作业四　连连看

郝千越 2018011153 无 85

（系统环境：Windows10，MATLAB 版本：R2020a）

目录

本实验涉及文件较多，详细说明见“4 附：文件清单”

1 制作自己的连连看

1.1 熟悉游戏界面

运行 linkgame.fig，游戏界面如下，规则同普通的连连看一致。相关程序代码已经通过 pcode 转换为可执行的.p 文件。

图 1 连连看游戏界面

由于系统防火墙限制，程序不能直接打开 auto_click.exe，因此直接点击“开外挂”不能正常运行自动消除程序，可以按照如下步骤操作：

- 首先手动运行 auto_click.exe，弹出如下窗口；

- 在 MATLAB 中运行 linkgame.fig，点击“开外挂”，在点击上面窗口中的“确定”；
- 松开鼠标，程序控制光标自动消除连连看，直到游戏完成。

1.2 实现 detect 函数

【题目描述】

实现 detect.m 中的 detect 函数，输入当前图像块的索引矩阵和要消去的两个方块的下标，输出 0 代表不能消去，输出 1 代表可以消去。

【主要思想】

连连看中可以消去的方块对分为三类：可以直接连接、经过一个弯折连接、经过两个弯折连接。完成 detect 函数时首先考虑按照上述三类，枚举方块对之间所有可能的位置关系，一一写出判断条件并用“或”运算连接。然而过程中发现方块对位置关系复杂多样，枚举类别很多导致代码冗长且容易出错，部分代码片段如下：

```
if (mtx(x1,y1)==mtx(x2,y2))
&((x1==1&x2==1)|(x1==m&x2==m)|(y1==1&y2==1)|(y1==n&y2==n)...
|(x1==x2&sum(mtx(x1,y1:-(y1-y2)/abs(y1-y2):y2))-mtx(x1,y1)-mtx(x2,y2)==0)...%判断能否纵向直接相连
|(y1==y2&sum(mtx(x1:-(x1-x2)/abs(x1-x2):x2,y1))-mtx(x1,y1)-mtx(x2,y2)==0)...%判断能否横向直接相连
```

```
|(sum(mtx(x1,y1:-(y1-y2)/abs(y1-y2):y2))+sum(mtx(x1:-(x1-x2)/abs(x1-x2):x2,y1))-mtx(x1,y1)-mtx(x2,y2)==0)...
|(sum(mtx(x2,y1:-(y1-y2)/abs(y1-y2):y2))+sum(mtx(x1:-(x1-x2)/abs(x1-x2):x2,y2))-mtx(x1,y1)-mtx(x2,y2)==0)...
|(sum(mtx(x1,y1:-(y1-y2)/abs(y1-y2):y2))+sum(mtx(x1:-(x1-x2)/abs(x1-x2):x2,y2))-mtx(x1,y1)-mtx(x2,y2)==0)...
|(sum(mtx(x2,y1:-(y1-y2)/abs(y1-y2):y2))+sum(mtx(x1:-(x1-x2)/abs(x1-x2):x2,y1))-mtx(x1,y1)-mtx(x2,y2)==0)...
%判断能否弯一次相连，8 种情况：←↑； ↑←； ↓→； →↓； ←↓； ↓←； ↑→； →↑；
```

因此放弃这种简单枚举所有位置关系的做法，从更高层次上考虑这一问题。仔细观察可以发现，弯折一次相连的情况可以由弯折两次相连退化而来，直接相连的情况又可以由弯折一次相连退化而来，三者关系如图：

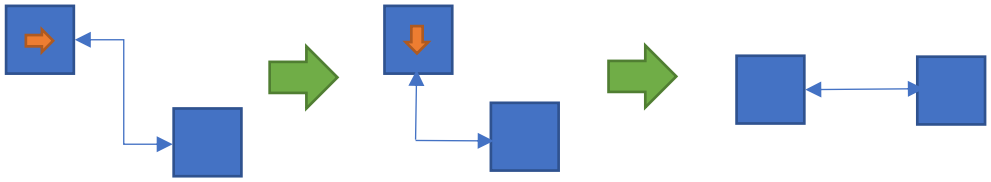


图 2 三种连接关系退化图

因此只要判断弯折两次能否连接，同时弯折的两臂长度都可以置为 0，即可覆盖所有三种情况。判断某一段空间是否有方块阻挡可以求该段空间对应 mtx 中元素的和，如果和不为 0 则有方块阻挡，反之没有。定义函数 sgn(a,b)，在 a<=b 时返回 1，在 a>b 时返回-1，通过此函数可以方便地索引 mtx(a:sgn(a,b):b)，而不必讨论 a 和 b 的大小关系。

【核心代码】

```
function bool = detect(mtx, x1, y1, x2, y2)
    [m, n] = size(mtx);
    bool=0;
    if mtx(x1,y1)==mtx(x2,y2)%判断两个方块是否相同
        bool=(sum(mtx(1:x1-1,y1))+sum(mtx(1:x2-1,y2))==0)|(sum(mtx(x1+1:m,y1))+sum(mtx(x2+1:m,y2))==0)...
            |(sum(mtx(x1,1:y1-1))+sum(mtx(x2,1:y2-1))==0)|(sum(mtx(x1,y1+1:n))+sum(mtx(x2,y2+1:n))==0);
        %如果两个方块都可以直接通向边缘则可以消去
        if ~bool
            for k=2:m-1%横向循环，寻找可以通过两次弯折连接的通路
                if sum(mtx(x1:sgn(x1,k):k,y1))+sum(mtx(k,y1:sgn(y1,y2):y2))+...
                        sum(mtx(k:sgn(k,x2):x2,y2))-2*mtx(x1,y1)-mtx(k,y1)-mtx(k,y2)==0
                    bool=1;
                    break;
                end
            end
            if ~bool%如果前面没有找到连接通路，则继续寻找
                for k=2:n-1%纵向循环，寻找可以通过两次弯折连接的通路
                    if sum(mtx(x1,y1:sgn(y1,k):k))+sum(mtx(x1:sgn(x1,x2):x2,k))+...
                            sum(mtx(x2,k:sgn(k,y2):y2))-2*mtx(x1,y1)-mtx(x1,k)-mtx(x2,k)==0
                        bool=1;
                        break;
                    end
                end
            end
        end
```

```
    end
end
```

【核心代码说明】

首先判断两个方块是否相同，如果相同则进入下面的步骤。先判断最简单的情况：两个方块都可以直接通向游戏区域的同一条边缘，则可以消去。如果不满足这种简单情况，则先分别画过两个方块的水平线，循环枚举两条水平线之间的每个位置的竖直线连接两条水平线，即判断是否存在如图通路

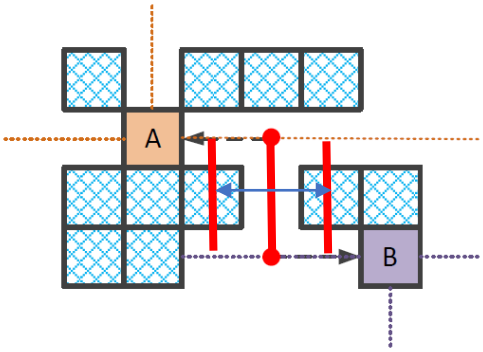


图 3 循环枚举两条水平线之间的每个位置的竖直线

如果仍未找到通路，再别画过两个方块的竖直线，循环枚举两条竖直线之间的每个位置的水平线连接两条竖直线，即判断是否存在类似通路。

经过上述操作，如果找到了通路则函数返回 1，否则返回 0。

【运行结果】

经过测试，自己编写的 detect 函数完全可以替代原 detect.p，使得连连看正常运行。

1.3 实现 omg 函数

【题目描述】

完成 omg.m 中的函数，实现自动消去方块的“外挂功能”。

【主要思想】

首先考虑如下简单算法：

1. 取一个方块，一一遍历其他方块并调用 1.2 中的 detect 函数判断能否消去；
2. 如果能够消去则将这对方块写入 steps 中，并将 mtx 数组中对应位置置为 0，
   如果不存在能消去的方块对，则取下一个方块，重复上述遍历操作 1、2。

这种算法下，每确定一步操作都需要进行大量遍历，复杂度较高。注意到遍历中不能消去的方块对大部分都是由于两者本身就不是一种方块，这种操作浪费了大量的时间，极大增多了遍历次数，因此设计了改进的算法。

改进算法中在开始确定第一步操作之前生成一个 temp 数组将方块分类，temp 数组中按照方块类别记录其位置。生成 temp 后，按照上面的算法开始执行，只是步骤 1 中只遍历同一类的其他方块而非全部方块。由此完全排除了遍历本身不是一种方块的方块对造成的额外开销。

【核心代码】

```
function steps = omg(mtx)
    [m, n] = size(mtx);
    temp=zeros(max(max(mtx)),2,m*n+1);%临时数组，储存方块分类
    temp(:,1,1)=1;%每页第一行第一个元素记录该页当前末尾位置
    steps=zeros(1,2*m*n+1);%steps 存放结果
    steps(1)=m*n/2;
```

```
    position=2;%辅助指针记录 steps 末尾位置
    for x=1:m
        for y=1:n
            if mtx(x,y)~=0
                temp(mtx(x,y),1,1)=temp(mtx(x,y),1,1)+1;%移动末尾位置
                temp(mtx(x,y),1,temp(mtx(x,y),1,1))=x;%归类当前方块
                temp(mtx(x,y),2,temp(mtx(x,y),1,1))=y;
            end
        end
    end
    while sum(sum(mtx))~=0%没有完全消除时循环
        flag=0;%标记是否找到能消去的方块对
        for x=1:m
            for  y=1:n
                if mtx(x,y)~=0
                    for k=2:temp(mtx(x,y),1,1)%遍历相同类型方块
                        if
(x~=temp(mtx(x,y),1,k)|y~=temp(mtx(x,y),2,k))&mtx(temp(mtx(x,y),1,k),temp(mtx(x,y),2,k))~=0&detect(mtx,
x,y,temp(mtx(x,y),1,k),temp(mtx(x,y),2,k))
                            %如果是尚未消去且能够消去的方块，则找到一组可消去的方块对
                            steps(position:position+3)=[x,y,temp(mtx(x,y),1,k),temp(mtx(x,y),2,k)];
                            %插入结果数组
                            position=position+4;%移动 steps 的末尾指针
                            mtx(temp(mtx(x,y),1,k),temp(mtx(x,y),2,k))=0;
                            %mtx 中已经消去的方块置为 0
                            mtx(x,y)=0;
                            flag=1;%标记置为 1，表示找到可以消去的方块对
                            break;
                        end
                    end
                end
                if flag==1
                    break;
                end
            end
            if flag==1
                break;
            end
        end
    end
end
```

【核心代码说明】

首先建立 temp 数组，对游戏区域中的方块进行归类，以便简化后续计算。归类完成后每次取一个方块，遍历所有和它同类型的方块，用 detect 函数判断能否消去。如果可以则将该步骤写入 steps 中并将 mtx 对应位置置为 0；如果不能则继续尝试下一对方块。直到 mtx 中所有元素均为 0，外挂步骤计算完成。完整代码文件为 omg.m。

【运行结果】

一次运行生成的 steps.txt 部分内容如图，可见一共需要 42 步消去全部方块，数据格式符合要求。

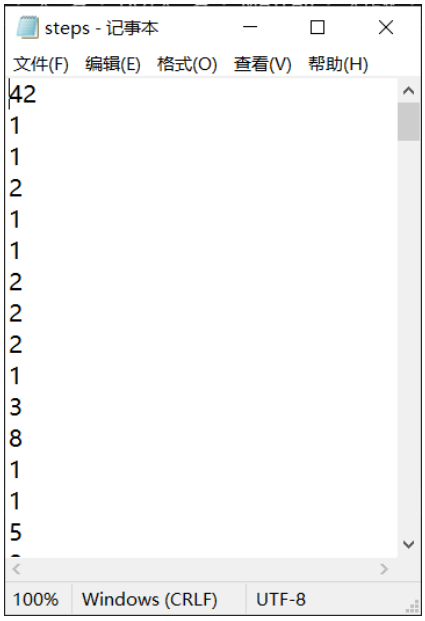


图 4 一次运行生成的 steps.txt 部分内容

【结果分析】

经过测试，自己编写的 omg 函数完全可以替代原 omg.p，使得连连看外挂功能正常运行。一些关于算法消耗时间的分析见 1.4.3 节。

1.4 （选做）连连看游戏中的若干问题

1.4.1 生成有意义、可完全消除的游戏区域

作业指导正文“第一章：制作自己的连连看，第四节：科学问题”中提到“如何生成一个有意义的游戏区域并保证他是可以完全消除的？”下面就这一问题进行讨论。

要满足游戏区域可以完全消除，有两个条件：

- 各类方块数目均为偶数，这一点是显然的；
- 方块的空间排列满足一定条件，不存在“自锁结构”，举一个极端的例子，如果游戏区域为 2×3，则如下方块排列（相同数字代表同种）是无法消去的

$$\begin{bmatrix} 1 & 2 & 3 \\ 3 & 1 & 2 \end{bmatrix}$$

如果扩大游戏区域，使得“自锁结构”中的一些方块可以与其他区域的方块消去，则“自锁结构”被解除，方块有可能能够消去。如上述区域扩展为 4×3，则可以消去。

$$\begin{bmatrix} 1 & 2 & 3 \\ 3 & 1 & 2 \\ 3 & 4 & 3 \\ 5 & 4 & 5 \end{bmatrix}$$

给定一个游戏区域（即一个矩阵），判断其是否可以完全消去是困难的，但是生成一个给定尺寸的、可以完全消去的游戏区域是相对容易的，下面给出一种根据“逆向方法”生成一个给定尺寸的、可以完全消去的游戏区域的算法。

1. 给定尺寸$L \times H$，判断若L、H均为奇数则尺寸错误，反之创建$L \times H$的全 0 矩阵mtx代表生成的游戏区域，创建$\left(\frac{L \times H}{2}\right) \times 4$的数组$pairs$，每行记录一对方块的坐标；
2. 考虑到连连看中往往有两个相同方块紧邻的情况，给定紧邻方块比例p，随机选取$\left(\frac{L \times H}{2}\right) \times p$对紧邻的坐标，如果所选择坐标不与已有坐标冲突，存入$pairs$中，否则重新生成这对坐标；
3. 继续随机生成$\left(\frac{L \times H}{2}\right) \times (1-p)$对非紧邻的坐标，如果所选择坐标不与已有坐标冲突且调用$detect$函数判断这两个位置是可以消去，则存入$pairs$中，否则重新生成这对坐标；
4. 如果 3 中生成某一对坐标时重新生成的次数达到一个上限仍未得到可行的结果，则清除 3 中已经生成的所有坐标对，全部重新生成；
5. 直到全部游戏区域都被两两配对，给定方块种类数目N，随机生成$\left(\frac{L \times H}{2}\right)$个位于$[1, N]$上的整数，用对应图片填充全部$\left(\frac{L \times H}{2}\right)$对方块。

利用上述算法，添加一对坐标时保证了他们是可以消去的，即生成游戏区域的过程实际上是按照消去的逆过程进行的，因此保证了生成的游戏区域至少有一种方式（即生成的逆过程）可以完全消去。同时，上述算法完全是随机进行的，保证了生成的游戏区域是丰富而多样的。代码文件为 create_game.m，其中调用生成游戏区域矩阵的函数文件 generate.m，调用给出的 pics.mat 中储存的图片数据画出生成的游戏区域。随机生成的一个结果如下图：

图 5 一次随机生成的结果

1.4.2 游戏难度的定义

容易想到，游戏区域越大、方块种类越多，则游戏难度越大。因此可以设计如下三级游戏难度设置：

表 1 游戏难度设置

难度	区域长度L	区域宽度H	方块种类N
入门	8	5	≤ 10
普通	12	7	≤ 15
困难	16	10	≤ 21

生成的一个入门模式游戏区域如图：

图 6 生成的一个入门模式游戏区域

1.4.1 节中算法使用完全随机生成的方式，当区域长度L、区域宽度H增大、紧邻方块比例p减小时，随机尝试需要的次数增多，生成游戏区域所需要的时间也延长了。固定紧邻方块比例$p = 0.2$，变化$L \times H$变化；固定$L \times H = 12 \times 7$，变化当紧邻方块比例p，生成游戏区域所需要的时间如下图（由于使用随机算法，每组数据均生成 5 次取平均结果）：

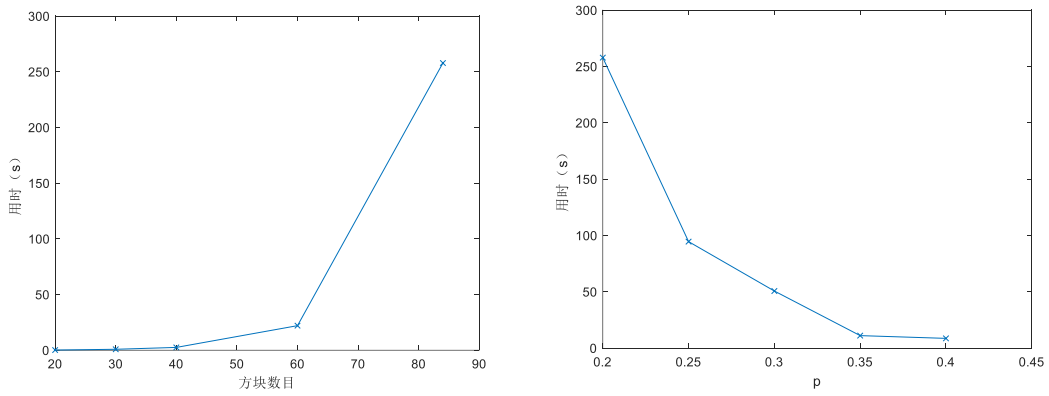


图 7 生成游戏区域所需时间随$L \times H$变化

可以发现，当所需要区域较大或设置p较小时，需要是时间较长，不适合在每次开始游戏时都重新生成游戏区域，结合算法特点可以采取如下方式生成游戏区域：

- 由于生成算法中先将游戏区域两两配对，得到游戏区域配对矩阵，再向每对方块上填充图案，实现了两者的分离。因此可以先生成若干个游戏区域配对矩阵，每次开始游戏前随机选取一个矩阵并对其随机填充方块图案，这样就可以快速地载入游戏，同时得到多样的游戏区域方块排列。

1.4.3 “外挂模式”中的算法讨论

1.3 中考虑的外挂算法在选取、遍历方块过程中，均是从下标较小的方块开始顺序遍历，下面讨论这种顺序遍历方法和另一种随机遍历方法消耗时间的长短。omg.m 为 1.3 中顺序遍历的外挂算法代码，omg_1.m 为重新编写的随机遍历的外挂算法代码。使用 1.4.1 中编写的 generate.m 生成一个 8×5 的游戏区域，在 compare.m 中使用顺序遍历和随机遍历分别求解 2000 次，用时结果如图

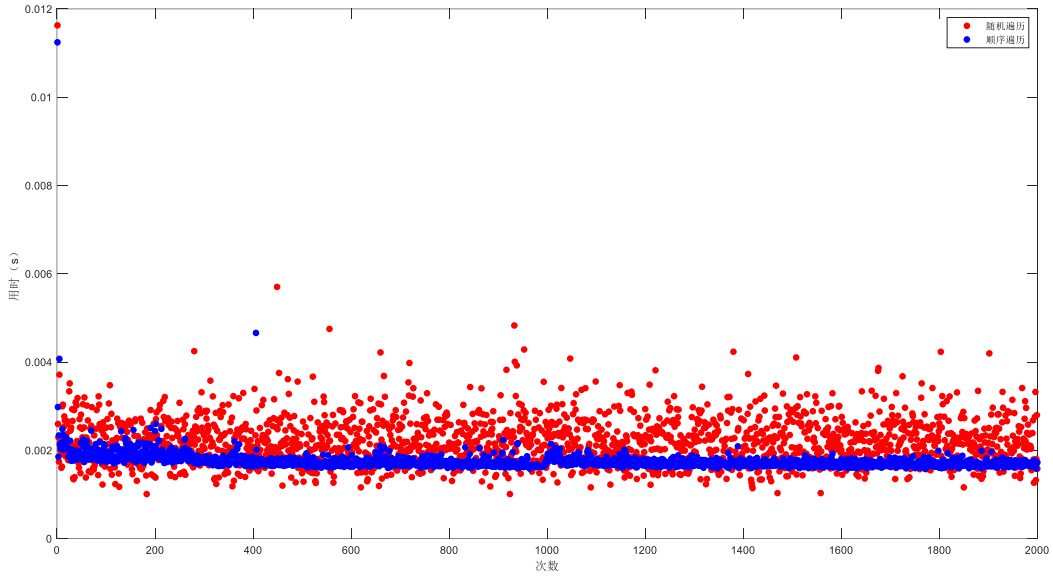


图 8 顺序遍历与随机遍历用时对比

可以发现两种算法第一次执行用时均较多，根据《数字逻辑与处理器基础》课程知识分析，原因应当是首次执行需要把对应指令从内存载入 cache，这一过程相当耗时，在此不做详述。后面的 1999 次执行中可以发现随机遍历用时方差较大，偶尔可以出现用时小于顺序遍历的情况，但均值大于顺序遍历；顺序遍历用时稳定在一个较小值附近。两种算法用时长短与游戏区域分布同样密切相关，但上述规律基本总是成立的，综合考虑后选择使用顺序遍历算法求解该问题。

2 攻克别人的连连看

2.1 对游戏区域屏幕截图进行分割

【题目描述】

对游戏区域的屏幕截图（灰度图像）进行分割，提取出所有的图像分块。在一个 figure 中用 subplot 的方式按照原始顺序绘出所有的图像分块。

【主要思想】

观察可以发现，每个方块的边缘为黑色（灰度较低），而内部为白色（灰度较高），因此按行扫描或按列扫描均呈现很明显的周期性，如图所示

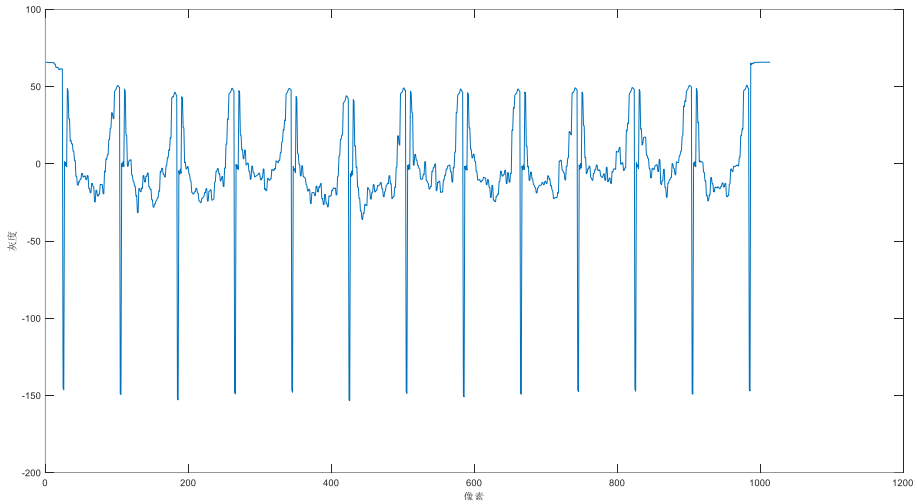


图 9 graygroundtruth 按行扫描平均结果

因此，可以利用傅里叶变换，得到基波周期即为一个方块占据的像素数目，下面以获得方块宽度（即横向周期长度）为例说明。

实际处理中，由于灰度波形比较复杂，而更据连连看游戏的实际情况，一般游戏区域内一行/一列中的方块数目一般不会小于 5 个或多于 30 个，即设整个图片横向长度中n个像素为单位 1 时，基频有

$$f_0 \in [5,30]Hz$$

因此首先对原始横向平均灰度x做带通滤波得到x_a，其频谱为F_{x_a}，得到基频f_0，方块宽度即为

$$width = \frac{n}{f_0}$$

根据观察，基频的最大值与游戏区域边缘对齐，因此设基频函数为

$$base = Acos(2\pi f_0 x + \phi)$$

则左侧空白长度为

$$l\_margin = -\frac{\phi}{2\pi f_0}$$

需要特别注意的是，前面的带通滤波器有其非平凡的相频响应，因此滤波后的函数相位发生了改变，不能用于计算左侧空白长度，需要用原始灰度函数的相位进行计算。原始灰度函数x的频谱为F_x，即有（式中方括号表示F_x在f_0处的取值）

$$l\_margin = -\frac{\phi}{2\pi f_0} = -\frac{\arg(F_x[f_0])}{2\pi f_0}$$

【核心代码】

```
pic=imread('graygroundtruth.jpg');%读取灰度图
[m,n]=size(pic);


x=mean(pic,1)-mean(mean(pic,1));%按行平均，除去直流分量
xa=bandpass(x,[5,30],n);%带通滤波
```

```
N=n;%时域采样点数
T=1;%时间范围
[t,omg,FT,IFT]=prefourier([0,T-T/N],N,[0,500],10000);
F=FT*xa.';%傅里叶变换
 [~,b]=max(abs(F));%找出基波频率
width=round(2*pi*n/omg(b));%根据基波频率计算横向周期

F=FT*x.';%计算原灰度函数的频谱
phase=angle(F(b));%计算基波频率的相位
t=[0:n];
base=abs(F(b))*cos(2*pi/width*t+phase);%生成基波函数
[~,l_margin]=max(base(1:width));%根据基波函数得到左侧空白距离
```

【核心代码说明】

此处以横向周期分析为例说明核心代码：首先将灰度图矩阵按列取平均得到行向量 x，使用 bandpass 函数对 x 低通滤波，得到 xa。对 xa 傅里叶变换得到频谱和并从中得出基频，从而计算出横向周期 width；再对 x 傅里叶变换，根据变换结果的相位得到基波函数 base，同时得出了左侧空白距离。上面的核心代码使用了 prefourier 函数进行傅里叶变换的计算，完整代码文件为 ex_2_1_b.m；同时还编写了使用 FFT 计算的代码，完整代码文件为 ex_2_1_a.m，两者的区别将在“结果分析”中说明。

【运行结果】

运行程序得到了对应的左侧空白距离l_margin，顶部空白距离t_margin，方块宽度$width$，方块高度$height$。由此划分出每个方块，结果图像见“结果分析”。

【结果分析】

按照上述步骤（下面均以横向分割为例），原灰度函数图像与带通滤波后函数图像为

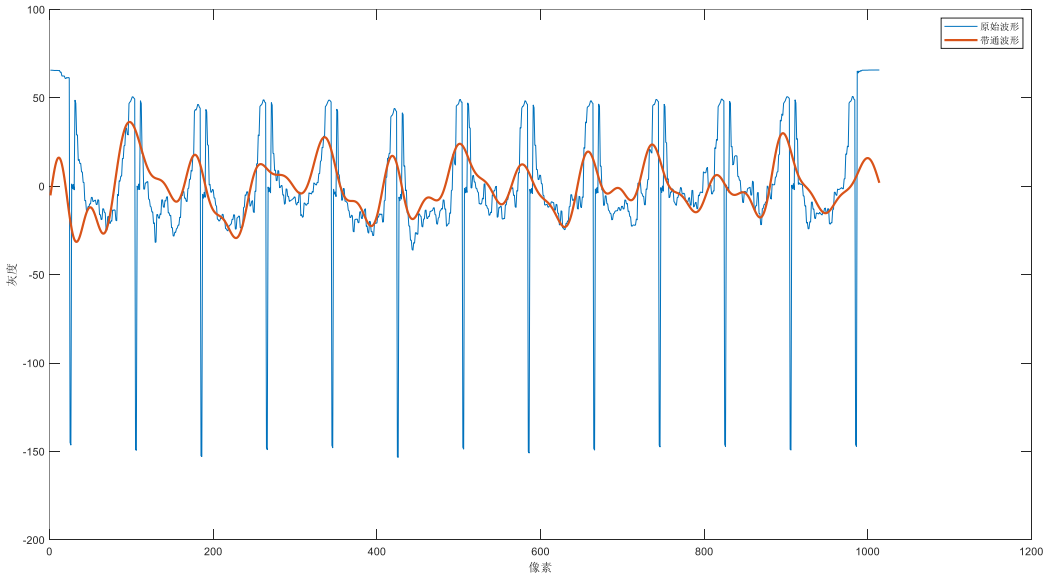


图 10 原灰度函数图像与带通滤波后函数图像

可以看出带通滤波后函数除去了原始函数中高频分量，但保留了周期信息，便于进行下一步操作。首先使用快速傅里叶变换方法得到频谱如图（频谱截取了峰值明显的部分，下同）

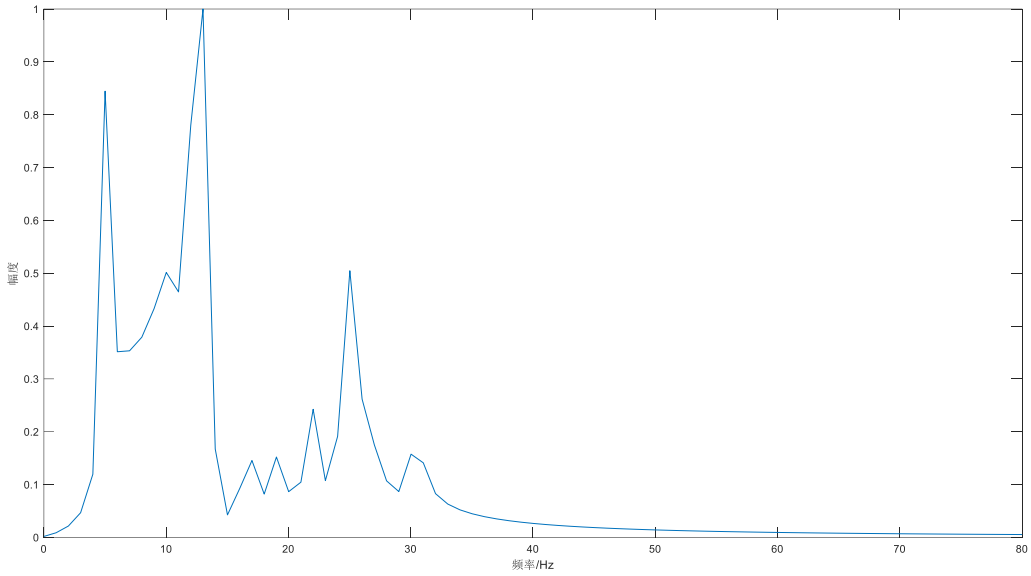


图 11 快速傅里叶变换方得到频谱

由此画出原灰度图像与其基波函数如图

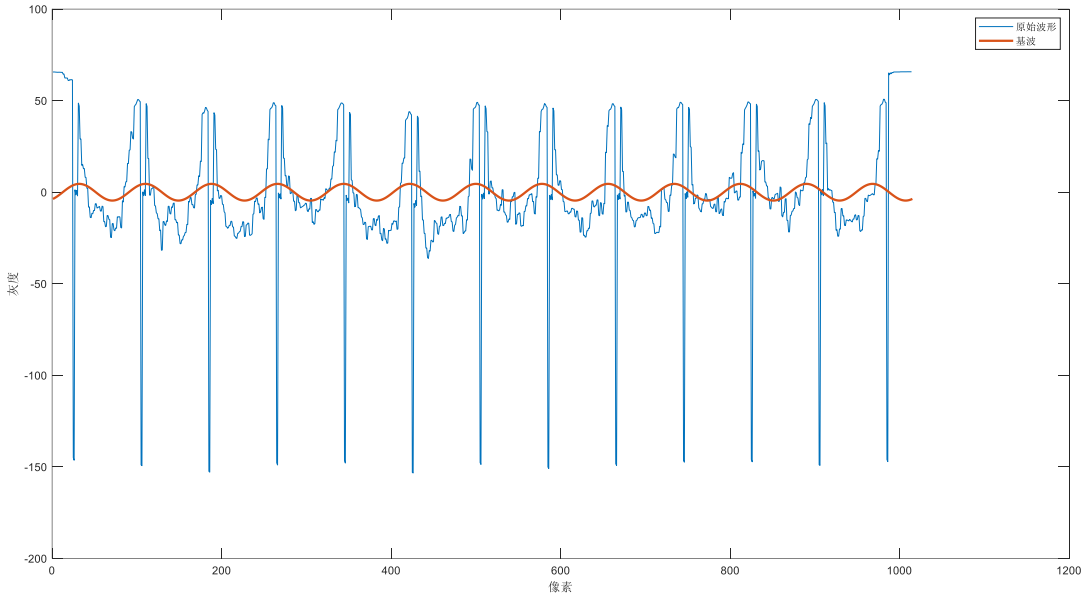


图 12 快速傅里叶变换得到的基波函数

可以看出基波函数的周期略小于原灰度图像的周期。这是由于 FFT 在频域采样的分辨率较低，真正的基波频率位于两个采样点之间，因此被近似为其右侧采样点，得到的基波频率略大，周期略小。如图，得到的分割结果存在一定偏差，尤其是越靠近右侧的方块累计误差越明显

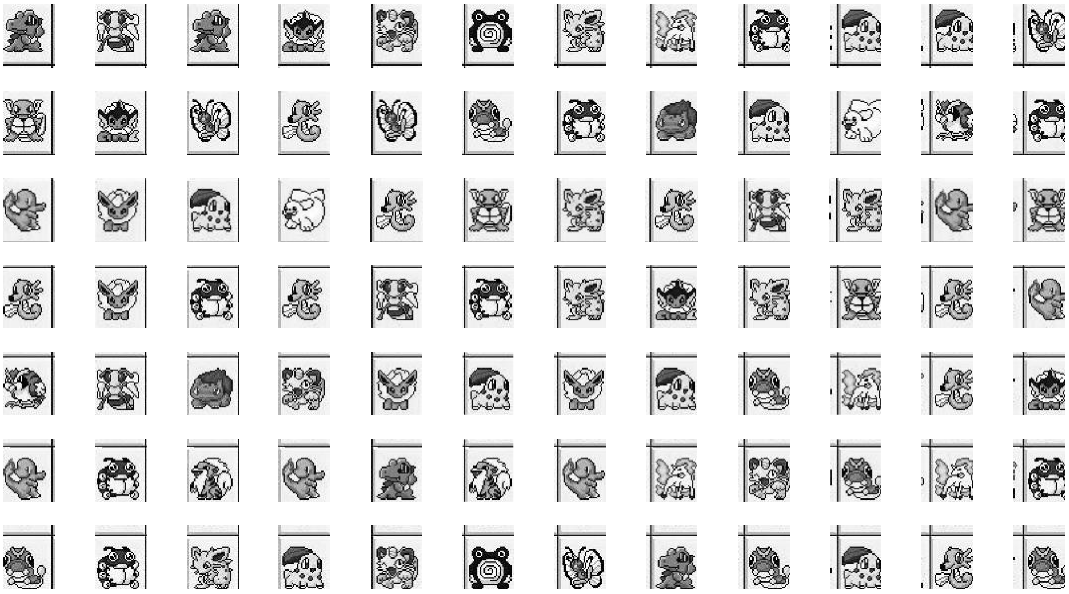


图 13 根据快速傅里叶变换得到的分割结果

可以明显看出这一分割结果并不理想，因此改为使用可以设定更高频域分辨率的 prefourier 函数，得到频谱如图，可以看出由于分辨率的提高，频谱变得光滑了很多，因此也能够更加准确地定位基波函数的频率

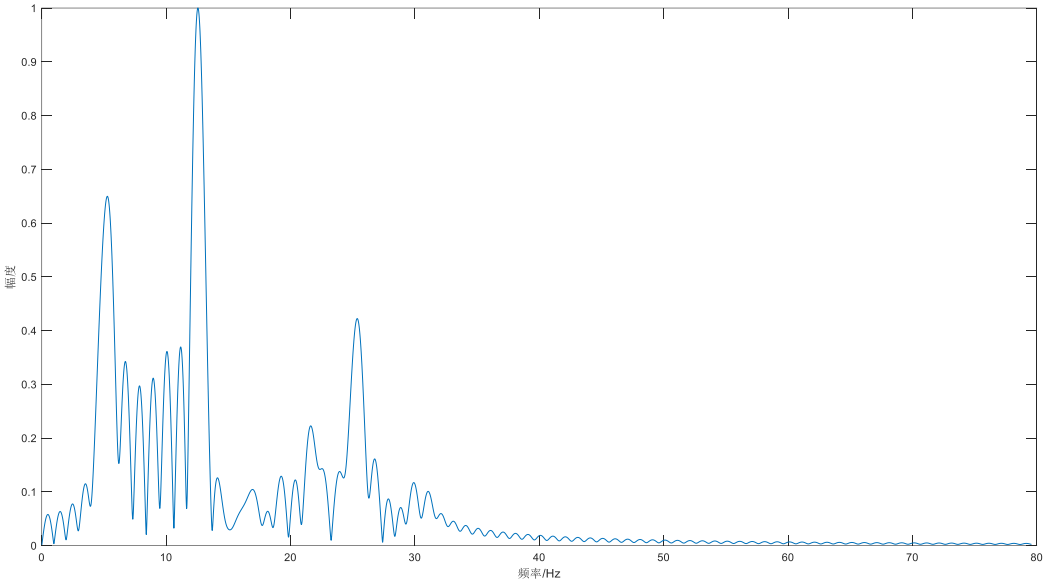


图 14 prefourier 函数得到的频谱图

根据基波函数的频率画出原灰度函数与基波函数的图像如图，可见两者的周期吻合地很好。

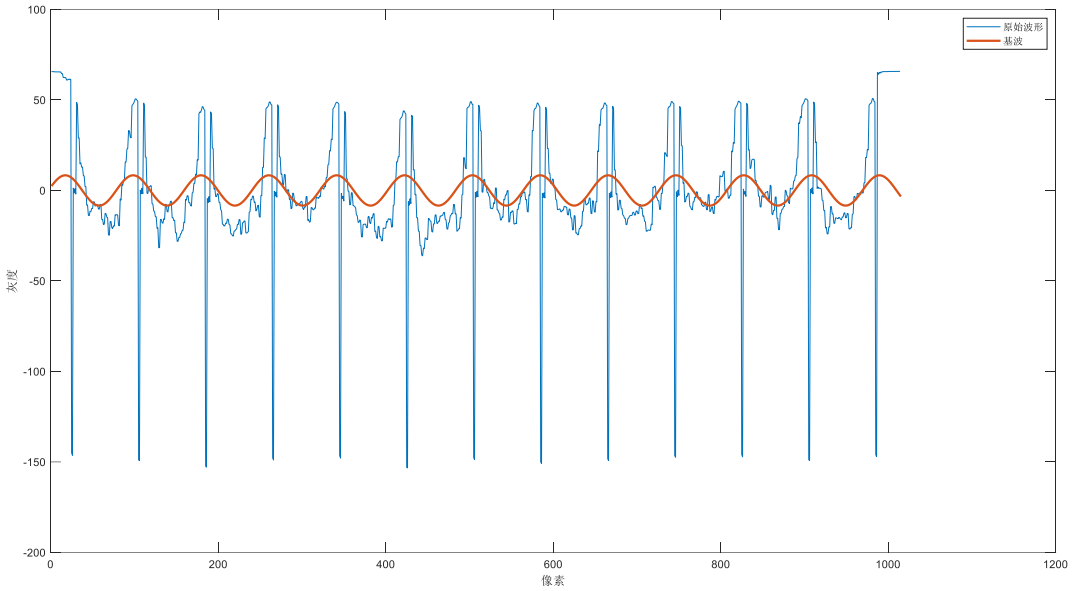


图 15 prefourier 函数得到的基波函数

据此得到最终的分割结果，效果比较理想，便于进行后续操作

图 16 graygroundtruth 最终分割结果

2.2 对摄像头采集的图像进行分割

【题目描述】

对摄像头采集的游戏区域（灰度图像）进行分割，提取出所有的图像分块。在一个 figure 中用 subplot 的方式按照原始顺序绘出所有的图像分块。

【主要思想】

经过 2.1 中在清晰的屏幕截图上的调试经验，确定使用“带通滤波——prefourier 函数高分辨率傅里叶变换——确定基波频率——根据相位确定空白距离”的方法可以比较准确地分割游戏区域。观察可以发现摄像头捕获的图像尽管比较模糊，但两个方块之间的暗色边界以及方块的明暗关系与屏幕截图一致，具有所需要的周期关系，因此可以使用 2.1 中类似方法进行分割。

【核心代码】

```
pic=imread('graycapture.jpg');%读取灰度图
[m,n]=size(pic);

x=mean(pic,1)-mean(mean(pic,1));%按行平均，除去直流分量
xa=bandpass(x,[5,30],n);%带通滤波

N=n;%时域采样点数
T=1;%时间范围
[t,omg,FT,IFT]=prefourier([0,T-T/N],N,[0,500],10000);
F=FT*xa.';%傅里叶变换
[~,b]=max(abs(F));%找出基波频率
width=round(2*pi*n/omg(b));%根据基波频率计算横向周期

F=FT*x.';%计算原灰度函数的频谱
phase=angle(F(b));%计算基波频率的相位
t=[0:n];
base=abs(F(b))*cos(2*pi/width*t+phase);%生成基波函数
[~,l_margin]=max(base(1:width));%根据基波函数得到左侧空白距离
```

【核心代码说明】

此处以横向周期分析为例说明核心代码：首先将灰度图矩阵按列取平均得到行向量 x，使用 bandpass 函数对 x 低通滤波，得到 xa。对 xa 用 prefourier 函数求傅里叶变换得到频谱和并从中得出基频，从而计算出横向周期 width；再对 x 傅里叶变换，根据变换结果的相位得到基波函数 base，同时得出了左侧空白距离。完整代码文件为 ex_2_2.m

【运行结果】

运行程序得到了对应的左侧空白距离l_margin，顶部空白距离t_margin，方块宽度$width$，方块高度$height$。由此划分出每个方块，结果如图

图 17 graycapture 分割结果

将分割后的图像保存为 spilit.mat 中，设有$M \times N$个方块，则 spilit 为$M \times N$个$cell$的单元数组，各个$cell$为一个$height \times width$的矩阵，保存分割后的一个方块图片。分割的距离信息保存在 info.mat 中，info 为 4 列的行向量，依次存储左侧空白距离l_margin，顶部空白距离t_margin，方块宽度$width$，方块高度$height$。

【结果分析】

对比屏幕截图与摄像头捕获图像的分割结果，两者均能够比较准确地得到方块的分割结果（摄像头捕获图像略有歪斜，但基本不影响分割结果）。由此可以发现，图像的模糊程度不影响图像分割，只要图像具有明显的明暗周期关系，就可以利用上述方法准确地分割出游戏区域中的方块。

为方便后续步骤中检验各种方法是否有效，此处手动标记 graycapture 的真值，通过代码文件 truth.m 储存为 truth.mat。

2.3 方块相似性分析

【题目描述】

计算所有图像分块的两两相似性，选出最相似的十对图像块在一个 figure 中画出并显示其相似性度量值。

【主要思想】

根据上一问划分得到的方块区域，进行方块之间的相似性分析。首先观察到方块具有区分性较强的纹理特征，因此可以使用高通滤波对图像进行预处理，突出纹理特性。将高通滤波后的图像两两进行互相关运算即可得到两图像之间的相似度。根据相似度的数值可以对方块进行后续分析。

【核心代码】

高通滤波

```
pic=load('../data/spilit.mat');
[M,N]=size(pic.spilit);
[height,width]=size(pic.spilit{1,1});


pic_h={};
h_c=(1+height)/2;%计算中心位置
w_c=(1+width)/2;
passband=2;%通带半径
for m=1:M
    for n=1:N
        temp=im2double(pic.spilit{m,n});%转化为 double 型
        temp=fftshift(fft2(temp));%傅里叶变换并将零频率移动到中心
        for x=1:floor(width/2)
            x2=x^2;
            for y=1:floor(height/2)
                y2=y^2;
                if x2+y2<passband^2%位于通带外则截断
                    temp(floor(h_c+y),floor(w_c+x))=0;%实信号频谱对称，可以由一个象限填充其他
                    temp(floor(h_c+y),ceil(w_c-x))=0;
                    temp(ceil(h_c-y),floor(w_c+x))=0;
                    temp(ceil(h_c-y),ceil(w_c-x))=0;
                end
```

```
            end
        end
        temp=real(ifft2(ifftshift(temp)));%傅里叶逆变换得到处理后的图像
        pic_h{m,n}=temp;
    end
end
```

计算相关系数

```
sim=zeros(M,N,M,N);
for m=1:M*N
    row1=ceil(m/N);%遍历所有图像对的组合
    col1=m-N*(row1-1);
    for n=m:M*N
        row2=ceil(n/N);
        col2=n-N*(row2-1);
        s=similarity(pic_h{row1,col1},pic_h{row2,col2});%计算相关系数
        sim(row1,col1,row2,col2)=s;%sim(picture1,picture2)=sim(picture2,picture1)
        sim(row2,col2,row1,col1)=s;%对称填充矩阵中两个位置
    end
end
sim=(sim-mean(sim,'all'))/sqrt(var(sim,1,'all'));%归一化[1]
```

【核心代码说明】

完整代码文件为 ex_2_3and4.m 对于图像的高通滤波与对于一维信号的完全相同，即首先使用 fft2 函数对图像进行快速傅里叶变换，得到图像的频谱。由于图像为二维矩阵，其频谱有两个方向的分量，为一个对应的二维矩阵。高通滤波操作即截断频谱的低频分量，保留高频分量，对应于一维信号高通滤波通带为一个区间外侧，图像高通滤波通带为一个圆外侧。截断后在将剩余的频谱用 ifft2 函数逆变换得到滤波后的图像。由于 MATLAB 中 fft2 函数变换结果为低频分量在频谱矩阵外围而高频分量再内侧，为便于截断操作，使用了 fftshift 函数将频谱一三象限互换，二四象限互换。

二维图像的互相关运算可以类比一维的相关定理，编写函数文件 similarity.m

$$\mathcal{F}\{R_{12}(\tau)\} = F_1(\omega)F_2^*(\omega)$$

得到二维相关定理，上标(2)表示为二维矩阵

$$\mathcal{F}\left\{R_{12}^{(2)}(\tau)\right\} = F_1^{(2)}(\omega)F_2^{(2)*}(\omega)$$

因此计算中首先对两幅待求图片做傅里叶变换，一者取共轭后相乘，再做傅里叶逆变换即可得到相关函数。相关函数最大值即两个图像处于最佳对齐位置时的相关系数。计算出两两方块的相关系数后，做均值为 0，方差为 1 的归一化处理，即

$$S = \frac{S - E(S)}{\sqrt{var(S)}}$$

其中S为两两相关系数矩阵，这一处理便于后续的分析。

[1] 此处‘all’参数表示对高维张量中全部元素求均值或方差，MATLAB 文档显示：
“M = mean(A,'all') 计算 A 的所有元素的均值。此语法适用于 MATLAB® R2018b 及更高版本。”
“V = var(A,w,'all') 计算 A 的所有元素的方差。此语法适用于 MATLAB® R2018b 及更高版本。”
因此可能存在版本兼容问题。我确保全部代码在本机环境运行无误，后面含这两个函数的代码同样如此。

【运行结果】

得到两两方块之间的相关系数的矩阵，保存为 similarity.mat，其中为$M \times N \times M \times N$的矩阵。同时得到相关系数最大的十对方块，图片见“结果分析”。

【结果分析】

首先讨论高通滤波器通带半径对相关系数计算的影响，利用 2.2 中标注的方块类别真值，定义如下裕度函数，其含义为同种方块中相关系数最小值与异种方块中相关系数最大值之差。

$$margin = min(S_{ij}[\epsilon_{ij}]) - max(S_{ij}[1-\epsilon_{ij}])$$

其中S_{ij}表示第i个方块和第j个方块的相关系数，且

$$\epsilon_{ij} = \begin{cases} 1, i与j为同种方块 \\ 0, i与j为不同种方块 \end{cases}$$

方括号中为 1 表示该元素被索引取出，0 标志该元素未被索引取出。

可见如裕度函数值为正，则存在一个阈值可以完全区分同种方块对和异种方块对；如裕度函数值为负则不存在这一阈值，同种方块对和异种方块对存在部分混淆。裕度函数值越大则说明分类效果越好。调节高通滤波器，裕度函数值随通带半径变化如图（此处通带半径为相对值，不含有单位）

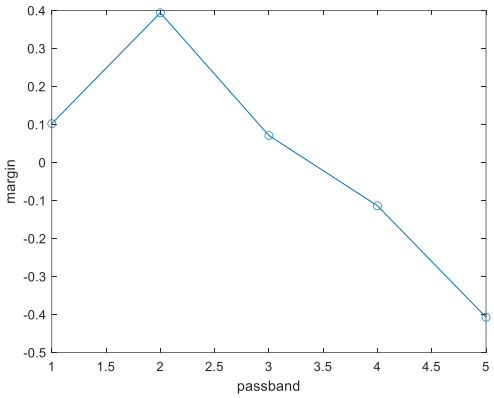


图 18 通带半径对分类效果的影响

可见通带半径为 2 时裕度函数值最大，分类效果最好，如果通带半径过大则会将太多“低频”分量滤去，损失信息而难以准确计算相关系数。此时分类结果如图

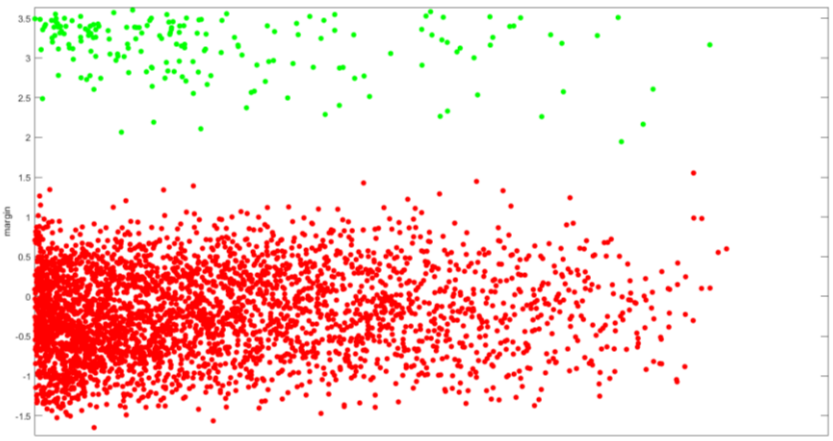


图 19 通带半径为 2 分类效果

图中红点表示异种方块对，绿点表示同种方块对，可以看出两者之间存在明显的分界线，即存在分类阈值。置通带半径为 2 时一个方块滤波前后对比如下：

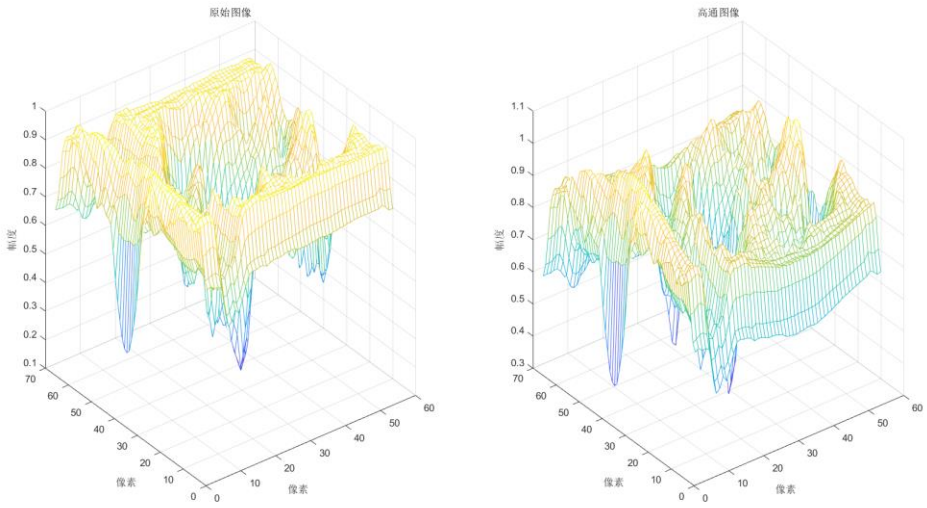


图 20 高通滤波前后对比

可见图像边缘平滑的部分被抑制了，而图像中心小精灵所在的部分由于纹理明显而被保留。

在高通滤波后的图像中两两计算相关系数，相关系数最大的十对如图

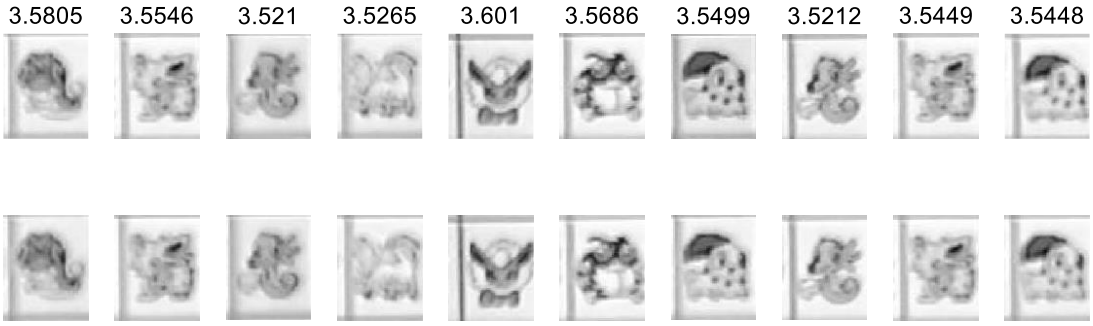


图 21 相关系数最大的 10 对方块

可以发现这些方块对均为同一种方块，而且进一步观察方块在原图中的位置（程序中 max_pair 数组中记录），相关系数最大的方块对大多是紧邻的同种方块。

2.4 非同种方块相关系数分析

【题目描述】

在 2.3 基础上找出不是同种方块但相似度最大的 10 对方块，在一个 figure 中画出并显示其相似性度量值。讨论：结果与主观感受一致吗？

【主要思想】

利用 2.3 中计算出的方块之间两两相关系数和 2.2 标注的真实类别，可以找出不是同种方块但相似度最大的 10 对方块。

【核心代码】

```
max_sim_wrong=zeros(10,1)-100;
max_pair_wrong=zeros(10,4);
for m=1:M*N
    row1=ceil(m/N);
    col1=m-N*(row1-1);
```

```
    for n=m:M*N
        row2=ceil(n/N);
        col2=n-N*(row2-1);
        if truth(row1,col1)~=truth(row2,col2)&(sim(row1,col1,row2,col2)>min(max_sim_wrong))
            %维护数组，记录相关系数最大但不是同一种图像的 10 对
            [~,a]=min(max_sim_wrong);
            max_sim_wrong(a)=sim(row1,col1,row2,col2);
            max_pair_wrong(a,:)=[row1,col1,row2,col2];
        end
    end
end
figure;
for k=1:10%显示相关系数最大但不是同一种图像的 10 对
    subplot(2,10,k);
    imshow(pic.spilit{max_pair_wrong(k,1),max_pair_wrong(k,2)});

title(num2str(sim(max_pair_wrong(k,1),max_pair_wrong(k,2),max_pair_wrong(k,3),max_pair_wrong(k,4))),'F
ontsize',20);
    subplot(2,10,10+k);
    imshow(pic.spilit{max_pair_wrong(k,3),max_pair_wrong(k,4)});
end
```

【核心代码说明】

完整代码文件为 ex_2_3and4.m，维护数组 max_sim_wrong 和 max_pair_wrong，遍历所有方块对，不断更新相关系数最大但不是同种方块的方块对，最终得到要求的 10 对方块，显示在图中。

【运行结果】

得到相关系数最大但不是同种方块的 10 个方块对，图片见“结果分析”。

【结果分析】

得到相关系数最大但不是同种方块的 10 个方块对如图

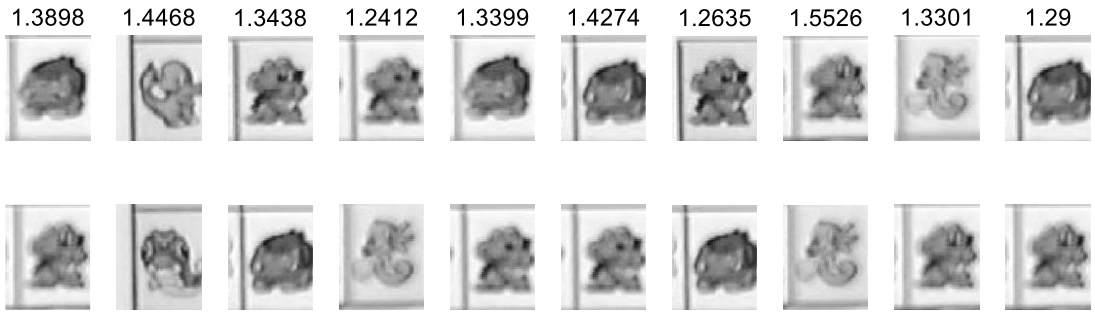


图 22 相关系数最大但不是同种方块的 10 个方块对如图

观察发现有如下规律

- 有多对涉及下面两个图案的方块对有着较大的相关系数。

这两个方块都具有纹理较不明显为占据方块中心较大面积区域的特定，因此计算中出现了较大的相关系数，可能对后续步骤造成干扰，这一点与直观感觉是一致的。但是利用图像颜色信息可以发现左边方块小精灵为蓝色而右边为绿色，可以更好地区分这两种方块，这一操作将在 2.7 中进一步讨论。

- 这 10 对方块中相关系数最大为 1.5526，因此可以据此设置阈值，即如果相关系数大于该数值，则是同种方块，小于等于该数值则无法判定是否为同种方块。这一阈值在后续步骤中有所用途。

2.5 游戏区域映射为索引值

【题目描述】

将游戏区域映射为索引值的数组，并列出索引值和图像分块的对照关系。讨论：你可以将全部图像分块正确映射到其索引值吗？哪些方块无法正确映射？为什么？

2.5.1 方法一：“无监督”分类

【主要思想】

对于这一已给出的游戏区域图像，2.4 中已经确定不同的方块对之间相关系数最大值为 1.5526，下面将进一步确定相同方块对之间相关系数的最小值。由于 2.3 中已经计算出阈值函数值为正，因此这一最小值一定大于 1.5526，因此可以在两个数之间确定一个阈值，两个方块相关系数大于阈值则为同种方块，否则为不同方块，由此可以 100%精确地将方块聚类，进而赋予每一类一个索引值。这种方法不需要依赖一个“标准方块类型图谱”，因此称为“无监督”分类[2]。

【核心代码】

确定相同方块对之间相关系数的最小值

```
sim=load('../data/similarity.mat').sim;
truth=load('../data/truth.mat').truth;
[M,N,~,~]=size(sim);
min_sim=100;
min_pair=zeros(4);
for m=1:M*N
    row1=ceil(m/N);
    col1=m-N*(row1-1);
    for n=m:M*N
        row2=ceil(n/N);
        col2=n-N*(row2-1);
        if truth(row1,col1)==truth(row2,col2)&(sim(row1,col1,row2,col2)<min_sim)
            %维护数组，记录相关系数最小但不的同一种图像的 10 对
            min_sim=sim(row1,col1,row2,col2);
            min_pair=[row1,col1,row2,col2];
        end
    end
end
```

将方块聚类并索引

```
thre=1.7;%阈值
sim=load('../data/similarity.mat').sim;
pics=load('../data/spilit.mat').spilit;
```

[2] 实际上寻找同种方块相关系数最小值过程中需要依赖游戏区域的真值标记，并非完全的无监督，此处为形象名称。

```
[M,N,~,~]=size(sim);
class_count=0;%类别总数
class_num_max=0;%一类中最多方块数
mtx=zeros(M,N);%索引数组
for m=1:M*N
    row1=ceil(m/N);
    col1=m-N*(row1-1);
    if mtx(row1,col1)~=0%如果方块已经被归类则跳过该方块
        continue;
    else
        count=0;%该类别方块数目
        class_count=class_count+1;
        for n=m:M*N
            row2=ceil(n/N);
            col2=n-N*(row2-1);
            if sim(row1,col1,row2,col2)>thre%大于阈值则归为同一类
                count=count+1;
                mtx(row2,col2)=class_count;
            end
        end
        if count>class_num_max
            class_num_max=count;
        end
    end
end
```

【核心代码说明】

首先遍历所有方块对，维护一个最小值，得到同种方块相关系数的最小值，根据这一最小值以及 2.4 中得到的不同方块相关系数最大值确定阈值，完整代码文件为 ex_2_5_a.m。根据这一阈值再次遍历所有方块对，大于阈值的归为同一类，在 mtx 数组中标记相同的索引值，同时记录类别总数和一类中最多含有的方块数目，完整代码文件为 ex_2_5_b.m。

【运行结果】

运行代码得到所有方块的准确聚类，具体结果见“结果分析”。

【结果分析】

可以确定相同方块对之间相关系数的最小值为 1.9463>1.5526，这对方块为

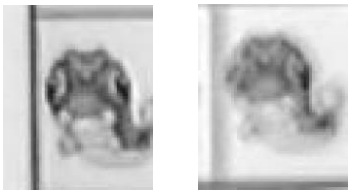


图 23 相关系数最小的一对同种方块

可以发现这两个方块一个较清晰而另一个很模糊，且前者由于拍摄原因有比较明显地歪斜，因此相关系数较小。因此根据这一结果可以设置阈值为 1.7，从而将所有方块准确聚类，聚类结果以及每类对应的索引值为

图 24 “无监督”分类索引结果

得到的 mtx 矩阵为[3]：

1	2	1	3	4	5	6	7	8	9	9	10
11	3	10	12	10	13	8	14	9	15	16	8
17	18	9	15	12	11	6	12	2	6	17	11
12	18	8	12	2	8	6	3	6	11	12	17
16	2	14	4	18	9	18	9	13	7	12	3
17	8	19	17	1	19	17	7	4	13	7	8
13	8	6	9	4	5	10	1	13	9	12	13

由此可以看出，如果能够确定一个合适的阈值，这一分类索引办法可以达到 100%的准确度。然而这一阈值并不一定对于所有游戏区域的图像都是存在的，因此下面讨论另一种方法。

2.5.2 方法二：“有监督”分类

【主要思想】

如果对于任意游戏区域图像，上述阈值存在变化，甚至可能存在不同方块相关系数最大值大于相同方块相关系数最小值的情况，因此这种“无监督”的方法可能无法准确将所有方块聚类。针对此问题，引入“标准方块类型图谱”，将分割出的每个方块经过高通滤波后依次与“标准方块类型图谱”中的方块计算相关系数，相关系数最大的一个即该方块所属类别。

【核心代码】

```
pic_s=load('pics.mat').pics;%载入标准图片
pic_t=load('../data/spilit.mat').spilit;%载入分割后的图片
info=load('../data/info.mat').info;%载入分割尺寸


[M,N]=size(pic_t);
pic_ht={};
h_c=(1+info(4))/2;%计算中心位置
w_c=(1+info(3))/2;
```

[3] 此处矩阵中元素顺序与实际游戏图像相同，与给出代码中表示顺序存在垂直翻转关系，并不影响后续操作

```
passband=2;%通带半径
for m=1:M
    for n=1:N
        temp=im2double(pic_t{m,n});%转化为 double 型
        temp=fftshift(fft2(temp));%傅里叶变换并将零频率移动到中心
        for x=1:floor(info(3)/2)
            x2=x^2;
            for y=1:floor(info(4)/2)
                y2=y^2;
                if x2+y2<passband^2%位于通带外则截断
                    temp(floor(h_c+y),floor(w_c+x))=0;%实信号频谱对称，可以由一个象限填充其他
                    temp(floor(h_c+y),ceil(w_c-x))=0;
                    temp(ceil(h_c-y),floor(w_c+x))=0;
                    temp(ceil(h_c-y),ceil(w_c-x))=0;
                end
            end
        end
        temp=real(ifft2(ifftshift(temp)));%傅里叶逆变换得到处理后的图像
        pic_ht{m,n}=temp;
    end
end

pic_hs={};
passband=2;
[height,width,~]=size(pic_s{1});
h_c=(1+height)/2;%计算中心位置
w_c=(1+width)/2;
figure;
for k=1:21
    subplot(2,21,k);
    imshow(pic_s{k});
    title(['类型',num2str(k)],'Fontsize',12);
    subplot(2,21,k+21);

temp=0.299*im2double(pic_s{k}(:,:,1))+0.587*im2double(pic_s{k}(:,:,2))+0.114*im2double(pic_s{k}(:,:,3));
%转为灰度图
    %高通滤波得到标准图谱
    temp=fftshift(fft2(temp));%傅里叶变换并将零频率移动到中心
    for x=1:floor(width/2)
        x2=x^2;
        for y=1:floor(height/2)
            y2=y^2;
            if x2+y2<passband^2%位于通带外则截断
                temp(floor(h_c+y),floor(w_c+x))=0;%实信号频谱对称，可以由一个象限填充其他
```

```
                    temp(floor(h_c+y),ceil(w_c-x))=0;
                    temp(ceil(h_c-y),floor(w_c+x))=0;
                    temp(ceil(h_c-y),ceil(w_c-x))=0;
                end
            end
        end
        temp=real(ifft2(ifftshift(temp)));%傅里叶逆变换得到处理后的图像
        pic_hs{k}=temp;
        imshow(pic_hs{k});
        pic_hs{k}=resize(pic_hs{k},info(4),info(3));
end

[M,N]=size(pic_ht);
mtx=zeros(M,N)+1;
sim=zeros(M,N,21);
for m=1:M
    for n=1:N
        for k=2:21
            %遍历所有方块，匹配最相近的标准图谱
            if
similarity(im2double(pic_ht{m,n}),pic_hs{k})>similarity(im2double(pic_ht{m,n}),pic_hs{mtx(m,n)})
                mtx(m,n)=k;
            end
        end
    end
end

function mtx=resize(mtx0,height,width)
    [m,n]=size(mtx0);
    mtx=resample(mtx0,height,m);%横向插值
    mtx=resample(mtx.',width,n);%纵向插值
    mtx=mtx.';
end
```

【核心代码说明】

完整代码文件为 ex_2_5_c.m，首先载入 pics.mat 中储存的彩色图片，根据公式

$$gray = 0.299R + 0.587G + 0.114B$$

转换为灰度图，使用与前面相同的二维高通滤波器进行滤波后得到“标准方块类型图谱”。后遍历游戏区域中分割出的所有方块，匹配到相关系数最大的一种“标准方块”，将索引值写入到 mtx 数组中。注意到这里分割出的方块和标准方块尺寸不同，因此编写了函数文件 resize.m，利用 resample 函数进行插值，实现图像的缩放。

【运行结果】

运行代码得到“标准方块类型图谱”，根据此图谱得到方块类型索引数组 mtx，见“结果分析”。

【结果分析】

通过上述方法得到“标准方块类型图谱”如下，共包含 21 种方块：

图 25 标准方块类型图谱

根据该图谱，得到方块类型索引数组 mtx 如下

6	10	6	8	18	13	4	5	15	7	7	19
17	8	19	21	19	3	15	1	7	2	9	15
16	12	7	2	21	17	4	21	10	4	16	17
21	12	15	21	10	15	4	8	4	17	21	16
9	10	1	18	12	7	12	7	3	5	21	8
16	15	11	16	6	11	16	5	18	3	5	15
3	15	4	7	18	13	19	6	3	7	21	3

这一方法得到的 mtx 数组同样正确索引了全部方块，由于两种方法中各类方块编号顺序不同，数组中元素并不相同，但同种元素的配对位置均是相同的，因此在后续操作中两种方法得到的 mtx 数组是具有等效作用的。同时注意到在给出的这一游戏区域中，第 14 类“皮卡丘”和第 20 类“可达鸭”没有出现。

2.6 模拟自动消去

【主要思想】

在上面图像分割、方块分类的基础上，模拟自动连连看，将方块涂黑表示消去。尽管上一题中用两种方法都将所有方块正确索引为类型数组，但并不能保证所有情况下捕获的游戏区域画质均能够实现这一点，因此自动连连看中不能够完全依赖索引数组进行操作，而是采取下面的方法：

- 使用 2.2 中的方法对图像进行分割并确定边缘空白距离
- 对分割后的方块对进行高通滤波
- 遍历所有方块对，得到能空间位置够消去且相关系数最大的一对，将其模拟消去
- 循环执行上一步骤，直到所有方块都被消去

这一方法从相关系数最大的开始操作，尽可能地保证了操作的正确性，同时不断消去把握较大的方块对减少了对后面方块的干扰，降低了误判的可能性。

【核心代码】

```
img=imread('graycapture.jpg');%载入整体图片并显示
info=load('../data/info').info;%载入分割尺寸
l_margin=info(1);
t_margin=info(2);
width=info(3);
height=info(4);
imshow(img);
sim=load('../data/similarity').sim;%载入相似性数据
[M,N,~,~]=size(sim);
pause(1);%暂停显示图像

%% 模拟消去
```

```
mtx=zeros(M,N)+1;
while sum(sum(mtx))~=0%未完全消去时循环遍历
    max=0;%维护相关系数最大值
    max_pair=[0,0,0,0];
    for m=1:M*N
        row1=ceil(m/N);%遍历所有图像对的组合
        col1=m-N*(row1-1);
        if mtx(row1,col1)~=0
            for n=m+1:M*N
                row2=ceil(n/N);
                col2=n-N*(row2-1);
                if mtx(row2,col2)~=0
                    if  (detect(mtx,row1,col1,row2,col2)==1)&(sim(row1,col1,row2,col2)>max)%寻找相
关系数最大值
                        max=sim(row1,col1,row2,col2);
                        max_pair=[row1,col1,row2,col2];
                    end
                end
            end
        end
    end
    mtx(max_pair(1),max_pair(2))=0;%模拟消去方块
    mtx(max_pair(3),max_pair(4))=0;
    %绘制黑矩形模拟消去
    rectangle('position',[l_margin+1+(max_pair(2)-1)*width,t_margin+1+(max_pair(1)-
1)*height,width,height],'LineWidth',1,'EdgeColor','none','FaceColor','black');
    rectangle('position',[l_margin+1+(max_pair(4)-1)*width,t_margin+1+(max_pair(3)-
1)*height,width,height],'LineWidth',1,'EdgeColor','none','FaceColor','black');
    pause(0.5);
end
```

【核心代码说明】

完整代码文件为 ex_2_6.m，首先载入分割尺寸数据、分割后的图像数据和整体图像。创建 mtx 数组记录各方块是否已经消去，维护 max 变量记录能够消去的相似度最大的方块对的相似度，max_pair 数组记录这一对方块的坐标。循环遍历所有方块对，确定能够消去的相似度最大的方块对，在 mtx 数组中记为 0 表示已经消去，同时在图像上用黑色矩形遮挡模拟消去。不断重复该过程直到所有方块均被消去。

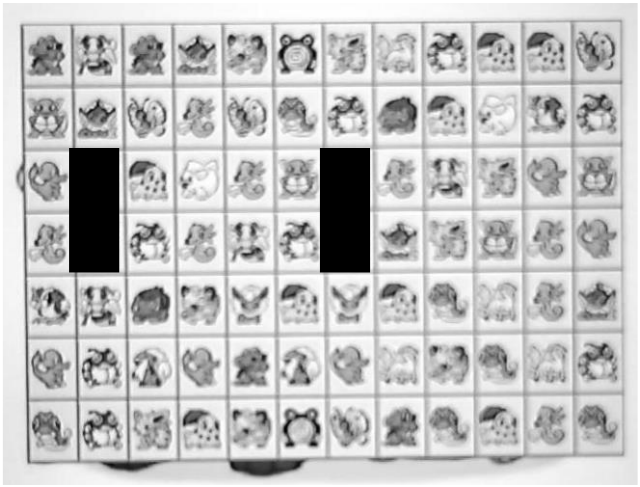
【运行结果】

代码中加入了适当的 pause 语句使得图窗可以刷新显示每一步消去的过程，运行代码得到图窗中展示模拟消去过程，过程录屏保存为“data/模拟消除_gray.mp4”，经过仔细检查，消除过程完全正确，部分截图见“结果分析”。

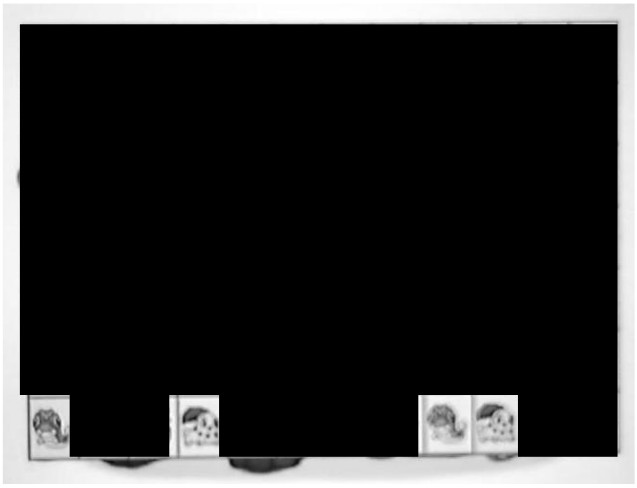
【结果分析】

上述过程能够正确消去所有方块，其中部分截图如下：

开始消除前



消除两对后



最后两对

图 26 模拟消除过程部分截图

2.7 彩色图像处理（选做）

【问题描述】

在摄像头采集的彩色图像上重做 2-6 题，注意不能简单地把彩色图像转化为灰度图。

2.7.1 图像分割

【主要思想】

使用与 2 题相似的方法，利用横向周期和相位以及纵向周期和相位得到方块宽度和高度以及左侧、顶部空白距离。彩色图像有 RGB 三个通道，此处对三个通道分别求取宽度和高度以及左侧、顶部空白距离，最后取平均得到整个图像中方块的宽度和高度以及左侧、顶部空白距离。

【核心代码】

```
pic0=imread('colorcapture.jpg');%读取彩色图
[m,n,~]=size(pic0);

width0=0;
height0=0;
l_margin0=0;
t_margin0=0;

for ch=1:3
    pic=pic0(:,:,ch);
    x=mean(pic,1)-mean(mean(pic,1));%按行平均，除去直流分量
    xa=bandpass(x,[5,30],n);%带通滤波

    N=n;%时域采样点数
    T=1;%时间范围
    [t,omg,FT,IFT]=prefourier([0,T-T/N],N,[0,500],10000);
    F=FT*xa.';%傅里叶变换
    [~,b]=max(abs(F));%找出基波频率
    width=round(2*pi*n/omg(b));%根据基波频率计算横向周期
    width0=width0+width;

    F=FT*x.';%计算原函数的频谱
    phase=angle(F(b));%计算基波频率的相位
    t=[0:n];
    base=abs(F(b))*cos(2*pi/width*t+phase);%生成基波函数
    [~,l_margin]=max(base(1:width));%根据基波函数得到左侧空白距离
    l_margin0=l_margin0+l_margin;

    %计算纵向周期、顶部空白距离步骤完全相同，不再画出图像
    y=mean(pic,2)-mean(mean(pic,2));
    ya=bandpass(y,[5,30],m);

    N=m;
    T=1;
```

```
    [t,omg,FT,IFT]=prefourier([0,T-T/N],N,[0,500],10000);
    F=FT*ya;
    [~,b]=max(abs(F));
    height=round(2*pi*m/omg(b));
    height0=height0+height;

    F=FT*y;
    phase=angle(F(b));
    t=[0:m];
    base=abs(F(b))*cos(2*pi/height*t+phase);
    [~,t_margin]=max(base(1:height));
    t_margin0=t_margin0+t_margin;
end

width0=round(width0/3);%计算平均值
height0=round(height0/3);
l_margin0=round(l_margin0/3);
t_margin0=round(t_margin0/3);

N=floor((n-l_margin0)/width0);%计算每行方块数目
M=floor((m-t_margin0)/height0);%计算每列方块数目
```

【核心代码说明】

利用 prefourier 函数求傅里叶变换矩阵，对三个颜色通道的空域波形分别做傅里叶变换，通过 2.2 中的方法依次求出方块的宽度和高度以及左侧、顶部空白距离，最后取平均得到整体方块分割。完整代码文件为 ex_2_7_a.m。

【运行结果】

运行程序得到了对应的左侧空白距离l_margin，顶部空白距离t_margin，方块宽度$width$，方块高度$height$。由此划分出每个方块，结果见“结果分析”。将分割后的图像保存为 spilit_c.mat 中，设有$M \times N$个方块，则 spilit_c 为$M \times N$个$cell$的单元数组，各个$cell$为一个$height \times width \times 3$的矩阵，保存分割后的一个方块图片。分割的距离信息保存在 info_c.mat 中，info 为 4 列的行向量，依次存储左侧空白距离l_margin，顶部空白距离t_margin，方块宽度$width$，方块高度$height$。

【结果分析】

得到 RGB 三个通道的空域波形和频谱如下：

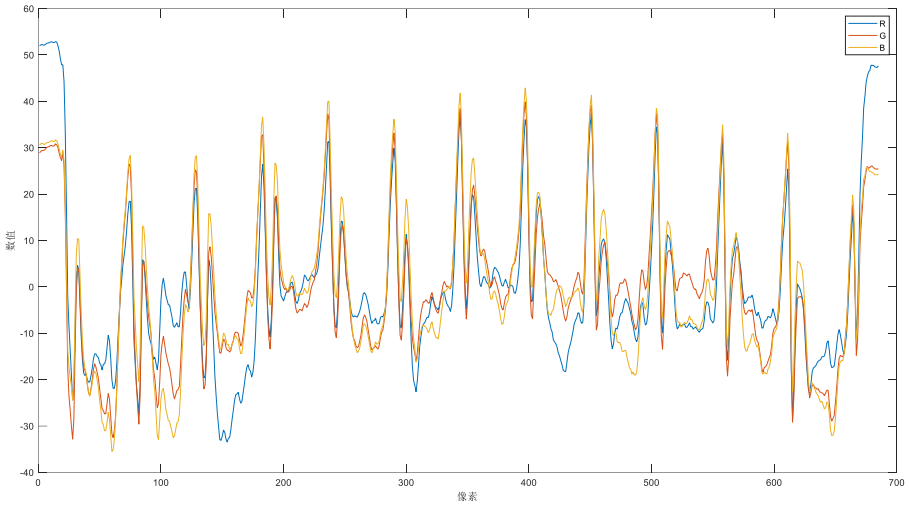


图 27 彩色图像三通道空域波形

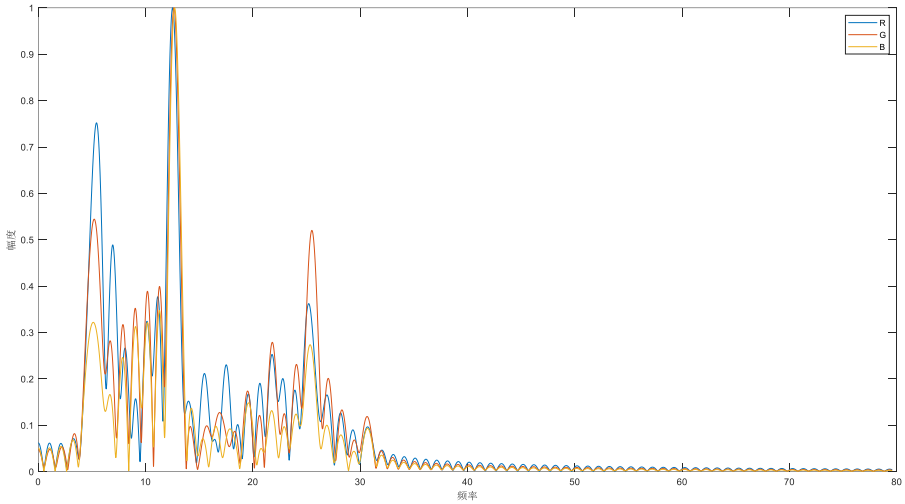


图 28 彩色图像三通道频谱

可以发现三个通道的空域波形有相似的周期性，因此频谱的主要峰值也是重合的。根据三个通道分别得到的方块长度、宽度、左侧空白距离和顶部空白距离是接近的，取均值后可以得到更加准确的结果。将分割后方块画出如下：

图 29 彩色图像分割结果

能够准确地分割出不同方块。

2.7.2 方块相似性分析

【主要思想】

充分利用彩色图像三个通道的信息计算方块的相似性。首先对三个通道分别做匹配滤波得到相关系数，三者取平均得到通道上的相关系数sim_1。再计算每个方块的颜色直方图，对每对方块的颜色直方图向量求得相关系数sim_C。线性加权sim_1与sim_C得到总相关系数sim。

【核心代码】

计算颜色直方向量

```
function v = color(mtx,k)
%COLOR 计算图像颜色直方向量
%    输入：mtx 图像 RGB 矩阵
%    输出：v 颜色直方向向量
%           n 1-8 整数，二进制保留位数
    [M,N,~]=size(mtx);
    v=zeros(1,power(2,3*k));
    for m=1:M
        for n=1:N
            R=dec2bin(mtx(m,n,1),8);%转换为二进制并截取高 n 位
            G=dec2bin(mtx(m,n,2),8);
            B=dec2bin(mtx(m,n,3),8);
            temp=bin2dec([R(1:k),G(1:k),B(1:k)])+1;%连接并转为 10 进制
            v(temp)=v(temp)+1;
        end
    end
end
```

计算综合相关系数

```
%% 高通滤波
pic_h={};
h_c=(1+height)/2;%计算中心位置
w_c=(1+width)/2;
passband=2;%通带半径
for m=1:M
    for n=1:N
        temp0=im2double(pic{m,n});
        for ch=1:3
            temp=temp0(:,:,ch);%转化为 double 型
            temp=fftshift(fft2(temp));%傅里叶变换并将零频率移动到中心
            for x=1:floor(width/2)
                x2=x^2;
                for y=1:floor(height/2)
                    y2=y^2;
                    if x2+y2<passband^2%位于通带外则截断
                        temp(floor(h_c+y),floor(w_c+x))=0;%实信号频谱对称，可以由一个象限填充其他
                        temp(floor(h_c+y),ceil(w_c-x))=0;
                        temp(ceil(h_c-y),floor(w_c+x))=0;
                        temp(ceil(h_c-y),ceil(w_c-x))=0;
                    end
                end
            end
            temp=real(ifft2(ifftshift(temp)));%傅里叶逆变换得到处理后的图像
            temp0(:,:,ch)=temp;
        end
        pic_h{m,n}=temp0;
    end
end

co={};%逐个方块计算色彩直方图
for m=1:M
    for n=1:N
        co{m,n}=color(pic{m,n},4);
    end
end

%RGB 通道分别做匹配滤波
%匹配滤波结果与色彩直方图结合
sim_1=zeros(M,N,M,N);
sim_c=zeros(M,N,M,N);
for m=1:M*N
```

```
    row1=ceil(m/N);%遍历所有图像对的组合
    col1=m-N*(row1-1);
    for n=m:M*N
        row2=ceil(n/N);
        col2=n-N*(row2-1);
        s_1=(similarity(pic_h{row1,col1}(:,:,1),pic_h{row2,col2}(:,:,1))+...
            similarity(pic_h{row1,col1}(:,:,2),pic_h{row2,col2}(:,:,2))+...
            similarity(pic_h{row1,col1}(:,:,3),pic_h{row2,col2}(:,:,3)))/3;%计算相关系数
        s_c=similarity_a(co{row1,col1},co{row2,col2});
        sim_1(row1,col1,row2,col2)=s_1;%sim(picture1,picture2)=sim(picture2,picture1)
        sim_1(row2,col2,row1,col1)=s_1;%对称填充矩阵中两个位置
        sim_c(row1,col1,row2,col2)=s_c;
        sim_c(row2,col2,row1,col1)=s_c;
    end
end
sim_1=(sim_1-mean(sim_1,'all'))/sqrt(var(sim_1,1,'all'));%归一化
sim_c=(sim_c-mean(sim_c,'all'))/sqrt(var(sim_c,1,'all'));
sim=1*sim_1+0*sim_c;%两者加权叠加
```

【核心代码说明】

color.m 中定义了计算颜色直方向量的函数，使用 dec2bin 函数将 RGB 数值转换为二进制，同时取高三位连接后转换回十进制。由此建立 RGB 颜色和十进制数的映射，得到长度为 512 的颜色直方向量。在 similarity_a.m 中定义计算一维矢量相关系数的函数，同样使用相关定理计算相关系数。

此处的高通滤波器对彩色图像的三个通道分别进行滤波后叠加，之后三个通道分别匹配滤波得到三个相关系数，取平均后得到通道上的相关系数矩阵 sim_1，计算得到颜色直方图的相关系数矩阵 sim_c，两者加权得到总的相关系数矩阵 sim，完整代码文件为 ex_2_7_b.m。

【运行结果】

探究各参数取值对结果的影响以及得到的结果见“结果分析”。

【结果分析】

首先在不加如颜色直方图的情况下分析高通滤波通带对于结果的影响，仍使用 2.3 中定义的裕度函数评价分类结果的优劣，结果如下：

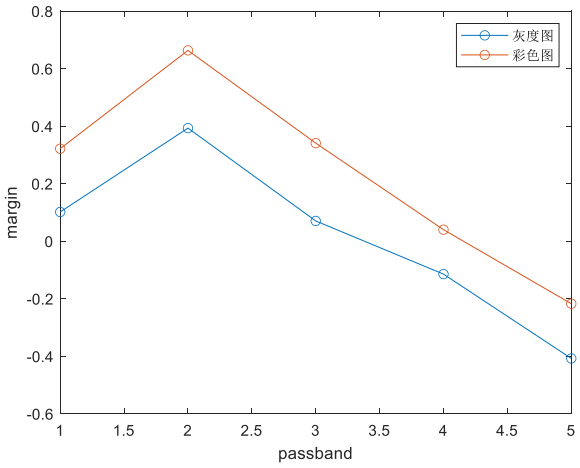


图 30 通带半径对分类效果的影响

可以看出仍是通带半径为 2 时阈值函数值最大，分类效果最佳。同时发现彩色图上的分类效果总是优于灰度图的，说明彩色图纳入了更丰富的信息，更有利于分类。

下面讨论色彩直方图在分类中的作用，部分方块的色彩直方图如下（考虑视觉效果，这里用 plot 函数绘制“直方图”）

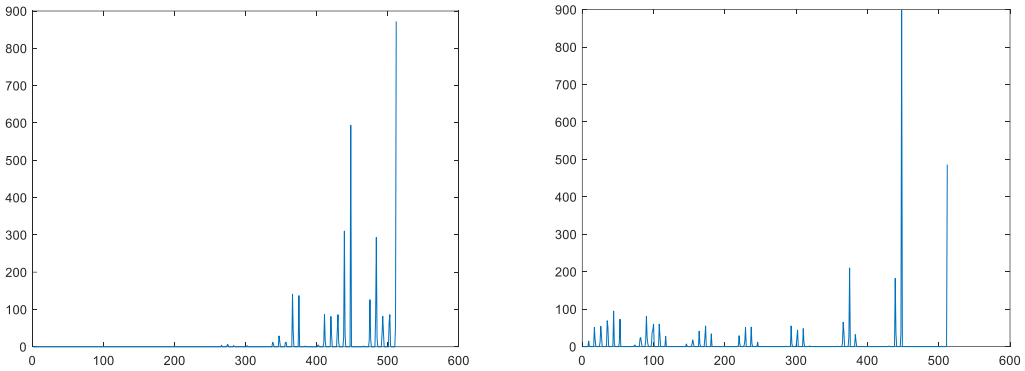


图 31 部分方块的色彩直方图

在计算总相关系数时存在一个加权系数，该系数的不同影响分类效果如下（高通滤波通带设置为 2），图中ρ为三通道分别匹配滤波得到的相关系数前的加权系数，同时有颜色直方图前的加权系数为$1-\rho$。

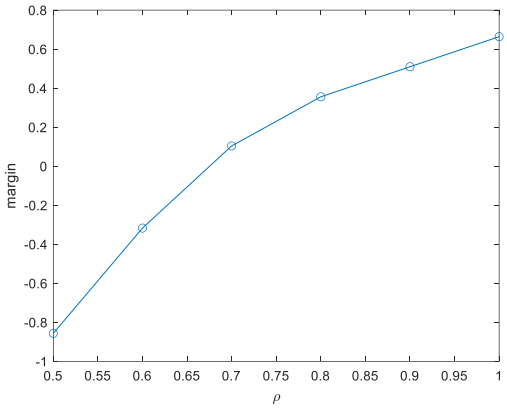


图 32 加权系数对分类效果的影响

可以发现，系数为 1 时，即只采用三通道匹配滤波分类效果最好，纳入颜色直方图后反而使得分类效果变差。观察方块种类的图像可以发现有多种方块有着相近的颜色，如图

图 33 色彩相近的方块

因此颜色直方图反而造成了不同方块之间的混淆，干扰了三通道匹配滤波的分类效果。显然这一结论仅仅适用于这一套“精灵连连看”方块，如果其他连连看游戏中不同方块之间颜色差异很大而，加入颜色直方图应当是有积极效果的。下面的问题都是针对这一套连连看方块进行，因此设置$\rho=1$，不纳入颜色直方图信息。

十对相关系数最大的方块为

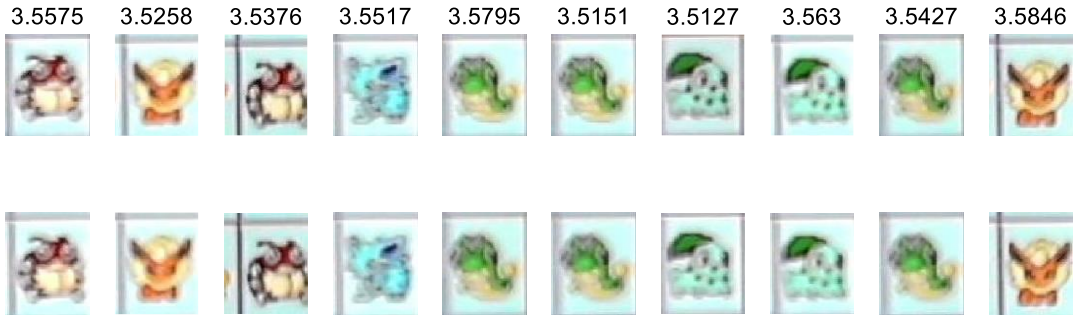


图 34 彩色图中十对相关系数最大的方块

可见这十对方块均属于同种类别。十对不同类别但相关系数最大的方块为

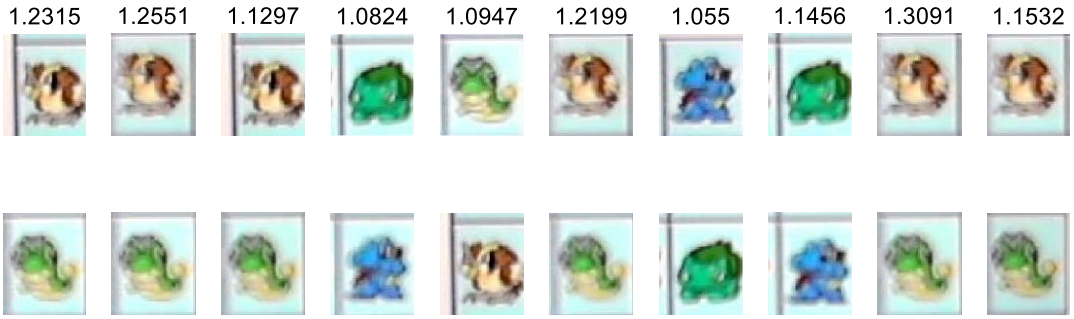


图 35 彩色图中十对相关系数最大的异种方块

这些方块对在视觉上看来并不相近，但是仔细观察可以发现这些方块对图像的倾斜方向、在块中的分布有相似之处

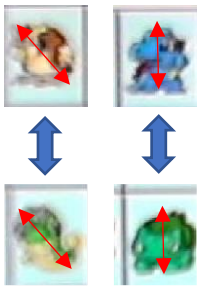


图 36 异种方块之间的相似性

因而计算相关系数中使用匹配滤波会得到较大的相关系数。

2.7.3 游戏区域映射为索引值

【主要思想】

根据 pic.mat 中的方块图像生成彩色的“标准方块类型图谱”，计算每个方块与标准图谱中方块的相关系数，相关系数最大的则为所属类别，此处计算相关系数使用三通道匹配滤波。

【核心代码】

```
[M,N]=size(pic_t);
mtx=zeros(M,N)+1;
sim=zeros(M,N,21);
for m=1:M
```

```
    for n=1:N
        for k=2:21
            %遍历所有方块，匹配最相近的标准图谱，利用三通道匹配滤波
            if similarity(im2double(pic_ht{m,n}(:,:,1)),pic_hs{k}(:,:,1))+...
                similarity(im2double(pic_ht{m,n}(:,:,2)),pic_hs{k}(:,:,2))+...
                similarity(im2double(pic_ht{m,n}(:,:,3)),pic_hs{k}(:,:,3))>...
                similarity(im2double(pic_ht{m,n}(:,:,1)),pic_hs{mtx(m,n)}(:,:,1))+...
                similarity(im2double(pic_ht{m,n}(:,:,2)),pic_hs{mtx(m,n)}(:,:,2))+...
                similarity(im2double(pic_ht{m,n}(:,:,3)),pic_hs{mtx(m,n)}(:,:,3))
                 mtx(m,n)=k;
            end
        end
    end
end
```

【核心代码说明】

完整代码文件为 ex_2_7_c.m，其思想与 2.5 中“有监督”分类完全相同，只是此处高通滤波器对彩色图像的三个通道分别滤波，计算相关系数使用三个通道依次匹配滤波并求平均值。

【运行结果及分析】

首先得到高通滤波后的彩色“标准方块类型图谱”

图 37 彩色“标准方块类型图谱”

根据此图谱得到索引数组为

6	10	6	8	18	13	4	5	15	7	7	19
17	8	19	21	19	3	15	1	7	2	9	15
16	12	7	2	21	17	4	21	10	4	16	17
21	12	15	21	10	15	4	8	4	17	21	16
9	10	1	18	12	7	12	7	3	5	21	8
16	15	11	16	6	11	16	5	18	3	5	15
3	15	4	7	18	13	19	6	3	7	21	3

与前面灰度图上得到的完全相同。

2.7.4 模拟自动消去

【主要思想】

使用与 2.6 中相同的方法，首先寻找能够消去的方块对中相关系数最大的模拟消去，不断循环此过程直到全部消去。不同之处在于这里的相关系数是通过三通道分别匹配滤波后取平均得到的。

【核心代码】

本题代码与 2.6 基本一致，只是将灰度图的相似度张量 similarity.m、分割尺寸 info.mat 替换成彩色图的相似度张量 similarity_c.m、分割尺寸 info_c.mat。在此不再做赘述，完整代码文件为 ex_7_2_d.m。

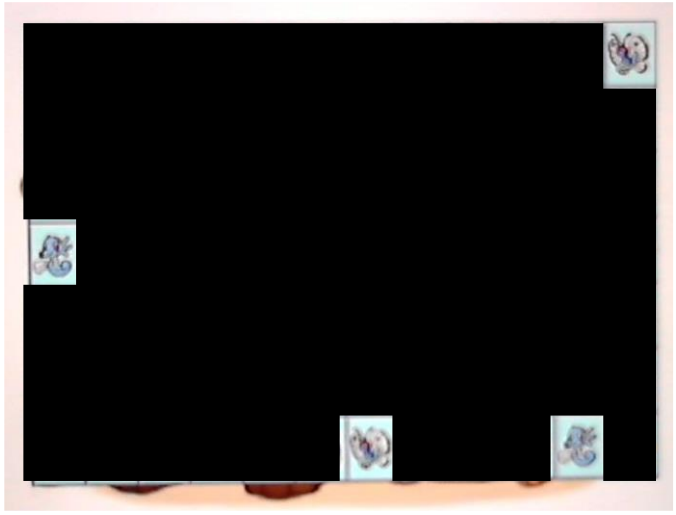
【运行结果】

运行代码得到图窗中展示模拟消去过程，过程录屏保存为“data/模拟消除_gray.mp4”，经过仔细检查，消除过程完全正确，部分截图见“结果分析”。

【结果分析】

模拟消除部分截图如下

消除两对后



最后两对

图 38 彩色图像模拟消除

彩色图像计算出的相关系数与灰度图像有所不同，因此模拟消去的顺序也有不同，但经过检查两者都是正确的。这也说明了一个游戏区域通常有不止一种全部消除的解法。

2.8 自动连连看（选做）

【问题描述】

基于上述步骤实现一个自动连连看。

【主要思想】

题目中给出的框架 ai.m 是基于 matlab 程序生成的连连看小游戏进行的，为将上面的图形处理步骤运用于更加真实的连连看游戏，本题中未使用该框架，而是编写了一个简单的 GUI 界面，实现对经典“宠物连连看”游戏的自动消除。该游戏界面如下

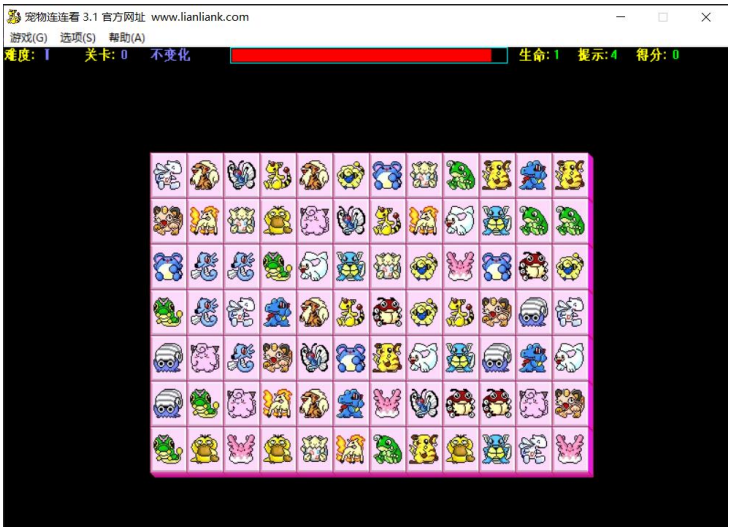


图 39 宠物连连看游戏界面

同时题目中给出的插件“按键精灵”受到计算机防火墙影响，调用比较不便，查阅资料发现 matlab 本身即可通过代码移动鼠标并点击，本题使用这种方式，编写函数文件 click.m 实现这一功能。

实现自动消去的功能步骤与前面相同，利用题目给出的 user_camera.m 中的函数通过摄像头捕获屏幕图像，首先由傅里叶变换计算得到方块宽度、高度、左侧空白和顶部空白距离；再对图像进行高通滤波后做三个通道上的匹配滤波得到方块对之间的相关系数；最后搜索当前能够消去的、相关系数最大的方块对，控制鼠标点击消去。

考虑到连连看游戏中存在随机打乱方块、消除后方块向下/上/左/右对齐的关卡，需要判断哪里有方块、哪里已经消去。利用该游戏背景为黑色的特点，计算每个分块的颜色直方图，如果有大于 60%的像素为黑色则判断该方块已经消去，这一功能编写 color_judge.m 函数实现。

由于图像分类识别并非完全准确，实现在试错一对方块后更改尝试其他方块对。编写 judge.m 函数实现判断一对方块是否已经被尝试消除而失败过。

【核心代码】

```
function pushbutton1_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
handles.text1.String='正在获取并分析图像...';
realcapture=user_camera();
imshow(realcapture);

%分割图像，代码略

previous=[];
mtx=zeros(M,N)+1;
mtx_sum=M*N;
while sum(sum(mtx))~=0
    realcapture=user_camera();
    imshow(realcapture);
```

```
pic={};
for k=1:N
    for t=1:M
        pic{t,k}=realcapture(t_margin0+(t-1)*height0+1:t_margin0+t*height0,l_margin0+(k-
1)*width0+1:l_margin0+k*width0,:);
    end
end

 %计算颜色直方图,30%以上为黑色则判定为已经消去
for k=1:M
    for t=1:N
        if color_judge(pic{k,t},3)==0
            mtx(k,t)=0;
        else
            mtx(k,t)=1;
        end
    end
end
s_temp=sum(sum(mtx));
if s_temp<mtx_sum
    previous=[];
    mtx_sum=s_temp;
end

%高通滤波，代码略

%消去一对
max_value=0;%维护相关系数最大值
max_pair=[0,0,0,0];
for m=1:M*N
    row1=ceil(m/N);%遍历所有图像对的组合
    col1=m-N*(row1-1);
    if mtx(row1,col1)~=0
        for n=m+1:M*N
            row2=ceil(n/N);
            col2=n-N*(row2-1);
            if mtx(row2,col2)~=0 & judge(previous,row1,col1,row2,col2)==1
                if detect(mtx,row1,col1,row2,col2)==1
                    s=(similarity(pic_h{row1,col1}(:,:,1),pic_h{row2,col2}(:,:,1))+...
                    similarity(pic_h{row1,col1}(:,:,2),pic_h{row2,col2}(:,:,2))+...
                    similarity(pic_h{row1,col1}(:,:,3),pic_h{row2,col2}(:,:,3)));
                    if s>max_value
                        max_value=s;
                        max_pair=[row1,col1,row2,col2];
```

```
                                end
                            end
                        end
                    end
                end
            end
        click(180+(max_pair(2)-1)*40,180+(max_pair(1)-1)*50);
        click(180+(max_pair(4)-1)*40,180+(max_pair(3)-1)*50);
        previous=[previous;max_pair];
        pause(0.3);
end
handles.text1.String='游戏结束！';
```

【核心代码说明】

完整代码文件为 autolink.m，mtx 数组记录每个位置方块是否已经被消去。维护 max_value 记录当前能够消去的相关系数最大的方块对相关系数，维护 max_pair 记录当前能够消去的相关系数最大的方块对的位置。previous 数组记录已经尝试错误的方块对，当 mtx 数组和减小，即成功消去一对方块后，previous 置空。

【运行结果】

运行 autolink.m 文件打开 GUI 界面，同时启动连连看游戏，将窗口拖动到屏幕左上角。点击“自动游戏”按钮开始自动消除，GUI 界面如下，节目中显示摄像头拍摄到的画面，下方文字显示当前运行状态。

图 40 GUI 界面

自动游戏场景如图，同时录制了两个视频文件：“第一关.mp4”为自动游戏第一关，方块不移动；“第二关.mp4”为自动游戏第二关，方块消除后向下对齐。

图 41 自动游戏场景

【结果分析】

本题实现了真实的自动连连看，能够自动破解经典连连看中各种关卡，同时使用了一些编程优化方法，尽可能地加快了计算速度，在限时模式中能够绰绰有余地过关。

由于涉及摄像头捕获画面并进行识别，摄像头摆放位置与角度、屏幕亮度和反光、周围环境光均对自动游戏有一定影响，一些条件下可以无法正确分析图像并进行消除。调节上述调节合适时即可实现自动消除效果。

3 实验总结

本实验中利用信号与系统的方法进行图像处理，实现了基本功能的自动连连看，在此过程中有如下几点体会。

首先，较为复杂的系统构建是难以一蹴而就的，往往需要拆分为多个部分逐一实现，最后再进行整体调试。本实验中分别实现图像分割、相似性计算等步骤，再由灰度图迁移到彩色图，最后将模拟图像替换成真实的摄像头画面，这一过程正体现了这种思想。

其次，实际中应用的系统需要面对相当复杂的环境而做出设计。本实验实现的自动连连看系统对于光线、摄像头位置等因素都较为敏感，在这一方面上距离真实的工程系统还有一定差距。

最后，几乎所有的实际系统中或多或少地都设计算法有关的问题，在优化系统时，尤其要对于多次反复执行的代码进行优化，最大限度地提高系统工作效率。

4 附：文件清单

附表 1 文件清单[4]

文件（夹）名	说明
实验报告.pdf（本文档）	实验报告
\code\linkgame	1“制作自己的连连看”1.1~1.3
detect.m	1.2 函数：实现 detect 函数（判断是否可消去）
omg.m	1.3 函数：实现 omg 函数（自动消去算法）
其余文件	作业中已给出的其他文件
\code\others	1“制作自己的连连看”1.4（选做）
generate.m	1.4.1 函数：生成有意义、可完全消除的游戏区域

4 为保证程序运行正确，部分文件同时存在于多个文件夹下，此处均重复列出

pics.mat	1.4.1 储存方块图片
create_game.m	1.4.1 生成游戏区域并显示
detect.m	1.4.3 函数：detect 函数（判断方块对是否可消去）
omg.m	1.4.3 函数：omg 函数（自动消去算法，顺序遍历）
omg_1.m	1.4.3 函数：omg 函数（自动消去算法，随机遍历）
compare.m	1.4.3 比较两种自动消去算法耗时
\code\process	2 攻克别人的连连看 2.1-2.7 代码文件
graygroundtruth.jpg	游戏区域截图（灰度图）
graycapture.jpg	游戏区域摄像头捕获图像（灰度图）
colorcapture.jpg	游戏区域摄像头捕获图像（彩色图）
pics.mat	储存标准方块图片
write_truth.m	生成游戏区域真值标记
similarity.m	函数：计算两矩阵相似性
similarity_a.m	函数：计算两向量相似性
prefourier.m	函数：计算傅里叶变换矩阵
ex_2_1_a.m	2.1 分割屏幕截图 FFT 方法
ex_2_1_b.m	2.1 分割屏幕截图傅里叶变换方法
ex_2_2.m	2.2 分割摄像头捕获图像（灰度图）
ex_2_3and4.m	2.3&2.4 计算方块相似性（灰度图）
ex_2_5_a.m	2.5.1 游戏区域映射为索引值（“无监督”分类）
ex_2_5_b.m	
ex_2_5_c.m	2.5.2 游戏区域映射为索引值（“有监督”分类）
resize.m	2.5.2 函数：插值改变图像尺寸
ex_2_6.m	2.6 模拟自动消去（灰度图）
detect.m	2.6 函数：detect 函数（判断方块对是否可消去）
ex_2_7_a.m	2.7.1 图像分割（彩色图）
ex_2_7_b.m	2.7.2 计算方块相似性（彩色图）
color.m	2.7.2 函数：计算方块颜色直方图
ex_2_7_c.m	2.7.3 游戏区域映射为索引值（彩色图）
resize_a.m	2.7.3 函数：插值改变三通道图像尺寸
ex_2_7_d.m	2.7.4 模拟自动消去（彩色图）
\code\data	2 攻克别人的连连看 2.1-2.7 运行结果
truth.mat	游戏区域真值标记
info.mat	2.2 游戏区域分割尺寸（灰度图）
spilit.mat	2.2 游戏区域分割出的方块（灰度图）
similarity.mat	2.3 游戏区域方块对相关系数（灰度图）
模拟消除_gray.mp4	2.6 灰度图模拟消除录屏
info_c.mat	2.7.1 游戏区域分割尺寸（彩色图）
spilit_c.mat	2.7.1 游戏区域分割出的方块（彩色图）
similarity_c.mat	2.7.2 游戏区域方块对相关系数（彩色图）
模拟消除_color.mp4	2.7.4 彩色图模拟消除录屏
\code\auto_game	2.8 自动连连看
autolink.fig	GUI 图形界面

autolink.m	GUI 主程序
click.m	函数：控制鼠标移动并点击
detect.m	函数：detect 函数（判断方块对是否可消去）
prefourier.m	函数：计算傅里叶变换矩阵
similarity.m	函数：计算两矩阵相似性
user_camera.m	函数：获取摄像头图像
color_judge.m	函数：根据颜色判断方块是否已被消去
judge.m	函数：判断当前方块是否已经尝试过
第一关.mp4	第一关（方块不移动）录像
第二关.mp4	第二关（方块向下移动）录像