به نام خدا

پروژه سوم درس سیگنال‌ها و سیستم‌ها

فاطمه‌زهرا برومندنیا-۸۱۰۱۰۰۰۹۴

۱.۱

```
pic=imread('a.jpg');
pic=rgb2gray(pic);
%pic=imresize(pic,0.3);
subplot(1,2,1);
%imshow(pic(1:10))

imshow(pic)
Alphabet='abcdefghijklmnopqrstuvwxyz .,!";';

numChar=length(Alphabet);
mapset=cell(2,numChar);
for i=1:numChar
    mapset{1,i}=Alphabet(i);
    mapset{2,i}=dec2bin(i-1,5);
end
msg='hello session signal;';

outPic=coding(msg,pic,mapset);
subplot(1,2,2);
%imshow(outPic(1:10));
imshow(outPic);

decodedMsg=decoding(outPic,mapset)
```

۲.۱

در تابع کدینگ ابتدا با دریافت پیام و مقایسه کاراکتر به کاراکتر آن با مپستی که در آن معادل باینری هر کاراکتر را ذخیره کرده‌ایم در ماتریس ایندکس ایندکس‌های متناظر از مپست با پیام خود را ذخیره می‌کنیم .نهایتاً پیام باینری شده را استخراج می‌کنیم پس از آن از پیکسل اول عکس شروع می‌کنیم، آنها را از دسیمال به باینری تبدیل می‌کنیم و بیت هشتم آن را به بیت متناظر از پیام باینری شده تغییر می‌دهیم سپس مقدار دسیمال آن را در عکس جایگزین می‌کنیم و به همین ترتیب کدگذاری را ادامه می‌دهیم تا رشته کد تمام شود.

```
function outPic=coding(msg,pic,mapset)
idx=[];
for i=1:length(msg)
    ch=msg(i);
    idx=[idx, find(strcmp(ch,mapset(1,:))==1)];
end
binMsg=cell2mat(mapset(2,idx));
outPic=pic;
for i=1:length(binMsg)
    num=dec2bin(pic(i),8);
    num(8)=binMsg(i);
    outPic(i)=bin2dec(num);
end
end
```

۳.۱

تغییرات در تصویر مشخص نیست زیرا کم ارزش‌ترین بیت هر پیکسل را تغییر داده‌ایم و از آن جایی که هر پیکسل عددی بین ۰ تا ۲۵۵ از سیاه تا سفید دارد، چشم قادر به تغییر مشاهده تغییر این بیت نیست.

۴.۱

با همان منطقی که کدگذاری کردیم از اولین پیکسل عکس شروع می‌کنیم. پس از باینری کردن، بیت هشتم هر کدام را استخراج می‌کنیم و آن بیت را رشته باینری خود اضافه می‌کنیم و هر بار چک می‌کنیم اگر به کاراکتر خاتمه دهنده رسیدیم این کار را خاتمه می‌دهیم. نهایتاً از پیام باینری ۵ بیت ۵ بیت برمی‌داریم و با مقایسه از مبست کاراکتر متناظر آن را پیدا می‌کنیم و پیام کدگذاری شده را به دست می‌آوریم.

```
function msg=decoding(pic,mapset)

binMsg='';
terminateChar=';';
terminateCharBin=dec2bin(find(strcmp(terminateChar,mapset(1,:))==1)-1);

for i=1:numel(pic)
    num=dec2bin(pic(i),8);
    binMsg=strcat(binMsg,num(8));
    if(rem(i,5)==0)
        if(binMsg(end-4:end)==terminateCharBin)
            break;
        end
    end
end

lenMsg=length(binMsg)/5;
idx=[];
for i=1:lenMsg
    idx=[idx, find(strcmp(binMsg(5*i-4:5*i),mapset(2,:))==1) ];
end
msg=cell2mat(mapset(1,idx));
end
```

۵.۱

خیر قادر به تشخیص، پس از افتادن نویز نخواهیم بود، اگر نویز بیت‌هایی که در آن کدگذاری کرده‌ایم را دچار تغییر و تحول کرده باشد. می‌توان به جهت اطمینان بیت هفتم پیکسل‌های تصویر gray scale را نیز کدگذاری کرد و در نهایت چک کرد که اگر رشته ساخته شده از کم ارزش‌ترین بیت با بیت هفتم برابری می‌کند مطمئن شویم پیام با نویزها دست نخورده است ولی اگر تفاوت داشتند یعنی نویزها پیام را جابجا کرده‌اند.

۶.۱

با استفاده از هیستوگرام عکس که توزیع گوسی و نرمال در حالت غیر کدگذاری شده دارد می‌توان تشخیص داد که آیا تصویر دست خورده است یا نه که البته کار ساده‌ای نیست ولی قطعاً تصویر را از حالت توزیع نرمال و گاوسی خود خارج می‌کند.

۱.۲

از اینپوت کاراکتر به کاراکتر جلو می‌رویم، برای هر کاراکتر فرکانس ردیف و ستون متناظر آن در صفحه کلید را پیدا می‌کنیم و در کا و جی می‌ریزیم. سپس سیگنال dtmfمتناظر ردیف و ستون آن را درست می‌کنیم و در ماتریس وای که برای ذخیره سازی صدای نهایی است، سیگنال ساخته شده را همراه با مدت زمانی سکوت append می‌کنیم.

```
fr = [697 770 852 941];
fc= [1209 1336 1477];
fs=8000;
Ts=1/fs;
Ton=0.1;
Toff=0.1;
t=0:Ts:Ton;
toff=0:Ts:Toff;
keys=['1','2','3','4','5','6','7','8','9','*','0','#'];
Y=[];
input='43218765';
for i=1:1:numel(input)
    idx=find(input(i)==keys);
    k=ceil(idx/3);
    j=rem(idx,3);
    if(j==0)
        j=3;
    end
    y1=sin(2*pi*fr(k)*t);
    y2=sin(2*pi*fc(j)*t);
    y_=(y1+y2)/2;
    Y=[Y y_ toff];
end
sound(Y, fs)
```

۲.۲

در حلقه فور با استپ‌هایی به اندازه بخش سکوت و بخش سیگنال جلو می‌رویم تا بخش‌های صوتی را از سیگنال جدا کنیم و روی هر سیگنال دی تی ام اف تحلیل انجام دهیم.

کورولیشن بخش صوتی با با هر کدام از فرکانس‌های داده شده را حساب می‌کنیم و به ترتیب با تابع xcorrr متلب کورلیشن می‌گیریم و نهایتاً مقدار ماکسیمم آن کرولیشن گیری را در تناظر با آن فرکانس در ماتریس های row_correlations و col_correlations ذخیره می‌کنیم. برای آنکه کلید متناظر آن را پیدا کنیم باید یک فرکانس از ردیف‌ها و یک فرکانس از ستون‌ها انتخاب شود؛ در نتیجه هم از ستون هم از ردیف آنی که ماکسیمم کرولیشنش از باقی بیشتر شده است را و از آستانه مشخص‌شده مقدارش بیشتر است را به عنوان کلید تشخیص داده شده در نظر می‌گیریم.

```
clc, clear, close all
[y, Fs] = audioread('a.wav');
figure
plot(1:length(y),y)


Ton=0.1;
Toff=0.1;
dtmf = round(Ton * Fs);
silence=round(Toff*Fs);

threshold = 0.2;
dtmf_keys = ['1', '2', '3'; '4', '5', '6'; '7', '8', '9'; '*', '0', '#'];

fr = [697, 770, 852, 941];
% row_tone=zeros(1,length(fr));
% for k=1:length(fr)
%     row_tone(k) = (sin(2 * pi * fr(k) * (0:dtmf) / Fs))';
% end
fc = [1209, 1336, 1477];
% col_tone=zeros(1,length(fc));
```

```
detected_keys = '';
for i = 1:silence+dtmf:(length(y) - silence)
    audio_part = y(i:i+dtmf);

    for j = 1:length(fc)
        col_tone = sin(2*pi*fc(j)*(0:dtmf)/Fs);
        col_correlations(j) = max(xcorr(audio_part, col_tone));
    end

    for j = 1:length(fr)
        row_tone = sin(2*pi*fr(j)*(0:dtmf)/Fs);
        row_correlations(j) = max(xcorr(audio_part, row_tone));
    end

    [u, row_idx] = max(row_correlations);
    [u, col_idx] = max(col_correlations);

    if row_correlations(row_idx) > threshold && col_correlations(col_idx) > threshold
        detected_keys = [detected_keys, dtmf_keys(row_idx, col_idx)];
    end
end
```

.۳

ابتدا عکس تمپلیت و عکس اصلی را در مطلب باز می‌کنیم، هر دو را گری اسکیل می‌کنیم بعد با کمک تابع normxcorr2 کرولیشن نرمالایز شده تمپلت و عکس اصلی را حساب می‌کنیم و در ماتریس corr_coef می‌ریزیم. حال برای پیدا کردن عکس مشابه، آستانه ۰.۷را برای کرولیشن مشخص می‌کنیم؛ سپس مقادیر کرولیشن که از آستانه بیشتر باشند ایندکسشان را در مارتیس‌های وای و ایکس می‌ریزیم. همین کار را برای تمپلیت

روتیت داده شده نیز تکرار می‌کنیم سپس با توجه به وای و ایکس که هرکدام ایندکس گوشه سمت راست پایین آن بخش در عکس را نشان می‌دهند، برای تمام تمپلت‌های پیدا شده مستطیل می‌کشیم.

تابع کورلیشن گیری با توجه به تعمیم‌داده‌شده فرمول داده شده هم در زیر آمده است.

```
clc, close all
IC_FILE = 'IC.png';
PICTURE_FILE = 'PCB.jpg';
template = imread(IC_FILE);
image = imread(PICTURE_FILE);

subplot(2,2,1);
imshow(template);
title('IC');
subplot(2,2,3);
imshow(image);
title('Board');

template_gray = rgb2gray(template);
image_gray = rgb2gray(image);
corr_coef = normxcorr2(template_gray, image_gray);
% normalized_corr =  normalizedCorr(template_gray, image_gray);
threshold = 0.7;
[y, x] = find(corr_coef > threshold);
template_rotated = imrotate(template_gray, 180);
```

```
[y, x] = find(corr_coef > threshold);
template_rotated = imrotate(template_gray, 180);
corr_coef_rotated = normxcorr2(template_rotated, image_gray);
[y_rotated, x_rotated] = find(corr_coef_rotated > threshold);

subplot(2,2,2);
imshow(image);
title('Detected IC');
hold on;

for i = 1:length(x)
    rectangle('Position', [x(i)-size(template_gray, 2)+1, y(i)-size(template_gray, 1)+1, size(template_gray, 2), size(template_g
end

for i = 1:length(x_rotated)
    rectangle('Position', [x_rotated(i)-size(template_gray, 2)+1, y_rotated(i)-size(template_gray, 1)+1, size(template_gray, 2),
end
```

```
function normalized_corr = normalizedCorr(image, template)
    mean_template = mean(template(:));
    mean_image = mean(image(:));

    numerator = sum(sum((image - mean_image) .* (template - mean_template)));
    denominator_image = sqrt(sum(sum((image - mean_image).^2)));
    denominator_template = sqrt(sum(sum((template - mean_template).^2)));

    normalized_corr = numerator / (denominator_image * denominator_template);
end
```

Algorithms

`normxcorr2` uses the following general procedure [1], [2]:

1. Calculate cross-correlation in the spatial or the frequency domain, depending on size of images.
2. Calculate local sums by precomputing running sums.
3. Use local sums to normalize the cross-correlation to get correlation coefficients.

The implementation closely follows the formula from [1]:

$$\gamma(u,v) = \frac{\sum_{x,y}\left[f(x,y) - \overline{f}_{u,v}\right]\left[t(x-u,y-v) - \overline{t}\right]}{\left\{\sum_{x,y}\left[f(x,y) - \overline{f}_{u,v}\right]^2 \sum_{x,y}\left[t(x-u,y-v) - \overline{t}\right]^2\right\}^{0.5}}$$

where

- f is the image.
- $\overline{t}$ is the mean of the template
- $\overline{f}_{u,v}$ is the mean of $f(x,y)$ in the region under the template.