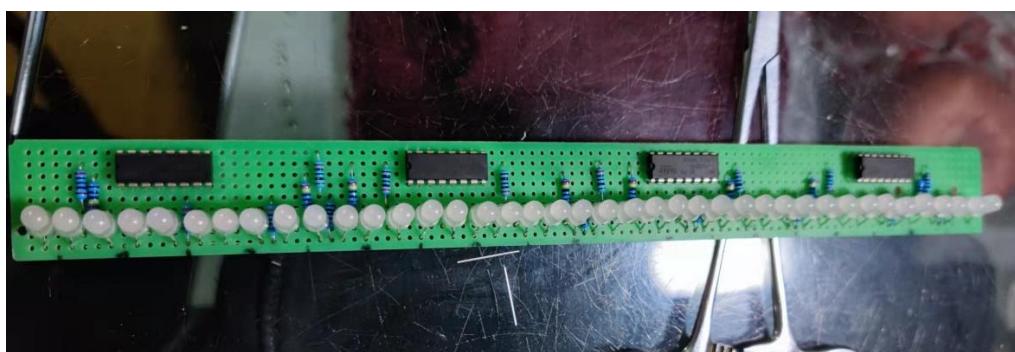
2.2 电路设计方案

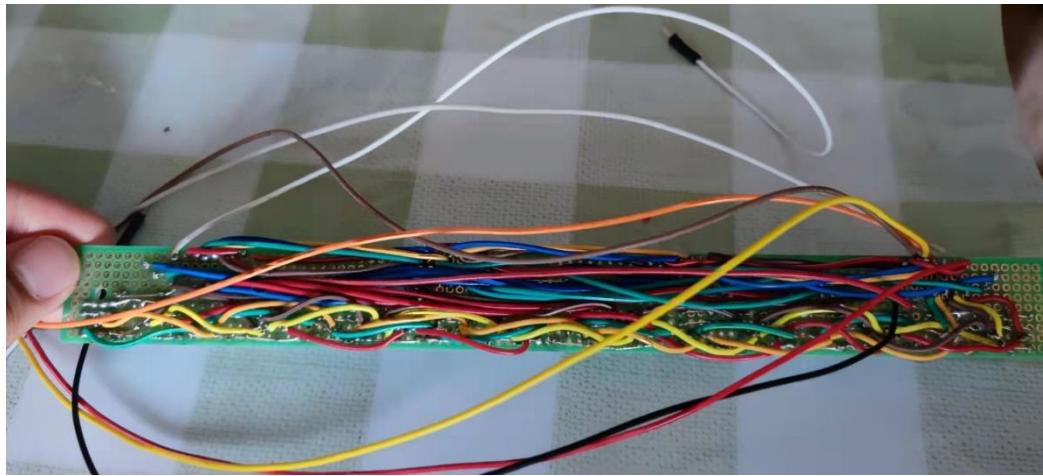
电路的设计方案可以说是我们项目最具有技术和创新性的方面之一，因为我们没有现成的封装，我们使用的都是最基础的电路元件，所以电路的设计存在相当大的困难。PCB 板的尺寸限制，RGB 灯三色电阻不均等，芯片工作的稳定性等因素，都对我们的设计造成了极大的困难。在设计的途中，有时因为焊接的困难，有时因为电路的错误，我们一次次优化，一次次修改，甚至经历了一次次推翻，最终终于得到了一份切实可行的设计方案：它的大概情况如下：





以上的这两幅图是我们最终设计方案的一部分截图，完整的方案要比这个大很多，也作为附件一起提交。通过这两幅图可以知道我们的大概的电路的构思，以及具体的布线的方法。这也是非常重要的。尤其是电路的布线，在这么一块长约 30 公分，宽不足 3 公分的板子上完成所有元件之间的布线其实是一件非常难的事情。不仅要考虑到焊接的难度，还要考虑到连接是否准确合理。最终我们顺利完成了电路的设计与布线工作。效果是这样的：





同时，焊接还是一个极富技术性、极其困难、极耗时间的操作，我们最终一共焊接了 4 块板子，能够正常使用的只有 3 个，这几乎耗费了我们小组 2 个组员整个 8 月份下午的时间。关于焊接，我们会在后面的第 6 部分再具体说明。

2.3 底层实现：

电路设计是和这个项目的底层实现密不可分的。我们的底层的设计大概是这样的：40 个灯，先是分成 8 组，相邻的每 5 个组成一组，这 5 个的 RGB 管脚是并联在一起的，也就是说，在阴极都为低电平的情况下，这 5 个灯的红色、绿色、蓝色是一起亮或灭的。然后，这 8 组中的每组的第一个组成一个新的小组，8 组中的每组的第二个又组成一个新的小组，然后，8 组中的每组的第三个又组成一个新的小组，以此类推，就分成了新的 5 组，每组的 8 个灯连成一个共阴极。换句话说，在这 8 个灯的红色管脚都是高电平的实现下，这 8 个灯是随着共阴极的高低一齐暗或者亮的。

也就是说，我们控制一个灯的亮灭有两种方法，一个是控制他的 RGB 管脚的高低电位，一个是控制他的阴极的高低电位。其中 RGB 管脚的高低电位是通过 3 个芯片去分别控制的，然后共阴极的效果是通过第 4 个芯片去控制的。至于具体如何通过芯片来控制，这会在下一个部分来详细解释了。这其中会涉及到扫描、锁存等相关知识。

第三部分：74HC595 寄存器相关知识的学习与使用

在我们的作品中，主要使用的是 74HC595 芯片。这是一种硅结构的 CMOS 器件，74HC595 的功能简单说来就是它具有 8 位移位寄存器和一个存储器，以及三态输出功能。所谓的三态输出功能，也就是具有高电平、低电平和高阻抗三种输出状态。而且输出寄存器是可以直接清除的。

至于为什么使用 74HC595 芯片，在前面也作出了解释，那就是为了节省 I/O 口，否则，我们后来一个车轮上装三个 PCB 板子，一块板子 40 个灯，一个灯 4 个引脚，一共就是 480 个引脚，这么多引脚是难以想象的。所以我们打算使用 74HC595 寄存器来实现对于数量如此庞大引脚的控制。我们通过查看 74HC595 芯片的手册结合上网搜索相关术语的解释，弄清楚了 74HC595 芯片各个引脚的作用，从而为之后设计整体 LED 控制电路打下了基础。

74HC595 芯片的一个好处是可以实现多个芯片的相互级联，采用的是 SPI 通

信方式，通过芯片的 14 引脚与 9 引脚可以实现不同芯片之间的信号传输。而 11 和 12 引脚是另外两个重要的引脚，其中 11 引脚是 SHIFT CLOCK，通过改变此引脚的电位可以向芯片传输是否进行移位的信号。而 13 引脚是 LATCH CLOCK，通过改变此引脚的电位可以向芯片传递是否锁存的信号。总的来说，每次传输都是需要移位+锁存，才能最终显示出我们想看到的 LED 显示。74HC595 芯片的 13 引脚是输出使能引脚，10 引脚是 RESET 引脚，我们都置为高电平，允许输出的同时也防止重置。

另一个我们使用的寄存器是 Arduino UNO 板上的 Timer1, Timer2。通过查阅相关的资料与手册，我们也学习了 TCCR_A, TCCR_B, OCR_A, OCR_B, TIMSK 等等寄存器，学会了如何从最底层激活、设置一个 Timer 的定时器中断。同时也了解到为什么使用 Timer1 的时候，7、8 引脚不能再产生 PWM 波的输出。同时我们也对于整块 UNO 板的时间分频有了更加深刻的理解，这对于我们之后在进行 SPI 通信时的时钟线的设置有着很强的指导意义。

第四部分:图像处理与转化（任意彩色图像转换成可识别编码）

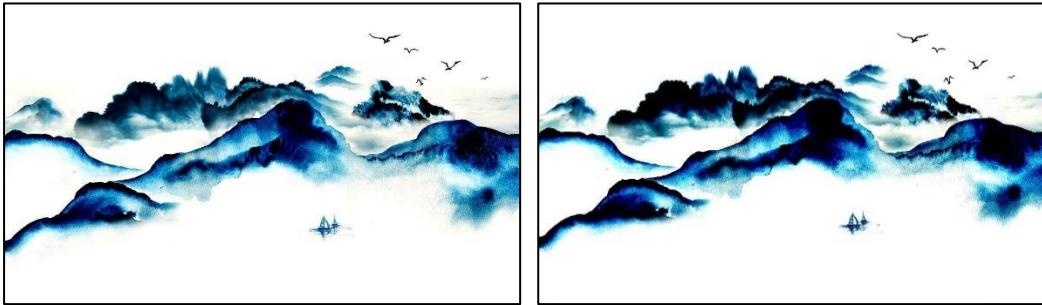
这个环节的工作，我们是采用 python 和 cpp 两种语言去综合实现的。其实本来我们是计划通过蓝牙把图片传输到单片机里，然后在单片机里自动完成编码的转换再在灯上表现出来。但是后来考虑到单片机的内存以及执行效率等等问题，所以我们就只能采取在烧录程序之前，通过 python 和 cpp 程序先把图片转换成编码，然后再把编码放进 arduino 程序里，最后再把 arduino 程序烧录进板子里面。这样确实复杂了一点，但是优点是保证了执行的效率。但凡是一张图片，都能通过这种方式显示在车轮上。（只是清晰度的问题而已）

这个环节是一个十分复杂的环节，它大概可以拆分成这样几个小的步骤：

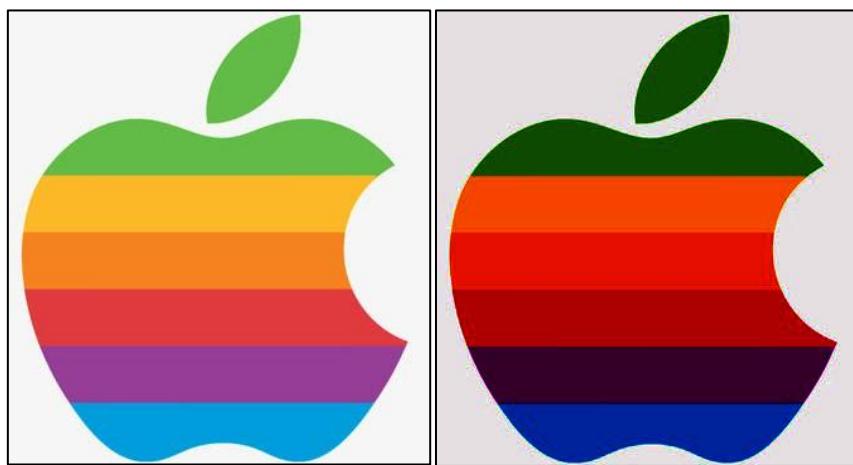
4.1 图像颜色增强。

由于我们最终是通过 40 个 RGB 灯珠（就是原始的那种灯）来实现颜色的显示的。而 RGB 的颜色显示效果并不很好，颜色偏淡，所以我们在所有的操作之前，先进行了一波图像颜色的增强，曝光度、对比度、色阶等参数和效果也会进行调整。这既可以在 ps 中实现，也可以通过简单的 python 代码实现：效果是这样的：





这个是对同一张图片进行了四次色彩加强的结果，可以见得，效果还是很不粗的。把颜色之间的差距拉开了。下面是对苹果这一张图片进行增强的结果。



这里给出色彩增强程序的 python 源代码：

```
# -*- coding: UTF-8 -*-
import os
from PIL import Image
from PIL import ImageEnhance

# 原始图像
def ImageAugument():
    path = r'./pic'  # 文件夹目录
    files = os.listdir(path)  # 得到文件夹下的所有文件名称
    # 遍历文件夹
    prefix = path + '/'
    for file in files:
        # print(file)
        image = Image.open(prefix + file)
        # image.show()

        # # 亮度增强
        # enh_bri = ImageEnhance.Brightness(image)
        # brightness = 1.5
        # image_brightened = enh_bri.enhance(brightness)
```

```

# image_brightened.save(prefix + file.strip('.jpg') + '-lightup' + '.jpg')

# 色度增强
enh_col = ImageEnhance.Color(image)
color = 1.5
image_colored = enh_col.enhance(color)
image_colored.save(prefix + file.strip('.jpg') + '-colorup' + '.jpg')

# 对比度增强
enh_con = ImageEnhance.Contrast(image)
contrast = 1.5
image_contrasted = enh_con.enhance(contrast)
image_contrasted.save(prefix + file.strip('.jpg') + '-contrastup' + '.jpg')

# # 锐度增强
# enh_sharpen = ImageEnhance.Sharpness(image)
# sharpness = 3.0
# image_sharpened = enh_sharpen.enhance(sharpness)
# image_sharpened.save(prefix + file.strip('.jpg') + '-moreSharp' + '.jpg')

if __name__ == '__main__':
    ImageAugument()

```

4.2 矩阵→极坐标展开。

因为我们是通过“线动成面”的方式进行图像的显示的，而这条线其实就相当于车轮这个圆形区域的一条半径，我们需要一个类似于极坐标系的 RGB 值数据，来指引这一条半径上的某个点在某个位置应该亮哪个灯。但是我们正常去拆解、获取一张图片的 RGB 值获得的都是一个三维的矩阵，也就是说，这是一个 xyz 坐标的矩阵，所以我们就要想办法把 xyz 坐标转换成极坐标。转换的过程还得在一定程度上保证图片的颜色不失真。

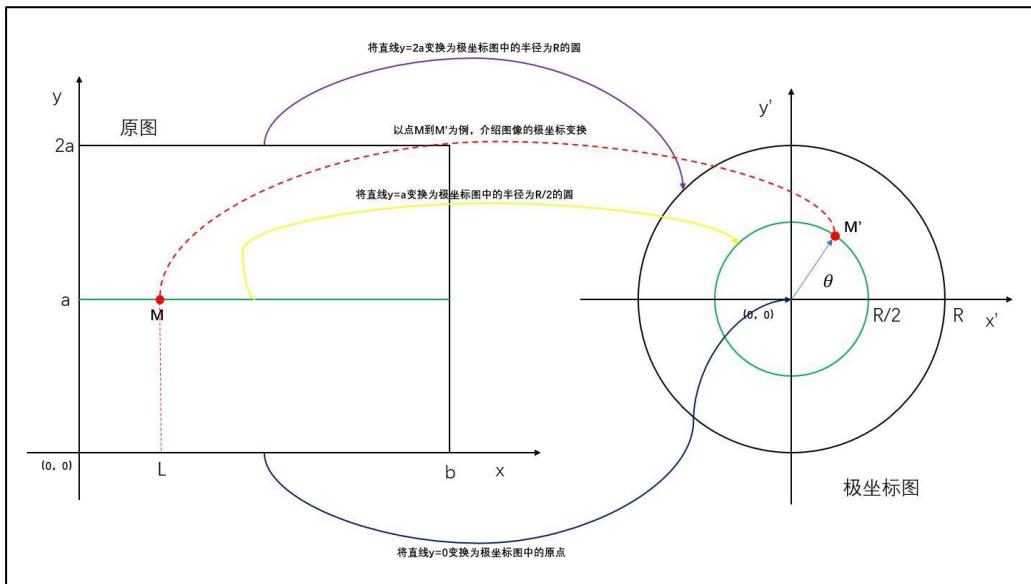


在本质上是一个坐标系变换的问题，但是换个角度思考，我们可以这样实现：我们假设把任意一张图片放在这个圆形的车轮上，然后把一根“半径”（其实是我们的 PCB 板子）绕着圆心绕一圈，每隔相等的角度就记录一下这个“半

径”上 40 个灯的 RGB 数值，这样就可以比较简单地记录这个“半径”上的各点的颜色。但这里会涉及到很多细节的问题，比如实际上这样的做法中，内层的颜色采集是比外层要密集的，这样的话，再转换成 RGB 数值就会导致图像的失真；再比如，图片的边界的选择，在转换过程中的边界问题应该如何解决才能尽可能还原本来的图片的模样。

还有一种思路就是，考虑一种类似全景展开的方式，一个圆环一个圆环地展开一个个细而长的矩形，最终再把一个个矩形拼接起来就可以了。这也可以叫做极坐标全景图变换。

最后，我们综合了两种方法的思路，完成了预想的功能。这段代码的大概思路如下：



已知原图中每个点（M）的灰度值及 RGB 值，通过极坐标系与平面直角坐标系的坐标转换公式，确定极坐标图中每个点（M'）处的灰度值及 RGB 值。

通过对上图中点 M 的坐标变换进行讲解，如图所示：原图中 M 的坐标为： (L, a) ，在极坐标图中 M' 的坐标为： (θ, r) 。

步骤 1：在极坐标图中 M' 的极坐标定位：

$$\theta = 2 * \pi * M \text{ 的横坐标} / \text{原图像的宽} = 2 * \pi * L / b;$$

$r = \text{半径缩放系数} * M \text{ 的纵坐标} = \rho * a$; 其中： $\rho = \text{极坐标图的 R} / \text{原图像的高} = R / (2 * a)$;

步骤 2：将极坐标图中的极坐标，转换为极坐标图中的平面直角坐标

$$x' = r * \cos\theta;$$

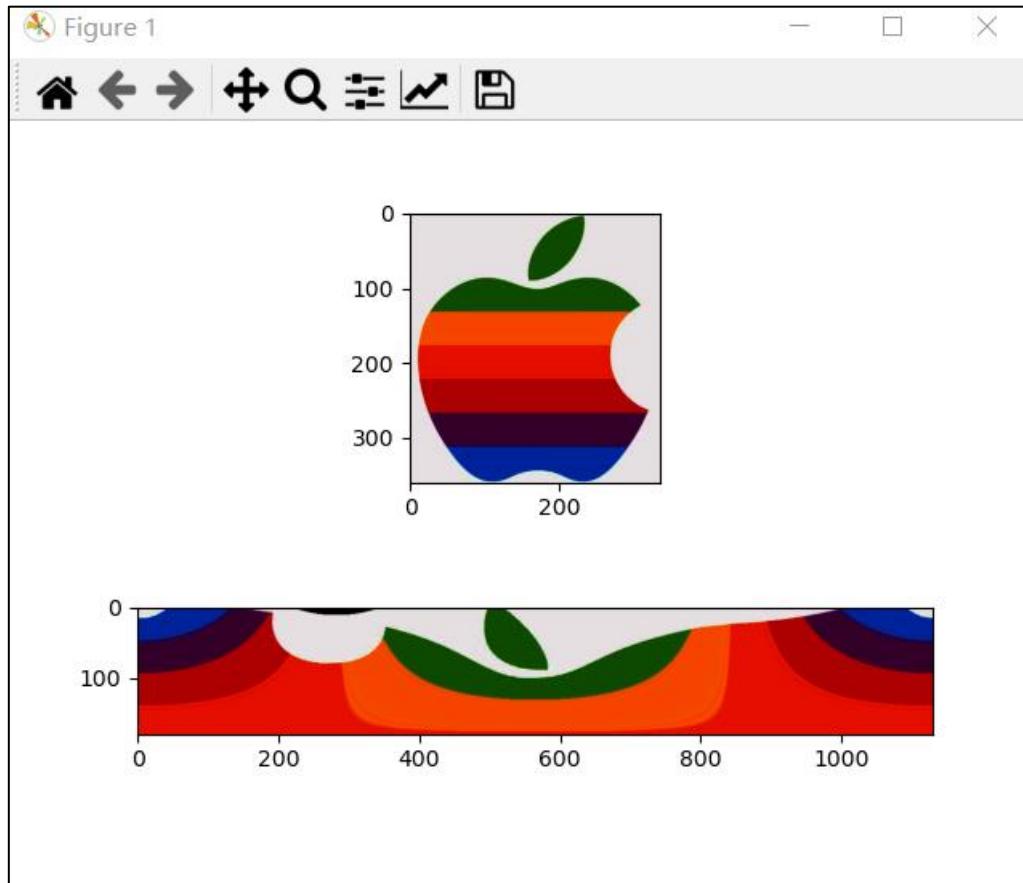
$$y' = r * \sin\theta;$$

步骤 3：将 M 处的灰度值、RGB 值，赋给 M' 处像素点的灰度值、RGB 值。

其实理论上，到这里也就结束了，但是这样做的话还有一点小缺陷，那就是：从原图像到极坐标图像的转换中，并非所有的点均可以一一对应，这个时候就需要对没有对应原图像中像素点的极坐标图中的点，进行插值操作，不然的话，显示的图片就像是一块完整的布被抽掉了很多丝一样，显示得很不好。

所以为了进一步提升效果，我们还加入了图像的差值操作以及图像的 Alpha 融合技术。由于篇幅的限制，这里就不再具体介绍了。

最终代码的效果如下：

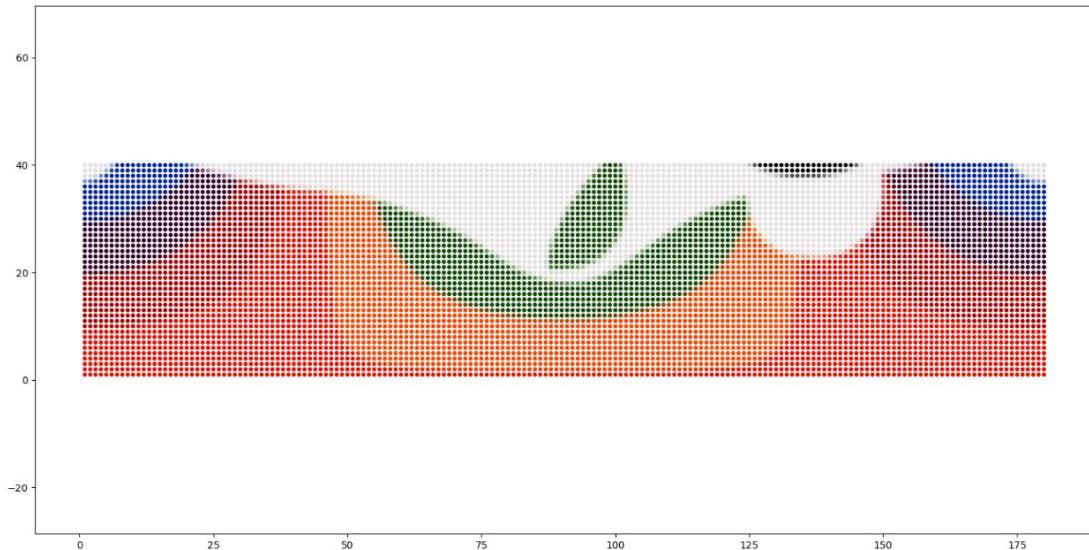


可以看出最后的效果就是把这个色彩增强后的苹果，以图片的中心为基准，按照一条条半径的方向进行了一个展开。可以注意到，图片还是相当连续的，而且左右两条边上的颜色是完全一样的，所以没有问题。

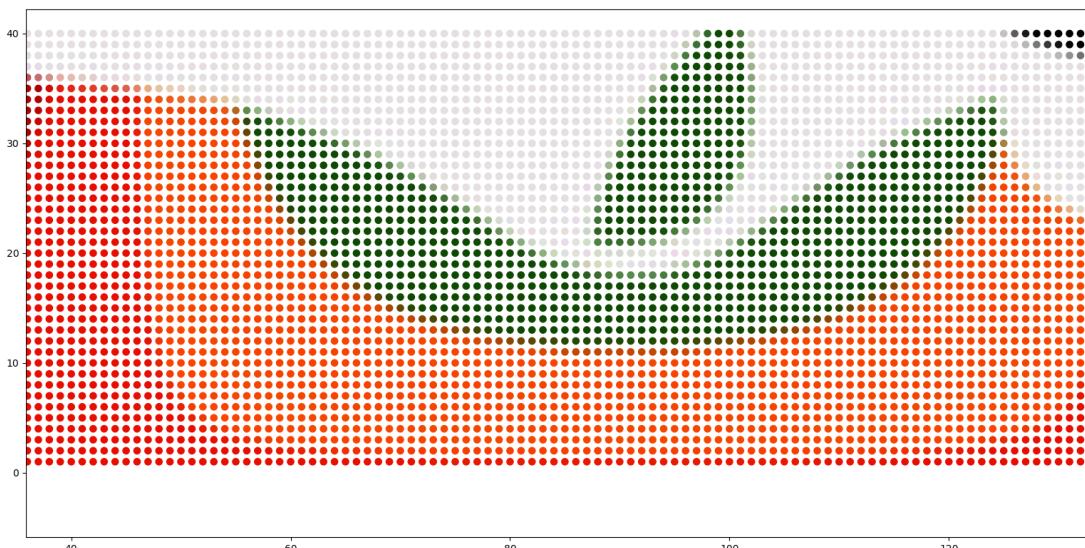
那么这一步就完美解决了。(4.3-4.4 的代码都是综合在一起，在 py 程序中进行展示)

4.3 极坐标展开后图像颜色的提取。

我们最后一条“半径”上只有 40 个灯，但是在上面的那个图中，一列的像素值显然不止 40，所以得从中提取 40 个值出来。我们尝试了很多先进的算法，但是效果都差不多。最后干脆就用了一种最简单的改变图片像素值大小的方法。经过提取后，我们就可以获得我们所需要的 40 个灯的全部的 RGB 值。为了展示得更加清楚，以及比较好的展现提取的效果，我们特地编写了一个程序，用点阵的形式来模拟最后的效果，效果如下：



放大看的效果就是这样：



可见效果还是非常好的。还是与原图非常接近的。

而在做到这一步的时候，我们也已经得到了所有的最终我们需要的 RGB 数值。RGB 值的具体形式将在 4.4 中展示。所以从 4.1-4.3 实际上完成的是所有关于图像的操作。4.2-4.3 的代码如下：

```

import numpy as np # 一堆库
import math
import cv2
import matplotlib.pyplot as plt
import torch
import numpy as np
from torchvision import datasets, transforms
from PIL import Image, ImageDraw
from matplotlib.patches import Circle
import os

```

```

os.environ["KMP_DUPLICATE_LIB_OK"] = "TRUE"

imgName = "apple2.png" # 这是需要处理的图片的名字
imgProName = "apple2-pro.jpg" # 这是作极坐标展开处理之后的图片的名字
img_first = imgProName # 之后的图像名字都是自动生成
img_second = img_first.replace("-pro", "-pro-min", 1) # 这是做尺寸转换之后的图片的名字
img_third = img_first.replace("-pro", "-final", 1) # 这是做最终点阵替代之后的图片的名字

img = cv2.imread(imgName) #
# CV2 库读入 numpy 数组, uint8 类型, 0-255 范围, 图像形状 (H,W,C) , 读入的顺序是 BGR
# cv2.imshow(imgName, img)

# 获取展开的参数
# 得到圆形区域的中心坐标
x0 = img.shape[0] // 2
y0 = img.shape[1] // 2
# 通过圆形区域半径, 构造展开后的图像
unwrapped_height = radius = img.shape[0] // 2
unwrapped_width = int(2 * math.pi * radius)
unwrapped_img = np.zeros((unwrapped_height, unwrapped_width, 3), dtype="u1") # u1
无符号整形成

# 具体的图像转换函数
# 由于在展开过程中难免遇到超出原图边界的情况发生, 因此加入了捕获异常的代码。
except_count = 0
for j in range(unwrapped_width):
    theta = 2 * math.pi * (j / unwrapped_width) # 1. 也可通过添加额外的选项调节起点
    # j / unwrapped_width 是每一步占圆周 2pi 的比例
    for i in range(unwrapped_height):
        unwrapped_radius = radius - i # 2. 展开过程中对于每一步的 j, r 都是一个从外向内
        动态变化的参数
        x = unwrapped_radius * math.cos(theta) + x0 # 3. sin 顺时针, cos 逆时针, 两者
        互换会有差别?
        y = unwrapped_radius * math.sin(theta) + y0
        x, y = int(x), int(y)
        try:
            unwrapped_img[i, j, :] = img[x, y, :]
        except Exception as e:
            except_count = except_count + 1
print(except_count)

cv2.imwrite(imgProName, unwrapped_img) # imwrite 用来保存图像

```

```

# cv2.imshow("Unwrapped", unwrapped_img)
# cv2.waitKey(0)
# 事实上, 到这里图像的处理已经结束了, 接下来使用 plt 来展示图像

for j in range(unwrapped_width):
    theta = 2 * math.pi * (j / unwrapped_width) # no offset
    for i in range(unwrapped_height):
        unwrapped_radius = radius - i
        x = unwrapped_radius * math.cos(theta) + x0 # "cos" 逆时针
        y = unwrapped_radius * math.sin(theta) + y0
        x, y = int(x), int(y)
        try:
            unwrapped_img[i, j, :] = img[x, y, :]
        except:
            continue

plt.subplot(2, 1, 1)
plt.imshow(img[:, :, ::-1])
plt.subplot(2, 1, 2)
plt.imshow(unwrapped_img[:, :, ::-1])
plt.show()

# 这一段的功能和上一段差不多, 只是显示出的图像的顺时针和逆时针的差别
# for j in range(unwrapped_width):
#     theta = 2 * math.pi * (j / unwrapped_width) - 1 / 2 * math.pi # + math.pi
#     for i in range(unwrapped_height):
#         unwrapped_radius = radius - i
#         x = unwrapped_radius * math.sin(theta) + x0 # "sin" is clockwise
#         y = unwrapped_radius * math.cos(theta) + y0
#         x, y = int(x), int(y)
#         try:
#             unwrapped_img[i, j, :] = img[x, y, :]
#         except:
#             continue
#     #
#     plt.subplot(2, 1, 1);
#     plt.imshow(img[:, :, ::-1])
#     plt.subplot(2, 1, 2);
#     plt.imshow(unwrapped_img[:, :, ::-1])
#     plt.show()

img = Image.open(img_first)
resize = transforms.Resize([40, 180])
img = resize(img)

```

```



```

```

# 开始画一个一个小圆
for i in range(180):
    for j in range(40):
        circle = Circle(xy=(180 - i, 40 - j), radius=0.3, alpha=1, color=(x[i, j],
y[i, j], z[i, j]))
        ax.add_patch(circle)

plt.axis('equal')
plt.savefig(img_third)
plt.show()

```

4.4 二进制编码转换

到此为止对于图片的操作已经结束了。接下来就是根据所提取到的 RGB 数值来转化成相应的能够被寄存器识别的编码。

首先展示一下经过上述的图片相关操作，我们提取到的 RGB 数值：

	try.py	img_RGB2.txt	4.3.png
1		178 29 31	
2		234 195 196	
3		244 245 247	
4		245 246 245	
5		247 246 246	
6		249 245 247	
7		245 247 250	
8		244 248 248	
9		246 247 245	
10		233 201 199	
11		189 83 85	
12		245 227 231	
13		223 158 160	
14		185 9 10	
15		187 1 3	
16		187 2 2	
17		188 1 2	
18		187 1 2	
19		187 1 2	
20		187 1 2	
21		188 1 2	
22		189 1 2	
23		190 1 3	
24		192 1 3	
25		191 2 5	
26		189 40 45	
27		236 219 219	

所有的 RGB 数据被存储在一个文档中，一共 7200 行。因为我们的 PCB 板子实际上是每 2° 为一个角度单位，也就是每 2° 换一下颜色，这样就有 $360/2=180$ 个角度位置，每个位置有 40 个灯。所以这样就应该有 $180*40=7200$ 个

灯的 RGB 数值信息。接下来就要把这 7200 个灯的 RGB 数值转化成编码。

转成编码这一块我是用 cpp 程序实现的。

这一块的困难在于编码的格式是非常特殊的。在我们的项目中，一个灯的 RGB 数值都是以一个四位二进制数字来存储的。之所以采用这样的形式，也是由于我们最终是要通过寄存器来实现用 4 根线（不包括 vcc、gnd 以及 MPU 模块的线）来精准控制 3 块板的 120 个 RGB 灯。【否则，将会需要 360 根线才能精准控制这 120 个 RGB 灯的三个管脚】

下面来具体描述一下最后 arduino 程序中的 RGB 的存储方式，下面先来看一下它的形式：

下面来做一些解释：所有的数据被存储在 byte 数据类型中。这是一个大小为 10800 的 byte 类型数组。所有的元素都是以“B11110110”这样的形式去存储的。10800 分成 180 份，每份 60 个。180 相当于每 2° 分隔后的 180 个位置，而 60 则是每个位置上 40 个灯的 RGB 信息。这 60 又划分成 20×3 。也就是 R、G、B 每种颜色 20 个元素。那么这 20 个元素又是如何记录一种颜色的 40 个灯的亮灭的呢。是这样的，从之前的介绍中，我们可以知道，40 个灯的阴极被分为 5 组，每组是有 8 个，这 8 个灯的阴极被连在一起，而 40 个灯的 R、G、B 管脚又各自被分为 8 组，每组 5 个。通过这两种分组，就可以实现精准的控制。而每种颜色的 20 个 byte 类型数据，则是这样起作用的：

前提是：一个灯的 RGB 数值都是以一个四位二进制数字来存储。然后 20 个 byte 类型被分成 4 组，每组 5 个。一个 byte 中有 8 位 0、1 数据，每组的 5 个就有 40 位。这 40 位则记录了所有 40 个灯的这一种颜色的 RGB 数值的某一位。所以，举个例子，你想要知道第一个灯的 R 管脚应该输出多大的电压，实际上你应该找到这 20 个 byte 类型中的，第 1 个的第一位，第 6 个的第一位，第 11 个的第一位，第 16 个的第一位。拼起来是一个 4 位二进制数值，也即管脚对应的输出电压的大小。（实际上这里输出电压的大小也是一种大约的说法，我们不是通过直接控制电压来控制亮度的，而是利用扫描的办法，通过极短时间内扫描的次数来控制灯的某个亮度值。效果就和控制电压是一样的。实际上，我们也没办法利用 arduino 板来控制 120 个 RGB 灯的三个管脚的串压值。）

由于四位二进制数的上限是 16，而图片中提取出的颜色值是 0-255，所以在这个转换程序中，先得把原来的 RGB 数值映射到 0-16 中，再转换成二进制，再按照规则分别拆解重组，形成最后的 10800 个 byte 的数据信息。

有了这个之后，我们只需要把这个 10800 个 byte 的数据信息放在 arduino 程

序中，直接烧录即可。

整个转换程序的源代码如下：

```
#include<iostream>
#include<fstream>
#include<sstream>
constexpr auto N = 90;
using namespace std;

int arr[180][40][3] = { 0 };

char red0[40];
char red1[40];
char red2[40];
char red3[40];

string ss[30] = { "" };

//180*40*3
//180*60
//180*3*20
//20:red0,red1,red2,red3
//180*3*4*5

int getII(int n,int loc)
{
    int k;
    k = n;
    switch (loc)
    {
        case 0: return k % 2;
        case 1: k = k / 2;    return k % 2;
        case 2: k = k / 4;    return k % 2;
        case 3: k = k / 8;    return k % 2;
    }
}

char nstr(int i)
{
    if (i == 0) return '0';
    if (i == 1) return '1';
}

int main()
```

```

{

cout << "hh";
ifstream infile;

infile.open("img_RGB2.txt", ios::in);
cout<<infile.is_open()<<endl;

int n,i,j,k;
for (i=0; i<180; i++)
    for (j=0;j<40;j++)
        for (k = 0; k < 3; k++)
    {
        infile >> n;
        arr[i][j][k] = ceil(n / 16);
    }
int l;
infile.close();

int m, p;
ofstream outfile;
outfile.open("img_RGB_pro.txt", ios::out);
//90-120
for (i = 0; i < 180; i++)
    for (k = 0; k < 3; k++)
    {
        for (j = 0; j < 40; j++)
        {
            //0-4 1-9 2-14 3-19 4-32      (i%5)*8+i/5
            //5-1 6-9 7-17 8-25 9-33
            //10-2 11-10 12-18 13-26 14-34

            red0[(j % 5) * 8 + j / 5] = nstr(getII(arr[i][j][k], 0));
            red1[(j % 5) * 8 + j / 5] = nstr(getII(arr[i][j][k], 1));
            red2[(j % 5) * 8 + j / 5] = nstr(getII(arr[i][j][k], 2));
            red3[(j % 5) * 8 + j / 5] = nstr(getII(arr[i][j][k], 3));

        }
        for (m = 4; m >= 0; m--)
        {
            outfile << "B";
            for (n = 0; n < 8; n++)
                outfile << red0[8*m+n];
            outfile << ",";
        }
    }
}

```

```

    }
    for (m = 4; m >= 0; m--)
    {
        outfile << "B";
        for (n = 0; n < 8; n++)
            outfile << red1[8 * m + n];
        outfile << ",";
    }
    for (m = 4; m >= 0; m--)
    {
        outfile << "B";
        for (n = 0; n < 8; n++)
            outfile << red2[8 * m + n];
        outfile << ",";
    }
    for (m = 4; m >= 0; m--)
    {
        outfile << "B";
        for (n = 0; n < 8; n++)
            outfile << red3[8 * m + n];
        outfile << ",";
    }

}

outfile.close();
return 0;
}

```

4.5 小结:

综合以上 4 个小步骤。我们在图像处理这一块需要做的工作顺次如下：

第一步：是把图像通过 py 程序进行色彩增强。（或 ps 手动调整）

第二步：是把色彩增强后的程序放在 py 程序中进行极坐标全景展开及 RGB 数值提取。

第三步：把提取之后的 RGB 数值放到 cpp 程序中，转换成后续 arduino 程序所需要的 10800 的 byte 数组。

第四步，也就是把数组拷贝进 arduino 程序中进行烧录即可。

第五部分：卡尔曼滤波学习应用

我们使用 MPU6050 测量 z 轴的绝对角度，同时采用 Kalman 滤波的技术来对所获得的误差较大的角度进行更一步精确，使得我们的程序在识别角度方面能表现得更好。

首先，我们查找了和 MPU6050 使用相关的文档，网上相关的资料鱼龙混杂，而且大部分是运用于无人机等具体案例的代码，缺少只测量具体方向角度的代码。接着，我们研究了一下 MPU6050 中的寄存器的分布与使用，探究出了如何从 MPU6050 中通过位运算等操作来提取角度。找到测量具体方向的角度的方法后，我们进行了代码尝试，又发现测量出的角度值并非绝对的角度制或者弧度制的值，还需要做进一步的转换。我们在尝试了网上许多乘除一定的比例系数的代码后，结果仍然不尽人意。因此我们想到可以使用 map 函数来进行线性映射把测量得出的值最终映射到 0°-360° 的区间。

接着，我们开始探索利用 Kalman 滤波的技术对从 MPU6050 获取的角度进行更进一步地减小误差，使得整个程序能进行的更加完美。关于 Kalman 滤波这一项技术，虽然在网上进行搜索可以得到很多的资料，但是要不就是代码是 Kalman 滤波在一些具体案例，如姿态解算的应用，我们不知所云，要不就是把代码复制下来发现根本运行不了。我们尝试调用 Kalman 相关的库，去 Github 上看了很久源码也不知道具体应该如何使用，同时又缺少清晰的说明文档，我们寸步难行。最后我们决定学习 Kalman 滤波的具体数学原理，自己从底层来写 Arduino 的代码来实现这一功能。我们在看了大量的视频后才明白 Kalman 滤波整个算法背后的本质思想：利用测量值和预估值二者的加权来得到一个相对来说更加准确的值。于是我们自己利用这样的思想和其背后严谨的数学原理自己写了一个 Arduino 的代码来实现角度的更精确的解算，运用于我们整体的项目中，使得整体的成果在测量角度方面有更加好的表现。

基于卡尔曼滤波的.MPU6050 调试的代码如下：

```
#include<Wire.h>
const int MPU_addr=0x68;
int16_t AcX,AcY,AcZ;
int minVal=265,maxVal=402;
double z;
double err_est[6];
double err_mea=2;
double x_est[6];
double kalman_gain[6];

void setup(){
  Wire.begin();
  Wire.beginTransmission(MPU_addr);
  Wire.write(0x6B);
  Wire.write(0);
  Wire.endTransmission(true);
  Serial.begin(9600);
}

void loop(){
  Wire.beginTransmission(MPU_addr);
  Wire.write(0x3B);
  Wire.endTransmission(false);
  Wire.requestFrom(MPU_addr,14,true);

  AcX=Wire.read()<<8|Wire.read();
```

```

AcY=Wire.read()<<8|Wire.read();
AcZ=Wire.read()<<8|Wire.read();

int xAng = map(AcX,minVal,maxVal,-90,90);
int yAng = map(AcY,minVal,maxVal,-90,90);

x_est[0] = RAD_TO_DEG * (atan2(-yAng,-xAng)+PI);
err_est[0]=2;
for(int i=1,i<6;i++){
    Wire.beginTransmission(MPU_addr);
    Wire.write(0x3B);
    Wire.endTransmission(false);
    Wire.requestFrom(MPU_addr,14,true);

    AcX=Wire.read()<<8|Wire.read();
    AcY=Wire.read()<<8|Wire.read();
    AcZ=Wire.read()<<8|Wire.read();

    int xAng = map(AcX,minVal,maxVal,-90,90);
    int yAng = map(AcY,minVal,maxVal,-90,90);

    z = RAD_TO_DEG * (atan2(-yAng,-xAng)+PI);

    kalman_gain[i]=err_est[i-1]/(err_est[i-1]+err_mea);
    x_est[i]=x_est[i-1]+kalman_gain[i]*(z-x_est[i-1]);
    err_est[i]=(1-kalman_gain[i])*err_est[i-1];
}
Serial.print("Angle Z=");
Serial.print(x_est[5]);
Serial.print("\n");

delay(100);
}

```

第六部分：具体困难及解决方案

如果用一个词语来概括我们项目的完成过程的话，那就是“千难万险”，用“千”和“万”也许有些夸张，但是“一波三折”也绝对不够用来形容。从7月末正式开始设计到9月初完成最后的作品，这其间几乎每天都有新的bug出现，也因此几乎每天都在debug的过程中。回顾这两个月的debug历程，大概有一半的bug通过查阅资料或者自己探索研究被顺利解决，但另一半的bug则是束手无策，即使我们穷尽手头上的所有资料也于事无补，但这并没有使得我们气馁或放弃，反而迫使我们去主动优化现有的方案，把所有可能存在风险和错误的地方全部避开、清除，甚至推翻现有的全部方法重新开始，使得我们的项目越来越趋于完善，不断努力，向成功的方向前进。

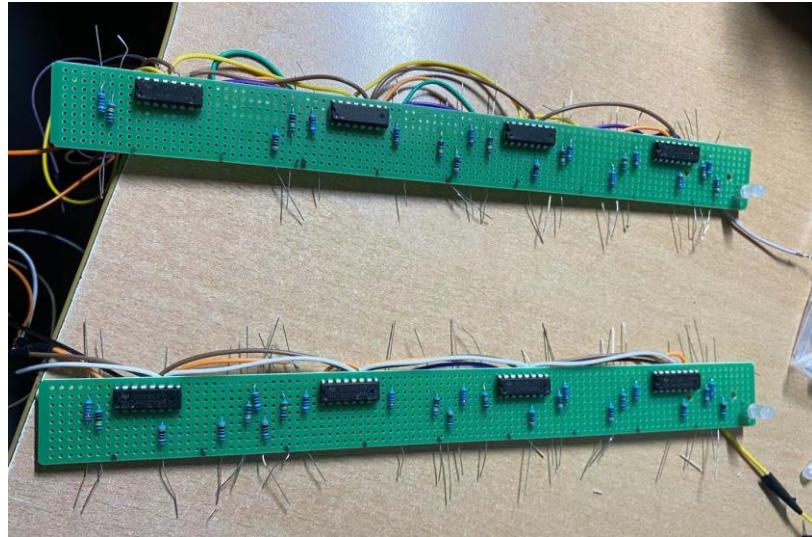
我们遇到的困难大体可以分为这样几个方面：焊接困难、调试困难、运行时间控制问题、uno板内寸不足。下面就详细介绍一下这几个方面的具体解决方案：

6.1 焊接困难：

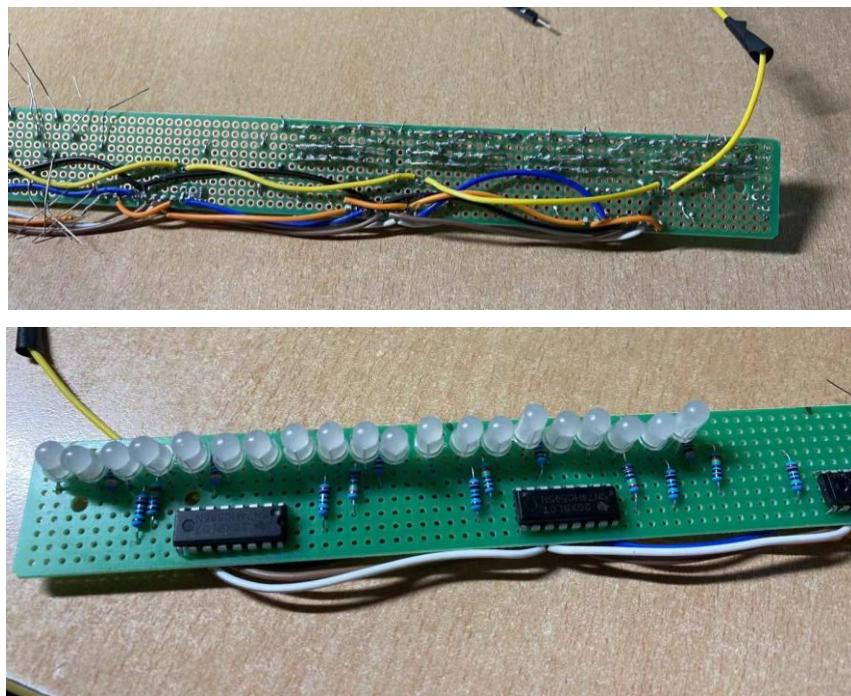
之前介绍过，我们的PCB板子是30cm*3cm的尺寸大小。也就是一块板子的面积不到100平方公分，但是我们却需要在这一块细长的板子上焊接40个RGB灯，24个电阻，4个芯片，每个RGB灯4个管脚，每个电阻2个管脚，每个芯片16个管脚。共有272个管脚，共布线143根。我们全部的电路线都是用杜邦线进行连接的，而杜邦线其实是比较粗的，所以这就加大了焊接电路的难度。很

多时候，两个焊点之间的距离只有 2mm，加上电线很密集，所以难度是很大的。

下面这幅图是我们的 PCB 板子上电阻和芯片的分布：

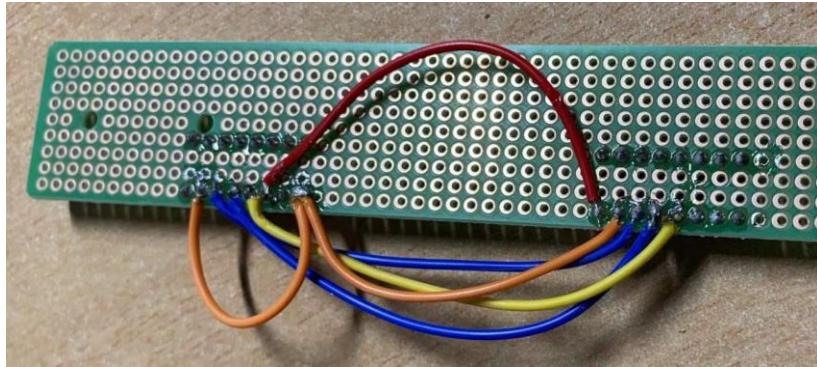


以下是 RGB 灯的正反面的分布图片：(从第一张图片中可以看到 RGB 灯连接的紧密程度)



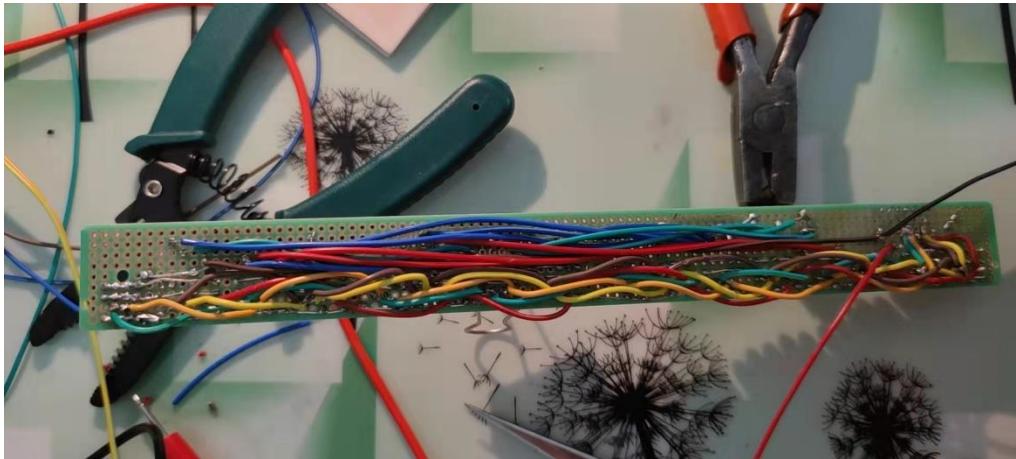
在我们的布线中，有很多这样的情况，就是一个管脚上需要连接两根杜邦线。在空间极其狭隘的情况下，用两根独立的杜邦线是很难实现的，所以我们想出了一个比较好的方法，就是把一根线从中间剥开，用中间剥开的地方去缠在这个管脚上，这样实际上这个管脚两边的线都是同一根线，连接是比较方便的。具体的样子可以参考下面这幅图橙色线的中间的那块部分。一般的线我们焊接都是以一个管脚为起点，一个管脚为终点，这里就要麻烦得多。这个方法在我们的焊接中使用了很多次。这个方法也存在缺点，那就是这根线在焊接的时候，免不了要被弯折，而且由于没有外皮的保护所以里面的铜线部分都暴露在外面，很容易收到

损坏，当线被弯折多了，也会出现断裂的现象。所以在我们后来的调试过程中，就屡屡发生这种状况，十分令人头疼。



在焊接的时候还有其他的问题，比如焊接点之间相连太紧密，导致锡全部焊接在一起，再比如 PCB 板子上铜圈的脱落导致焊锡焊接不上去，再导致线也焊接不上去等等。再比如焊头的热度不够以及稳定性差。最糟糕的问题莫过于芯片在调试过程中烧坏，导致我们不得不更换芯片，更换芯片，就意味着需要把二三十根线全部重新焊接，不仅改动大，而且还很难保证质量。

我们焊接最后的成品是这样的：



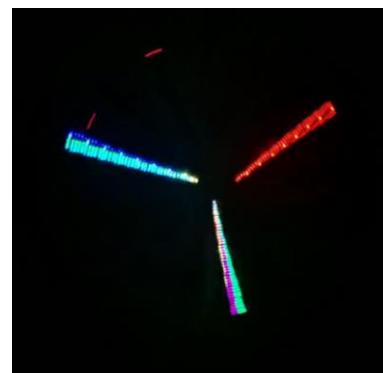
这样的板子我们一共焊接了 4 块。有一块板子实在是 bug 太多，de 不完，只好选择放弃。其他三块板子都能够正常运行，大致的效果在我们附加的视频 6.1 与 6.2 中可以体现。

6.2 调试困难

为什么说调试有困难呢，因为我们的最终的效果是要把板子放在自行车的后轮上，通过骑行时高速的转动来形成图片。在桌子上的话，是很难通过手动改变 MPU 的角度来判断灯亮的正确与否。也就是说每次调试都最好安装在自行车轮子上，而这恰恰就是最难的地方。对于 PCB 板子而言，板子上面的线太多，难以固定，而且有的焊点并不牢固，如果用力过大就会损坏它。其次，我们的 PCB 板子看起来很扁，但由于 RGB 灯的高度，PCB 板子整体的高度将近 2cm，这就给安装在车轮上造成了很大很大的麻烦。再者，理想情况下，是所有的板子以及线随着车轮一起旋转，但是有的时候，线会和轮子上的辐条缠绕在一起。严重的话甚至会破坏整个板子。进一步说，当车速过快时，离心力也会增加，所有的东西都在向外甩，质量比较大的 uno 板子、小面包板、电池就有被甩出去的风险。

所以调试是很难的。

而且由于，我们的效果只能在轮子高速旋转的时候，用肉眼才能看出来。在室外，车速过高时颠簸也是十分严重的，我们手头的摄像机也无法完成对焦的工作，即使人眼看到的是连续的图案，摄像机拍到的也并不连续的。更关键的是，一边高速骑车，一边用手拿着摄像机拍摄是一件非常危险的事，有一次我们就由于忙着拍视频没有及时调整方向而差点跟别的车子相撞。所以我们后来就想了一种办法，那就是把车抬到宿舍，在宿舍把后轮架起来，用手快速旋转踏板，带动后轮在原地高速旋转，这时候摄像机的拍摄效果是很不错的。所以我们最终也基本都采用这种图片的形式来呈现我们的效果。



上图是转动过慢导致颜色显示得不够连续，形成不了图片。





上面两张图是在宿舍中厅安装和调试我们的作品。

6.3 运行时间控制问题

运行时间控制问题是这样的情况：我们如果想要使得灯在转动的时候利用视觉暂留显示出图片，那么转速就要有一定的保证。最慢也得 6-7 米每秒，车轮的转速是非常快的。这里就出现了问题，在车轮转得比较快的时候，一秒可能就要转 4-6 圈，而我们设定的角度是每 2° 划分一个显示的位置，所以轮子转动一圈，板子上 40 个灯的颜色就要改变 180 次。这样，一秒可能就要改变八九百甚至上千次，也就是 1ms 左右必须完成对所有灯的一次显示工作。而这样的运行速度是我们原来的程序远远不够的。所以我们尝试了很多方法，比如

所以我们利用缓冲器的机制进行了一个优化，事实证明这是很有效的。先把数据传进缓冲器，再从缓冲器中读出数据进行显示，这样就会比较高效。

6.4 Uno 板内存不足

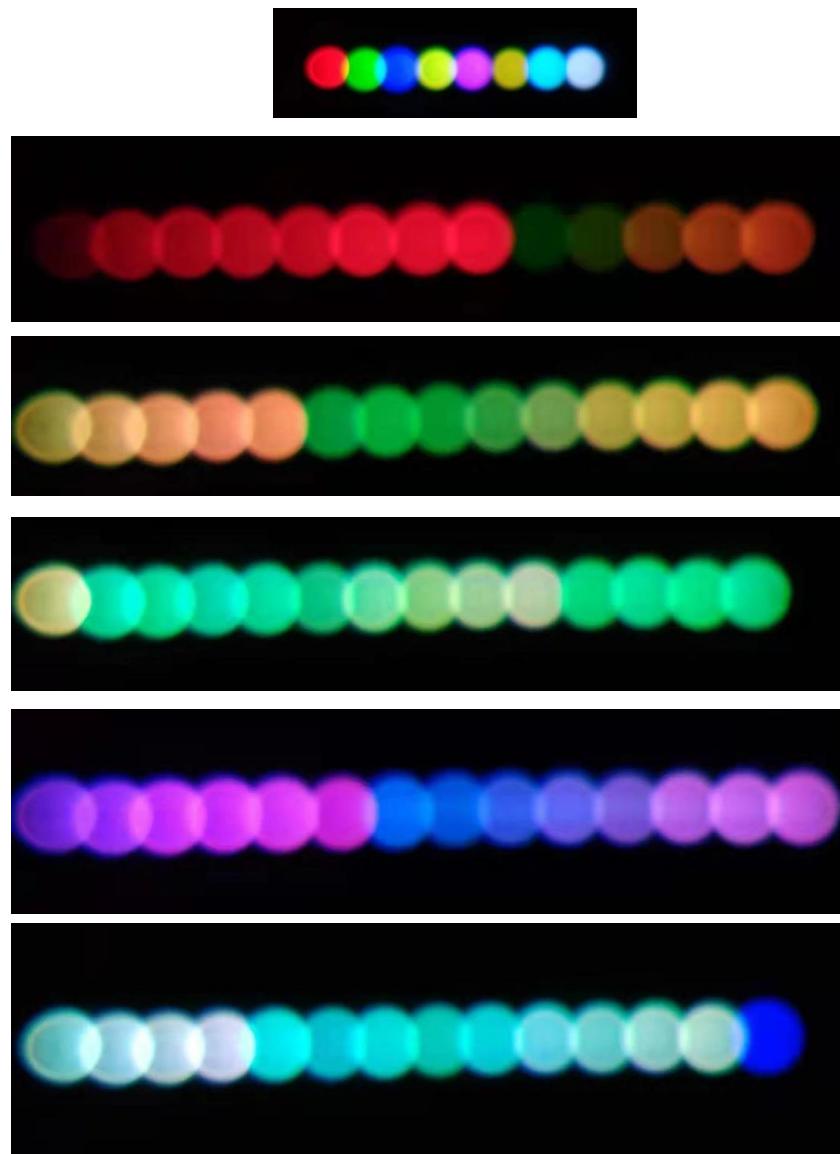
arduino 的内存是很小的，程序的存储空间，最大为 32256 字节。如果用 RGB 值去存储亮灯的信息的话，一张图片需要 21600 个 unsigned int 类型，一个 unsigned int 占 4 个字节，一共就需要 86400 字节，这是不可能实现的，如果利用这种方式，恐怕连一张图片也存储不了。所以我们就想出了这样一种存储数据的方式。那就是用 4 位二进制数去存储一个灯的一种颜色的亮度。8 位二进制共同组成一个 byte 类型。这样用一定的顺序和方法去读取，就可以用 32256 字节去存储 3-4 张照片，这是我们巨大的进步。

第七部分：成品展示与实测效果

效果的展示一共可以分为这样几个部分：单个板的亮灯展示、骑行时图片显示、骑行时动图 gif 显示、骑行时车速显示这几个部分。每个部分我们都会给出必要的图片以及视频说明。

7.1 单个板的亮灯展示：

这个效果的展示可以见附件中的视频 6.1 和 6.2，在文件夹 7.1 中也有展示的视频。以下是我们单独控制亮灯的颜色的图片，目的是来准确控制 RGB 显示不同的颜色。以探究 RGB 的色域。



7.2 骑行时图片显示：

之前提到过，由于摄像机的设备原因，以及出于安全因素的考虑（极高车速下拍摄视频真的很危险），我们没有能够拍出在真实路况上骑行时效果很好的视频，所以我们采取另一种展示的方法，那就是把车抬到宿舍，把后轮撑起来，通

过手转踏板来带动后轮高速旋转，从而拍出效果。我们拍出的效果是这样的：

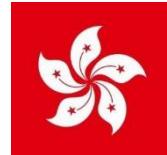
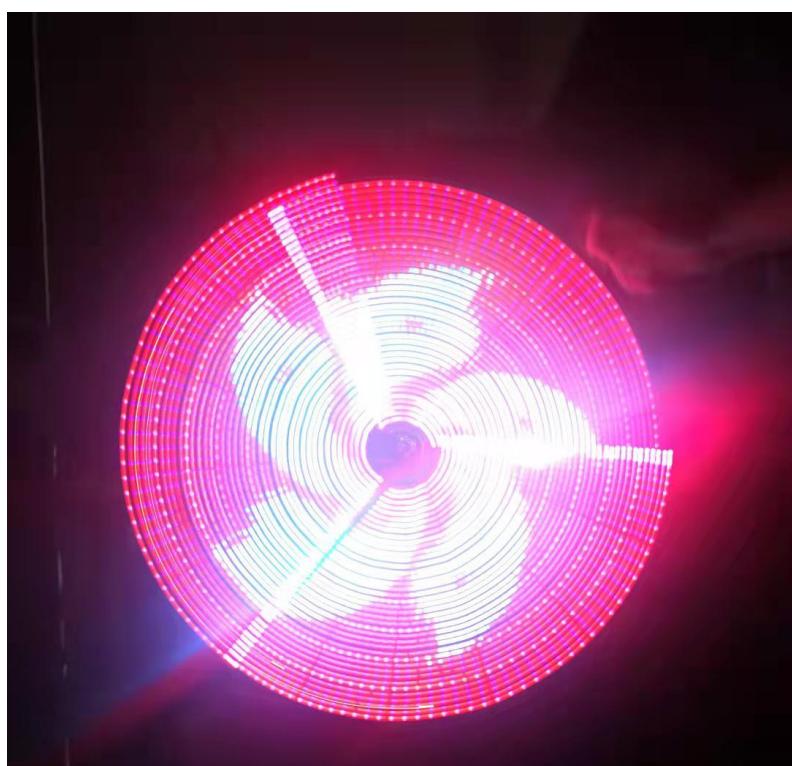
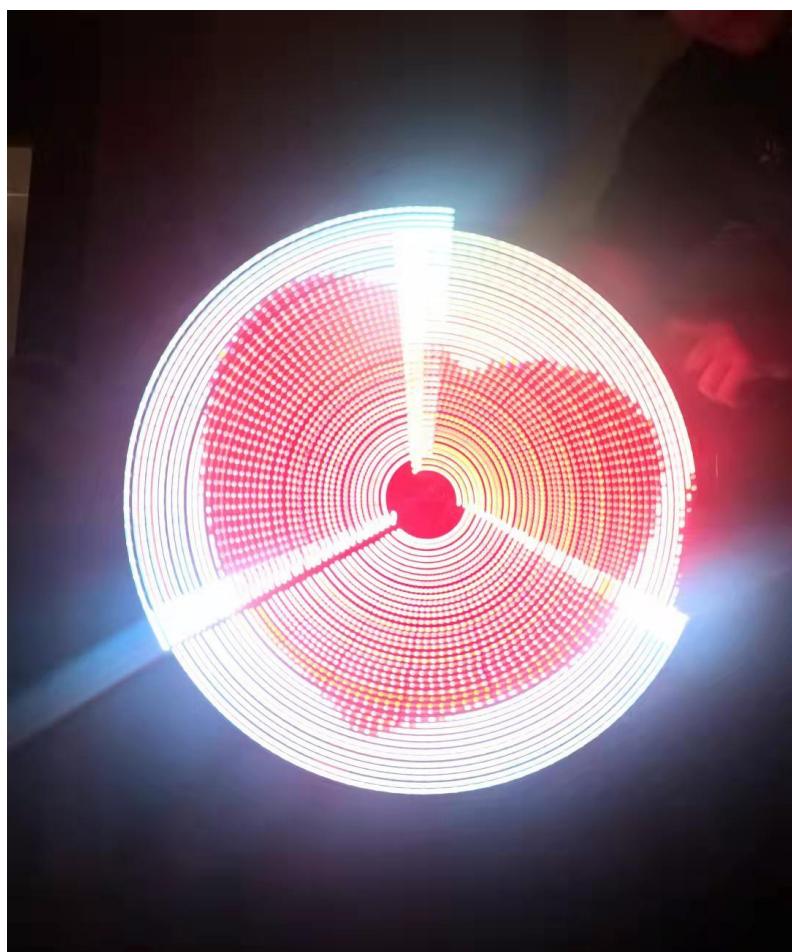


右下角是我们原来找到的图片，经过处理、转换、烧录等一系列操作，最终呈现在板子上的就是上图这样的效果。(其实照相机拍摄的还是有一点失真，肉眼的效果会更加好一点)

这里其实可以顺便展示一下这张图片的所有的处理流程：先色彩增强，再极坐标展开，最后压缩大小，以及像素提取。



下面是另外几张效果图片，看起来也是很不错的：



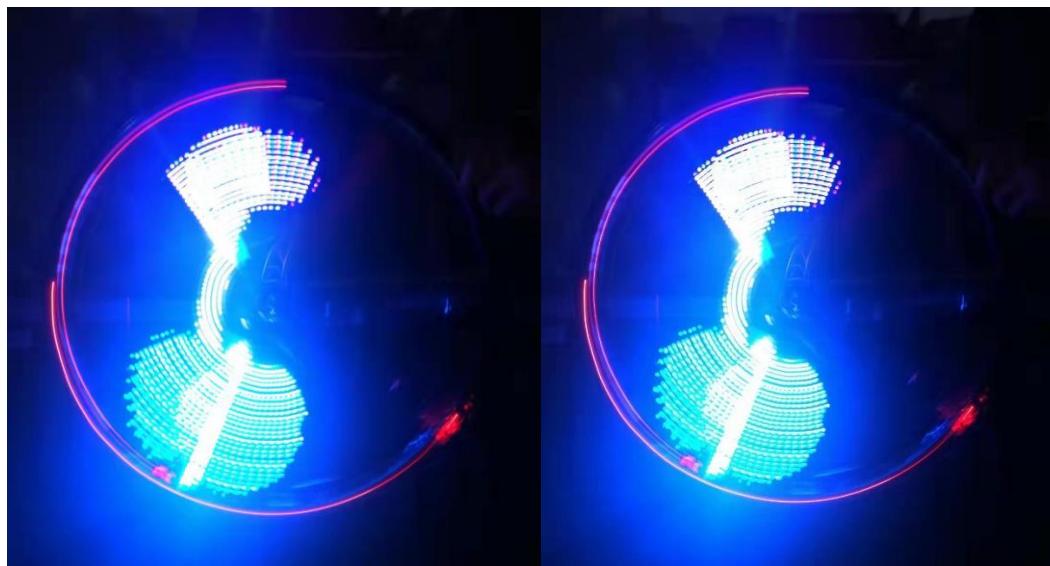


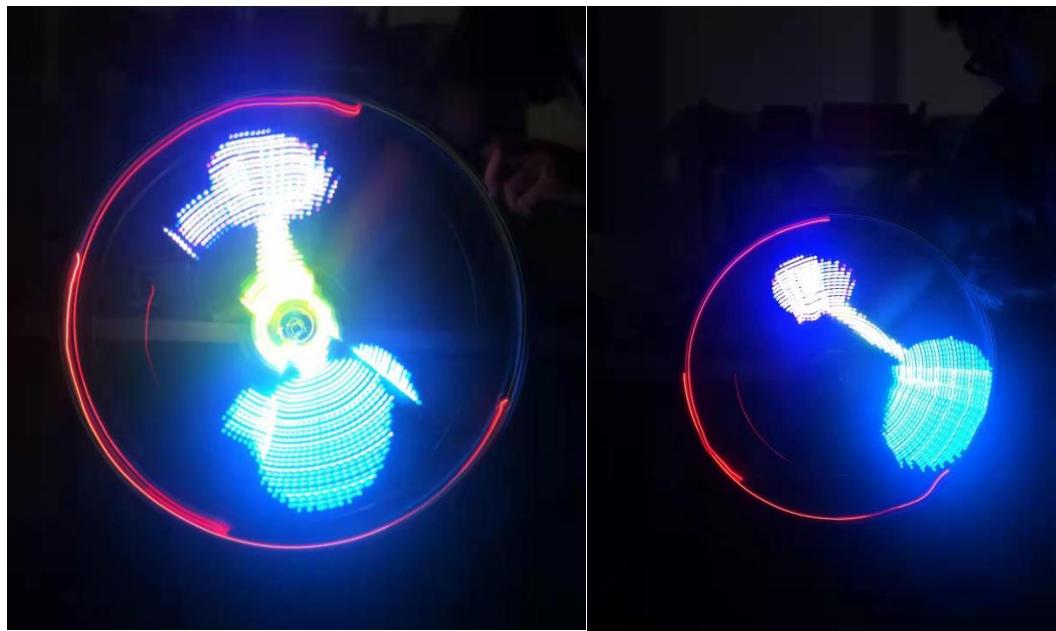


其实可以注意到，我们显示的图片也是有限制的。如果是图案花纹比较精细的话，精度就会不够。比如北大校徽周围的字母。我们支持半径为 40 像素的图片，这是很难显示比较精细的部分的。但是对于大部分图案还是能够比较好地显示出来的。

7.3 骑行时动图 gif 显示：

我们做了一个简易的摆锤 gif 动图。其实肉眼的效果还不错，但是用视频或者图片拍摄出来有些滞留：





周围的那一圈红色，其实是 uno 板子和 MPU 的红色亮灯，本来想遮住，后来干脆不遮了，这样也挺酷。

7.4 骑行时车速显示：

车速显示是我们后期升级的一个功能，就是根据 MPU 在固定时间内的旋转角度，以及辐条半径长度等参数计算出当前车轮的转速是多少，然后再把这个数字以图片的形式显示在车轮上，单位是米每秒。效果如下：



有的数字可能有些弯折扭曲，比如“8”，但是都很容易识别，整体的效果还是不错的。

第八部分：调试代码与优化过程

8.1 单个板的亮灯代码：

```
#include <SPI.h>
#include "HardwareSerial.h"
#define latch_pin 2
#define blank_pin 4
#define data_pin 51
#define clock_pin 52

#define IMGS 1
#define PADS 1
//板子的总数，每块板子 40 个 LED 灯

byte red0[5*PADS],red1[5*PADS],red2[5*PADS],red3[5*PADS];
byte green0[5*PADS],green1[5*PADS],green2[5*PADS],green3[5*PADS];
byte blue0[5*PADS],blue1[5*PADS],blue2[5*PADS],blue3[5*PADS];
//每个数组中存有每组共阴极 LED 的各色亮度值的各个位，blue0[3]代表第一块板第四组共阴极灯蓝色
亮度值的第 0 位

byte node[5];

//用于标记共阴极该如何输出，板子最下面的共阴极为 node[0]

int nodelevel=0;
int trans_bit, trans_count = 0;
/*用于标记 ISR 传输函数
trans_bit 指当前传输的亮度的位数
trans_count 用以标记当前的传输次数，从而控制亮度的传输
nodelevel 指当前扫描的共阴极是哪一个
*/
unsigned long start;
double pre_angle = 359;
byte**** img;
/*用以记录到达某个角度时灯该如何点亮，尚未想到更好的存储方式
img[imagenum][anglenum][lednum/2][r1-r2/g1-g2/b1-b2]
imagenum 为要播放的照片序号
anglenum 为角度序号，355°-1°为 0, 1°-3°为 1, 依此类推
lednum 下每个记录了两盏灯的播放方式
每个 lednum 下还包含三项：0 为 r, 1 为 g, 2 为 b
每项的前四位记录了这两盏灯较为小号的灯的相应四位亮度值，后四位记录了后一盏灯相应的四位亮度
值*/
int imgnow=0;
char rgb[30]="";

void setup() {
    // put your setup code here, to run once:
    SPI.setBitOrder(MSBFIRST);
    SPI.setDataMode(SPI_MODE0);
    SPI.setClockDivider(SPI_CLOCK_DIV2);
    /*对 SPI 初始化，SPI 模式设置为
    1.高位优先传入;
    2.上升沿采样，下降沿置位，SCK 闲置时为 0;
```

```

3. 每位传输时钟设置为系统时钟的二分频，也即 8Mhz*/
cli();

TCCR1A = B00000000;
TCCR1B = B00001011;
TIMSK1 = B00000010;
OCR1A = 5;
/*ISR 设置 (TIMER1)
1. 因无需使用，禁用 PWM 模式;
2. 分频为 64;
3. 禁用所有溢出中断，使用比较中断;
4. 以 64 为分频时，使用三块板及更少，则 5 个分频后时钟中断一次；使用六块板则 10 个分频后时钟
中断一次*/

TCCR2A = B00000000;
TCCR2B = B00000100;
TIMSK2 = B00000100;
OCR2A = 25;
//与上面类似的 ISR 设置 (TIMER2) , 每 100μs 中断一次

for(int i=0;i<5;i++)
node[i]=~(2<<i);

img= new byte***[IMGS];
for(int i=0;i<IMGS;i++)
{
    img[i]=new byte**[180];
    for(int j=0;j<180;j++)
    {
        img[i][j]=new byte*[20];
        for(int k=0;k<20;k++)
        {
            img[i][j][k]=new byte[3];
            for(int l=0;l<3;l++)
                ;//img[i][j][k][l]=0;
        }
    }
}
//为 img 分配空间并初始化 img 数据

pinMode(latch_pin, OUTPUT);
pinMode(data_pin, OUTPUT);
pinMode(clock_pin, OUTPUT);
pinMode(blank_pin, OUTPUT);
SPI.begin();
sei();
clean();
//初始化

Serial.begin(9600);

}

int N=0;
void loop() {
// N++;
// Serial.println("loop");
// put your main code here, to run repeatedly:
}

```

```

//clean();
//testrgb(); //理论上为全亮红 5s, 全亮绿 5s, 全亮蓝 5s, 全亮白 5s, 清空中断 1s, 用以基础测试
testshow();
}
double get_angle()
{
    return pre_angle+2;
}
//模拟陀螺仪输出绝对角度的函数
void LED(int pad,int light,byte red, byte green, byte blue)
{
    if (pad < 0)
        pad = 0;
    if (pad > PADS-1)
        pad = PADS-1;
    if (light < 0)
        light = 0;
    if (light > 39)
        light = 39;
    if (red < 0)
        red = 0;
    if (red > 15)
        red = 15;
    if (green < 0)
        green = 0;
    if (green > 15)
        green = 15;
    if (blue < 0)
        blue = 0;
    if (blue > 15)
        blue = 15;
    bitWrite(red0[5*pad+light%5],int(light/5), bitRead(red, 0));
    bitWrite(red1[5*pad+light%5],int(light/5), bitRead(red, 1));
    bitWrite(red2[5*pad+light%5],int(light/5), bitRead(red, 2));
    bitWrite(red3[5*pad+light%5],int(light/5), bitRead(red, 3));

    bitWrite(green0[5*pad+light%5],int(light/5), bitRead(green, 0));
    bitWrite(green1[5*pad+light%5],int(light/5), bitRead(green, 1));
    bitWrite(green2[5*pad+light%5],int(light/5), bitRead(green, 2));
    bitWrite(green3[5*pad+light%5],int(light/5), bitRead(green, 3));

    bitWrite(blue0[5*pad+light%5],int(light/5), bitRead(blue, 0));
    bitWrite(blue1[5*pad+light%5],int(light/5), bitRead(blue, 1));
    bitWrite(blue2[5*pad+light%5],int(light/5), bitRead(blue, 2));
    bitWrite(blue3[5*pad+light%5],int(light/5), bitRead(blue, 3));
}
/*
用以编辑当前 LED 应有的显示情况。
其中：
    pad 范围为 0 - PADS-1, 表示的是板子顺序;
    light 范围为 0-39, 表示的是板子上由下端到上端（有孔端）的灯序号;
    red, green, blue 范围为 0-15, 表示的是 4-bit 下的 rgb 亮度。
*/
void LED_from2(int pad,int halflight,byte red_double,byte green_double,byte blue_double)
{
    if (pad < 0)
        pad = 0;

```

```

    if (pad > PADS-1)
        pad = PADS-1;
    if (halflight < 0)
        halflight = 0;
    if (halflight > 19)
        halflight = 19;
    if (red_double < 0)
        red_double = 0;
    if (green_double < 0)
        green_double = 0;
    if (blue_double < 0)
        blue_double = 0;
    int light=2*halflight;
    bitWrite(red0[5*pad+light%5],int(light/5), bitRead(red_double, 4));
    bitWrite(red1[5*pad+light%5],int(light/5), bitRead(red_double, 5));
    bitWrite(red2[5*pad+light%5],int(light/5), bitRead(red_double, 6));
    bitWrite(red3[5*pad+light%5],int(light/5), bitRead(red_double, 7));

    bitWrite(green0[5*pad+light%5],int(light/5), bitRead(green_double, 4));
    bitWrite(green1[5*pad+light%5],int(light/5), bitRead(green_double, 5));
    bitWrite(green2[5*pad+light%5],int(light/5), bitRead(green_double, 6));
    bitWrite(green3[5*pad+light%5],int(light/5), bitRead(green_double, 7));

    bitWrite(blue0[5*pad+light%5],int(light/5), bitRead(blue_double, 4));
    bitWrite(blue1[5*pad+light%5],int(light/5), bitRead(blue_double, 5));
    bitWrite(blue2[5*pad+light%5],int(light/5), bitRead(blue_double, 6));
    bitWrite(blue3[5*pad+light%5],int(light/5), bitRead(blue_double, 7));

    light++;

    bitWrite(red0[5*pad+light%5],int(light/5), bitRead(red_double, 0));
    bitWrite(red1[5*pad+light%5],int(light/5), bitRead(red_double, 1));
    bitWrite(red2[5*pad+light%5],int(light/5), bitRead(red_double, 2));
    bitWrite(red3[5*pad+light%5],int(light/5), bitRead(red_double, 3));

    bitWrite(green0[5*pad+light%5],int(light/5), bitRead(green_double, 0));
    bitWrite(green1[5*pad+light%5],int(light/5), bitRead(green_double, 1));
    bitWrite(green2[5*pad+light%5],int(light/5), bitRead(green_double, 2));
    bitWrite(green3[5*pad+light%5],int(light/5), bitRead(green_double, 3));

    bitWrite(blue0[5*pad+light%5],int(light/5), bitRead(blue_double, 0));
    bitWrite(blue1[5*pad+light%5],int(light/5), bitRead(blue_double, 1));
    bitWrite(blue2[5*pad+light%5],int(light/5), bitRead(blue_double, 2));
    bitWrite(blue3[5*pad+light%5],int(light/5), bitRead(blue_double, 3));

}

//供 TIMER2 中断函数使用，便于将显示矩阵中的图像信息转化为要显示的图像信息
ISR(TIMER1_COMPA_vect)
{
    PORTD |= 1 << blank_pin;
    if (trans_count == 5)
        trans_bit++;
    else if (trans_count == 15)
        trans_bit++;
    else if (trans_count == 35)
        trans_bit++;
}

```

```

trans_count++;

for (int pad = 0; pad < PADS; pad++) {
    switch (trans_bit) {
        case 0:SPI.transfer(red0[nodelevel+5*pad]);SPI.transfer(green0[nodelevel+5*pad]);SPI.transfer(blue0[nodelevel+5*pad]);
            break;
        case 1:SPI.transfer(red1[nodelevel+5*pad]);SPI.transfer(green1[nodelevel+5*pad]);SPI.transfer(blue1[nodelevel+5*pad]);
            break;
        case 2:SPI.transfer(red2[nodelevel+5*pad]);SPI.transfer(green2[nodelevel+5*pad]);SPI.transfer(blue2[nodelevel+5*pad]);
            break;
        case 3:SPI.transfer(red3[nodelevel+5*pad]);SPI.transfer(green3[nodelevel+5*pad]);SPI.transfer(blue3[nodelevel+5*pad]);
            break;
    }
    SPI.transfer(node[nodelevel]);
}
if (trans_count == 75) {
    trans_count = 0;
    trans_bit = 0;
}
PORTD |= 1 << latch_pin;
PORTD &= ~(1 << latch_pin);
PORTD &= ~(1 << blank_pin);
nodelevel++;
if (nodelevel == 5)
    nodelevel = 0;
}

//用于传输信息的中断函数，每 20μs 传输一次所有板子单个共阴极的显示方式，每 100μs 扫描一次，每 1.5ms 完成一轮显示
ISR(TIMER2_COMPB_vect)
{
    double anglenow=get_angle();
    if (pre_angle>=359)
        if (anglenow<359&&anglenow>1)
    {
        pre_angle =1;
        for(int led=0;led<20;led++)
            for(int pad=0;pad<PADS;pad++)
                LED_from2(pad,led,*(*(*(*img+imgnow)+(60*pad+int(pre_angle/2))%180)+led)+0),
                *(*(*(*img+imgnow)+(60*pad+int(pre_angle/2))%180)+led)+1),
                *(*(*(*img+imgnow)+(60*pad+int(pre_angle/2))%180)+led)+2));
    }
    else if (anglenow-2>pre_angle)
    {
        pre_angle+=2;
        for(int led=0;led<20;led++)
            for(int pad=0;pad<PADS;pad++)
                LED_from2(pad,led,*(*(*(*img+imgnow)+(60*pad+int(pre_angle/2))%180)+led)+0),
                *(*(*(*img+imgnow)+(60*pad+int(pre_angle/2))%180)+led)+1),
                *(*(*(*img+imgnow)+(60*pad+int(pre_angle/2))%180)+led)+2));
    }
}

//用于检测角度来更改当前应有的输出
void purecolor(int r,int g,int b)

```

```
{  
    for(int i=0;i<PADS;i++)  
        for(int j=0;j<40;j++)  
            LED(i,j,r,g,b);  
}  
//用于让所有灯显示 r, g, b 下的纯色  
void clean()  
{  
    for(int i=0;i<PADS;i++)  
        for(int j=0;j<40;j++)  
            LED(i,j,0,0,0);  
}  
//用于让所有灯灭掉清零  
void testrgb()  
{  
    int i=3000;  
    purecolor(15,15,15);  
    delay(i);  
    purecolor(15,0,0);  
    delay(i);  
  
    purecolor(0,15,0);  
    delay(i);  
  
    purecolor(0,0,15);  
    delay(i);  
  
    for(int i=0;i<PADS;i++)  
        for(int j=0;j<10;j++)  
            LED(i,j,15,15,0);  
  
        for(int j=10;j<13;j++)  
            LED(i,j,15,0,0);  
        for(int j=13;j<15;j++)  
            LED(i,j,15,15,0);  
  
        for(int j=15;j<40;j++)  
            LED(i,j,0,0,15);  
    delay(i);  
  
    clean();  
    delay(5000);  
}  
//用于测量各个 rgb 引脚及芯片连接情况  
void testshow()  
{  
    int tim=30;  
    //Serial.print("蓝色 1");  
    for(int i=0;i<PADS;i++)  
        for(int j=0;j<40;j++)  
    {  
        LED(i,j,0,0,15);  
        delay(tim);  
    }  
    // Serial.print("红色");
```

```
    for(int i=0;i<PADS;i++)
        for(int j=0;j<40;j++)
    {
        LED(i,j,15,0,0);
        delay(tim);
    }

    // Serial.print("绿色");
    for(int i=0;i<PADS;i++)
        for(int j=0;j<40;j++)
    {
        LED(i,j,0,15,0);
        delay(tim);
    }
    // Serial.print("红色 2");
    for(int i=0;i<PADS;i++)
        for(int j=0;j<40;j++)
    {
        LED(i,j,15,0,0);
        delay(tim);
    }

    // Serial.print("绿色 2");
    for(int i=0;i<PADS;i++)
        for(int j=0;j<40;j++)
    {
        LED(i,j,0,15,0);
        delay(tim);
    }

    // Serial.println("白色");
    for(int i=0;i<PADS;i++)
        for(int j=0;j<40;j++)
    {
        LED(i,j,15,15,15);
        delay(tim);
    }
    /*
    for(int i=0;i<PADS;i++)
        for(int j=0;j<40;j++)
    {
        LED(i,j,15,15,15);
        delay(50);
    }
    for(int i=0;i<PADS;i++)
        for(int j=0;j<40;j++)
    {
        LED(i,j,0,0,0);
        delay(50);
    }
    */
}

}

//滚动播放测试亮灯情况
void colordefine()
{
    int r=0;
    int g=0;
```

```

int b=0;
int i=0;
int j=0;
while(b<16)
{
    if(j>39)
    {
        j=0;delay(5000);
    }
    if(r>15)
    {
        r=0;
        if(g) g+=2;
        else g++;
    }
    if(g>15)
    {
        g=0;
        if(b) b+=2;
        else b++;
    }
    if(i>PADS-1)
    {
        i=0;
        delay(490);
        clean();
        delay(10);
    }
    LED(i,j,r,g,b);
    if(r) r+=2;
    else r++;
    j+=3;
}
}

//用于遍历(为了减少工作量将档位压缩为 0,1,3,5,7,9,11,13,15 这九种)显示颜色的所有情况, 用以校准 rgb 转化情况
void showclr()
{
    for(int i=0;i<40;i++)
        LED(0,i,15,0,0);
    delay(1000);

    for(int i=0;i<40;i++)
        LED(0,i,0,15,0);
    delay(1000);

    for(int i=0;i<40;i++)
        LED(0,i,0,0,15);
    delay(1000);

    for(int i=0;i<40;i++)
        LED(0,i,15,15,0);
    delay(1000);

    for(int i=0;i<40;i++)
        LED(0,i,15,0,15);
    delay(1000);
}

```

```

for(int i=0;i<40;i++)
    LED(0,i,15,11,0);
delay(1000); //这是我校正过的黄色

for(int i=0;i<40;i++)
    LED(0,i,0,15,15);
delay(1000);

for(int i=0;i<40;i++)
    LED(0,i,15,15,15);
delay(1000);

for(int i=0;i<40;i++)
    LED(0,i,15,11,11);
delay(1000); //猜测校正后的白色

for(int i=0;i<40;i++)
    LED(0,i,0,0,0);
delay(1000);
}

//用于显示三原色和基本混合色
int rgbtrans(int value)
{
    return ((value%16)/8+value/16);
}

//用以将 8-bit 标准 rgb 颜色信息转化为 LED 显示的 4-bit 颜色信息，暂设定为线性映射
void imginput(int imgnumber,byte*** k)
{
    for(int ang=0;ang<180;ang++)
        for(int light=0;light<40;light+=2)
            for(int clr=0;clr<3;clr++)
            {
                byte colortmpL=rgbtrans(*(*(*(k+ang)+light)+clr));
                byte colortmpR=rgbtrans(*(*(*(k+ang)+light+1)+clr));
                for(int whichbit=0;whichbit<4;whichbit++)
                {
                    bitWrite(*(*(*(*img+imgnumber)+ang)+light/2)+clr),4+whichbit, bitRead(colortmpL, whichbit));
                    bitWrite(*(*(*(*img+imgnumber)+ang)+light/2)+clr),whichbit, bitRead(colortmpR, whichbit));
                }
            }
    }

/*假设输入的图像矩阵为 picture[angle(0-179)][light(0-39)][rgb(0 is r,1 is g,2 is b,
8-bit color)]
   则此函数可以将此图像矩阵转换为压缩后的编码，后期可改造为蓝牙传输，则会最大程度压缩内存
*/

```

8.2 图片显示代码：

这部分代码中有一个全局数组是这样的：

```

const byte IMGMEM[IMGS*10800]
PROGMEM={B00110000,B10110000,B00110000,.....,B11000000,B11000000,B11000000};

```

在原本的程序中，这个数组大小是 10800，也就是说有 10800 个像“B11000010”这样的元素，所以这里用省略号代替，以节省空间。后面的代码也都采取这种形

式来减小篇幅。

```
#include <TimerOne.h>
#include <SPI.h>
#include "HardwareSerial.h"
#include <avr/pgmspace.h>
#include<Wire.h>
#define latch_pin 2
#define blank_pin 4
#define data_pin 11
#define clock_pin 13
#define IMGS 1
#define PADS 3
byte red0[5*PADS],red1[5*PADS],red2[5*PADS],red3[5*PADS];
byte green0[5*PADS],green1[5*PADS],green2[5*PADS],green3[5*PADS];
byte blue0[5*PADS],blue1[5*PADS],blue2[5*PADS],blue3[5*PADS];
int16_t AcX,AcY,AcZ;
int minValue=265,maxValue=402;
double z;
double err_est[6];
double err_mea=2;
double x_est[6];
double kalman_gain[6];
uint8_t angle_now = 0;// 0-179 区间
byte* to_read;
byte* red[4]={red0,red1,red2,red3};
byte* green[4]={green0,green1,green2,green3};
byte* blue[4]={blue0,blue1,blue2,blue3};
byte* imgnow;
byte node[5];
int8_t nodelevel=0;
int8_t trans_bit, trans_count = 0;
unsigned int begintime=0,dftime=0,endtime=0;
unsigned int angle=0,beginangle=0,difangle=0,endangle=0;
unsigned int speed1;
const byte IMGMEM[IMGS*10800] PROGMEM = {B00110000,B10110000,B00110000,.....,B11000000
,B11000000,B11000000,B11000000};

void setup()
{
    int i;
    SPI.setBitOrder(MSBFIRST);
    SPI.setDataMode(SPI_MODE0);
    SPI.setClockDivider(SPI_CLOCK_DIV2);
    /*对 SPI 初始化, SPI 模式设置为
```

```

1.高位优先传入;
2.上升沿采样, 下降沿置位, SCK 闲置时为 0;
3.每位传输时钟设置为系统时钟的二分频, 也即 8Mhz*/
cli();
//VCC GND BLANK LATCH CLOCK
imgnow = (byte*) IMGMEM;

for(int i=0;i<5;i++)
node[i]=~(2<<i);

pinMode(latch_pin, OUTPUT);
pinMode(data_pin, OUTPUT);
pinMode(clock_pin, OUTPUT);
pinMode(blank_pin, OUTPUT);
SPI.begin();
sei();
//初始化

Serial.begin(9600);

Wire.begin();
Wire.beginTransmission(0x68);
Wire.write(0x6B);
Wire.write(0);
Wire.endTransmission(true);

//Timer1.initialize( 500000 );
//Timer1.attachInterrupt( Isr );
}

double MPU_use(){

Wire.beginTransmission(0x68);
Wire.write(0x3B);
Wire.endTransmission(false);
Wire.requestFrom(0x68,14,true);

AcX=Wire.read()<<8|Wire.read();
AcY=Wire.read()<<8|Wire.read();
AcZ=Wire.read()<<8|Wire.read();

int xAng = map(AcX,minVal,maxVal,-90,90);
int yAng = map(AcY,minVal,maxVal,-90,90);

x_est[0] = RAD_TO_DEG * (atan2(-yAng,-xAng)+PI);

```

```

    return x_est[0];

}

void to_shine()
{
    trans_bit=0;
    trans_count=0;
    to_read=imgnow+angle_now*60;
    nodelevel=0;

    for(;trans_count<75;trans_count++)
    {
        PORTD |= 1 << blank_pin;
        if (trans_count == 5)
            trans_bit++;
        else if (trans_count == 15)
            trans_bit++;
        else if (trans_count == 35)
            trans_bit++;

        for (int pad = 0;pad < PADS;pad++)
        {
            to_read=imgnow+((angle_now+(PADS-1-pad)*60)%180)*60;
            SPI.transfer(pgm_read_byte(to_read+trans_bit*5+nodelevel));
            SPI.transfer(pgm_read_byte(to_read+trans_bit*5+nodelevel+20));
            SPI.transfer(pgm_read_byte(to_read+trans_bit*5+nodelevel+40));
            SPI.transfer(node[nodelevel]);
        }

        PORTD |= 1 << latch_pin;
        PORTD &= ~(1 << latch_pin);
        PORTD &= ~(1 << blank_pin);
        nodelevel++;

        if (nodelevel == 5)
            nodelevel = 0;

        delayMicroseconds (150) ;
    }
}

void loop()
{
    angle_now = int(MPU_use()/2);
    to_shine();
}

```

```
}
```

8.3 动图 gif 显示代码:

```
#include <TimerOne.h>
#include <SPI.h>
#include "HardwareSerial.h"
#include <avr/pgmspace.h>
#include<Wire.h>
#define latch_pin 2
#define blank_pin 4
#define data_pin 11
#define clock_pin 13
#define IMGS 1
#define PADS 3
byte red0[5*PADS],red1[5*PADS],red2[5*PADS],red3[5*PADS];
byte green0[5*PADS],green1[5*PADS],green2[5*PADS],green3[5*PADS];
byte blue0[5*PADS],blue1[5*PADS],blue2[5*PADS],blue3[5*PADS];
int16_t AcX,AcY,AcZ;
int minVal=265,maxVal=402,flag = 1;
double z;
double err_mea=2;
double x_est[6];
uint8_t angle_now = 0;// 0-179 区间
byte* to_read;
byte* red[4]={red0,red1,red2,red3};
byte* green[4]={green0,green1,green2,green3};
byte* blue[4]={blue0,blue1,blue2,blue3};
byte* imgnow;
byte node[5];
int8_t nodelevel=0;
int8_t trans_bit, trans_count = 0;
unsigned int begintime=0,dftime=0,endtime=0,loop_start = 0;
unsigned int angle=0,beginangle=0,difangle=0,endangle=0;
unsigned int speed1;
const byte IMGMEM[IMGS*10800] PROGMEM = {B00110000,B10110000,.....,B11000000,B11000000
,B11000000};

byte* save_angle[3] = {0};
byte* beginning;
int img_num_now = 0;

void setup()
{
    int i;
    SPI.setBitOrder(MSBFIRST);
    SPI.setDataMode(SPI_MODE0);
    SPI.setClockDivider(SPI_CLOCK_DIV2);
    /*对 SPI 初始化, SPI 模式设置为
    1.高位优先传入;
    2.上升沿采样, 下降沿置位, SCK 空置时为 0;
    3.每位传输时钟设置为系统时钟的二分频, 也即 8Mhz*/
    cli();
    //VCC GND BLANK LATCH CLOCK
    imgnow = (byte*) IMGMEM;

    for(int i=0;i<5;i++)
        node[i]=~(2<<i);

    pinMode(latch_pin, OUTPUT);
    pinMode(data_pin, OUTPUT);
    pinMode(clock_pin, OUTPUT);
    pinMode(blank_pin, OUTPUT);
    SPI.begin();
}
```

```

sei();
//初始化

Serial.begin(9600);

Wire.begin();
Wire.beginTransmission(0x68);
Wire.write(0x6B);
Wire.write(0);
Wire.endTransmission(true);

// Timer1.initialize( 1500 );
// Timer1.attachInterrupt( Isr );
}

double MPU_use(){

    Wire.beginTransmission(0x68);
    Wire.write(0x3B);
    Wire.endTransmission(false);
    Wire.requestFrom(0x68,14,true);

    AcX=Wire.read()<<8|Wire.read();
    AcY=Wire.read()<<8|Wire.read();
    AcZ=Wire.read()<<8|Wire.read();

    int xAng = map(AcX,minVal,maxVal,-90,90);
    int yAng = map(AcY,minVal,maxVal,-90,90);

    x_est[0] = RAD_TO_DEG * (atan2(-yAng,-xAng)+PI);
    return x_est[0];

}

void to_shine()
{
    trans_bit=0;
    trans_count=0;
    // to_read=imgnow+angle_now*60;
    nodelevel=0;
    imgnow = (byte*)IMGMEM + img_num_now*10800;
    for(;trans_count<75;trans_count++)
    {
        PORTD |= 1 << blank_pin;
        if (trans_count == 5)
            trans_bit++;
        else if (trans_count == 15)
            trans_bit++;
        else if (trans_count == 35)
            trans_bit++;

        if(angle_now<60){
            save_angle[0] = imgnow + 60 * (angle_now + 120);
            save_angle[1] = save_angle[0] - 3600;
            save_angle[2] = save_angle[1] - 3600;
        }
        else if(angle_now < 120){
            save_angle[0] = imgnow + 60*angle_now - 3600;
            save_angle[1] = save_angle[0] + 7200;
            save_angle[2] = save_angle[0] + 3600;
        }
        else{
            save_angle[0] = imgnow + 60*angle_now - 3600;
            save_angle[1] = save_angle[0] - 3600;
        }
    }
}

```

```

        save_angle[2] = save_angle[0] + 3600;
    }
    for (int pad = 0; pad < PADS; pad++)
    {
        beginning = save_angle[pad] + trans_bit * 5 + nodelevel;
        SPI.transfer(pgm_read_byte(beginning));
        SPI.transfer(pgm_read_byte(beginning+20));
        SPI.transfer(pgm_read_byte(beginning+40));
        SPI.transfer(node[nodelevel]);
    }

    PORTD |= 1 << latch_pin;
    PORTD &= ~(1 << latch_pin);
    PORTD &= ~(1 << blank_pin);
    nodelevel++;

    if (nodelevel == 5)
        nodelevel = 0;

    delayMicroseconds (150) ;
}
}

void loop()
{
    if (flag){
        loop_start = millis();
        flag = 0;
    }
    if(millis()-loop_start>1000){
        img_num_now += 1;
        if(img_num_now == IMGS) img_num_now = 0;
        loop_start = millis();
    }
    angle_now = int(MPU_use()/2);
    to_shine();
}
}

```

8.4 车速显示代码:

```

#include <TimerOne.h>
#include <SPI.h>
#include "HardwareSerial.h"
#include <avr/pgmspace.h>
#include<Wire.h>
#define latch_pin 2
#define blank_pin 4
#define data_pin 11
#define clock_pin 13
#define IMGS 1
#define PADS 1
byte red0[5*PADS],red1[5*PADS],red2[5*PADS],red3[5*PADS];
byte green0[5*PADS],green1[5*PADS],green2[5*PADS],green3[5*PADS];
byte blue0[5*PADS],blue1[5*PADS],blue2[5*PADS],blue3[5*PADS];
int16_t AcX,AcY,AcZ;
int minVal=265,maxVal=402;
double z;
double err_est[6];
double err_mea=2;
double x_est[6];
double kalman_gain[6];
uint8_t angle_now = 0;// 0-179 区间
byte* to_read;
byte* red[4]={red0,red1,red2,red3};

```

```

byte* green[4]={green0,green1,green2,green3};
byte* blue[4]={blue0,blue1,blue2,blue3};
byte* imgnow,numnow;
byte node[5];
int8_t nodelevel=0;
int8_t trans_bit, trans_count = 0;
unsigned int begintime=0,dftime=0,endtime=0;
unsigned int angle=0,beginangle=0,difangle=0,endangle=0;
float speed1;
int num1=5,num2=8;
unsigned int num0=0;

const byte IMGNUM[IMGS*19800] PROGMEM = {
B00000000,B00000000,.....,B00000000,B00000000,
B00000100,B00000100,.....,B00000000,B00000000,
B00000000,B00000000,.....,B00000000,B00000000,
B00000000,B00000000,.....,B00000000,B00000000,
B00000000,B00000000,.....,B00000000,B00000000,
B00000000,B00000000,.....,B00000000,B00000000,
B00000000,B00000000,.....,B00000000,B00000000,
B00000000,B00000000,.....,B00000000,B00000000,
B00000000,B00000000,.....,B00000000,B00000000,
B00000000,B00000000,.....,B00000000,B00000000,
B00000000,B00000000,.....,B00000000,B00000000
};

const byte IMGMEM[IMGS*10800] PROGMEM = {};

void setup()
{
    int i;
    SPI.setBitOrder(MSBFIRST);
    SPI.setDataMode(SPI_MODE0);
    SPI.setClockDivider(SPI_CLOCK_DIV2);
    /*对 SPI 初始化， SPI 模式设置为
    1.高位优先传入;
    2.上升沿采样，下降沿置位， SCK 闲置时为 0;
    3.每位传输时钟设置为系统时钟的二分频，也即 8Mhz*/
    cli();

    //imgnow = (byte*) IMGMEM;
    numnow = (byte*) IMGNUM;

    for(int i=0;i<5;i++)
        node[i]=~(2<<i);

    pinMode(latch_pin, OUTPUT);
    pinMode(data_pin, OUTPUT);
    pinMode(clock_pin, OUTPUT);
    pinMode(blank_pin, OUTPUT);
    SPI.begin();
    sei();
    //初始化

    Serial.begin(9600);

    Wire.begin();
    Wire.beginTransmission(0x68);
    Wire.write(0x6B);
    Wire.write(0);
    Wire.endTransmission(true);

    Timer1.initialize( 20000 );
    Timer1.attachInterrupt( Isr );
}

void Isr( )

```

```

{
    endangle=angle_now*2;
    if ((endangle<beginangle)and(endangle<360))
        endangle+=360;
    difangle=endangle-beginangle;

    //speed1=difangle*3.14159*0.3/180/0.2;
    speed1 = difangle*0.026;

    //Serial.println(speed1);
    num0=ceil(speed1);
    num1=num0/10;
    num2=num0%10;
    Serial.println(num2);

    beginangle=endangle;
    while (beginangle>=360)
        beginangle-=360;
}

double MPU_use(){

    Wire.beginTransmission(0x68);
    Wire.write(0x3B);
    Wire.endTransmission(false);
    Wire.requestFrom(0x68,14,true);

    AcX=Wire.read()<<8|Wire.read();
    AcY=Wire.read()<<8|Wire.read();
    AcZ=Wire.read()<<8|Wire.read();

    int xAng = map(AcX,minVal,maxVal,-90,90);
    int yAng = map(AcY,minVal,maxVal,-90,90);

    x_est[0] = RAD_TO_DEG * (atan2(-yAng,-xAng)+PI);
    //Serial.println(x_est[0]);
    return x_est[0];
}

/*
0-30*60:0
30*60-60*60:1
60*60-90*60:2
90*60-120*60:3
120*60-150*60:4
150*60-180*60:5
180*60-210*60:6
210*60-240*60:7
240*60-270*60:8
270*60-300*60:9
300*60-330*60:black
*/
int temp;
void to_shine()
{
    trans_bit=0;
    trans_count=0;

    //Serial.println(angle_now);

    /*
        60-90 为左半边数字 num1
        90-120 为右半边数字 num2
    */
}

```

```

if ((angle_now>=60)and(angle_now<90))
{
    temp=angle_now-60;
    to_read=numnow+temp*60+1800*num1;
}
else if((angle_now>=90)and(angle_now<=120))
{
    temp=angle_now-90;
    to_read=numnow+temp*60+1800*num2;
}
else to_read=numnow+18000;

for(;trans_count<75;trans_count++)
{
    PORTD |= 1 << blank_pin;
    if (trans_count == 5)
        trans_bit++;
    else if (trans_count == 15)
        trans_bit++;
    else if (trans_count == 35)
        trans_bit++;

    for (int pad = 0;pad < 1;pad++)
    {
        SPI.transfer(pgm_read_byte(to_read+trans_bit*5+nodelevel));
        SPI.transfer(pgm_read_byte(to_read+trans_bit*5+nodelevel+20));
        SPI.transfer(pgm_read_byte(to_read+trans_bit*5+nodelevel+40));
        SPI.transfer(node[nodelevel]);
    }

    PORTD |= 1 << latch_pin;
    PORTD &= ~(1 << latch_pin);
    PORTD &= ~(1 << blank_pin);
    nodelevel++;
}

if (nodelevel == 5)
    nodelevel = 0;

delayMicroseconds (175) ;
}

void loop()
{
    angle_now = int(MPU_use()/2);
    //Serial.println(angle_now);
    to_shine();
}

```

8.5 利用缓冲器的时间优化-代码:

之前提到过我们遇到过程序运行时间不够快的困难。在车轮转得比较快的时候，一秒可能就要转 4-6 圈，而我们的角度是每 2° 划分一个显示位置，所以一圈就要改变 180 次颜色，这样，一秒可能就要改变上千次，也就是 1ms 左右必须完成对所有灯的显示工作。而这是我们原来的程序远远不够的。所以我们利用缓冲器的机制进行了一个优化，事实证明这是很有效的。

缓冲区实际上就是在 Arduino 的 RAM 上开辟的临时存储空间，因此缓冲区的设定大小即使可以调整，但也不能超过 Arduino 本身的 RAM 大小；又因为我们还要在 RAM 上进行其他数据的存储，所以并不能将所有 RAM 空间都分配作

串口缓冲区。我们使用的就是默认的大小，64字节。当超过64字节时，之后接收到的数据会被直接丢弃。使用buffer可以让我们数据传输的速度快很多。

运行速度快了将近5倍。程序优化之后，每20 μ s就可以传输一次所有板子单个共阴极的显示方式，每100 μ s就可以完成一次扫描，每1.5ms就可以完成一轮显示。

```
#include <SPI.h>
#include "HardwareSerial.h"
#include <avr/pgmspace.h>
#define latch_pin 2
#define blank_pin 4
#define data_pin 11
#define clock_pin 13

#define IMGS 1
#define PADS 3
//板子的总数，每块板子40个LED灯
#define BUFFERNUM 6

byte red0[5*PADS],red1[5*PADS],red2[5*PADS],red3[5*PADS];
byte green0[5*PADS],green1[5*PADS],green2[5*PADS],green3[5*PADS];
byte blue0[5*PADS],blue1[5*PADS],blue2[5*PADS],blue3[5*PADS];
//每个数组中存有每组共阴极LED的各色亮度值的各个位，blue0[3]代表第一块板第四组共阴极灯蓝色亮度值的第0位
byte buffer[BUFFERNUM][12][5*PADS];
/*buffer[buffernum][color*4+bit][ICdata]用于储存从flashrom中加载完的缓冲数据
buffernum为缓冲区顺序的数字
color为颜色，0为r，1为g，2为b
bit为颜色对应的亮度位数，0-3位
ICdata为每种颜色与位数要传输到芯片中的相应数据，可以直接将指针赋值过去，达到显示某个
buffer的目的
*/
byte* red[4]={red0,red1,red2,red3};
byte* green[4]={green0,green1,green2,green3};
byte* blue[4]={blue0,blue1,blue2,blue3};

int16_t buffernow=0;
int16_t refreshnow=0;

int16_t angle_to_buffer, buffernum0, buffernum1, buffernum2, bufferkey;

int16_t bufferprocess=0;

byte node[5];
//用于标记共阴极该如何输出，板子最下面的共阴极为node[0]

int8_t nodelevel=0;
int8_t trans_bit, trans_count = 0;
/*用于标记ISR传输函数
trans_bit指当前传输的亮度的位数
trans_count用以标记当前的传输次数，从而控制亮度的传输
nodelevel指当前扫描的共阴极是哪一个
*/
unsigned long start;
double pre_angle = 359;
double anglenew=0;

const byte IMGMEM[IMGS*10800] PROGMEM={ B00000000,B00000000,B00000000,.....,B11111111,
B11111111,B11111111,B11111111 };
```

```

/*用以记录到达某个角度时灯该如何点亮，尚未想到更好的存储方式
 img[imagenum][anglenum][color][bit][ICdata]
 imagenum 为要播放的照片序号
 anglenum 为角度序号，359°-1°为 0, 1°-3°为 1, 依此类推
 color 为颜色, 0 为 r, 1 为 g, 2 为 b
 bit 为颜色对应的亮度位数, 0-3 位
 ICdata 为每种颜色与位数要传输到芯片中的相应数据
 */

int imgnum=0;
byte* imgnow=NULL;byte* imgover=NULL;
byte* bufferaddr[3]={NULL,NULL,NULL};

void setup() {
    // put your setup code here, to run once:
    SPI.setBitOrder(MSBFIRST);
    SPI.setDataMode(SPI_MODE0);
    SPI.setClockDivider(SPI_CLOCK_DIV2);
    /*对 SPI 初始化, SPI 模式设置为
     1.高位优先传入;
     2.上升沿采样, 下降沿置位, SCK 闲置时为 0;
     3.每位传输时钟设置为系统时钟的二分频, 也即 8Mhz*/
    cli();
}

TCCR1A = B00000000;
TCCR1B = B00001011;
TIMSK1 = B00000010;
OCR1A = 5;
/*ISR 设置 (TIMER1)
1.因无需使用, 禁用 PWM 模式;
2.分频为 64;
3.禁用所有溢出中断, 使用比较中断;
4.以 64 为分频时, 使用三块板及更少, 则 5 个分频后时钟中断一次; 使用六块板则 10 个分频后时钟
中断一次*/
TCCR2A = B00000000;
TCCR2B = B00000100;
TIMSK2 = B00000100;
OCR2A = 32;
//与上面类似的 ISR 设置 (TIMER2) , 每 128μs 中断一次

imgnow=(byte*)IMGMEM+imgnum*10800;
imgover=imgnow+10800;

for(int i=0;i<5;i++)
node[i]=~(2<<i);

int16_t anglenow=int16_t(get_angle())+1)/2;
if(anglenow>=180)
anglenow-=180;
if(anglenow<60)
{
    bufferaddr[0]=imgnow+anglenow*60;
    bufferaddr[1]=bufferaddr[0]+3600;
    bufferaddr[2]=bufferaddr[1]+3600;
}
else if(anglenow<120)
{
    bufferaddr[0]=imgnow+anglenow*60;
    bufferaddr[1]=bufferaddr[0]+3600;
    bufferaddr[2]=bufferaddr[0]-3600;
}
else

```

```

    {
        bufferaddr[0]=imgnow+anglenow*60;
        bufferaddr[2]=bufferaddr[0]-3600;
        bufferaddr[1]=bufferaddr[2]-3600;
    }
    for(bufferkey=0;bufferkey<BUFFERNUM;bufferkey++)
    {
        for(int j=0;j<12;j++)
        for(int i=0;i<5;i++)
        {
            angle_to_buffer=bufferkey+anglenow;
            buffer[bufferkey][j][i]=pgm_read_byte(bufferaddr[0]+i+5*j);
            buffer[bufferkey][j][5+i]=pgm_read_byte(bufferaddr[1]+i+5*j);
            buffer[bufferkey][j][10+i]=pgm_read_byte(bufferaddr[2]+i+5*j);
        }

        for(int temp=0;temp<3;temp++)
        {
            bufferaddr[temp]+=60;
            if(bufferaddr[temp]>=imgover)
            {
                bufferaddr[temp]=imgnow+int(bufferaddr[temp]-imgover);
            }
        }
    }
    refreshnow=BUFFERNUM-1;
    pre_angle=anglenow;
    buffernow=0;
    for(int tmp=0;tmp<4;tmp++)
    {
        red[tmp]=*(*(buffer+buffernow)+tmp);
        green[tmp]=*(*(buffer+buffernow)+tmp+4);
        blue[tmp]=*(*(buffer+buffernow)+tmp+8);
    }

pinMode(latch_pin, OUTPUT);
pinMode(data_pin, OUTPUT);
pinMode(clock_pin, OUTPUT);
pinMode(blank_pin, OUTPUT);
SPI.begin();
sei();
clean();
//初始化

Serial.begin(9600);

}

void loop() {
    // put your main code here, to run repeatedly:
    delay(1);
    //clean();
    //testrgb(); //理论上为全亮红 5s, 全亮绿 5s, 全亮蓝 5s, 全亮白 5s, 清空中断 1s, 用以基础测试
}
double get_angle()
{
    anglenew+=0.1;
    if(anglenew>=360)
        anglenew-=360;
    return anglenew;
}
//模拟陀螺仪输出绝对角度的函数
void LED(int pad,int light,byte red, byte green, byte blue)
{

```

```

    if (pad < 0)
        pad = 0;
    if (pad > PADS-1)
        pad = PADS-1;
    if (light < 0)
        light = 0;
    if (light > 39)
        light = 39;
    if (red < 0)
        red = 0;
    if (red > 15)
        red = 15;
    if (green < 0)
        green = 0;
    if (green > 15)
        green = 15;
    if (blue < 0)
        blue = 0;
    if (blue > 15)
        blue = 15;
    bitWrite(red0[5*pad+light%5],int(light/5), bitRead(red, 0));
    bitWrite(red1[5*pad+light%5],int(light/5), bitRead(red, 1));
    bitWrite(red2[5*pad+light%5],int(light/5), bitRead(red, 2));
    bitWrite(red3[5*pad+light%5],int(light/5), bitRead(red, 3));

    bitWrite(green0[5*pad+light%5],int(light/5), bitRead(green, 0));
    bitWrite(green1[5*pad+light%5],int(light/5), bitRead(green, 1));
    bitWrite(green2[5*pad+light%5],int(light/5), bitRead(green, 2));
    bitWrite(green3[5*pad+light%5],int(light/5), bitRead(green, 3));

    bitWrite(blue0[5*pad+light%5],int(light/5), bitRead(blue, 0));
    bitWrite(blue1[5*pad+light%5],int(light/5), bitRead(blue, 1));
    bitWrite(blue2[5*pad+light%5],int(light/5), bitRead(blue, 2));
    bitWrite(blue3[5*pad+light%5],int(light/5), bitRead(blue, 3));
}
*/
用以编辑当前 LED 应有的显示情况。
其中:
    pad 范围为 0 - PADS-1, 表示的是板子顺序;
    light 范围为 0-39, 表示的是板子上由下端到上端 (有孔端) 的灯序号;
    red, green, blue 范围为 0-15, 表示的是 4-bit 下的 rgb 亮度。
*/
ISR(TIMER2_COMPB_vect)
{
    PORTD |= 1 << blank_pin;
    if (trans_count == 5)
        trans_bit++;
    else if (trans_count == 15)
        trans_bit++;
    else if (trans_count == 35)
        trans_bit++;

    trans_count++;

    for (int pad = 0; pad < 1; pad++)
    {
        switch (trans_bit) {
            case 0:SPI.transfer(*(red[0]+nodelevel+5*pad));SPI.transfer(*(green[0]+nodelevel+5*pad));SPI.transfer(*(blue[0]+nodelevel+5*pad));
                break;
            case 1:SPI.transfer(*(red[1]+nodelevel+5*pad));SPI.transfer(*(green[1]+nodelevel+5*pad));SPI.transfer(*(blue[1]+nodelevel+5*pad));
                break;
        }
    }
}

```



```

        }
    }
}

else if(refreshnow==1 || !refreshnow)//如果缓冲区中没有或者仅有 1 个已经加载好的模块,
则将加载的速度翻倍
{
    bufferkey=(buffernow+refreshnow+1);
    if(bufferkey>=BUFFERNUM)
        bufferkey-=BUFFERNUM;
    int8_t i=0;
    for(i=0;i<5;i++)
        buffer[bufferkey][bufferprocess][i]=pgm_read_byte(bufferaddr[0]+i);
    for(i=0;i<5;i++)
        buffer[bufferkey][bufferprocess][5+i]=pgm_read_byte(bufferaddr[1]+i);
    for(i=0;i<5;i++)
        buffer[bufferkey][bufferprocess][10+i]=pgm_read_byte(bufferaddr[2]+i);
    bufferprocess++;
    bufferaddr[0]+=5;
    bufferaddr[1]+=5;
    bufferaddr[2]+=5;
    for(i=0;i<5;i++)
        buffer[bufferkey][bufferprocess][i]=pgm_read_byte(bufferaddr[0]+i);
    for(i=0;i<5;i++)
        buffer[bufferkey][bufferprocess][5+i]=pgm_read_byte(bufferaddr[1]+i);
    for(i=0;i<5;i++)
        buffer[bufferkey][bufferprocess][10+i]=pgm_read_byte(bufferaddr[2]+i);
    bufferprocess++;
    bufferaddr[0]+=5;
    bufferaddr[1]+=5;
    bufferaddr[2]+=5;
    if(bufferprocess!=12)//所有模块加载过程的最小单元速度皆为 2 个 bufferprocess, 此处
的检测可以避免发生不对齐的情况
    {
        for(i=0;i<5;i++)
            buffer[bufferkey][bufferprocess][i]=pgm_read_byte(bufferaddr[0]+i);
        for(i=0;i<5;i++)
            buffer[bufferkey][bufferprocess][5+i]=pgm_read_byte(bufferaddr[1]+i);
        for(i=0;i<5;i++)
            buffer[bufferkey][bufferprocess][10+i]=pgm_read_byte(bufferaddr[2]+i);
        bufferprocess++;
        bufferaddr[0]+=5;
        bufferaddr[1]+=5;
        bufferaddr[2]+=5;
        for(i=0;i<5;i++)
            buffer[bufferkey][bufferprocess][i]=pgm_read_byte(bufferaddr[0]+i);
        for(i=0;i<5;i++)
            buffer[bufferkey][bufferprocess][5+i]=pgm_read_byte(bufferaddr[1]+i);
        for(i=0;i<5;i++)
            buffer[bufferkey][bufferprocess][10+i]=pgm_read_byte(bufferaddr[2]+i);
        bufferprocess++;
        bufferaddr[0]+=5;
        bufferaddr[1]+=5;
        bufferaddr[2]+=5;
    }

    if(bufferprocess==12)
    {
        bufferprocess=0;
        refreshnow++;
        for(int temp=0;temp<3;temp++)
            if(bufferaddr[temp]>=imgover)
            {
                int temp1=bufferaddr[temp]-imgover;
                bufferaddr[temp]=imgnow+temp1;
            }
    }
}

```

```

        }
    }
    else if(refreshnow!=BUFFERNUM-1)
        //如果加载进度为 7, 也就是除了当前使用的部分全都加载好了, 则不予加载;
        //如果加载进度不在应有的 0-7 的范围之内, 则清空加载进度, 并要显示的区块加紧加载完成, 防
        止显示出现错误
    {
        refreshnow=0;
        if(anglenow<60)
        {
            bufferaddr[0]=imgnow+anglenow*60;
            bufferaddr[1]=bufferaddr[0]+3600;
            bufferaddr[2]=bufferaddr[1]+3600;
        }
        else if(anglenow<120)
        {
            bufferaddr[0]=imgnow+anglenow*60;
            bufferaddr[1]=bufferaddr[0]+3600;
            bufferaddr[2]=bufferaddr[0]-3600;
        }
        else
        {
            bufferaddr[0]=imgnow+anglenow*60;
            bufferaddr[2]=bufferaddr[0]-3600;
            bufferaddr[1]=bufferaddr[2]-3600;
        }
        int8_t i=0;int8_t j=0;
        for(j=0;j<12;j++)
        for(i=0;i<5;i++)
            buffer[buffernow][j][i]=pgm_read_byte(bufferaddr[0]+i+5*j);
        for(j=0;j<12;j++)
        for(i=0;i<5;i++)
            buffer[buffernow][j][5+i]=pgm_read_byte(bufferaddr[1]+i+5*j);
        for(j=0;j<12;j++)
        for(i=0;i<5;i++)
            buffer[buffernow][j][10+i]=pgm_read_byte(bufferaddr[2]+i+5*j);
        for(int temp=0;temp<3;temp++)
        {
            bufferaddr[temp]+=60;
            if(bufferaddr[temp]>=imgover)
            {
                bufferaddr[temp]=imgnow+int(bufferaddr[temp]-imgover);
            }
        }
    }
    //completed
    else if(anglenow>pre_angle)
    {
        if(anglenow<=pre_angle+refreshnow&&refreshnow<BUFFERNUM)
        {
            buffernow=anglenow-pre_angle+buffernow;
            while(buffernow>=BUFFERNUM)
                buffernow=buffernow-BUFFERNUM;
            for(int tmp=0;tmp<4;tmp++)
            {
                red[tmp]=*(*(buffer+buffernow)+tmp);
                green[tmp]=*(*(buffer+buffernow)+tmp+4);
                blue[tmp]=*(*(buffer+buffernow)+tmp+8);
            }
            refreshnow=pre_angle+refreshnow-anglenow;
            pre_angle=anglenow;
        }
        else
        {
            refreshnow=0;
        }
    }
}

```

```

        buffernow++;
        if(buffernow>=BUFFERNUM)
            buffernow-=BUFFERNUM;
    if(anglenow<60)
    {
        bufferaddr[0]=imgnow+anglenow*60;
        bufferaddr[1]=bufferaddr[0]+3600;
        bufferaddr[2]=bufferaddr[1]+3600;
    }
    else if(anglenow<120)
    {
        bufferaddr[0]=imgnow+anglenow*60;
        bufferaddr[1]=bufferaddr[0]+3600;
        bufferaddr[2]=bufferaddr[0]-3600;
    }
    else
    {
        bufferaddr[0]=imgnow+anglenow*60;
        bufferaddr[2]=bufferaddr[0]-3600;
        bufferaddr[1]=bufferaddr[2]-3600;
    }
    int8_t i=0,int8_t j=0;
    for(j=0;j<12;j++)
    for(i=0;i<5;i++)
        buffer[buffernow][j][i]=pgm_read_byte(bufferaddr[0]+i+5*j);
    for(j=0;j<12;j++)
    for(i=0;i<5;i++)
        buffer[buffernow][j][5+i]=pgm_read_byte(bufferaddr[1]+i+5*j);
    for(j=0;j<12;j++)
    for(i=0;i<5;i++)
        buffer[buffernow][j][10+i]=pgm_read_byte(bufferaddr[2]+i+5*j);
    for(int temp=0;temp<3;temp++)
    {
        bufferaddr[temp]+=60;
        if(bufferaddr[temp]>=imgover)
        {
            bufferaddr[temp]=imgnow+int(bufferaddr[temp]-imgover);
        }
    }
    for(int tmp=0;tmp<4;tmp++)
    {
        red[tmp]=*(*(buffer+buffernow)+tmp);
        green[tmp]=*(*(buffer+buffernow)+tmp+4);
        blue[tmp]=*(*(buffer+buffernow)+tmp+8);
    }
    pre_angle=anglenow;
}
}
else
{
    if(anglenow+180<=pre_angle+refreshnow&&refreshnow<BUFFERNUM)
    {
        buffernow=anglenow+180-pre_angle+buffernow;
        while(buffernow>=BUFFERNUM)
            buffernow=buffernow-BUFFERNUM;
        for(int tmp=0;tmp<4;tmp++)
        {
            red[tmp]=*(*(buffer+buffernow)+tmp);
            green[tmp]=*(*(buffer+buffernow)+tmp+4);
            blue[tmp]=*(*(buffer+buffernow)+tmp+8);
        }
        refreshnow=pre_angle-180+refreshnow-anglenow;
        pre_angle=anglenow;
    }
    else
    {

```

```

refreshnow=0;
buffernow++;
if(buffernow>=BUFFERNUM)
buffernow-=BUFFERNUM;
if(anglenow<60)
{
    bufferaddr[0]=imgnow+anglenow*60;
    bufferaddr[1]=bufferaddr[0]+3600;
    bufferaddr[2]=bufferaddr[1]+3600;
}
else if(anglenow<120)
{
    bufferaddr[0]=imgnow+anglenow*60;
    bufferaddr[1]=bufferaddr[0]+3600;
    bufferaddr[2]=bufferaddr[0]-3600;
}
else
{
    bufferaddr[0]=imgnow+anglenow*60;
    bufferaddr[2]=bufferaddr[0]-3600;
    bufferaddr[1]=bufferaddr[2]-3600;
}
int8_t i=0,int8_t j=0;
for(j=0;j<12;j++)
for(i=0;i<5;i++)
    buffer[buffernow][j][i]=pgm_read_byte(bufferaddr[0]+i+5*j);
for(j=0;j<12;j++)
for(i=0;i<5;i++)
    buffer[buffernow][j][5+i]=pgm_read_byte(bufferaddr[1]+i+5*j);
for(j=0;j<12;j++)
for(i=0;i<5;i++)
    buffer[buffernow][j][10+i]=pgm_read_byte(bufferaddr[2]+i+5*j);
for(int temp=0;temp<3;temp++)
{
    bufferaddr[temp]+=60;
    if(bufferaddr[temp]>=imgover)
    {
        bufferaddr[temp]=imgnow+int(bufferaddr[temp]-imgover);
    }
}
for(int tmp=0;tmp<4;tmp++)
{
    red[tmp]=*(*(buffer+buffernow)+tmp);
    green[tmp]=*(*(buffer+buffernow)+tmp+4);
    blue[tmp]=*(*(buffer+buffernow)+tmp+8);
}
pre_angle=anglenow;
}
}

//**(*(*(*img+IMGMEM)+(60*pad+int(pre_angle/2))%180)+led)+0)
//用于检测角度来更改当前应有的输出
void purecolor(int r,int g,int b)
{
    for(int i=0;i<PADS;i++)
    for(int j=0;j<40;j++)
        LED(i,j,r,g,b);
}
//用于让所有灯显示r, g, b 下的纯色
void clean()
{
    for(int i=0;i<PADS;i++)
    for(int j=0;j<40;j++)

```

```
    LED(i,j,0,0,0);
}
//用于让所有灯灭掉清零
void testrgb()
{
    purecolor(15,0,0);
    delay(5000);
    purecolor(0,15,0);
    delay(5000);
    purecolor(0,0,15);
    delay(5000);
    purecolor(15,15,15);
    delay(5000);
    clean();
    delay(1000);
}
//用于测量各个 rgb 引脚及芯片连接情况
void testshow()
{
    for(int i=0;i<PADS;i++)
        for(int j=0;j<39;j++)
    {
        LED(i,j,0,0,15);
        delay(50);
    }
    for(int i=0;i<PADS;i++)
        for(int j=0;j<39;j++)
    {
        LED(i,j,0,0,0);
        delay(50);
    }
    for(int i=0;i<PADS;i++)
        for(int j=0;j<40;j++)
    {
        LED(i,j,0,15,0);
        delay(50);
    }
    for(int i=0;i<PADS;i++)
        for(int j=0;j<4;j++)
    {
        LED(i,j,0,0,0);
        delay(50);
    }
    for(int i=0;i<PADS;i++)
        for(int j=0;j<4;j++)
    {
        LED(i,j,15,0,0);
        delay(50);
    }
    for(int i=0;i<PADS;i++)
        for(int j=0;j<4;j++)
    {
        LED(i,j,0,0,0);
        delay(50);
    }
    for(int i=0;i<PADS;i++)
        for(int j=0;j<4;j++)
    {
        LED(i,j,15,15,15);
        delay(50);
    }
    for(int i=0;i<PADS;i++)
        for(int j=0;j<4;j++)
    {
```

```

        LED(i,j,0,0,0);
        delay(50);
    }
}

//滚动播放测试亮灯情况
void colordefine()
{
    int r=0;
    int g=0;
    int b=0;
    int i=0;
    int j=0;
    while(b<16)
    {
        if(j>39)
        {
            j=0;delay(5000);
        }
        if(r>15)
        {
            r=0;
            if(g) g+=2;
            else g++;
        }
        if(g>15)
        {
            g=0;
            if(b) b+=2;
            else b++;
        }
        if(i>PADS-1)
        {
            i=0;
            delay(490);
            clean();
            delay(10);
        }
        LED(i,j,r,g,b);
        if(r) r+=2;
        else r++;
        j+=3;
    }
}

//用于遍历(为了减少工作量将档位压缩为 0,1,3,5,7,9,11,13,15 这九种)显示颜色的所有情况, 用以校准rgb转化情况
void showclr()
{
    for(int i=0;i<40;i++)
        LED(0,i,15,0,0);
    delay(1000);

    for(int i=0;i<40;i++)
        LED(0,i,0,15,0);
    delay(1000);

    for(int i=0;i<40;i++)
        LED(0,i,0,0,15);
    delay(1000);

    for(int i=0;i<40;i++)
        LED(0,i,15,15,0);
    delay(1000);
}

```

```

for( int i=0;i<40;i++)
    LED(0,i,15,0,15);
delay(1000);

for( int i=0;i<40;i++)
    LED(0,i,15,11,0);
delay(1000); //这是我校正过的黄色

for( int i=0;i<40;i++)
    LED(0,i,0,15,15);
delay(1000);

for( int i=0;i<40;i++)
    LED(0,i,15,15,15);
delay(1000);

for( int i=0;i<40;i++)
    LED(0,i,15,11,11);
delay(1000); //猜测校正后的白色

for( int i=0;i<40;i++)
    LED(0,i,0,0,0);
delay(1000);
}

//用于显示三原色和基本混合色
int rgbtrans(int value)
{
    return ((value%16)/8+value/16);
}

```

第九部分：参考资料

1、CSDN 博客《Arduino 详细讲解和资料》。

<https://blog.csdn.net/hlitt3838/article/details/109128925>

2、CSDN 博客《如何在 Arduino Uno 中使用移位寄存器 74HC595》

https://blog.csdn.net/acktomas/article/details/116127820?ops_request_misc=%257B%2522request%2522%253A%2522163126236816780271525130%2522%2522C%2522scm%2522%253A%252220140713.130102334.%2522%257D&request_id=163126236816780271525130&biz_id=0&utm_medium=distribute.pc_search_result.none-task-blog-2~all~sobaiduend~default-2-116127820.pc_search_result_control_group&utm_term=arduino+uno%E5%AF%84%E5%AD%98%E5%99%A8&spm=1018.2226.3001.4187

3、CSDN 博客《超全 Python 图像处理讲解！花五天才整理的！」》

https://blog.csdn.net/weixin_43881394/article/details/105730236

4、CSDN 博客《超全 Python 图像处理讲解！花五天才整理的！」》

https://blog.csdn.net/weixin_43881394/article/details/105730236

5、CSDN 博客《Arduino MPU6050 学习资料总结》

https://blog.csdn.net/u010006102/article/details/46667603?utm_medium=distribute.pc_aggpage_search_result.none-task-blog-2~aggregatepage~first_rank_v2~rank_aggregation-1-46667603.pc_agg_rank_aggregation&utm_term=arduino+mpu6050+%E8%A7%92%E5%BA%A6&spm=1000.2123.3001.4430

6、位运算相关知识学习：

<https://www.runoob.com/w3cnote/bit-operation.html>

7、B 站课程：Kalman 滤波知识学习：

https://www.bilibili.com/video/BV1ez4y1X7eR?from=search&seid=6698134637322287149&spm_id_from=333.337.0.0

8、《TIMER INTERRUPTIONS》

https://electrooobs.com/eng_arduino_tut140.php

<https://blog.csdn.net/hltt3838/article/details/109128925>