

(سوال یک)

(بخش اول)

بخش دوم)

در این قسمت، به این صورت عمل میکنیم که ابتدا پیام مورد نظر را به باینری تبدیل کرده و سپس هر بیت از پیام باینری شده را در کم ارزش ترین خانه ی هر پیکسل قرار می دهیم. ترتیب انتخاب پیکسل ها، از سمت چپ و بالای عکس است و به صورت ستونی پر میشوند.

کد مربوط به این بخش را در فایل coding.m میتوانید مشاهده کنید.

بخش سوم)

خروجی برای تصویر اول:



خروجی برای تصویر دوم:



همانطور که مشاهده میشود، تفاوتی که با چشم بتوان تشخیص داد میان این دو عکس وجود ندارد. علت آن هم این است که مقدار هر پیکسل، یک عدد بین 0 تا 255 است. زمانی که ما بیت آخر آن را که کم ارزش ترین بیت آن است تغییر میدهیم، مقدار هر پیکسل، تنها یک واحد میتواند تغییر کند. که تفاوت بین دو پیکسل که تنها یک واحد با هم اختلاف دارند، با چشم میسر نیست.

## بخش چهارم)

برای این قسمت، دقیقا بالعکس تابع coding عمل می کنیم به این صورت که به ترتیب ستونی، بیت های آخر هر پیکسل را ذخیره میکنیم. و سپس رشته ی بدست آمده را با استفاده از mapSet، رمزگشایی میکنیم. تنها نکته ای که در کد باید به آن دقت کرد این است که لازم نیست تمام پیکسل ها را پیمایش بکنیم، زیرا هرگاه به ";" رسیدیم، می توان گفت پیام رمزنگاری شده در پیکسلی که در آن هستیم، تمام شده است. پس تا زمانی پیش میرویم که به ";" برسیم. حال اگر همیشه به 5 بیت آخر رشته ای که داریم تشکیل میدهیم، نگاه بکنیم، ممکن است اشتباهها یک ";" ببینیم. چرا که مثلا 3 بیت آخر حرف h که کد متناظر آن "00111" است و 2 بیت اول حرف y که کد متناظر آن "11000" است، اگر پشت سر هم بیایند به صورت (hy)، ما در حین رمزگشایی به رشته ی "0011111" میرسیم که پنج تای آخر آن را اگر بخوانیم، به ; که کد متناظر آن "1111" است، میرسیم. برای رفع این مشکل، کافی است تنها زمانی 5 تای آخر را چک کنیم که رشته ای که تا الان تشکیل دادیم مضرب 5 باشد، که نشان دهنده این است که حرف بعدی کامل خوانده شده است.

کد مربوط به این بخش را در فایل decoding.m میتوانید مشاهده کنید.

خروجی این تابع به ازای پیام رمزگذاری شده ی قسمت قبل:

```
>> p1_1  
fx signal;>>
```

## بخش پنجم)

از آنجایی که هر بیت پیام را در کم ارزش ترین بیت هر پیکسل میگذاریم، با کوچکترین نویز وارد شده به تصویر، حتی در بهترین حالت که باعث جابجایی 1 واحدی مقادیر پیکسل ها میشود، باعث میشود دیگر نتوانیم پیام مورد نظر را دیکود کنیم، برای این موضوع، میتوانیم چند approach متفاوت در نظر بگیریم. برای مثال، یکی از آنها این است که بیت های یکی مانده به آخر هر پیکسل را هم برابر پیام مورد نظر قرار می دهیم. که در اینصورت اگر نویز کمی وارد شود، شاید پیام مورد نظر را از بیت های یکی مانده به آخر بتوانیم استخراج کنیم. یا یک راه دیگر این است که پیام مورد نظر را به تعدادی که در پیکسل ها جا میشوند، تکرار میکنیم. به امید اینکه یک بخشی از تصویر دست نخورده باقی بماند.

## بخش ششم)

چند راه مختلف برای پی بردن به این موضوع داریم که در اینجا به دو تا از آنها میپردازیم:

1) تحلیل هیستوگرام: در این روش، هیستوگرام مقادیر پیکسل ها را رسم می کنیم. در یک عکس نرمال، یک توزیع نسبتاً نرمال وجود دارد، در یک عکس کدگذاری شده، جهش های ناگهانی یا یکسری الگوهای خاص دیده میشود که میتوان آن را تشخیص داد. (که از توزیع نرمال هم خارج شده است).

2) تحلیل نویز: میتوان تکنیک های تحلیل نویز را روی بیت های کم ارزش پیکسل های عکس را (که عموماً روش های آماری مانند Chi-Squared هستند) پیاده سازی کرد. در عکس های کد شده، این نویز روی بیت های کم ارزش هر پیکسل، مقدار بسیار بالاتری نسبت به عکس های نرمال دارند که میتوانیم یک threshold برای این مقدار نویز تعیین کنیم که بالاتر از آن نشان دهنده ی رمز نهفته در تصویر میباشد.

## سوال دوم)

### بخش اول)

برای این بخش، دو تابع کمکی به نام `generateTon` و `generateToff` پیاده سازی میکنیم.

تابع `generateTon`: عددی که صدای آن قرار است تولید شود را به همراه صدایی که تا اینجا تولید شده را میگیرد و صدای عدد ورودی داده شده را به طول فرکانس ضرب در `Ton` تولید کرده و به انتهای صدای تولید شده ی قبلی اضافه میکند.

تابع `generateTof`: صدای تولید شده تا اینجا را به عنوان ورودی میگیرد و به طول فرکانس ضرب در `Toff`، سیگنال برابر با صفر تولید میکند و به انتهای صدای تولید شده قبلی اضافه می کند.

حال به کمک این دو توابع، به آسانی در فایل `p2_1.m`، صدای متناظر با 43218765 را تولید کرده و در فایل `y.wav` ذخیره میکنیم.

کد مربوط به این بخش و دو تابع کمکی را در فایل `p2_1.m` و دو فایل هم نام با آن توابع میتوانید مشاهده کنید.

### بخش دوم)

ایده پیاده سازی این بخش این است که طول فایل صوتی داده شده را به مجموع `Ton` و `Toff` تقسیم می کنیم. حاصل نشان میدهد در مجموع چند رقم مختلف در این فایل صوتی کدگذاری شده اند. سپس با استفاده از `Ton`، سیگنال های مربوط به هر رقم را جدا میکنیم. حال به ازای هر رقم، سیگنال DTMF همه ی کلید ها را با استفاده از تابع `generateTon` بدست میاوریم و حاصل را با این رقم `correlation` می گیریم. ماکسیمم این `correlation` ها نشان میدهد که کدام کلید، معادل این صدا است. با یک حلقه این کار را به ازای تمامی ارقام تکرار می کنیم و جواب مورد نظر بدست می آید.

پیاده سازی این بخش را در قسمت `p2_2func` میتوانید مشاهده کنید.

نتیجه استفاده ازین تابع در بخش اول این سوال و تست فایل صوتی ای که در قسمت قبل رمزگذاری کردیم:

---

```
>> p2_1  
fx Decoded number is: 43218765>>
```

و تست این تابع در فایل p2\_2.m که فایل صوتی قراقره داده شده را رمزنگاری می کند:

---

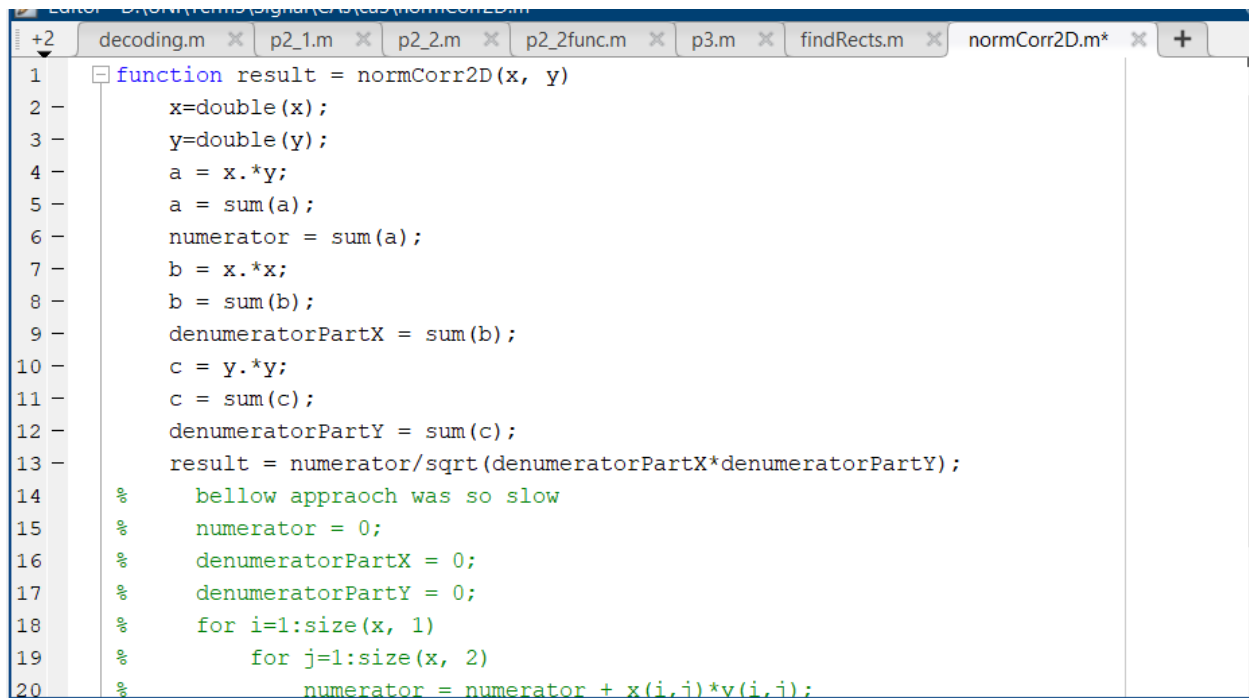
```
>> p2_2  
fx Decoded number is: 810198>> |
```

## سوال سوم)

ابتدا تعریف ضریب همبستگی نرمالایز شده برای دو سیگنال دو بعدی  $x$  و  $y$  را تعریف می کنیم:

$$\text{corr}(x, y) = \frac{\sum_n \sum_m x(n, m) y(n, m)}{\sqrt{(\sum_n \sum_m x^2(n, m)) * (\sum_n \sum_m y^2(n, m))}}$$

این تعریف در تابع `normCorr2D` در فایلی به همین اسم ذخیره شده است: (یک راه با `for` پیاده سازی شده و در کد زیر با کامنت نمایش داده شده است که به دلیل بودن بسیار زیاد، تصمیم گرفتم از عملیات ماتریسی برای پیاده سازی این راه استفاده کنم).



```
1 function result = normCorr2D(x, y)
2     x=double(x);
3     y=double(y);
4     a = x.*y;
5     a = sum(a);
6     numerator = sum(a);
7     b = x.*x;
8     b = sum(b);
9     denominatorPartX = sum(b);
10    c = y.*y;
11    c = sum(c);
12    denominatorPartY = sum(c);
13    result = numerator/sqrt(denominatorPartX*denominatorPartY);
14    % bellow appraoch was so slow
15    % numerator = 0;
16    % denominatorPartX = 0;
17    % denominatorPartY = 0;
18    % for i=1:size(x, 1)
19    %     for j=1:size(x, 2)
20    %         numerator = numerator + x(i,j)*y(i,j);
```

```
+2 decoding.m p2_1.m p2_2.m p2_2func.m p3.m findRects.m normCorr2D.m +
1 function findRects(PCBImage, ICIImage)
2     rotatedIC = imrotate(ICIImage, 180);
3     corrThr = 0.93;
4     foundIdx = [];
5     for i=1:size(PCBImage,1)-size(ICIImage,1)
6         for j=1:size(PCBImage,2)-size(ICIImage,2)
7             takeICSizeOut=PCBImage(i:i+size(ICIImage,1)-1,j:j+size(ICIImage,2)-1, :);
8             corrValueR = normCorr2D(ICIImage(:, :, 1), takeICSizeOut(:, :, 1));
9             corrValueG = normCorr2D(ICIImage(:, :, 2), takeICSizeOut(:, :, 2));
10            corrValueB = normCorr2D(ICIImage(:, :, 3), takeICSizeOut(:, :, 3));
11            corrValue = (corrValueR+corrValueG+corrValueB)/3;
12            rotatedCorrValueR = normCorr2D(rotatedIC(:, :, 1), takeICSizeOut(:, :, 1));
13            rotatedCorrValueG = normCorr2D(rotatedIC(:, :, 2), takeICSizeOut(:, :, 2));
14            rotatedCorrValueB = normCorr2D(rotatedIC(:, :, 3), takeICSizeOut(:, :, 3));
15            rotatedCorrValue=(rotatedCorrValueR+rotatedCorrValueG+rotatedCorrValueB)
16            if((corrValue > corrThr) || (rotatedCorrValue > corrThr))
17                foundIdx = [foundIdx i; j];
18            end
19        end
20    end
```

حال کاری که نیاز است انجام دهیم این است که عکس IC داده شده و دوران یافته ی آن به اندازه ی 180 درجه را ذخیره کنیم. (خط 2 کد) تفاوت راه این مسئله با مسئله ی پروژه دوم که پیدا کردن قاب پلاک در ماشین بود، این است که در آنجا در هر عکس، تنها یک قاب وجود داشت پس ما به دنبال ماکسیم correlation value پیدا شده بودیم. اما در اینجا از آنجایی که در هر تصویر ممکن است چند IC وجود داشته باشد، ما نیاز داریم یک threshold برای correlation value تعریف کنیم، و هرکجا که مقدار همبستگی ازین مقدار بیشتر شد، آن جا را هم به عنوان یک IC شناسایی شده اضافه کنیم. (خط 16 تا 18 کد) این باعث میشود در صورت وجود چند IC، همه ی آن ها شناسایی شوند.

و نکته دیگر این است که از آنجایی که عکس RGB است، یکبار همبستگی میان ماتریس های مربوط به رنگ قرمز آن ها (خط 8)، یکبار همبستگی میان ماتریس های مربوط به رنگ آبی آنها (خط 9) و یکبار هم همبستگی میان ماتریس های مربوط به رنگ سبز آنها (خط 10) را محاسبه کرده و از آنها میانگین گرفته (خط 11) و آن را به عنوان ضریب همبستگی در نظر میگیریم. حال شروع به correlation گرفتن یکبار میان خود IC (خط 8 تا 11) و تصویر PCB و یکبار میان دوران یافته ی آن و تصویر PCB می کنیم. (خط 12 تا 15) در صورتی که هرکدام از آنها بیشتر از threshold در نظر گرفته شده باشند، باید این نقطه را به عنوان یکی از مکان های پیدا شده در نظر بگیریم.



```

21 - subplot(1,2,1);
22 - imshow(PCBImage);
23 - subplot(1,2,2);
24 - imshow(ICImage);
25 - figure;
26 - imshow(PCBImage);
27 - for i=1:size(foundIdx, 2)
28 -     rect = [foundIdx(2,i)
29 -             foundIdx(1,i)
30 -             size(ICImage, 2)
31 -             size(ICImage, 1)];
32 -     rectangle('Position', rect, 'EdgeColor','g','LineWidth',2);
33 - end
34 - end

```

در ادامه ی کد که در بالا نشان داده شده، تنها کاری که میکنیم این است که مستطیل های متناظر با نقطه هایی که پیدا کردیم را رسم می کنیم.(هر نقطه ی یافت شده، سمت چپ بالای هر مستطیل را نشان میدهد. و طول تصویر IC، طول مستطیلی که باید رسم شود و عرض آن، عرض مستطیلی که باید رسم شود را نشان میدهد).

فایل p3، تنها کاری که میکند این است که تصویر PCB و تصویر IC را می گیرد و تابع findRects را به ازای آن دو صدا میزند. با اجرا کردن p3، تصویر خروجی این است:

