MATLAB 综合实验之连连看*

聂浩 无 31 2013011280

2015 年 9 月 12 日

1 制作自己的连连看

(1) MBATLAB 为环境下，设置当前路径为 linkgame 行，运行 linkgame （打开 linkgame.fig 键或右键 p linkgame.p 点“运行”），熟悉游戏。（上述程序已经过的测试。）

(2) 注意 linkgame 目录下有个 detect.p。它的功能是检测块是否可以消除。现在请你把它移动到其他文件夹或删掉! 把然后把 linkgame/reference 目录下的 detect.m 复制到 linkgame。目录下。detect.m 文件中是 detect 函数，函数以图像块的索号矩阵与要判断的两个块的下标为输入，如果两个块能消掉则输出 11，否则输出 00。请根据文件中的注释提示，实现判断块是否可以消除的功能。写完后再次运行 linkgame，检验游戏是否仍然可以正确运行，当你的程序的判断结果有误时，在游戏界面右下角会有提示。（注意：当 detect.p 文件存在时，detect.m 文件将不会被执行，所以测试时一定要移走 detect.p）

利用讲义中的十字判断法进行了判断，使用求和的方式检测通路上是否都是零。

值得说明的是边缘检测是比较麻烦；当两个块相等时才应该进行下一步判断，这里值得注意。具体可见 check.m 代码如下：detect.m

```
function bool = detect(mtx, x1, y1, x2, y2)
    %================================ 参数说明 ================================

    % 输入参数中，mtx为图像块的矩阵，类似这样的格式：
    % [ 1 2 3;
    %   0 2 1;
    %   3 0 0 ]
    % 相同的数字代表相同的图案，0代表此处没有块。
    % 可以用[m, n] = size(mtx)获取行数和列数。
```

*所有的.m 文件均采用 utf8 编码，windows 版 matlab 中打开可能会出现中文乱码的情况，请用其它编辑器打开

```
% (x1, y1)与（x2, y2）为需判断的两块的下标，即判断mtx(x1, y1)与mtx(x2, y2)
% 是否可以消去。

% 注意mtx矩阵与游戏区域的图像不是位置对应关系。下标(x1, y1)在连连看界面中
% 代表的是以左下角为原点建立坐标系，x轴方向第x1个，y轴方向第y1个

% 输出参数bool = 1表示可以消去，bool = 0表示不能消去。

%% 在下面添加你的代码O(_)O
    [m, n] = size(mtx);
    x=[x1 x2];
    y=[y1 y2];
    [x_max,which_x_max]=max(x);
    [x_min,which_x_min]=min(x);
    [y_max,which_y_max]=max(y);
    [y_min,which_y_min]=min(y);
    if(mtx(x1,y1)~=mtx(x2,y2))
        bool=0;
        return;
    end

    bool = 0;
    %直线情况
    if x1==x2
        if sum(mtx(x1,y_min:y_max))==(mtx(x1,y1)+mtx(x2,y2))
            bool=1;
            return
        end
    end
    if y1==y2
        if sum(mtx(x_min:x_max,y1))==(mtx(x1,y1)+mtx(x2,y2))
            bool=1;
            return
        end
    end
    %单折线情况和边缘
    if (sum(mtx(1:x(1),y(1)))==mtx(x(1),y(1))&&sum(mtx(1:x(2),y
```

```
    (2)))==mtx(x(2),y(2)))
47          bool=1;
48          return;
49      end
50      if (sum(mtx(x(1):m,y(1)))==mtx(x(1),y(1))&&sum(mtx(x(2):m,y
        (2)))==mtx(x(2),y(2)))
51          bool=1;
52          return;
53      end
54      if (sum(mtx(x(1),1:y(1)))==mtx(x(1),y(1))&&sum(mtx(x(2),1:y
        (2)))==mtx(x(2),y(2)))
55          bool=1;
56          return;
57      end
58      if (sum(mtx(x(1),y(1):n))==mtx(x(1),y(1))&&sum(mtx(x(2),y(2):
        n))==mtx(x(2),y(2)))
59          bool=1;
60          return;
61      end
62      if check(x,y,mtx,m,n)==1
63          bool=1;
64          return;
65      end
66      %双折线x方向
67      s=0;
68      x_tmp=x;
69      for i=x_min-1:-1:1
70          s=s+mtx(i,y(which_x_min));
71          if s~=0;
72              break;
73          end
74          x_tmp(which_x_min)=i;
75          if check(x_tmp,y,mtx,m,n)==1
76              bool=1;
77              return;
78          end
79      end
80      s=0;
81      for i=x_min+1:m
```

```
82                  s=s+mtx(i,y(which_x_min));
83                  if s~=0;
84                      break;
85                  end
86                  x_tmp(which_x_min)=i;
87                  if check(x_tmp,y,mtx,m,n)==1
88                      bool=1;
89                      return;
90                  end
91              end
92              s=0;
93              y_tmp=y;
94              for i=y_min-1:-1:1
95                  s=s+mtx(x(which_y_min),i);
96                  if s~=0;
97                      break;
98                  end
99                  y_tmp(which_y_min)=i;
100                 if check(x,y_tmp,mtx,m,n)==1
101                     bool=1;
102                     return;
103                 end
104             end
105             s=0;
106             for i=y_min+1:n
107                 s=s+mtx(x(which_y_min),i);
108                 if s~=0;
109                     break;
110                 end
111                 y_tmp(which_y_min)=i;
112                 if check(x,y_tmp,mtx,m,n)==1
113                     bool=1;
114                     return;
115                 end
116             end
117     end
```

check.m

```
1  function [bool]=check(x,y,mtx,m,n)
```

```
    if mtx(x(1),y(2))==0
        if (sum(mtx(x(1),min(y):max(y)))+sum(mtx(min(x):max(x),y(2)))
            )==(mtx(x(1),y(1))+mtx(x(2),y(2)))
            bool=1;
            return
        end
    end
    if mtx(x(2),y(1))==0
        if (sum(mtx(x(2),min(y):max(y)))+sum(mtx(min(x):max(x),y(1)))
            )==(mtx(x(1),y(1))+mtx(x(2),y(2)))
            bool=1;
            return
        end
    end
    %边缘
    if (x(1)==1&&x(2)==1)||(x(1)==m&&x(2)==m)||(y(1)==1&&y(2)==1)||(y
        (1)==n&&y(2)==n)
            bool=1;
            return;
    end
    bool=0;
    return;
end
```

(3) 你一定发现了“外挂”模式，是不是很有趣？逐一自动消除所有的块的功由能是由 link 的目录的 omg.p 实现的。现在请你把它也删掉！然后把 link/reference 目录下的 omg.m 复制到 link 目录下。omg.m 文件的注释中对输入输出变量做了详细说明，请以这个文件为基础，实现逐一自动消除所有块的功能。（同上题要移走 omg.p 文件。）写完后再次运行 linkgame，检验自动模式是否正确。（在自动点击过程中可接 F12）

这里有两种思路，一种是从外圈逐渐向内消；另一种是逐一消除同一种块，多次循环即可，这也是手动消除的常用方法，这里选用这种。有趣的是，这并不能保证一定能够解出来所有有解的连连看，一些精巧设计的连连看需要按照一定的顺序消除，甚至是唯一的顺序才能解开——实际上在本代码的消除顺序下也无法解开第二部分图像中的连连看。至于是否能设计一种一定能找到合理解的算法，这涉及到了图论和组合数学的知识，不进行详细讨论。本部分代码如下 (omg.m):

```
function steps = omg(mtx)
    % ———————— 输入参数说明 ————————
```

```
3
4       %    输入参数中，mtx为图像块的矩阵，类似这样的格式：
5       %    [ 1 2 3;
6       %      0 2 1;
7       %      3 0 0 ]
8       %    相同的数字代表相同的图案，0代表此处没有块。
9       %    可以用[m, n] = size(mtx)获取行数和列数。
10
11      %    注意mtx矩阵与游戏区域的图像不是位置对应关系。下标(x1, y1)
            在连连看界面中
12      %    代表的是以左下角为原点建立坐标系，x轴方向第x1个，y轴方向第y1
            个
13
14      %————————————— 输出参数说明 —————————————%
15
16      %    要求最后得出的操作步骤放在steps数组里,格式如下：
17      %    steps(1)表示步骤数。
18      %    之后每四个数x1 y1 x2 y2，代表把mtx(x1,y1)与mtx(x2,y2)
            表示的块相连。
19      %    示例： steps = [2, 1, 1, 1, 2, 2, 1, 3, 1];
20      %    表示一共有两步，第一步把mtx(1,1)和mtx(1,2)表示的块相连，
21      %    第二步把mtx(2,1)和mtx(3,1)表示的块相连。
22
23      %%————————————— 请在下面加入你的代码 O(_)O~ —————————————
24
25      steps(1) = 0;
26      mtx_tmp=mtx;
27      size=max(max(mtx));
28      count=0;
29      while(size~=0)
30          for i=1:size
31              [m,n]=find(mtx==i);
32              for j=1:length(m)
33                  for k=j+1:length(n)
34                      if mtx(m(j),n(j))~=0&&mtx(m(k),n(k))~=0;
35                          if detect(mtx,m(j),n(j),m(k),n(k))==1
36                              count=count+1;
37                              steps=[steps m(j) n(j) m(k) n(k)];
38                              mtx(m(j),n(j))=0;
```

```
                                mtx(m(k),n(k))=0;
                            end
                        end
                    end
                end
            end
            size=max(max(mtx));
            if(mtx_tmp==mtx) break;
            end
            mtx_tmp=mtx;%避免死循环
        end
        steps(1)=count;
        return
end
```

(4) 自由发挥

这里自己设计了一个连连看矩阵生成程序，生成一个随机矩阵，然后用上一问的方法检测是否有解。这里没有进行难度判断的代码实现，但设想为使用不同的图像判断顺序对生成的矩阵进行判断，能够过得出解的顺序越少难度越高。代码如下 generate.m

```
clear;clc;close all;
check=0;
r=[];
while(check==0)
    r=ceil(21*rand(1,42));
    r=[r r];
    r=reshape(r,[7,12]);
    check=omg_check(r);
end
r
```

2 攻克别人的连连看

(1) 在 MATLAB 环境下, 将路径设置至 process 文件夹下。对游戏区域的屏幕截图(灰度图像)graygroundtruth 进行分割, 提取出所有图像分块。在一个 figure 中用 subplot 方式按照原始页序绘出所有的图像分块

这里按照实验指导书的方案，对图像在行和列上求平均后再利用傅里叶变换得到边宽和图像宽度，但傅里叶变换后的高频分量很强，难以分割，所以进行了高通滤波，同时进行频率限制[1]。最终的结果不错，得到的块宽 79，块高 102，上边沿 24，左边沿 26. 行列的傅里叶变换分别见图1和图2, 切割线见3, 切割后的图像见4。代码如下（a4_2_1.m）

```
clear;clc;close all;
img=imread('graygroundtruth.jpg');
%利用高通滤波器进行处理
high_pass_1=fir1(22,0.5,'high');
[imp,n]=impz(high_pass_1);
stem(n,imp);
len=length(imp);
[~,max_i]=max(imp);
tmp_r=repmat(1:len,len,1)-max_i;
tmp_c=repmat((1:len)',1,len)-max_i;
tmp=round(sqrt(tmp_r.^2+tmp_c.^2));
t_i=find(tmp>len/2);
tmp(t_i)=max_i-1;
high_pass_2=imp(max_i-tmp);
figure;
img_div=conv2(img-128,high_pass_2,'same');
%行处理
[m,n]=size(img);
aver_row=mean(img_div,1);
aver_row=aver_row-mean(aver_row);
subplot(2,1,1)
plot((1:n),aver_row);
hold;
L=length(aver_row);
NFFT=2^nextpow2(L);
F_row=fft(aver_row,NFFT)/L;
nf=NFFT/2+1;
f= 1/2*linspace(0,1,nf);
```

[1]这里利用先验知识，大致给出了宽度的下限

```
reso=1/2/nf;
[value,index]=max(abs(F_row(f<0.02)));
an=angle(F_row(index));
left=(pi-an)/2/pi/f(index)
width=1/f(index)
plot((1:n),255*cos(2*pi*f(index)*(1:n)+an),'r');
subplot(2,1,2);
plot(f(f<0.02),abs(F_row(f<0.02)));
title('Fourier for average of rows')
%列处理
figure
aver_column=mean(img_div,2);
aver_column=aver_column-mean(aver_column);
subplot(2,1,1)
plot((1:m),aver_column);
hold;
L=length(aver_column);
NFFT=2^nextpow2(L);
F_column=fft(aver_column,NFFT)/L;
nf=NFFT/2+1;
f= 1/2*linspace(0,1,nf);
reso=1/2/nf;
[value,index]=max(abs(F_column(f<0.02)));
an=angle(F_column(index));
top=(pi-an)/2/pi/f(index)
height=1/f(index)
plot((1:m),255*cos(2*pi*f(index)*(1:m)+an),'r');
subplot(2,1,2);
plot(f(f<0.02),abs(F_column(f<0.02)));
title('Fourier for average of columns')

figure

width=round(width);
height=round(height);
top=round(top);
left=round(left);
c_count=fix((n-left)/width);
r_count=fix((m-top)/height);
```

```
imshow(img);
hold on;
%用红线分割
for i=0:r_count
    plot(left:left+c_count*width,top+height*i,'r');
end
hold on;
for i=0:c_count
    plot(left+width*i,top:top+height*r_count,'r');
end
%切割
figure
pic=zeros(height,width,r_count*c_count);
for i=1:r_count
    for j=1:c_count
        subplot(r_count,c_count,j+i*c_count-c_count);
        pic(:,:,j+(i-1)*c_count)=img((top+height*(i-1)):top+height*i
            -1,left+width*(j-1):left+width*j-1);
        imshow(uint8(pic(:,:,j+(i-1)*c_count)));
    end
end
save graygroundtruth
```

(2) 对摄像头采集的图像 (灰度图像)graycapture, 参考第 1 题要求进行处理。讨论: 和干净图像相比, 被噪声污染的图像给分块带来了什么样的困难?

这里的处理代码和上题基本一致，模糊等问题利用高通滤波可以解决，但是 graycapture 这张图有明显的透视畸变，这里参照了网上的代码[2]形成了 perspective 这一函数进行修正。矫正后的图像如图5, 可以看出效果不错。

最终的分割结果不错，主观上甚至好于上一题，得到的块宽 54，块高 68，上边沿 17，左边沿 23. 行列的傅里叶变换分别见图6和图7, 切割线见8, 切割后的图像见9。代码如下（a4_2_2.m）

```
clear;clc;close all;
img=imread('graycapture.jpg');

img=perspective(img);

high_pass_1=fir1(22,0.5,'high');
[imp,n]=impz(high_pass_1);
```

[2]手工标出图像的四个顶点http://www.cnblogs.com/tiandsp/archive/2012/12/16/2820916.html

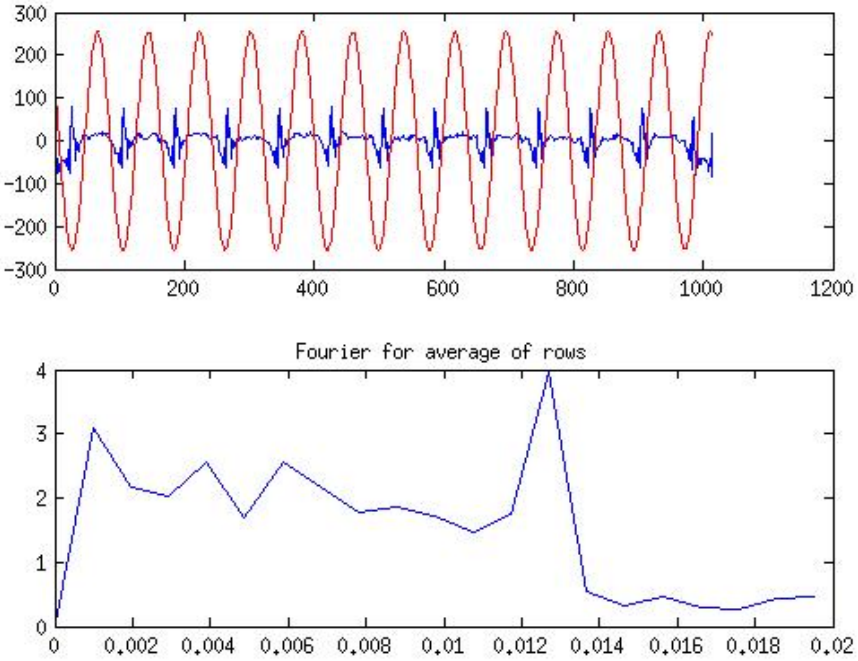

图 1: groundtruth 行平均值和其傅里叶变换

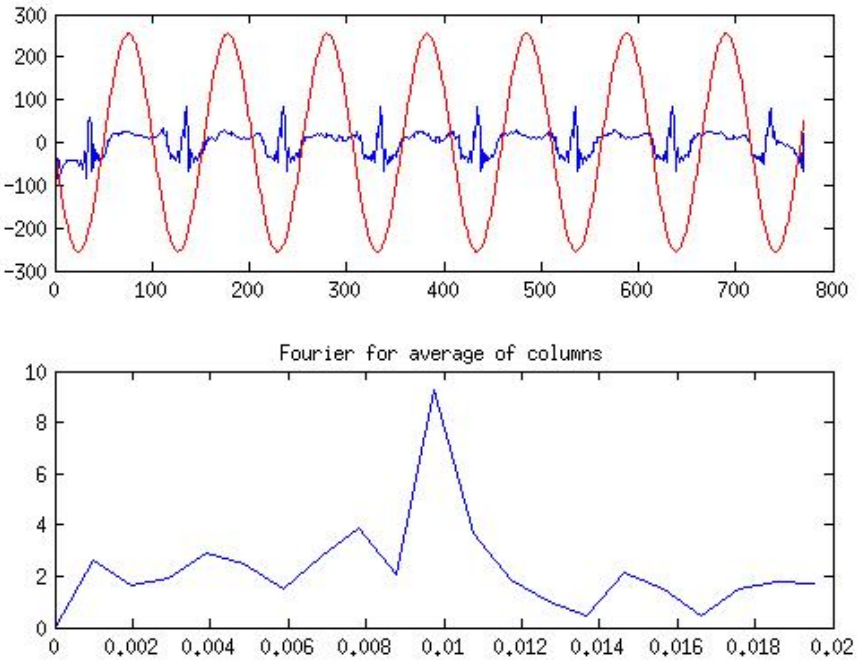

图 2: groundtruth 列平均值和其傅里叶变换

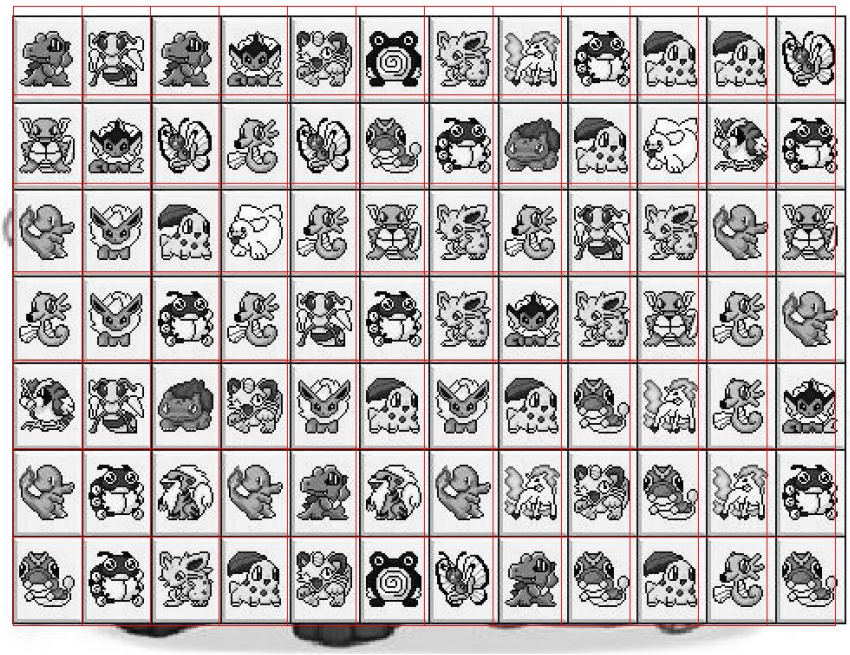

图 3: groundtruth 切割线

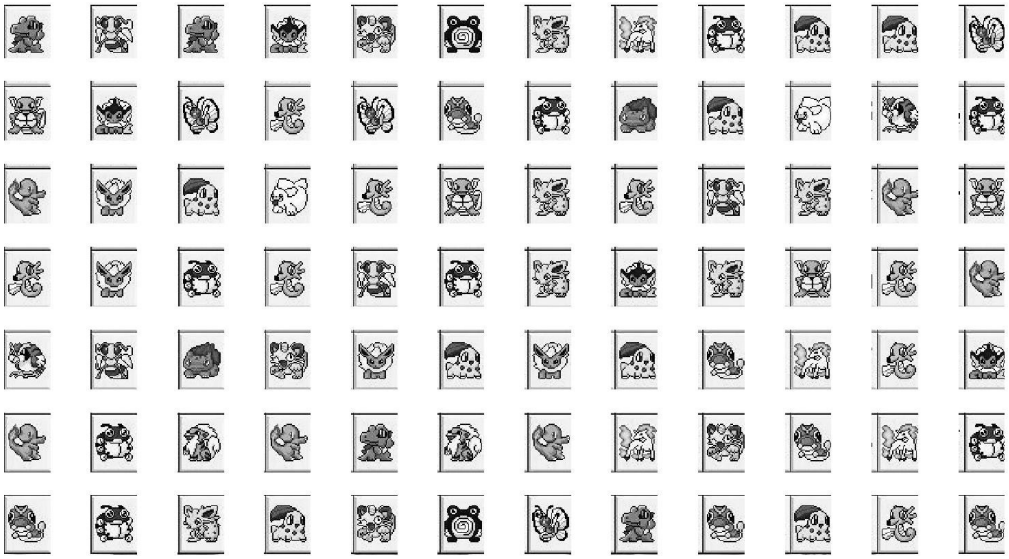

图 4: groundtruth 切割分块

```
len=length(imp);
[~,max_i]=max(imp);
tmp_r=repmat(1:len,len,1)-max_i;
tmp_c=repmat((1:len)',1,len)-max_i;
tmp=round(sqrt(tmp_r.^2+tmp_c.^2));
t_i=find(tmp>len/2);
tmp(t_i)=max_i-1;
high_pass_2=imp(max_i-tmp);
figure;


img_div=conv2(img-128,high_pass_2,'same');
[m,n]=size(img);
aver_row=mean(img_div,1);
aver_row=aver_row-mean(aver_row);
subplot(2,1,1)
plot((1:n),aver_row);
hold;
L=length(aver_row);
NFFT=2^nextpow2(L);
F_row=fft(aver_row,NFFT)/L;
nf=NFFT/2+1;
f= 1/2*linspace(0,1,nf);
reso=1/2/nf;
[value,index]=max(abs(F_row(f<0.02)));
an=angle(F_row(index));
left=(pi-an)/2/pi/f(index)
width=1/f(index)
plot((1:n),255*cos(2*pi*f(index)*(1:n)+an),'r');
subplot(2,1,2);
plot(f(f<0.02),abs(F_row(f<0.02)));
title('Fourier for average of rows')

figure
aver_column=mean(img_div,2);
aver_column=aver_column-mean(aver_column);
subplot(2,1,1)
plot((1:m),aver_column);
hold;
```

```
47 L=length(aver_column);
48 NFFT=2^nextpow2(L);
49 F_column=fft(aver_column,NFFT)/L;
50 nf=NFFT/2+1;
51 f= 1/2*linspace(0,1,nf);
52 reso=1/2/nf;
53 [value,index]=max(abs(F_column(f<0.02)));
54 an=angle(F_column(index));
55 top=(pi-an)/2/pi/f(index)
56 height=1/f(index)
57 plot((1:m),255*cos(2*pi*f(index)*(1:m)+an),'r');
58 subplot(2,1,2);
59 plot(f(f<0.02),abs(F_column(f<0.02)));
60 title('Fourier for average of columns')
61
62
63 figure
64 width=round(width);
65 height=round(height);
66 left=round(left);
67 top=round(top);
68 c_count=fix((n-left)/width);
69 r_count=fix((m-top)/height);
70 imshow(uint8(img));
71 hold on;
72 for i=0:r_count
73     plot(left:left+c_count*width,top+height*i,'r');
74 end
75 hold on;
76 for i=0:c_count
77     plot(left+width*i,top:top+height*r_count,'r');
78 end
79
80 figure
81 pic=zeros(height,width,r_count*c_count);
82 for i=1:r_count
83     for j=1:c_count
84         subplot(r_count,c_count,j+i*c_count-c_count);
85         pic(:,:,j+(i-1)*c_count)=img((top+height*(i-1)):top+height*i
```

```
            -1,left+width*(j-1):left+width*j-1);
        imshow(uint8(pic(:,:,j+(i-1)*c_count)));
    end
end
save graycapture
```

perspective.m:

```
function [imgn]=perspective(img)
figure;
imshow(img);
[M N] = size(img);

dot=ginput();          %取四个点，依次是左上，右上，左下，右下，
    这里我取的是书的四个角
w=round(sqrt((dot(1,1)-dot(2,1))^2+(dot(1,2)-dot(2,2))^2));      %
    从原四边形获得新矩形宽
h=round(sqrt((dot(1,1)-dot(3,1))^2+(dot(1,2)-dot(3,2))^2));      %
    从原四边形获得新矩形高

y=[dot(1,1) dot(2,1) dot(3,1) dot(4,1)];          %四个原顶点
x=[dot(1,2) dot(2,2) dot(3,2) dot(4,2)];

%这里是新的顶点，我取的矩形，也可以做成其他的形状
%大可以原图像是矩形，新图像是从dot中取得的点组成的任意四边形.:)
Y=[dot(1,1) dot(1,1) dot(1,1)+h dot(1,1)+h];
X=[dot(1,2) dot(1,2)+w dot(1,2) dot(1,2)+w];

B=[X(1) Y(1) X(2) Y(2) X(3) Y(3) X(4) Y(4)]';    %变换后的四个顶点，
    方程右边的值
%联立解方程组，方程的系数
A=[x(1) y(1) 1 0 0 0 -X(1)*x(1) -X(1)*y(1);
   0 0 0 x(1) y(1) 1 -Y(1)*x(1) -Y(1)*y(1);
    x(2) y(2) 1 0 0 0 -X(2)*x(2) -X(2)*y(2);
   0 0 0 x(2) y(2) 1 -Y(2)*x(2) -Y(2)*y(2);
    x(3) y(3) 1 0 0 0 -X(3)*x(3) -X(3)*y(3);
   0 0 0 x(3) y(3) 1 -Y(3)*x(3) -Y(3)*y(3);
    x(4) y(4) 1 0 0 0 -X(4)*x(4) -X(4)*y(4);
   0 0 0 x(4) y(4) 1 -Y(4)*x(4) -Y(4)*y(4)];

```

```
fa=inv(A)*B;          %用四点求得的方程的解，也是全局变换系数
a=fa(1);b=fa(2);c=fa(3);
d=fa(4);e=fa(5);f=fa(6);
g=fa(7);h=fa(8);

rot=[d  e  f;
     a  b  c;
     g  h  1];        %公式中第一个数是x,Matlab第一个表示y，所以我矩阵1,2行互换了

pix1=rot*[1  1  1]'/(g*1+h*1+1);  %变换后图像左上点
pix2=rot*[1  N  1]'/(g*1+h*N+1);  %变换后图像右上点
pix3=rot*[M  1  1]'/(g*M+h*1+1);  %变换后图像左下点
pix4=rot*[M  N  1]'/(g*M+h*N+1);  %变换后图像右下点

height=round(max([pix1(1) pix2(1) pix3(1) pix4(1)])-min([pix1(1) pix2(1) pix3(1) pix4(1)]));      %变换后图像的高度
width=round(max([pix1(2) pix2(2) pix3(2) pix4(2)])-min([pix1(2) pix2(2) pix3(2) pix4(2)]));       %变换后图像的宽度
imgn=ones(height,width)*mean2(img(1:10,1:10));

delta_y=round(abs(min([pix1(1) pix2(1) pix3(1) pix4(1)])));
                %取得y方向的负轴超出的偏移量
delta_x=round(abs(min([pix1(2) pix2(2) pix3(2) pix4(2)])));
                %取得x方向的负轴超出的偏移量
inv_rot=inv(rot);

for  i = 1-delta_y:height-delta_y                                %
    从变换图像中反向寻找原图像的点，以免出现空洞，和旋转放大原理一样
     for  j = 1-delta_x:width-delta_x
          pix=inv_rot*[i  j  1]';        %求原图像中坐标，因为[YW XW W]=fa*[y  x  1],所以这里求的是[YW XW W],W=gy+hx+1;
          pix=inv([g*pix(1)-1 h*pix(1);g*pix(2) h*pix(2)-1])*[-pix(1) -pix(2)]'; %相当于解[pix(1)*(gy+hx+1) pix(2)*(gy+hx+1)]=[y x],这样一个方程，求y和x，最后pix=[y  x];

          if  pix(1)>=0.5 && pix(2)>=0.5 && pix(1)<=M && pix(2)<=N
               imgn(i+delta_y,j+delta_x)=img(round(pix(1)),round(pix(2)));      %最邻近插值,也可以用双线性或双立方插值
```

图 5: 透视矫正后的 capture 图像

```
58            end
59        end
60    end
61    imshow(uint8(imgn));
```

(3) 在第 2 题基础上, 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 中绘出, 并显示其相似性度量值。(建议先利用 graygroundtruth 测试代码的正确性, 然后再用 graycapture 完成本题。)

首先利用 fir1 函数产生高通滤波器如图10, 将其冲激响应旋转一周后得到图11, 将其与各分块后的图像卷积即可得到其高通滤波后的值。

随后两两计算各个图片的内积，这里经过查询利用 normxcorr2 函数得到归一化的比较值。因为两个图片交换位置求相关的值有一定的差异，为了下一题好算这里取更大的那个。

这里的计算量比较大，我的笔记本大概需要运算 15s 左右，感觉也很难进一步压缩时间。得到图12 代码如下 a4_2_3.m

```
1  clear;clc;close all;
2
3  load 'graycapture.mat';
4
```

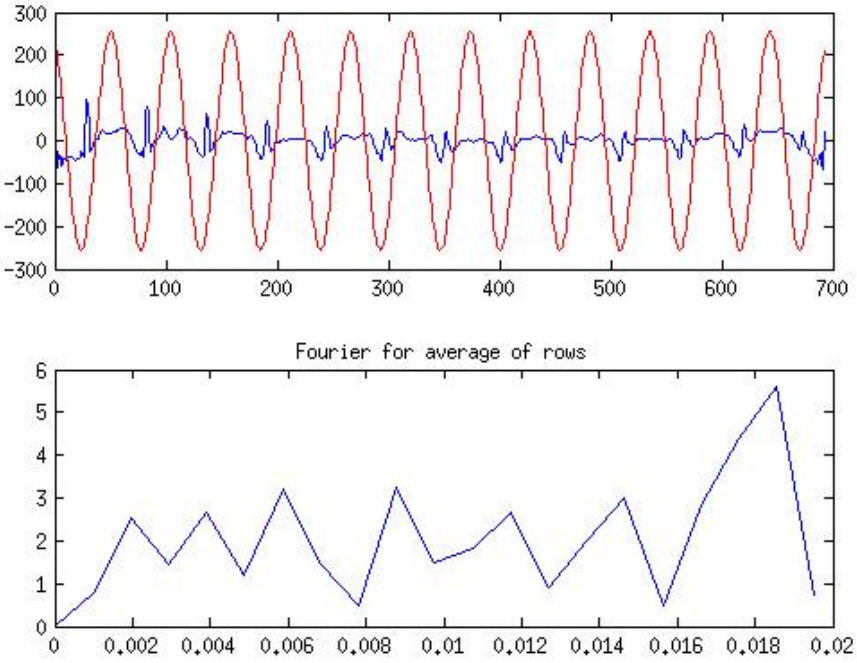

图 6: capture 行平均值和其傅里叶变换

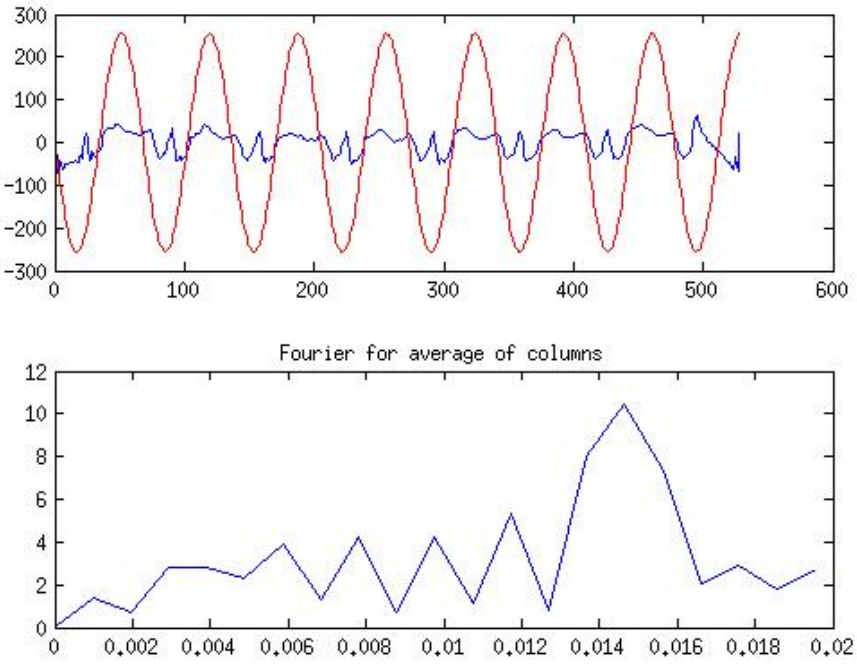

图 7: capture 列平均值和其傅里叶变换

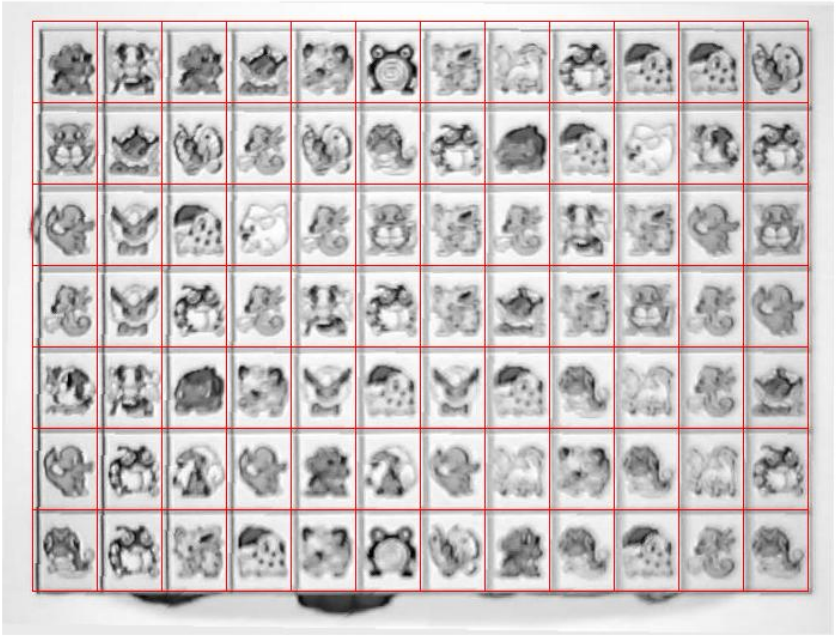

图 8: capture 切割线

图 9: capture 切割分块

```
high_pass_1=fir1(22,0.5,'high');
[imp,n]=impz(high_pass_1);
stem(n,imp);
len=length(imp);
[~,max_i]=max(imp);
tmp_r=repmat(1:len,len,1)-max_i;
tmp_c=repmat((1:len)',1,len)-max_i;
tmp=round(sqrt(tmp_r.^2+tmp_c.^2));
t_i=find(tmp>len/2);
tmp(t_i)=max_i-1;
high_pass_2=imp(max_i-tmp);
figure;
mesh(high_pass_2);
figure;
pic_div=zeros(height,width,r_count*c_count);
for i=1:r_count*c_count
    pic_div(:,:,i)=conv2(pic(:,:,i)-128,high_pass_2,'same');
 %   subplot(r_count,c_count,i);
 %   imshow(uint8(128+pic_div(:,:,i)));
end

corr=zeros(r_count*c_count,c_count*r_count);
siz=r_count*c_count;
for i=1:siz
    for j=1:siz
        if i~=j
        corr(j,i)=max(max(normxcorr2(pic_div(:,:,i),pic_div(:,:,j))))
            ;
        end
    end
end
corr=max(corr,corr');
siz=r_count*c_count;
corr_tmp=reshape(corr,[],1);
[value,index]=sort(corr_tmp,'descend');
for i=1:2:20%因为每个值存了两次
    a=fix((index(i)-1)/siz)+1;
    b=mod(index(i)-1,siz)+1;
    subplot(2,10,(i+1)/2);
```

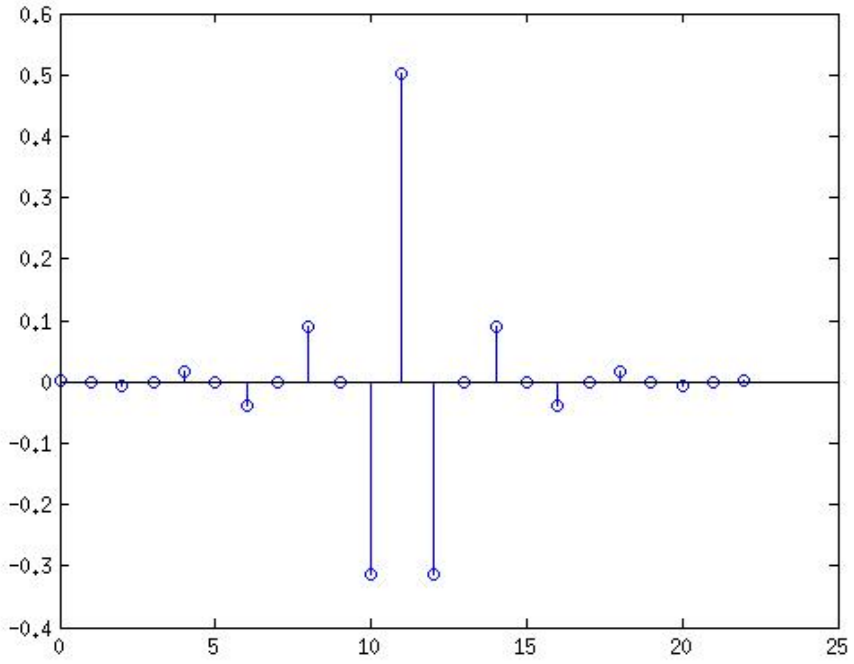

图 10: 高通滤波器的冲激响应

```
43      imshow(uint8(pic(:,:,a)));
44      title(value(i))
45      subplot(2,10,(i+1)/2+10);
46      imshow(uint8(pic(:,:,b)));
47  end
48
49  save corr_capture.mat
```

(4) 第 3 题基础上, 找出相似度最大却不是同一种精灵的十对图像块。在一个 figure 中绘出, 并显示其相似性度量值。讨论: 这个结果和你的主观感受一致吗?

挑选出来的十对图像如图13, 可以看出，虽然人能比较容易的区分它们，但它们都是颜色比较深的图像，相似度值还是比较高的。

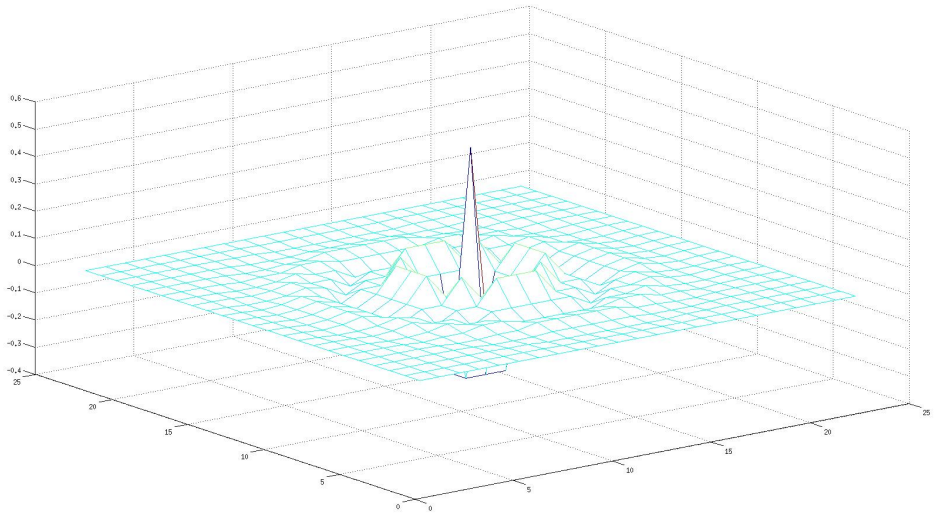

图 11: 旋转后的高通滤波器

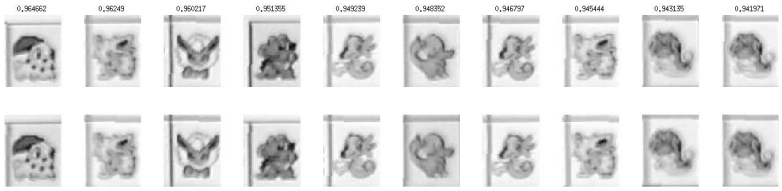

图 12: 最相近的十对

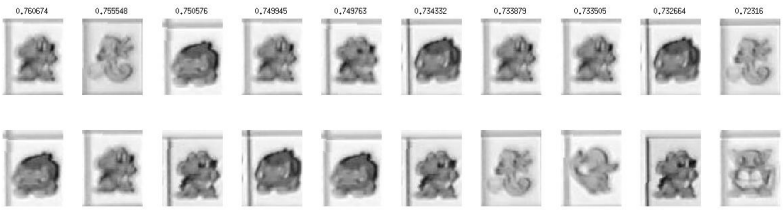

图 13: 相似度最大却不同的十对

图 14: 每种编号对应的图像

(5) 在第 3 题基础上, 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表。讨论: 你可以将全部图像分块正确映射到其索引值吗? 哪些分块无法正确映射? 为什么?

不得不提的是，海狮，也就是图14中编号 15 的块的两块之间相似性度量值只有 0.6 左右[3]，从上一题的结果即可看出，若直接用设定阈值的方法，这两块是无法被识别的，我开始这样试的时候也是如此。

因此我对这一方法进行了增强。这里的做法是先寻找每一个块的最相似的块，然后两两分组（此时也会得到因为两两最相似而多个块被放在一个集合里的情况）。再处理这些集合，利用阈值选择一些关系（取大于 0.77，这一系数由上一问得到）把包含着一样图像的集合合并，各个集合去重后再依次给 mtx 的对应点赋值。

得到的矩阵如下, 其中各个数字对应的图像如图14所示。比对后是完全正确的。

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

代码如下（a4_2_5.m）

```
clear;clc;close all;
load 'corr_capture.mat';

```

[3]应该是颜色太淡的原因

```
[m,n]=size(corr);
mtx=zeros(r_count,c_count);
tmp=mtx';
[~,same]=max(corr);
where=zeros(12*7,1);
A=repmat({[]},12*7,1);
for i=1:n%最大值匹配
    j=min(i,same(i));
    k=max(i,same(i));
    if where(j)~=0 && where(k)~=0
        A{min(where(j),where(k))}=[A{min(where(j),where(k))} A{max(
            where(j),where(k))}];
        if where(j)~=where(i)
            A{max(where(j),where(k))}=[];
        end
        where(k)=where(j);
    else if where(j)~=0
            A{where(j)}=[A{where(j)} i same(i)];
            where(i)=where(j);
            where(same(i))=where(j);
        else if where(k)~=0
                A{where(k)}=[A{where(k)} i same(i)];
                where(i)=where(k);
                where(same(i))=where(k);
            else
                A{j}=[A{j} i same(i)];
                where(i)=j;
                where(same(i))=j;
            end
        end
    end

end
[m,n]=find(corr>0.77);
for i=1:length(m)%阈值匹配
    mi=min(m(i),n(i));
    ma=max(m(i),n(i));
    if where(m(i))~=where(n(i))
        A{min(where(mi),where(ma))}=[A{where(mi)} A{where(ma)}];
```

```
42          A{max(where(mi),where(ma))}=[];
43          where(ma)=where(mi);
44      end
45
46  end
47  type=1;
48  for i=1:84%mtx赋值
49      B=unique(A{i});
50      if B
51          r_a=fix((B-1)/c_count)+1;
52          c_a=mod(B-1,c_count)+1;
53          tmp=mtx';
54          tmp(B)=type;
55          mtx=tmp';
56          type=type+1;
57      end
58  end
59  count=max(max(mtx));
60  for i=1:count%显示各种块的标号
61      t=reshape(mtx',1,[]);
62      in=find(t==i);
63      subplot(3,8,i);
64      imshow(uint8(pic(:,:,in(1))));
65      title(i);
66  end
67  save mtx.mat mtx pic img r_count c_count
```

(6) 在上述工作基础上, 设计实现一个模拟的自动连连看。对摄像头采集的图像 (灰度图像)graycapture 进行分块并找出最相似的一对可消除分块后, 将这图片上两个块的位置设为黑色或其他特定颜色 (即模拟消除操作), 并将图片展示在 figure 上。然后继续找出下一对可消除的分块并模拟消除, 直至消除所有的分块或找不到可消除的分块对。设计一种方法验证并展示上述工作的正确性。

我的做法是没延时 1.5s 给被消除的一对加上标记，最终结果如图15

这里通过观察消除过程可见程序是成功的，屏幕录像可以看根目录的 solve.mp4 文件。有趣的是，当我想调用之前的 omg.m 时却发现不能解出这一连连看。在调整了图像的扫描顺序后就得到了解。可见对于之前的算法，不同的扫描策略会导致有解无解的差异，但如何去覆盖所有的情况已经超出了讨论范围，这里也就不再修改。

代码如下 a4_2_6.m

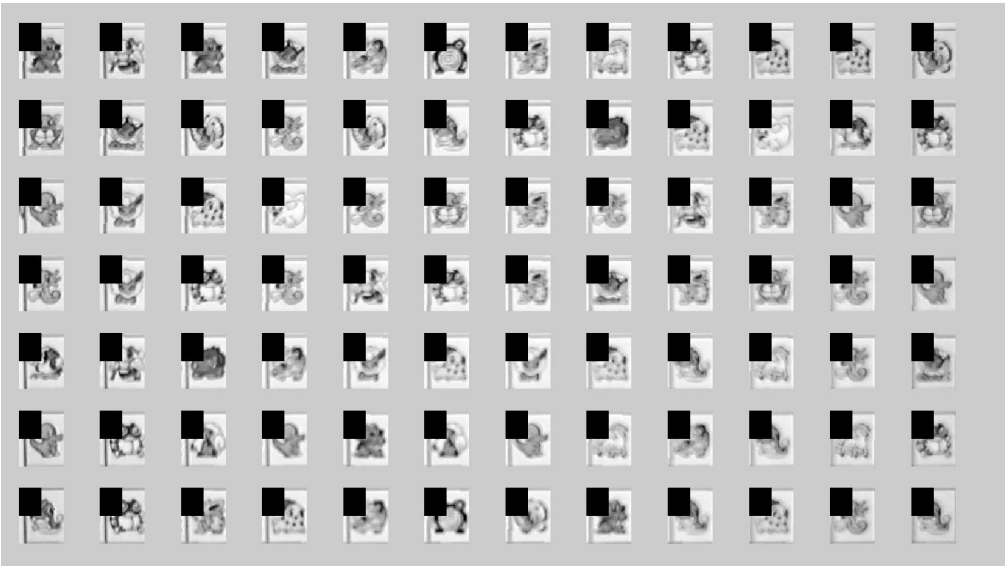

图 15: 模拟连连看最终结果

```
clear;close all; clc;
load  'mtx.mat';
figure
for i=1:r_count
    for j=1:c_count
        subplot(r_count,c_count,j+i*c_count-c_count);
        imshow(uint8(pic(:,:,j+(i-1)*c_count)));
        hold on;
    end
end
hold on;
s=zeros(size(pic(:,:,1))/2);
steps(1) = 0;
mtx_tmp=zeros(size(mtx));
siz=max(max(mtx));
pause(4);

while(siz~=0)
    for i=siz:-1:1
        [m,n]=find(mtx==i);
        for j=1:length(m)
            for k=j+1:length(n)
                if mtx(m(j),n(j))~=0&&mtx(m(k),n(k))~=0;
```

```
                    if detect(mtx,m(j),n(j),m(k),n(k))==1
                        steps=[steps m(j) n(j) m(k) n(k)];
                        subplot(r_count,c_count,m(j)*c_count-c_count+
                            n(j));
                        imshow(uint8(s));
                        subplot(r_count,c_count,m(k)*c_count-c_count+
                            n(k));
                        imshow(uint8(s));
                        pause(1.5);
                        mtx(m(j),n(j))=0;
                        mtx(m(k),n(k))=0;
                    end
                end
            end
        end
    end
    siz=max(max(mtx));
    if(mtx_tmp==mtx)
        break;
    end
    mtx_tmp=mtx;%避免死循环
end
```

3 实验总结

本试验非常有趣，选取了连连看这一常见的游戏结合图像处理。

相对来说，连连看的逻辑比较好完成，而图像处理比较复杂。实验中我虽然使用了畸变矫正、高通滤波等方法，但是识别的效果依然不理想。这时只能用更强的判断算法去解决。这应该也是信号处理中经常要去面对的问题——输入信号好则算法简单但是收集难度大，输入信号差则收集难度小但算法复杂，实际的系统设计也一定是在这两者之间寻找平衡。本次实验是我对这一过程有了更深刻的理解。

除此之外，在进行映射为数组的过程中，因为 matlab 的二维矩阵各行必须等长，所以我不得不使用了元胞数组（cell）这一数据结构，同时还知道了 fir1 和 normxcorr2 这两个系列函数的使用，收货很大。

虽然没有进一步做的选作部分，但是我设想，彩色图像可以在三个维度上进行综合比较（平均或最大值），实际的摄像头肯定在去畸变和去噪后也是一样的处理方法，但是很难做到实时读取。