Matlab 高级编程与工程应用
图像处理大作业实验报告

姓名　　王禹

学号　　2014011241

班级　　无 48

日期　二〇一六年 八月 二十五日

目录

1 原创性声明

本实验完全采用原创设计代码，仅在自主设计 JPEG 信息隐藏的时候，参考了李思涵同学的思想。

2 实验目的

- 了解计算机存储和处理图像的基础知识；
- 掌握 JPEG 标准的基本原理；
- 变化域编码和量化的基本思想；
- MATLAB 处理矩阵和图像的常用命令；
- 在变换域进行信息隐藏的方法；
- 学习人脸检测的基本方法。

3 基础知识

3.1 MATLAB 提供了图像处理工具箱，请阅读并大致了解这些函数的基本功能

3.2 利用 MATLAB 提供的 Image file I/O 函数分别完成以下处理：

3.2.1 以测试图像的中心为圆心，图像的长和宽中较小值的一半为半径画一个红色的圆；

图像的读写 IO 主要依靠 MATLAB 自带的图像处理工具箱的 imread 和 imwrite 函数。将图像读入后，彩色图像为三维矩阵，灰度矩阵为二维矩阵。而本实验中，则是直接 load 已经准备好的.mat 文件，获得图像矩阵。彩色图像的前两维分别为高和宽的像素值，第三维按照顺序为 RGB，矩阵的值为 0 255 的 RGB 三色的亮度值；灰度图像则没有第三维，只有一个灰度亮度值的二维矩阵。第一问直接获得了图像的矩阵后，找到距离中心小于所指定半径的所有点，将其 R 分量值调为 255，G 和 B 分量都调为 0，即可得到所需的图像，最终的结果如下图 Figure 1a所示，而代码如 Listing 1所示。

3.2.2 将测试图像涂成国际象棋状的“黑白格”的样子，其中“黑”即为黑色，“白”则意味着保留原图。

对于分割为棋盘，为了判断是涂黑还是保持原样，只需判断所要处理的格子的横纵坐标序列数之和的奇偶性即可。结果图见 Figure 1b, 具体代码见 Listing 2。

(a) 红圆 (b) 棋格

Figure 1: 题 3.1 结果图

```
clear all; close all;
load hall.mat
r = floor(min(size(hall_color(:,:,1)))/2);
center = floor(size(hall_color)/2);
temp_hall = im2double(hall_color);
for i = center(1)-r+1:1:center(1)+r
    for j = 1:size(hall_color(:,:,1),2)
        if((i - center(1))^2 + (j - center(2))^2 <= r^2)
            temp_hall(i,j,1)=255;
            temp_hall(i,j,2)=0;
            temp_hall(i,j,3)=0;
        end
    end
end

imshow(im2uint8(temp_hall));
imwrite(im2uint8(temp_hall),'a.jpg');
```

Listing 1: 画红色圆代码

```
clear all; close all;
load hall.mat
len = size(hall_color);
temp_hall = im2double(hall_color);

for i = 1:len(1)/3
    for j = 1:len(2)/3
        if(mod(i+j,2))
            temp_hall(3*i-2:3*i,3*j-2:3*j,1:3)=0;
        end
    end
end

imshow(im2uint8(temp_hall));
imwrite(im2uint8(temp_hall),'b.jpg');
```

Listing 2: 涂棋格代码

4 图像压缩编码

4.1 图像的预处理是将每个像素灰度值减去 128，这个步骤是否可以在变换域进行？

由于变换域的第一个分量便是直流分量，因此这个步骤可以在变换域进行。块是 8×8 的像素块，他的 DC 分量基底为 $\frac{1}{8}$，因此将 DCT 变换后的直流分量减去 128 即可得到预处理图像。具体代码见 Listing 3。

根据结果来看，结果相差为 10^{-13} 数量级。

```
clear all; close all;
load hall.mat

part = hall_gray(11:18,21:28);
a1 = dct2(part);
a1(1) = a1(1) -128/(1/8);
a2 = dct2(part-128);
```

```
8
9  ans = norm(a1-a2)
10
11 % ans = 3.5762e-13
```

Listing 3: 涂棋格代码

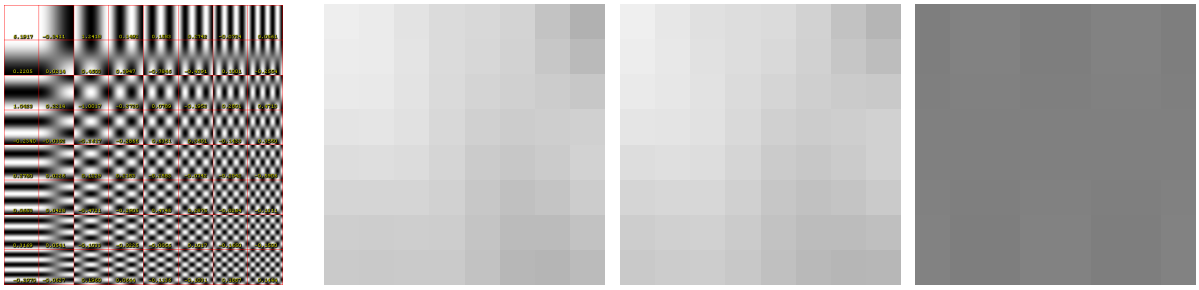
4.2 请编程实现二维 DCT，并和 MATLAB 自带的库函数 dct2 比较是否一致。

根据实验指导书前述的二维 DCT 原理，实现 DCT 的基底矩阵后，利用矩阵相乘的性质取得 dct 系数。经过对比后，范数相差仅为 10^{-12} 数量级，可能是由计算中的近似而产生的，因此可以认为自行实现的二维 DCT 变换和自带库函数 dct2 是一致的。程序代码参见 Listing 4。

```
1  clear all; close all;
2  load hall.mat
3
4  part = hall_gray(11:18,21:28);
5  N = 8;
6
7  D = zeros(N,N);
8  for i = 1:N
9      for j = 1:N
10         D(i,j) = cos(pi*(i-1)*(2*j-1)/(2*N));
11     end
12 end
13 D(1,:) = sqrt(1/2);
14 D = D*sqrt(2/N);
15
16 C_my = D*double(part)*D';
17 C = dct2(part);
18
19 norm(C_my-C)
20
21 % ans = 1.0899e-12
```

Listing 4: 涂棋格代码

4.3 如果将 DCT 系数中右侧四列的系数全部置零，逆变换后图像会发生什么变化？选取一块图像证明你的结论。如果左边四列置零呢？



(a) 二维 DCT 变换基底图像[1] (b) 原始图像（左）、右边四列置零（中）、左边四列置零（右）

Figure 2: 题 4.3 结果图

从 Figure 2a [1] 中可以看出 dct 系数右侧四列基底在横向上都有高频变化，而左边四列在横向上有低频变化。因此当右侧四列置零时，图像中的横向高频分量丢失，反之当左侧四列置零时，图像中横向低频分量丢失。

测试结果如图 Figure 2b所示。由于我所选择的图片在横向上没有高频变化，主要集中在低频变化上。因此右侧四列原本的值即较小，置零后对图片没有较大的影响，而左侧四列置零后，对图片的影响较大。如结果图所示，左侧四列置零后，整个图片都是黑蒙蒙的一片。

具体代码如 Listing 5所示。

```
clear all; close all;
load hall.mat

part = hall_gray(11:18,21:28);

C = dct2(part-uint8(128));

C1 = C; C2 = C;
C1(:,5:8) = 0;
C2(:,1:4) = 0;

rightzeropart = uint8(idct2(C1) + 128);
leftzeropart = uint8(idct2(C2) + 128);

imshow(part);
imwrite(part,'origin.jpg');
figure();
imshow(rightzeropart);
imwrite(rightzeropart,'rightzero.jpg');
figure();
imshow(leftzeropart);
imwrite(leftzeropart,'leftzero.jpg')
```

Listing 5: 涂棋格代码

4.4 若对 DCT 系数分别转置、旋转 90 度和旋转 180 度操作 (rot90)，逆变换后恢复的图像有何变化？

若对 DCT 系数进行转置，则横纵分量的系数发生互换，因此恢复出来的原图像发生了转置。

若对 DCT 系数逆时针旋转 90°，则大部分能量集中到左下角。而左下角系数代表的是横向低频变化、纵向高频变化，因此恢复出来的图像应当在横向上是低频变化，而纵向上呈现量化高频变化。

若对 DCT 系数逆时针旋转 180°，则大部分能量集中到右下角。而右下角系数代表的横向纵向都是高频变化，因此恢复出来的在横纵向上都有量化高频变化。

最终恢复出来的结果如图 Figure 3所示，与理论匹配。具体的代码如 Listing 6所示。

(a) 原始图像 (b) 转置图像 (c) 旋转 90° (d) 旋转 180°

Figure 3: 题 4.4 结果图

```
clear all; close all;
load hall.mat

part = hall_gray(11:18,21:28);
C = dct2(part - uint8(128));

C0 = C'; C1 = rot90(C); C2 = rot90(C1);
transpose = uint8(idct2(C) + 128);
rightrot90 = uint8(idct2(C1) + 128);
rightrot180 = uint8(idct2(C2) + 128);

imshow(part);
imwrite(part,'origin_rot.jpg');
figure();
imshow(transpose);
imwrite(transpose,'transpose.jpg');
figure();
imshow(rightrot90);
imwrite(rightrot90,'rightrot90.jpg');
figure();
imshow(rightrot180);
imwrite(rightrot180,'rightrot180.jpg');
```

Listing 6: 涂棋格代码

4.5 如果认为差分编码是一个系统，请绘出这个系统的频率响应，说明他是怎样的滤波器。DC 系统先进行差分编码在进行熵编码，说明他是一个怎样的系统。

不考虑初值，差分方程为：

$$\hat{c}_D(n) = c_D(n-1) - c_D(n)$$

即 $A = [1], B = [-1\ 1]$，直接使用 MATLAB 的 freqz 函数即可画出频率响应，即 Figure 4。代码如 Listing 7所示。

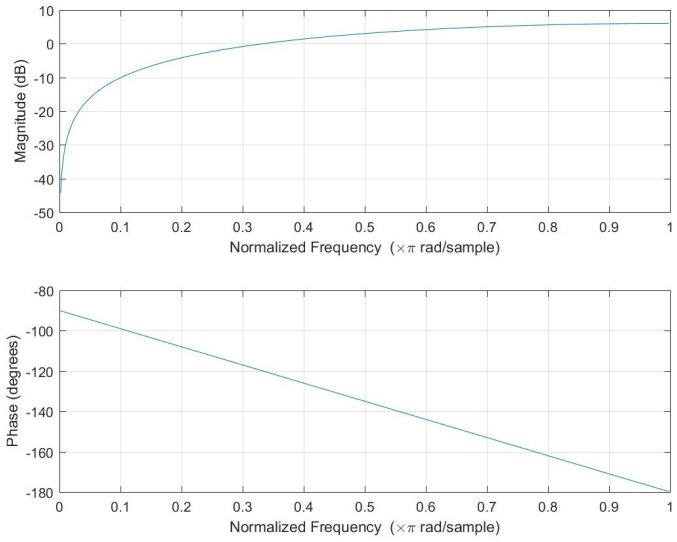


Figure 4: 频率响应

```
freqz([-1 1], 1);
saveas(gca, 'frequency_plot.jpg');
```

Listing 7: 频率响应代码

因此这是一个高通系统。而 DC 系统先进行差分编码再进行熵编码说明 DC 系统的低频分量较多，故经过系统后能量会有很大的衰减，从而实现信息压缩。

4.6 DC 预测误差取值和 Category 值有何关系？如何利用预测误差计算出其 Category？

从表中不难观察出，DC 误差的取值所对应的二进制长度即为 Category 值。用数学表达式表述为：

$$Category = ceil(log_2(|value| + 1))$$

4.7 你知道哪些实现 Zig-Zag 扫描方法？请利用 MATLAB 强大功能设计一种最佳方法。

共有两种方法：

第一种方法是知已将表格制作好，然后直接存在 MATLAB 中。这种方法需要自己实现编写好已知位数的 zig-zag 顺序列表，较为笨拙，但是实际程序运行时效率较高。本实验中采用的便是这种方法，但是效率较高。具体代码见 Listing 8。

```
function index = zigzag(r)
    if r ~= 8
        error(['Please input 8!!!!'])
    end
    index =[1;9;2;3;10;17;25;18;11;4;5;12;19;26;33;41;34;27;20;13;6;7;...
        14;21;28;35;42;49;57;50;43;36;29;22;15;8;16;23;30;37;44;51;58;...
        59;52;45;38;31;24;32;39;46;53;60;61;54;47;40;48;55;62;63;56;64];
```

Listing 8: zigzag 代码

第二种方法是设置较为复杂的边界条件，每次碰到边界则横移转向/数移转向。这样的方法可以方便的生成任意尺寸的 zig-zag 序列，但是每次生成一次序列所需的时间较长，效率较低。因此我采用的是前一种 zig-zag 生产方法。

4.8 对测试图像分块、DCT 和量化，将量化后的系数写成矩阵的形式，其中每一列为一个块的 DCT 系数 Zig-Zag 扫描后形成的列矢量，第一行为各个块儿的 DC 系数。

有了上述的 Zig-Zag 列后，可以很容易的将 DCT 系数按顺序变为一个列向量，然后放入矩阵中，代码清单 Listing 10中的代码可以实现这个目的。

```
clear all; close all;

load hall.mat;
load JpegCoeff.mat;

hwlen = size(hall_gray);
newhwlen = ceil(hwlen/8)*8;
newimg = zeros(newhwlen);

newimg(1:hwlen(1),1:hwlen(2)) = hall_gray;
```

```
11 if(hwlen(1) < newhwlen(1))
12     newimg(hwlen(1)+1:newhwlen(1),:) = hall_gray(hwlen(1),:);
13 end
14 if(hwlen(2) < newhwlen(2))
15     newimg(:,hwlen(2)+1:newhwlen(2)) = hall_gray(:,hwlen(2));
16 end
17
18 coef = zeros(64,newhwlen(1)*newhwlen(2)/64);
19
20 for i = 1:newhwlen(1)/8
21     for j = 1:newhwlen(2)/8
22         c = dct2(newimg(i:i+7,j:j+7)-128);
23         c = round(c ./ QTAB);
24         coef(:,(i-1)*8+j) = c(zigzag(8));
25     end
26 end
```

Listing 9: coef 矩阵代码

4.9 请实现 JPEG 编码，输出为 DC 系数的码流、AC 系数的码流、图像高度和图像宽度，将这四个变量写入 jpegcodes.mat 文件

为实现 JPEG 编码需要按照分块、DCT、量化、熵编码的顺序进行编码。

4.9.1 分块、DCT、量化

为了实现分块，需要原图像的宽和高的像素数目都为 8 的倍数。本大作业使用的 hall_gray 的宽高都恰为 8 的倍数，若宽高不为高的倍数时，将其扩展到离他最近的 8 的倍数的宽高。新增的像素点用与其相邻的左侧的像素点或上侧的像素点填充。

获得新的图像后，便可以进行 8×8 的分块了与 DCT 变换了。修改图片像素尺寸的代码见 Listing 10 10-20 行。

而获得了 8×8 的分块后便可以直接进行 DCT 变换。为了实现量化，使用的是./的运算，之后使用 zigzag(8) 的序列，将其放入矩阵中。这部分代码见 Listing 10 20-26 行。

4.9.2 DC 熵编码

DC 系数即为 coef 矩阵的第一行。首先我们要对这一行系数做一个差分处理，从而消除低频分量达到压缩的目的。差分部分的代码见 Listing 10 30-36 行。差分处理完成后将

```
1  clear all; close all;
2
3  load hall.mat;
4  load JpegCoeff.mat;
5
6  hwlen = size(hall_gray);
7  newhwlen = ceil(hwlen/8)*8;
8  newimg = zeros(newhwlen);
9
10 newimg(1:hwlen(1),1:hwlen(2)) = hall_gray;
11 if(hwlen(1) < newhwlen(1))
12     newimg(hwlen(1)+1:newhwlen(1),:) = hall_gray(hwlen(1),:);
13 end
14 if(hwlen(2) < newhwlen(2))
```

```
    newimg(:,hwlen(2)+1:newhwlen(2)) = hall_gray(:,hwlen(2));
end

coef = zeros(64,newhwlen(1)*newhwlen(2)/64);

for i = 1:newhwlen(1)/8
    for j = 1:newhwlen(2)/8
        c = dct2(newimg(8*i-7:8*i,8*j-7:8*j)-128);
        c = round(c ./ QTAB);
        coef(:,(i-1)*newhwlen(2)/8+j) = c(zigzag(8));
    end
end

%% DC coef

DC = coef(1,:);
% DC = [10 8 60];
Diff = zeros(size(DC));
Diff(1) = DC(1);
for i = 2:length(Diff)
    Diff(i) = DC(i-1) - DC(i);
end
Dc_ceof = [];
Category_DC = ceil(log2(abs(Diff)+1));
for i = 1:length(Category_DC)
    huff = DCTAB(Category_DC(i)+1, 2:1+DCTAB(Category_DC(i)+1,1));
    dc = Diff(i);
    if(dc==0)
        Dc_ceof = [Dc_ceof huff];
    else
        amp = dec2bin(abs(dc)) - '0';
        if(dc<0)
            amp = 1 - amp;
        end
        Dc_ceof = [Dc_ceof huff amp];
    end
end

%% AC coef
AC = coef(2:64,:);
% AC = [10 3 0 0 2 zeros(1,20) 1 zeros(1,37)]';
Ac_ceof = [];

for i = 1:size(AC,2);
    currentAC = AC(:,i);
    pos = find(currentAC~=0);
    if(~isempty(pos))
        previous = 1;
        for j = 1:length(pos)
            ac = currentAC(pos(j));
            run = pos(j)-previous;
            while(run>15)
                Ac_ceof = [Ac_ceof 1 1 1 1 1 1 1 1 0 0 1];
                run = run - 16;
            end
            amp = dec2bin(abs(ac)) - '0';
            if(ac<0)
```

```
                amp = 1 - amp;
            end
            Huff = ACTAB(run*10+length(amp),4:ACTAB(run*10+length(amp),3)+3);
            Ac_ceof = [Ac_ceof Huff amp];
            previous = pos(j) + 1;
        end
    end
    Ac_ceof = [Ac_ceof 1 0 1 0];
end
height = newhwlen(1);
width = newhwlen(2);
save jpegcodes.mat Dc_ceof Ac_ceof height width
```

Listing 10: JPEG 编码代码

4.10 计算压缩比

asdasd

4.11 请实现 JPEG 解码，输入是你生成的 jpegcodes.mat 文件。分别用客观 (PSNR) 和主观方法评价编解码效果如何。

asdasd

4.12 将量化步长减小为原来一半，重做编解码。同标准量化步长的情况比较压缩比和图像质量。

asdasd

4.13 看电视时偶尔能看到美丽的雪花图像 (见 snow.mat)，请对其编解码。和测试图像的压缩比和图像质量进行比较，并解释比较结果。

asdasd

5 信息隐藏

5.1 实现本章介绍的空域隐藏方法和提取方法。验证其抗 JPEG 编码能力。

asdasd

ALU 部分没有经过硬件调试，由于是组合逻辑电路，仿真通过后，便不会产生较大的硬件错误。本部分是一次编写代码便成功通过，且直接嵌入到处理器中进行使用，故本部分无硬件调试过程。

5.2 依次实现本章介绍的三种变换域信息隐藏方法和提取方法，分析嵌密方法的隐蔽性以及嵌密后 JPEG 图像的质量变化和压缩比变化

MIPS 单周期处理器在最开始用 Vivado 进行综合与实现时，总是显示不能满足时序要求。我刚开始不太明白如何处理这一错误，仍然直接将 bit 文件烧录进 FPGA 中，结果是 CPU 不能正常的工作。之后了解到，我们的 FPGA 的硬件性能有限，不能够直接接入 100M 的时钟晶振作为单周期处理器的主频工作。在将

100M 的晶振经过二分频得到的 50M 时钟接入后，Vivado 便能够正常的进行综合与仿真而不再报错了。由于最终验收时，不考虑单周期处理器的最高主频，因此也没有对单周期处理器的主频最高性能进行测试。

单周期处理器另一个调试的问题在控制模块的编写上。由于控制模块所控制的信号太多，信号名称的大小写最开始在各个模块中没能匹配，导致硬件实现出错。在后续的调试中，发现了这个问题所在，然后将其修复。

其他的模块没有太多的问题需要硬件调试。

5.3 请设计实现新的隐藏算法并分析其优缺点

流水线的硬件调试给予我们的最大启示是，verilog 语言对每个模块都有固定的写法需要遵守，尤其是逻辑判断。具体情况是：单个条件若是一位信号则直接用该信号（或取反），若是多位信号则用 == 或!= 目标值；条件之间的运算用按位或、与。流水线硬件调试一开始到处报错，与 ModelSim 调试结果大相径庭。按照上述原则修改之后便一次性通过了。

6 人脸识别

6.1 所给资料 Faces 目录下包含从网络中截取的 28 张人脸，试以其作为样本训练人脸标准 v

6.1.1 样本人脸大小不一致，是否需要首先将图像调整为相同大小

asdasd

6.1.2 假设 L = 3,4,5, 所得的三个 v 之间有什么关系？

asdasd

6.2 设计一种从任意大小的图片中检测任意多张人脸的算法并编程实现 (输出图像在判定为人脸的位置加上红色的方框)。随意选取一张多人的照片，对程序进行测试。尝试 L 分别取不同的值，评价检测结果的区别。

asdasd

6.3 对上述图像分别进行如下处理后

6.3.1 顺时针旋转 90 度

asdasd

6.3.2 保持高度不变，宽度变为原来的 2 倍

asdasd

6.3.3 适当改变颜色

asdasd

6.4 如果可以重新选择人脸样本的训练标准，你觉得应该如何选取？

asdasd

7 实验总结

以下为我组三名同学的实验总结：

通过本次实验，我将在数字逻辑与处理器基础理论课上学习到的 MIPS 周期处理器理论，在 FPGA 上用硬件将其实现。我在小组实现 CPU 中，负责的是 ALU 和单周期 MIPS 的编写。

对于具体的编写实现，ALU 部分的编写方式和理论课有所不同。在理论课上，我并没有将 ALU 分为四个部分分别实现，而是将整体进行编写。而实验指导书要求将 ALU 分四个部分。在实现层次编写后，我认为这样的编写方式，使得整体的结构更为清晰。而对于四个部分的具体代码，我觉得最有趣的部分应该是比较器的代码。因为比较器内部没有实际的使用比较逻辑判断符，而是直接使用由加法器计算产生的结构进行判断与比较。这是我觉得 ALU 编写过程中最有趣的部分。

而在单周期处理器的实现中，大部分内容和理论课所讲授的一致。但是外设的编写是一大难点。最开始并没有理解外设的具体实现方法，后来经过慢慢的摸索和理解，了解到外设可以认为是一段新的内存，用以存储和外部部件相关的数据，包括要显示的数据或者是控制状态的数据。通过汇编程序对这部分内存进行访问，从而实现与外设的交互。这样慢慢的理解了外设的地位与 RAM 是等价的，应当放在与 RAM 并行的位置进行实现。而单周期的实现中另一大难点在于对于 IRQ 信号的理解。IRQ 信号是由外设模块产生，作用于控制模块，从而使得处理器进入内核态进行处理。这对单周期处理器而言，并没有太大的难点，但对于流水线处理器而言，处理 IRQ 信号至关重要。

总的而言，本次实验对我来说，收获颇丰，第一次实现了自己编写的 CPU，还是一个非常有意义的事。不过想对老师提一点建议，就是 CPU 性能的衡量标准是 CPU **执行时间 = 指令数** × CPI × **时钟周期** (1/**主频**)。因此，并不能单单根据主频来判断 CPU 的性能，因为不同实现方法，对 CPI 以及程序的行数的影响也会比较大，应当根据三者的综合来进行判断。

References

[1] WikiPedia. Discrete cosine transform. WikiPedia, the Free Encyclopedia, 2016.

A 代码清单

根目录：/source

1. /3.1
2. /3.2