پروژه کامپیوتری دوم

آرین باستانی 810100088

قسمت اول:

تمرین 1-1) اول کاراکتر ها را به ترتیب در یک متغیر ذخیره میکنیم؛ سپس تصویر را از کاربر ورودی میگیریم.

حالا متغیر مپست را تعریف کرده و کاراکتر ها و اعداد 0 تا 31 را به ترتیب در آن ذخیره میکنیم.

```
chars = ['a' 'b' 'c' 'd' 'e' 'f' 'g' 'h' 'i'...
    'j' 'k' 'l' 'm' 'n' 'o' 'p' 'q' 'r' 's' 't' 'u' ...
    'v' 'w' 'x' 'y' 'z' ' ' '.' ',' '!' '"' ';'];

[file, path] = uigetfile(...
    {'*.jpeg;*.jpg;*.bmp;*.png;*.tif'}...
    , 'Please choose image:');
my_image = imread([path, file]);
my_image = rgb2gray(my_image);

Mapset = cell(2,32);
char_id = 0:1:31;
binaryNums = dec2bin(char_id, 5);
for map_index=1:32
    Mapset{1,map_index} = chars(map_index);
    Mapset{2,map_index} = binaryNums(map_index, :);
end
```

تولید مپ ست

تمرین 1_2: ابتدا ارور احتمالی را هندل میکنیم. یعنی سایز مورد نیاز و سایز تصویر را محاسبه کرده و اگر سایز مورد نیاز از سایز عکس بزرگتر بود، یک ارور پرینت کرده و خروجی را خالی برمیگردانیم.

حال برای باینری کردن مسیج، با استفاده از تابع code_the_massg که خودم نوشتم، پیام را باینری

میکنم. کار این تابع بدین صورت است که با دو حلقه ی تودرتو در مپست و کاراکتر های مسیج، عدد باینری متناظر با کاراکتر مسیج را به متغیر خروجی اضافه میکند.

حال با استفاده از پیام باینری شده و تابع code_the_image خروجی ساخته میشود.

در این تابع طبق صحبت های سرکلاس، از بالاسمت چپ تصویر شروع میکنیم و به صورت ستونی حرکت کرده و پس از تبدیل پیکسل به عدد باینری 8 بیتی، کم ارزش ترین بیت آن را برابر بیت متناظر در مسیج باینری شده قرار میدهیم.

```
function coded_image = coding(message, image, Mapset)

    char_id_len = size(Mapset{2,1}, 2);
    if((char_id_len * size(message, 2)) >...    //error handling
            (size(image,1) * size(image, 2)))

        coded_image = NaN;
        fprintf('Invalid length of massage!\n')
        return;
    end

    coded_massg = code_the_massg(message, Mapset);

    coded_image = code_the_image(coded_massg, image);
end
```

تولید تصویر کد شده

```
function coded_massg = code_the_massg(message, Mapset)
    coded_massg = [];
    for massg_index = 1 : size(message, 2)
        for Mapset_index = 1 : size(Mapset, 2)
            if(message(1, massg_index) == Mapset{1, Mapset_index})
                coded_massg = [coded_massg Mapset{2, Mapset_index}];
            end
        end
    end
    coded_massg = [coded_massg Mapset{2, size(Mapset, 2)}];
end

function coded_image = code_the_image(coded_massg, image)
    for massg_index = 1 : size(coded_massg, 2)
        [row, col] = ind2sub(size(image), massg_index);
        current_pixel = image(row, col);
        binPix = dec2bin(current_pixel, 8);
        binPix(8) = coded_massg(1, massg_index);
        image(row, col) = bin2dec(binPix);
    end
    coded_image = image;
end
```

توابع استفاده شده

تمرین 1_3: خیر. از آنجایی که برای هرپیکسل کم ارزش ترین بیت را تغییر دادیم، تغییرات با چشم قابل مشاهده نیستند!

```
subplot(1,2,1);
imshow(my_image);
title('Image before coding')

my_coded_image = coding('signal', my_image, Mapset);

subplot(1,2,2);
imshow(my_coded_image);
title('Image after coding')
```

رسم تصویر، قبل و بعد از کد کردن

تصویر، قبل و بعد از کد کردن

<u>تمرین 1_4)</u> ابتدا با استفاده از تصویر داده شده، اعداد مسیج را درمیاوریم. که برای اینکار در یک تابع، اعداد داخل پیکسل هارا میخوانیم و بعد از هر 5 بار خواندن(یعنی بعد از خواندن یک کاراکتر) چک میکنیم اگر سمی کالن بود، یعنی مسیج تمام شده و خروجی را برابر اعداد کاراکتر های قبلی قرار میدهیم.

سپس این اعداد داخل مسیج را با استفاده از مپست به کاراکتر تبدیل کرده و پرینت میکنیم.

```
function decoding(image, Mapset)
    message_ids = binerize_massg(image, Mapset);

    message = [];

    for i = 1 : 5 : size(message_ids,2)
        for j=1:size(Mapset,2)
            if(message_ids(i:i+4)==Mapset{2,j})
                message = [message Mapset{1,j}];
            end
        end
    end

    fprintf('%s', message);
end
```

دیکود تصویر

```
function bin_massg = binerize_massg(image, Mapset)
    flag = true;
    col = 1;
    message_ids = [];

    while(flag && (col <= size(image, 2)))
        for row = 1 : size(image, 1)
            current_pixel = image(row, col);
            pix_id = dec2bin(current_pixel, 8);
            message_ids = [message_ids, pix_id(8)];
            if(mod(size(message_ids,2), 5) == 0)
                sz = size(message_ids,2);
                if(message_ids(1,sz-4:sz) == Mapset{2, size(Mapset, 2)})
                    flag = false;
                    break;
                end
            end
        end
        col = col + 1;
    end

    bin_massg = message_ids(1 : size(message_ids, 2) - 5);
end
```

تابع استفاده شده

قسمت دوم:

الف)

به ازای هر یک از اعداد گفته شده، تابع make_audio را صدا میکنیم و سپس آرایه ی تولید شده را به صورت فایل صوتی ویو ذخیره میکنیم.

در تابع گفته شده، ابتدا تابع make_on را صدا میکنیم و سپس ادامه ی جواب را بصورت فاصله ی بین صداها اضافه میکنیم(سکوت).

تابع Make_on :

در این تابع، مانند کد داده شده در صورت سوال عمل میکنیم؛ با این تفاوت که به ازای تمام اعداد این کار را چک کرده و انجام میدهیم و در انتها سکوت را اضافه نمیکنیم.

```
audio = [];
numbers = '43218765';

for number_id = 1 : size(numbers, 2)
    audio = make_audio(numbers(number_id), audio);
end

audiowrite('y.wav', audio, 8000);
```

کد اصلی

```
function y = make_audio(num, previous_y)
    y = make_on(num, previous_y);

    fs = 8000;
    ts = 1/fs;
    t_len = 0.1;
    t = ts : ts : t_len;

    off_y = zeros(size(t));
    y = [y off_y];
end
```

تابع اول

```
function y = make_on(num, pre)
    fs = 8000;
    ts = 1/fs;
    t_len = 0.1;
    t = ts : ts : t_len;

    col = [1209 1336 1477];
    row = [697 770 852 941];

    if(num == '0')
        y1 = sin(2 * pi * row(4) * t);
        y2 = sin(2 * pi * col(2) * t);
    elseif(num == '1')
        y1 = sin(2 * pi * row(1) * t);
        y2 = sin(2 * pi * col(1) * t);
    elseif(num == '2')
        y1 = sin(2 * pi * row(1) * t);
        y2 = sin(2 * pi * col(2) * t);
    elseif(num == '3')
        y1 = sin(2 * pi * row(1) * t);
        y2 = sin(2 * pi * col(3) * t);
    elseif(num == '4')
        y1 = sin(2 * pi * row(2) * t);
        y2 = sin(2 * pi * col(1) * t);
    elseif(num == '5')
        y1 = sin(2 * pi * row(2) * t);
        y2 = sin(2 * pi * col(2) * t);
    elseif(num == '6')
        y1 = sin(2 * pi * row(2) * t);
        y2 = sin(2 * pi * col(3) * t);
    elseif(num == '7')
        y1 = sin(2 * pi * row(3) * t);
        y2 = sin(2 * pi * col(1) * t);
    elseif(num == '8')
        y1 = sin(2 * pi * row(3) * t);
        y2 = sin(2 * pi * col(2) * t);
    elseif(num == '9')
        y1 = sin(2 * pi * row(3) * t);
        y2 = sin(2 * pi * col(3) * t);
    elseif(num == '*')
        y1 = sin(2 * pi * row(4) * t);
        y2 = sin(2 * pi * col(1) * t);
    elseif(num == '#')
        y1 = sin(2 * pi * row(4) * t);
        y2 = sin(2 * pi * col(3) * t);
    end

    y = ( y1 + y2 ) / 2;
    y = [pre y];
end
```

تابع دوم

ب)

برای این بخش باید صدای خوانده شده را بخش بخش کنیم.

از آنجایی که مدت زمان صداهای تولید شده از شماره ها و سکوت بین آنها برابر با 0.1 است، و فرکانس نمونه برداری برابر 8000 است، پس بخش های تقسیم شده باید به طول 800 = 0.1 × 8000 باشند.

در یک حلقه، از ابتدای صدا شروع کرده و با گام های 1600 = 800 + 800 جلو میرویم، چراکه باید سکوت هارا ازشان بگذریم.

و در این قسمت های به طول 800، به ازای تمام اعداد تابع make_on را صدا میکنیم تا صدای متناضر با آن شماره را تولید کند و سپس بین صدای تولید شده و صدای قسمتی که در آن هستیم، کورولیشن گرفته و در نهایت این صدا شماره اش برابر با ماکسیمم این کورولیشن ها میشود.

سپس اعداد تشخیص داده شده را پرینت میکنیم.

```
[audio, fs_audio] = audioread('y.wav');
audio = audio';

nums = ['0' '1' '2' '3' '4' '5' '6' '7' '8' '9' '*' '#'];
t_len = 800; % fs = 8000, t = 0.1 ===>> fx * t_len = 800

finded = [];
for number_index = 1 : (2 * t_len) : size(audio, 2)
    current_num = audio(number_index : number_index + t_len - 1);

    index_max = 1;
    corr_max = 0;
    for make_index= 1 : size(nums, 2)
        current_cor = corr2(current_num, make_on(nums(make_index), []));
        if(current_cor > corr_max)
            corr_max = current_cor;
            index_max = make_index;
        end
    end

    finded = [finded nums(index_max)];
end

fprintf('%s', finded);
```

تشخیص شماره ها

شماره ی رمز گشایی شده

قسمت سوم:

پس از خواندن تصویر مدار و آی سی مورد نظر، تابعی را برای پیدا کردن آی سی ها صدا میکنیم.

این تابع ورودی های مدار، آی سی و مرز مورد نظر برای کورولیشن را میگیرد. سپس با دو حلقه ی تودرتو به ازای تمام قسمت های عکس مدار (به اندازه ی سایز عکس آی سی) این قسمت را با خود عکس آی سی و دوران یافته ی آن کورولیشن میگیرد و اگر یکی از این کورولیشن ها بیشتر از مرز تعیین شده در ورودی بود، بنابراین یکی از آی سی هارا پیدا کرده ایم.

برای کورولیشن گرفتن نیز در یک تابع به ازای هر یک از پیکسل های قرمز، سبز و آبی به طریقه ای که در صورت سوال گفته شد کورولیشن گرقته و سپس میانگین این سه را برمیگردانیم.

و سپس عکس مدار و آی سی را رسم کرده و زیر آنها با استفاده از لوکیشن هایی که برای مستطیل های آی سی پیدا شد، دور آی سی ها خط آبی کشیده و مدار را رسم میکنیم.(با داشتن یکی از گوشه های مستطیل و سایز های عمودی و افقی آی سی، این مستطیل قابل تشخیص است).

```
[pcd_f, pcb_p] = uigetfile({'*.jpeg;*.jpg;*.bmp;*.png;*.tif'},...
    'Choose your PCB image');
pcb = imread([pcb_p, pcd_f]);

[ic_f, ic_p] = uigetfile({'*.jpeg;*.jpg;*.bmp;*.png;*.tif'},...
    'Choose your IC image');
ic = imread([ic_p, ic_f]);

founded = find(pcb, ic, 0.93);

subplot(2, 2, 1);
imshow(pcb);

subplot(2, 2, 2);
imshow(ic);

subplot(2, 2, 3 : 4);
imshow(pcb);
for i=1:size(founded, 2)
    rectangle('Position', [founded(2, i) ...
        founded(1, i) size(ic, 2) size(ic, 1)],...
        'EdgeColor','b','LineWidth',2);
end
```

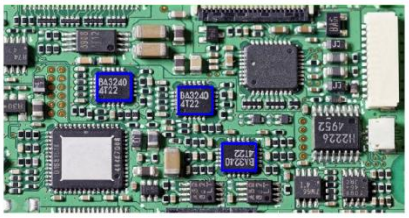
تشخیص و رسم آی سی ها

```
function founded = find(pcb, ic, threshold)
    rotatedIC = imrotate(ic, 180);

    founded = [];
    for row = 1 : size(pcb, 1) - size(ic, 1)
        for col = 1 : size(pcb, 2) - size(ic, 2)
            part_of_pcb = pcb(row : row + size(ic, 1) - 1,...
                col : col + size(ic, 2) - 1, :);

            currnet_corr = rgb_norm_corr(ic, part_of_pcb);
            current_rotat_corr = rgb_norm_corr(rotatedIC, part_of_pcb);

            if((currnet_corr > threshold) || (current_rotat_corr > threshold))
                founded = [founded [row; col]];
            end
        end
    end
end
```

تابع تشخیص لوکیشن آی سی ها در مدار

```
function corr = rgb_norm_corr(x, y)

    r_corr = norm_corr(x(:,:,1), y(:,:,1));
    g_corr = norm_corr(x(:,:,2), y(:,:,2));
    b_corr = norm_corr(x(:,:,3), y(:,:,3));

    corr = (r_corr + g_corr + b_corr) / 3;
end

function corr = norm_corr(x, y)
    x=double(x);
    y=double(y);

    xy_sum = sum(sum(x.*y));
    x2_sum = sum(sum(x.*x));
    y2_sum = sum(sum(y.*y));

    corr = xy_sum/sqrt(x2_sum*y2_sum);
end
```

تابع های مخصوص کورولیشن گیری



خروجی آی سی های تشخیص داده شده