بنام خدا

دانشگاه صنعتی اصفهان
دانشکده برق و کامپیوتر

تمرین سری چهارم –پردازش تصاویر دیجیتال

استاد: دکتر سعید صدری

پروانه رشوند ۹۴۱۰۱۲۴
رضا سعادتی فرد ۹۴۱۱۳۹۴

تاریخ تحویل ۹۵/۰۸/۱۶

سوال اول) : (کد این قسمت در فایل HW4_Q1.m قرار دارد)

در این سوال یک تصویر به ما داده شده که این تصویر توسط DCT رمزنگاری شده است. می دانیم نقطه‌ی (۱و۴) یکی از پیکسل های مورد نظر است ولی درمورد آدرس پیکسل دیگر بین (۳و۲) و (۲و۳) شک داریم. می‌خواهیم پیام کد شده در تصویر را بیابیم.

تصویر داده شده به صورت زیر می‌باشد:

قرار است پیام نهان شده در این تصویر را بیابیم. برای این کار به این طریق عمل می‌کنیم که تصویر را به بلوک‌های ۸ × ۸ تقسیم کرده و سپس از هرکدام از آن‌ها DCT می‌گیریم. بعد از این‌که DCTی دوبعدی گرفته شد سپس باید برای هر بلوک، DCTی (۱و۴) را با (۳و۲) و یا (۲و۳) مقایسه نماییم. از آن‌جا که در مورد (۳و۲) و یا (۲و۳) شک داریم، این مراحل را دو مرتبه انجام می‌دهیم. یک بار مقایسه بین (۱و۴) و(۲و۳) و یک بار هم مقایسه بین (۱و۴) و(۳و۲). مثلا اگر قرار باشد برای (۲و۳) انجام دهیم، بعد از این‌که DCT گرفته شد به این ترتیب عمل می‌کنیم:

$$if\ DCT(4,1) > DCT(2,3) \quad \rightarrow \quad pixel = 1$$
$$if\ DCT(4,1) < DCT(2,3) \quad \rightarrow \quad pixel = 0$$

به همین شیوه‌ای که اشاره شد، کد برنامه را به صورت زیر می‌نویسیم:

```
load('exersize4.mat');

figure;
imshow(F)
title('embededd message')

[M,N] = size(F);
p=0;
for m=1:8:M
    p=p+1; q=0;
    for n=1:8:N
        q=q+1;
   temp = F(m:m+7,n:n+7);
   DCT_temp = dct2(temp);
   DCT_41(p,q) = DCT_temp(4,1);
   DCT_23(p,q) = DCT_temp(2,3);
   DCT_32(p,q) = DCT_temp(3,2);
    end
end

Message_1 = zeros(M/8,N/8);
Message_2 = zeros(M/8,N/8);

temp_1 = DCT_41 - DCT_23;
a_1=find(temp_1>=0);
Message_1(a_1) = 1;

temp_2 = DCT_41 - DCT_32;
a_2=find(temp_2>=0);
Message_2(a_2) = 1;

figure;
subplot(211),imshow(Message_1),title('if we choses pixel(2,3)');
subplot(212),imshow(Message_2),title('if we choses pixel(3,2)');
```

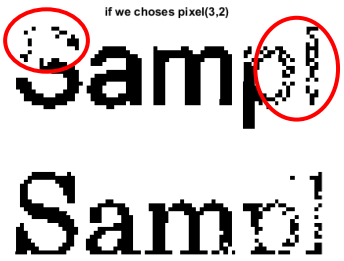
در کد بالا در مرحله‌ی اول پیکسل (۳و۲) را انتخاب کرده و مقایسه را برای این پیکسل انجام داده‌ایم که نتیجه به صورت زیر مشاهده می‌شود:

از نتیجه‌ی بالا هیچ مفهومی دریافت نمی‌شود. بنابراین پیکسل (۳و۲) اشتباه بوده و بنابراین باید مقایسه را بین (۱و۴) و (۲و۳) انجام دهیم که کد بالا این مقایسه را نیز در بر دارد.

نتیجه‌ی حاصل به صورت زیر است:

از پیام مشاهده شده با اطمینان می‌توان گفت آدرس پیکسل‌های مورد نظر، (۱و۴) و (۲و۳) بوده‌است و پیام مخفی شده در عکس نیز متن بالا می‌باشد.

اما اگر در متن بالا دقت کنیم، مشاهده می‌نماییم که قسمت‌هایی از متن تخریب شده‌اند. نظیر قسمت‌هایی که در زیر مشخص کرده ایم:



علت این امر را می‌توان بدین صورت توضیح داد که از آن‌جا که گوشه‌های تصویر اصلی قسمت‌های کاملا یکنواخت و بدون تغییر دارد یعنی تغییرات سطوح روشنایی تصویر پایین می‌باشد در نتیجه شاهد تغییرات فرکانسی اندکی در ماتریس DCT می‌باشیم.
اگر ماتریس DCT ی تصویر را برای یکی از این نواحی بخواهیم نشان دهیم به‌صورت زیر خواهد بود:

1.0e+03 *

1.9040	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

همان‌طور که در ماتریس به دست آمده نیز مشاهده می‌شود، هر دو درایه‌ی (۱و۴) و(۲و۳) صفر می‌باشند. لذا مقایسه‌ای نمی‌توان بین آن‌ها صورت داد و لذا برنامه به صورت رندم برای این مقادیر تصمیم‌گیری نموده‌است که در نتیجه در متن پیام قسمت‌هایی ایجاد شده‌اند که تخریب شده‌اند.

سوال دوم) (کد این قسمت در فایل HW4_Q2.m قرار دارد)

در این سوال، یک متن به ما داده شده‌است و قرار است یک تصویر را به دلخواه انتخاب نموده و متن مورد نظر را در این عکس با DCT رمزنگاری نماییم. تصویر مورد نظر ما که قرار است متن را در آن کد کنیم تصویر زیر می‌باشد:

و متنی که قرار است کد شود به صورت زیر می‌باشد:

راست ساکت و باربز باش

متن مورد نظر ما دارای ابعاد ۹۷*۲۰ می‌باشد. در قدم اول ما تمام پیکسل‌ها را درون یک بردار می‌ریزیم. برداری به طول ۱۹۴۰(۹۷×۲۰). ابعاد عکس درنظر گرفته شده‌ی ما نیز ۳۵۲*۳۶۰ می‌باشد. برای این‌که بتوان تصویر را به بلوک‌های ۸*۸ تقسیم نمود از آنجا که $\frac{360*352}{64} = 1980$ ، باید برداری که پیکسل‌های متن در آن است را zeropad نماییم (۴۰ صفر به انتهای بردار اضافه می‌کنیم). سپس از بلوک‌های مورد نظر DCT گرفته و دو پیکسل از این بلوک را انتخاب نموده و اولین بیت اطلاعات را می‌خوانیم که یا صفر می‌باشد و یا یک. اگر بیت اطلاعات برابر با یک بود و مقدار DCT پیکسل اول بزرگتر از دومی بود تغییری ایجاد نمی‌کنیم در غیر این صورت جای آن‌ها را عوض می‌نماییم. برای بیت صفر نیز اگر پیکسل اول کوچکتر از دومی بود تغییری ایجاد نمی‌کنیم در غیر این صورت جای آن‌ها را عوض می‌نماییم.

$$if \quad pixel\ of\ code = 1 \quad \rightarrow \quad must \quad DCT(pixel\_1) > DCT(pixel\_2)$$
$$if \quad pixel\ of\ code = 0 \quad \rightarrow \quad must \quad DCT(pixel\_2) > DCT(pixel\_1)$$

ما بدنبال انتخاب یک معیار بهینه برای انتخاب دو پیکسل مورد نظر می‌باشیم. برای نمونه، ماتریس ۸×۸ تصویر که از آن DCT گرفته شده است به صورت زیر می‌باشد:

1.0e+03 *

فرکانس‌های پایین

1.0714	-0.0003	0.0045	0.0003	-0.0119	-0.0005	0.0002	-0.0003
0.0179	0.0002	0.0000	0.0099	0.0000	-0.0002	-0.0000	0.0003
-0.0067	0.0002	-0.0001	-0.0001	-0.0003	0.0001	0.0003	-0.0006
-0.0002	-0.0000	-0.0002	0.0000	-0.0003	-0.0001	0.0004	0.0001
-0.0004	-0.0006	-0.0001	0.0001	-0.0001	0.0003	0.0002	0.0004
-0.0001	-0.0004	-0.0001	-0.0001	-0.0001	0.0002	0.0002	0.0001
0.0005	0.0004	-0.0002	0.0010	-0.0003	-0.0002	0.0004	0.0001
-0.0003	0.0003	-0.0002	0.0000	-0.0003	0.0001	0.0004	0.0001

فرکانس‌های میانی

فرکانس‌های بالا

مشاهده می‌شود که در فرکانس‌های پایین، تغییرات بین پیکسل‌ها بسیار زیاد می‌باشد و لذا اگر قرار باشد جای آن‌ها عوض شود، تصویر به شدت تخریب می‌شود. لذا بهتر است این دو پیکسل را از فرکانس‌های پایین انتخاب نکنیم. در فرکانس‌های بالا نیز تغییرات بین پیکسل‌ها بسیار ناچیز می‌باشد و ممکن است گاهی DCT دو پیکسل مقدار یکسان داشته باشد و نتوانیم کد مورد نظر را در آن قرار دهیم و بنابراین در متن بازیابی شده ممکن است یک سری پیکسل‌های تخریب شده ایجاد شوند. بنابراین لازم ست که این دو پیکسل را از فرکانس‌های میانی انتخاب نماییم تا به نوعی، هم تصویری که کد در آن قرار گرفته است کیفیت مطلوبی داشته باشد و هم متن بازیابی شده خیلی تخریب نشود. برای چند نمونه از این نقاط در فرکانس‌های بالا، پایین و میانی کد را اجرا می‌نماییم تا نتیجه را مشاهده نماییم:

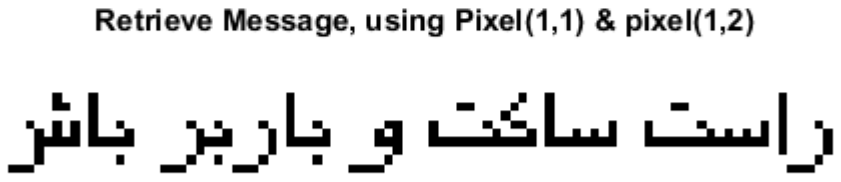
کد برنامه به صورت زیر است:

```
load('exersize4_2.mat');
original_code = round(z);

f=imread('lena.jpg');
figure
imshow(f,[]);

%%%%% first we must put our code in a vector
[M,N] = size(f);
```

```
code_length = (M*N/64);
code = zeros(1,code_length); %%% we must do zeropad , at the end we will remove additional
zeros
temp =original_code.';
code(1,1:20*97) = temp(:);
%%%%
f_new = zeros(M,N);
p=0; counter_code =0;

%%%%%% selecting 2 points in order to coding process
pix_1_x = 1; pix_1_y = 1;
pix_2_x = 1; pix_2_y = 2;

for m=1:8:M
    p=p+1; q=0;
    for n=1:8:N
        q=q+1; counter_code = counter_code+1;

        DCT_temp = dct2(f(m:m+7,n:n+7));

        pixel_code = code(counter_code);

        switch pixel_code
            case 0
               if DCT_temp(pix_1_x,pix_1_y) >= DCT_temp(pix_2_x,pix_2_y)
                    temp = DCT_temp(pix_1_x,pix_1_y);
                    DCT_temp(pix_1_x,pix_1_y) = DCT_temp(pix_2_x,pix_2_y);
                    DCT_temp(pix_2_x,pix_2_y) = temp;
                end
            case 1
                if DCT_temp(pix_1_x,pix_1_y) <= DCT_temp(pix_2_x,pix_2_y)
                    temp = DCT_temp(pix_1_x,pix_1_y);
                    DCT_temp(pix_1_x,pix_1_y) = DCT_temp(pix_2_x,pix_2_y);
                    DCT_temp(pix_2_x,pix_2_y) = temp;
                end
        end

       f_new(m:m+7,n:n+7) = idct2(DCT_temp);
    end
end


figure,subplot(121),imshow(f),title('Original Image');
subplot(122),imshow(f_new,[]),title(['Image with DCT Steganography ,Using pixel('
num2str(pix_1_x),','...
    ,num2str(pix_1_y),') & pixel(',num2str(pix_2_x),',',num2str(pix_2_y),')']);

%%%%% decoding

[M,N] = size(f_new);
index_1 = zeros(1,code_length); index_2 = zeros(1,code_length);
q=0;
for m=1:8:M
    for n=1:8:N
        q=q+1;
   temp = f_new(m:m+7,n:n+7);
   DCT_temp = dct2(temp);
   index_1(1,q) = DCT_temp(pix_1_x,pix_1_y);
   index_2(1,q) = DCT_temp(pix_2_x,pix_2_y);
    end
end
temp_code = zeros(1,code_length);
index_1 = find(index_1 - index_2 > 0); %%% find index that the code is 1
temp_code(index_1) = 1;

vector_code = temp_code(1,1:20*97);         %%% delete the zeropad that we added at the first
retrieve_code = reshape(vector_code ,[97 20]);

% gg = original_code - retrieve_code;

figure,subplot(211),imshow(original_code),title('Oroginal Message');
subplot(212),imshow(retrieve_code.',[]),title(['Retrieve Message, using Pixel('
num2str(pix_1_x),',',num2str(pix_1_y)...
    ') & pixel(',num2str(pix_2_x),',',num2str(pix_2_y),')'])
```

در مرحله اول، از فرکانس‌های پایینی پیکسل‌های (۱و۱) و(۲و۱) را انتخاب نماییم، نتیجه‌ی تصویر به صورت زیر خواهد بود:

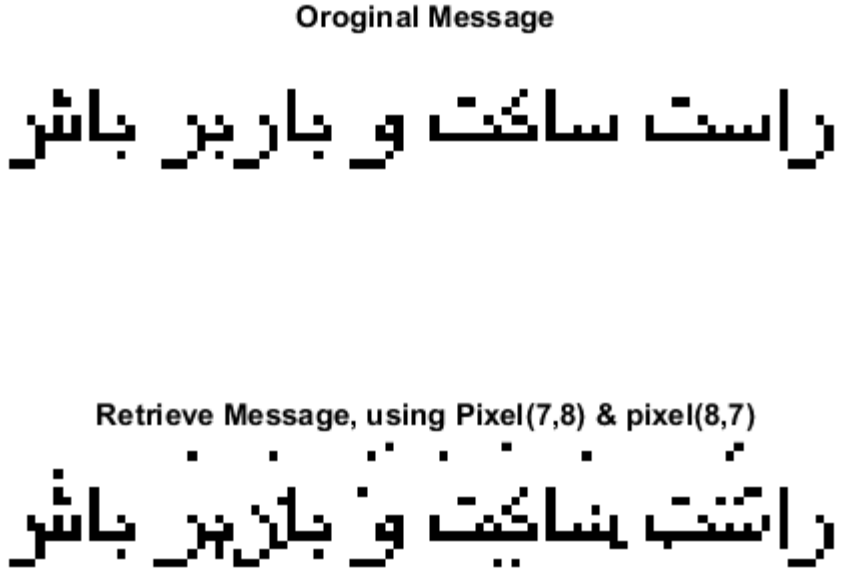
و متن دیکد شده به صورت زیر می‌باشد:



مشاهده می‌شود که دیکدینگ بسیار خوب انجام شده است زیرا وقتی اختلاف بین پیکسل‌ها در فرکانس‌های پایینی زیاد استُ مقایسه بسیار دقیق انجام می‌شود.اما تصویر به شدت به هم ریخته است که بدلیل جابجایی پیکسل‌هایی است که اختلاف زیادی با هم دارند. در واقع همان نتیجه مورد انتظار را مشاهده نمودیم.

در مرحله‌ی بعد پیکسل‌ها را از فرکانس‌های بالا انتخاب می‌نماییم. مثلا پیکسل‌های (۷و۸) و(۸و۷). نتیجه‌ی تصویر به صورت زیر می‌باشد:

و نتیجه‌ی دیکدینگ به صورت زیر است:

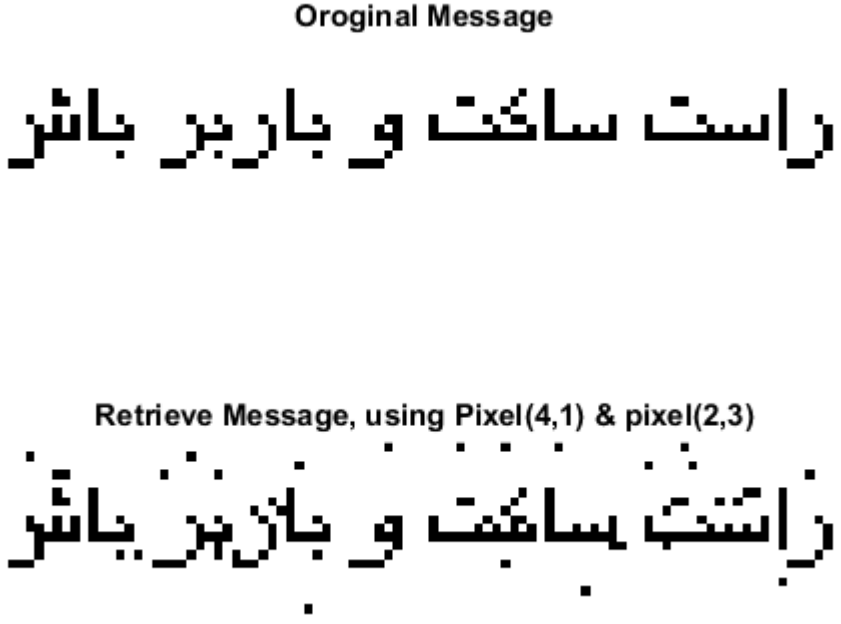


همان‌طور که انتظار داشتیم از آن‌جایی که اختلافی بین DCT برخی پیکسل‌ها در فرکانس‌های بالا وجود ندارد، لذا قادر به قرار دادن کد در این پیکسل ها و بازیابی آن نیستیم، که این امر باعث ایجاد نویز می‌شود. به علت توضیح داده شده، دیکدینگ بسیار بد شده است.

در مرحله‌ی بعد پیکسل‌ها را از فرکانس‌های میانی انتخاب می‌نماییم. مثلا پیکسل (۱و۴) و(۳و۲). که نتایج زیر را مشاهده می‌نماییم:

نتیجه‌ی مشاهده شده از تصویر به صورت زیر است:

و نتیجه‌ی دیکدینگ به صورت زیر می‌باشد:



مشاهده می‌شود همان‌طور که انتظار داشتیم، هم تصویر کدگذاری شده کیفیت مطلوبی دارد و هم دیکدینگ قابل قبول می‌باشد. بنابراین بهترین انتخاب پیکسل‌ها مربوط به فرکانس‌های میانی می‌باشد که علت آن اختلاف اندک بین DCT پیکسل های فرکانس میانی است.

سوال سوم-الف) (کد این قسمت در فایل HW4_Q3.m می‌باشد)

در این سوال قرار است که از تبدیل rotation برای یک تصویر استفاده نموده و تصویر را به اندازه‌ی ۳۸ درجه چرخش دهیم. ماتریس تابع تبدیل برای عملگر rotation به صورت زیر می‌باشد:

$$A = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

اگر U مختصات تصویر اصلی باشد و X مختصات تصویر روتیت شده باشد، تصویر روتیت شده از طریق ماتریس تبدیل بدین صورت بدست می‌آید:

$$X = AU$$

که برای چرخش ۳۸ درجه، این ماتریس تبدیل به صورت زیر درخواهد آمد:

$$A = \begin{bmatrix} 0.7880 & -0.6157 \\ 0.6157 & 0.7880 \end{bmatrix}$$

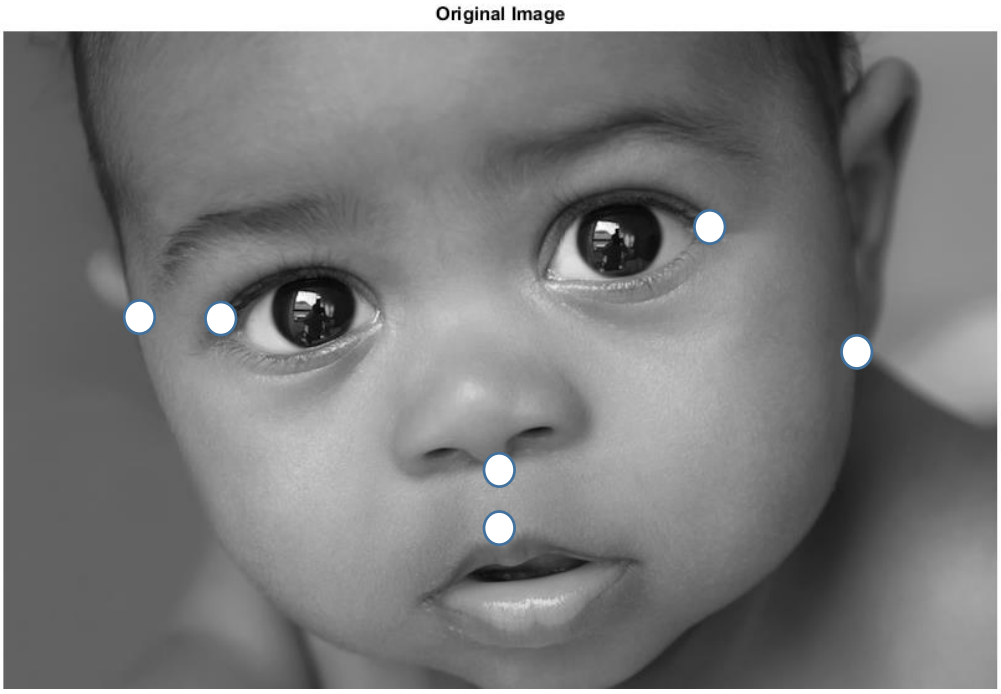
حال قرار است با توجه به تصویر اصلی و تصویر روتیت شده، ماتریس تبدیل را خودمان بدست آوریم:

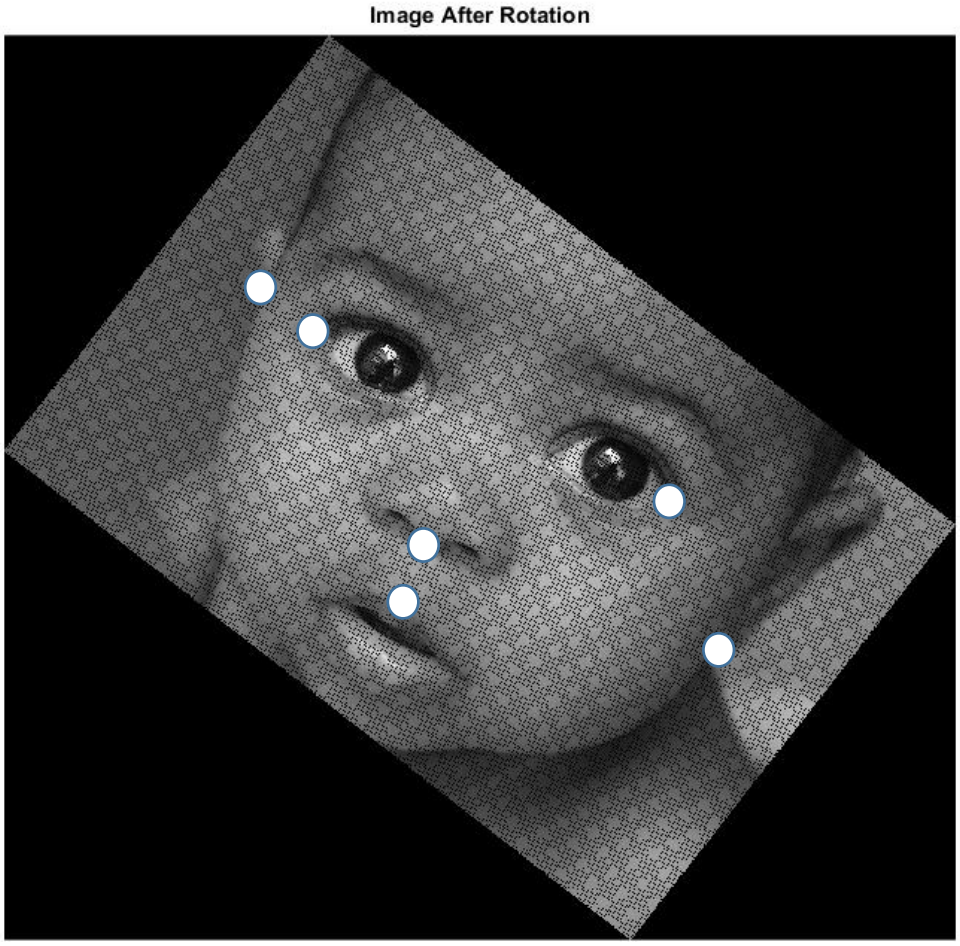
کد برنامه به صورت زیر است:

```
% clearvars -except A_Retrieve2 f_retrieve;
close all;

f = imread('baby2.jpg');
f = rgb2gray(f);

figure(1),imshow(f),title('Original Image');

[M,N] = size(f);
theta = degtorad(38);

A = [cos(theta) -sin(theta) 0; sin(theta) cos(theta) 0; 0  0 1];

%%%%%%%%%%%%%%%%%%%%%%%%%%%% calculate rotation Image

for m=1:M
    for n=1:N

        U = [m n 1];
        uv = (U*A).';
        u=round(uv(1,1)); v= round(uv(2,1));
```

```
        f_rotation(u-min_u+1 , v-min_v+1) = f(m,n);
    end
end

figure(2), imshow(f_rotation),title('Image After Rotation');

%%%%%%%%%%%%% calculating Transformation matrix with 6 point
[x,y] = getpts(figure(2));
X = [x  y ones(6,1)];

[u,v] = getpts(figure(1));
U = [u  v ones(6,1)];

A_Retrieve2 = inv(X.'*X)*X.'*U;
```

تصویر اصلی و تصویر روتیت شده با استفاده از کد بالا به‌صورت زیر خواهد بود:





حال، همان‌طور که در کد بالا نیز آورده شده است، ۶ نقطه از تصویر اصلی و ۶ نقطه در همان محل‌ها در تصویر روتیت شده را در نظر گرفته و از طریق رابطه‌ی زیر، ماتریس تبدیل را محاسبه می‌نماییم:

$$X = AU \rightarrow U^T X = U^T U A \rightarrow A = (U^T U)^{-1} U^T X$$

۶ نقطه‌ی مورد نظر ما در تصاویر بالا مشخص شده‌اند و ماتریس تبدیل موردنظر به صورت زیر به‌دست می‌آید:

$$A = \begin{bmatrix} 0.8000 & -0.5983 & -5.85469173142172e-18 \\ 0.5965 & 0.7819 & -1.51788304147971e-18 \\ -253.1892 & 197.4356 & 1 \end{bmatrix}$$

اگر این ماتریس به‌دست امده را با ماتریس تبدیل اصلی مقایسه نماییم، نتیجه‌ی تقریبا قابل قبولی به‌دست آمده است و اندک خطای ایجاد شده به دلیل کمبود دقت ما در انتخاب ۶ نقطه در تصویر روتیت شده و تصویر اصلی می‌باشد.

همان‌طور که مشاهده می‌شود، درایه‌های (۱و۳) و (۲و۳) در این ماتریس دارای مقدار شده است در صورتی که در ماتریس اصلی این درایه‌ها صفر بود. علت آن این است که به هنگام روتیت کردن تصویر، به علت وجود ترم

$$y = -\sin(\theta)\, u + \cos(\theta) v$$

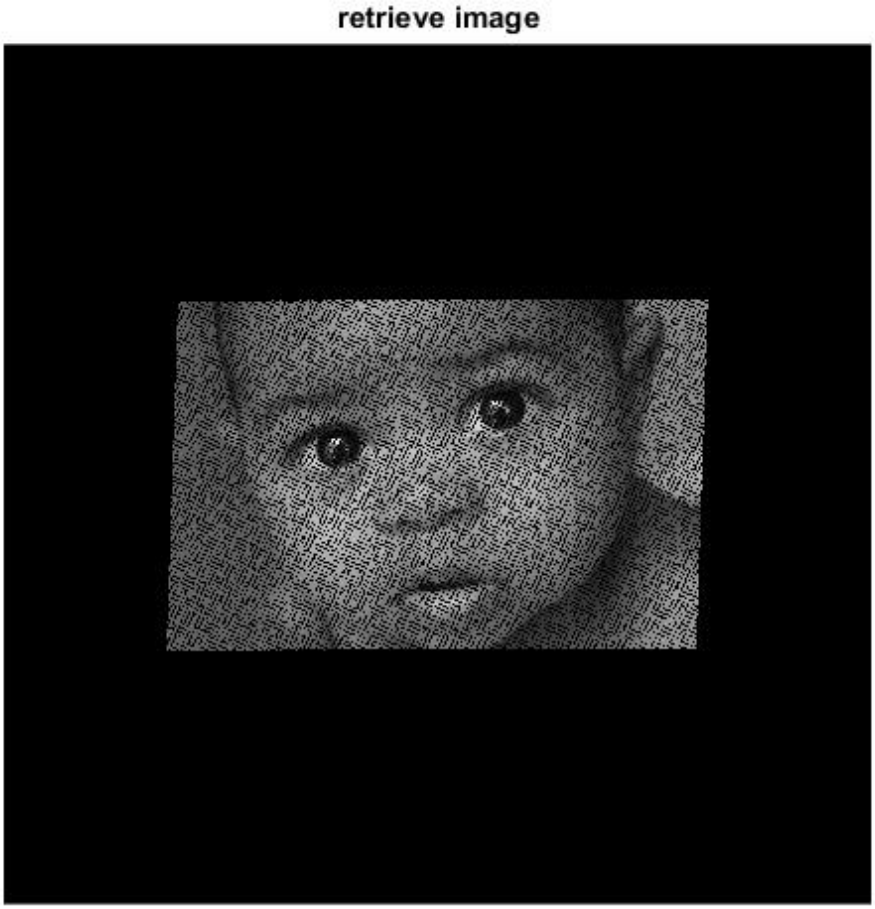
اندیس‌های منفی ایجاد می‌گردد و ما برای جلو گیری از منفی و صفر شدن اندیس ها نیاز داریم اندیس‌ها را شیفت دهیم که این کار باعث غیر صفر شدن این دو اندیس (که مربوط به شیفت هستند) می شود.

در ادامه قرار است توسط این ماتریس تبدیل به‌دست آمده و از روی نقاط در نظر گرفته شده، تصویر اصلی را از تصویر روتیت شده بازیابی نماییم:

کد برنامه برای بازیابی تصویر اصلی از تصویر روتیت شده به صورت زیر می‌باشد:

```
%%%%%%%%%%%% Retrive Original Image with A Coeficient
%%%%%%%%% get min max
[M1 , N1] = size(f_rotation);
inv_A = inv(A_Retrieve2);
uv=[];
p=0;
for m=1:M1
    for n=1:N1
        p=p+1;
        U = [m n 1];
        uv(:,p) = (U*inv_A).';
    end
end
u = uv(1,:); v=uv(2,:);
max_u = round(max(u(:))); max_v = round(max(v(:)));
min_u = round(min(u(:))); min_v = round(min(v(:)));

%%%%%%%%%%%%%%%%%%%%

for m=1:M1
    for n=1:N1
        U = [m n 1];
        uv = (U*inv_A).';
        u=round(uv(1,1)); v= round(uv(2,1));

        f_retrieve(u-min_u+2 , v-min_v+1) = f_rotation(m,n);
    end
end

figure(4), imshow(f_retrieve);
```

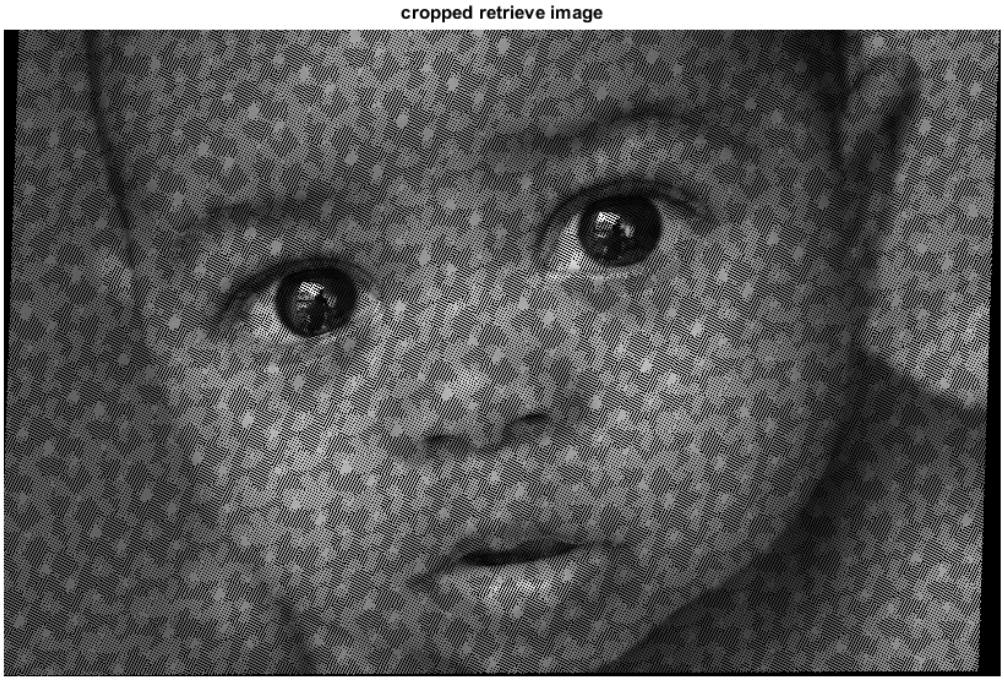
و نتیجه، به صورت زیر مشاهده می‌شود:



برای حذف حاشیه‌های تاریک اطراف تصویر، از الگوریتم زیر جهت کراپ کردن نتیجه استفاده می‌نماییم:

```
%%%%%%%%%%  crop
[r,c] = find(f_retrieve~=0);
min_r = min(r)+5;
min_c = min(c)+5;
temp_1 = f_retrieve(min_r,min_c:end);
max_c  = max(find(temp_1 ~= 0)) + min_c;


temp_2 = f_retrieve(min_r:end,min_c);
max_r = max(find(temp_2 ~= 0)) + min_r;

f_crop = f_retrieve(min_r:max_r , min_c:max_c);
figure(5),imshow(f_crop)
```

و درنهایت، پس از کراپ کردن نتیجه‌ی زیر را مشاهده می‌نماییم:



ب) بهبود تصاویر با حذف نقاط سیاه

همان‌طور که در نتایج مشاهده می‌کنیم، در تمامی نتایج، یک سری نقاط سیاه رنگی مشاهده می‌شود که علت آن‌ها این است که به دلیل وجود توابع مثلثاتی، در اندیس‌ها مقادیر غیر صحیح مشاهده می‌شود و ما تمامی این مقادیر را در برنامه صحیح نموده‌ایم و این نقاط ایجاد شده اند. برای رفع این مشکل، ما برای هر پیکسل، ۸ پیکسل همسایه‌ی آن را در نظر گرفته و میانگین آن‌ها را حساب می‌کنیم تا پیکسل مورد نظر تعیین شود و بدین ترتیب، تصویر بهبود داده شده به دست می‌آید.

کد مورد نظر به صورت زیر می‌باشد:

```
%% Improved Image
%%%%%%%%%%  Clear Black dots
%%%%%%%%%%%%%%%%%%%%%%%%%% using 8 points around black dots to removo them
[R,C] = find(f_rotation ==0);
%%%% remove indice that row is minimum or moximum on them
remove_min_max_R = find(R~=min(R) & R~=max(R));
R = R(remove_min_max_R); C = C(remove_min_max_R);
%%%% remove indice that coloum is minimum or moximum on them
remove_min_max_C = find(C~=min(C) & C~=max(C));
R = R(remove_min_max_C); C = C(remove_min_max_C);

zero_index = [R.';C.'];

f_rotation_Improved = f_rotation;
```

```
for i=1:length(zero_index);
    r = zero_index(1,i);
    c = zero_index(2,i);
    temp = f_rotation(r-1:r+1,c-1:c+1);
    index_non_0 = find(temp~=0);
    l_non_0 = length(index_non_0);
    if l_non_0 > 0
           f_rotation_Improved(r,c) = sum(temp(:))/l_non_0;
    end

end
 figure(5), imshow(f_rotation_Improved)


%%%%%%
[x,y] = getpts(figure(5));
X = [x  y ones(6,1)];

[u,v] = getpts(figure(1));
U = [u  v ones(6,1)];

A_Retrieve_Improved = inv(X.'*X)*X.'*U;

%%%%%%%%%%%% Retrive Original Image with A Coeficient
%%%%%%%%% get min max
[M1 , N1] = size(f_rotation_Improved);
inv_A = inv(A_Retrieve_Improved);
uv=[];
p=0;
for m=1:M1
    for n=1:N1
        p=p+1;
        U = [m n 1];
        uv(:,p) = (U*inv_A).';
    end
end
u = uv(1,:); v=uv(2,:);
max_u = round(max(u(:))); max_v = round(max(v(:)));
min_u = round(min(u(:))); min_v = round(min(v(:)));

%%%%%%%%%%%%%%%%%%%

for m=1:M1
    for n=1:N1
        U = [m n 1];
        uv = (U*inv_A).';
        u=round(uv(1,1)); v= round(uv(2,1));

        f_retrieve_Improved(u-min_u+2 , v-min_v+1) = f_rotation_Improved(m,n);
    end
end

figure(6), imshow(f_retrieve_Improved);

%%%%%%%%%%  Clear Black dots
%%%%%%%%%%%%%%%%%%%%%%%%%% using 8 points around black dots to removo them
R=[]; C=[]; remove_min_max_R=[]; remove_min_max_C=[]; zero_index=[];
[R,C] = find(f_retrieve_Improved ==0);
%%%% remove indice that row is minimum or moximum on them
remove_min_max_R = find(R~=min(R) & R~=max(R));
R = R(remove_min_max_R); C = C(remove_min_max_R);
%%%% remove indice that coloum is minimum or moximum on them
remove_min_max_C = find(C~=min(C) & C~=max(C));
R = R(remove_min_max_C); C = C(remove_min_max_C);

zero_index = [R.';C.'];

f_retrieve_Improved2 = f_retrieve_Improved;
for i=1:length(zero_index);
    r = zero_index(1,i);
    c = zero_index(2,i);
    temp = f_retrieve_Improved(r-1:r+1,c-1:c+1);
    index_non_0 = find(temp~=0);
    l_non_0 = length(index_non_0);
    if l_non_0 > 0
           f_retrieve_Improved2(r,c) = sum(temp(:))/l_non_0;
    end
```

```
end
 figure(7), imshow(f_retrieve_Improved2)


%%%%%%%%%%  crop
[r,c] = find(f_retrieve_Improved2~=0);
min_r = min(r)+15;
min_c = min(c)+15;
temp_1 = f_retrieve_Improved2(min_r,min_c:end);
max_c  = max(find(temp_1 ~= 0)) + min_c;


temp_2 = f_retrieve_Improved2(min_r:end,min_c);
max_r = max(find(temp_2 ~= 0)) + min_r;

f_crop = f_retrieve_Improved2(min_r:max_r , min_c:max_c);
figure(8),imshow(f_crop)

error_1 = norm(A(1:2,1:2)-A_Retrieve_Improved(1:2,1:2))*100; %%% norm of error without
removing black dots
error_2 = norm(A(1:2,1:2)-A_Retrieve_Improved(1:2,1:2))*100; %%% norm of error after removing
black dots
```

پس از آن‌که نقاط سیاه را حذف نمودیم و تصویر روتیت شده را بهبود دادیم نتیجه‌ی زیر مشاهده می‌شود:

مشاهده می‌شود که با روش میانگین گیری ۸ پیکسل همسایه، تصویر روتیت شده بسیار بهبود یافته است.

مشابه حالت قبل، ۶ نقطه از این تصویر و ۶ نقطه متناظر آن‌ها در تصویر اصلی را انتخاب نموده تا ماتریس تبدیل به‌دست آید که پس از ران کردن برنامه، ماتریس تبدیل به صورت زیر به دست خواهد آمد:

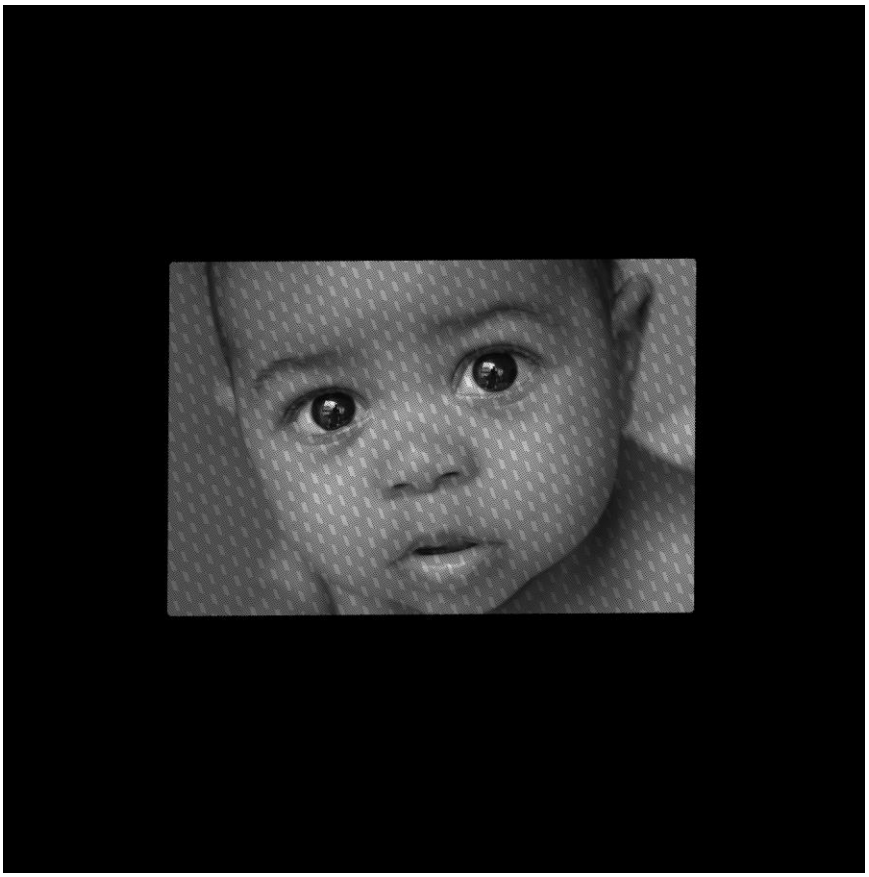
$$A = \begin{bmatrix} 0.7872 & -0.6304 & -5.85469173142172e-18 \\ 0.6169 & 0.7877 & -4.33680868994202e-18 \\ -264.6731 & 208.1463 & 1 \end{bmatrix}$$

مشاهده می‌شود که با بهبود دادن تصویر روتیت شده، نتیجه بسیار به ماتریس تبدیل اصلی نزدیک شده است چرا که کیفیت تصویر روتیت داده شده بهبود یافته و ما بهتر می‌توانیم ۶ نقطه موردنظر را در تصویر روتیت شده و تصویر اصلی انتخاب نماییم.

برای این‌که مشاهده نماییم تا چه حد اختلاف بین ماتریس تبدیل تصویر بهبود داده شده و ماتریس تبدیل تصویر با نقاط سیاه، وجود دارد نرم خطای بین این دو ماتریس را در این دو حالت با ماتریس اصلی را به دست آوردیم. که خطای بین ماتریس تبدیل تصویر بهبود داده شده و ماتریس تبدیل اصلی برابر با 1.43 درصد شد و خطای بین ماتریس تبدیل تصویر با نقاط سیاه و ماتریس تبدیل اصلی برابر با 2.15 درصد به دست آمد.

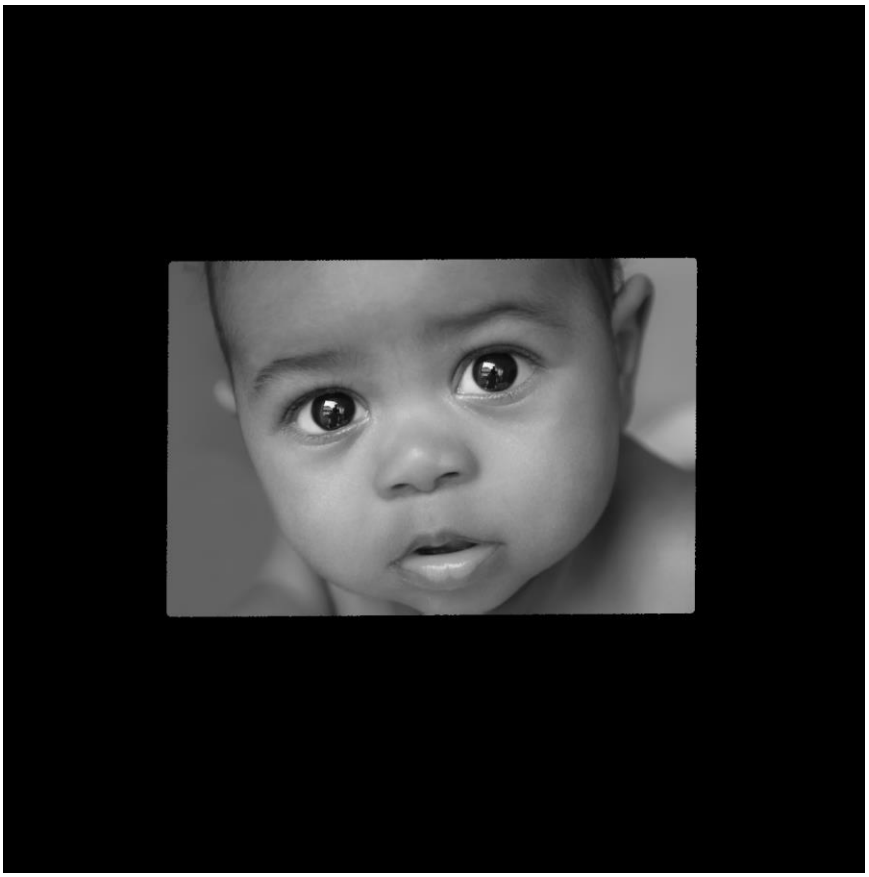
حال می‌خواهیم تصویر اصلی را از تصویر روتیت داده شده در حالت بهبود یافته به دست آوریم. نتیجه زیر را مشاهده می‌نماییم:

retrieve improved image



چون تصویر روتیت داده شده را مجددا روتیت نمودیم نقاط سیاه دوباره ایجاد شد. لذا باز هم از الگوریتم ۸ پیکسل همسایه برای از بین بردن نقاز سیاه استفاده می‌نماییم تا نتیجه زیر حاصل شود:

removing black dots from retrieved image



برای از بین بردن حاشیه ها نیز از همان متد قسمت اول سوال استفاده می‌نماییم که در نهایت نتیجه به صورت زیر خواهد بود:
توجه شود که نتیجه کراپ کردن در صورتی مناسب خواهد بود که نقاط انتخاب شده درست باشند و ماتریس معکوس به خوبی بتواند شکل را به حالت مستطیلی بازگرداند چون روش کراپ استفاده شده براساس مستطیلی بودن شکل می‌باشد.

croped retrieve image

مشاهده می‌شود که به نسبت حالتی که نقاط سیاه در تصویر وجود داشت نتیجه بسیار خوبی به دست آمد.

سوال ۴) کد مربوط به این قسمت در فایل HW4_Q4.m قرار گرفته است.

در این سوال هدف بررسی اثر فشرده‌سازی توسط صفر کردن تعدادی از المان های کم اهمیت DCT است. برای این کار طبق روال گفته شده در سوال، تصویر را به بلوک های ۸×۸ تقسیم کرده و از آن‌ها اخذ تبدیل DCT می‌کنیم سپس در دوحالت خواسته شده در سوال، یک بار المان‌های ۴×۴ بالا و چپ و یک‌بار ۲×۲ از ضرائب DCT را نگه می‌داریم و سایر ضرائب DCT را ۰ می‌کنیم. سپس از این ضرائب که مقدار زیادی از المان‌های آن صفر شده اخذ تبدیل معکوس DCT می‌کنیم و با کنار هم قرار دادن این بلوک‌های ۸×۸ تصویر را بدست می‌آوریم که این صفر کردن ضرائب DCT به منزله فشرده سازی تصویر می‌باشد. لازم به ذکر است که با توجه به اینکه می‌خواهیم تصویر را به بلوک های ۸×۸ تقسیم کنیم نیازمند به zeropad کردن تصویر، به طوری که تصویر ایجاد شده ابعادش مضربی از ۸ باشد.

در زیر تصویر اصلی و نتایج تصاویر بدست آمده را برای دو حالت خواسته شده، را شاهد هستیم:

using only 4*4 left up of DCT coefficients , compression rate is 75 %

using only 2*2 left up of DCT coefficients, compression rate is 93.75 %

در قسمت بعد از ما خواسته شده است میزان فشرده سازی را نیز بدست آوریم که در زیر شاهد مقادیر بدست آمده هستیم:

برای محاسبه میزان فشرده سازی از رابطه زیر استفاده می‌کنیم:

$$\text{میزان فشرده سازی} = \left(1 - \frac{\text{طول انتخابی}^2}{8^2}\right) \times 100$$

۲×۲	۴×۴	
٪۹۳,۷۵	٪۷۵	نرخ فشرده سازی

واضح است که در شکلی ۹۳ درصد فشرده سازی انجام شده است میزان جزئیات که مربوط به فرکانس بالا می‌باشد به شدت کاهش یافته است (در ماتریس ضرائب DCT فرکانس بالا در سمت راست و پایین قرار دارد که ما تمامی این ضرائب را ۰ کرده‌ایم) اما مشاهده می‌شود که برای با وجود این میزان فشرده سازی شدید همچنان شکل و کلیت آن به خوبی حفظ شده‌است. در حالتی که ۷۵ درصد فشرده سازی داریم با توجه به اینکه تعداد بیشتری از ضرائب فرکانس بالا را حفظ کرده‌ایم انتظار داریم نتیجه از حالت قبل بهتر باشد که در تصاویر فوق شاهد این نکته هستیم اما همچنان در نواحی از تصویر که تغییرات شدید داریم نظیر مژه‌ها جزئیات را از دست داده‌ایم.

در قسمت بعد از ما خواسته شده‌است که در گام اول ابتدا تمام تصاویر را نرمالیزه کنیم به گونه ای که نرم هر کدام از تصاویر ۱ شود. برای این کار ابتدا برای هر تصویر نرم۲ آن را حساب کرده سپس تصویر را بر نرم بدست آمده تقسیم میکنیم، تصویر نهایی بدست آمده دارای نرم برابر ۱ می‌باشد. این کار را باید برای هر ۳ تصویر انجام دهیم.

پس از نرمالیزه کردن تصاویر، برای محاسبه میزان rms خطا، تصاویر بدست را از تصویر مرجع کم کرده و از مقدار حاصل اخذ نرم۲ می‌کنیم.

نتیجه حاصل را در زیر شاهد هستیم:

۲×۲	۴×۴	
٪۲٫۵۸	٪۱٫۷۹	rms خطا

این میزان برای حالتی است که نقاط zeropad شده را پس از بازسازی تصویر حذف نکرده باشیم. در این حالت با توجه به اینکه باید در سطرهای اخر تنها شاهد ۰ باشیم اما چون ما تنها از مقداری از اطلاعات DCT استفاده کرده‌ایم پس از تبدیل معکوس گیری این نقاط مقادیر غیر صفر اتخاذ میکنند که این امر باعث افزایش میزان نرم خطا می‌گردد (ما ۵ سطر ۰ به انتهای تصویر اضافه کرده بودیم اما در تصویر بدست آمده بعد از فشرده سازی این سطر ها مخالف ۰ هستند که در زیر چند نمونه از این سطور را شاهد هستیم)

F obtained from preserving 2×2

0.1885 0.1885 0.1885 0.1885 0.1885 0.1885 0.1885 0.1885 0.1881 0.1881

0.0997 0.0997 0.0997 0.0997 0.0997 0.0997 0.0997 0.0997 0.0996 0.0995

0.0177 0.0177 0.0177 0.0177 0.0177 0.0177 0.0177 0.0177 0.0177 0.0177

-0.0450 -0.0450 -0.0450 -0.0450 -0.0450 -0.0450 -0.0450 -0.0450 -0.0449 -0.0449

F obtained from preserving 4×4

0.1249	0.1249	0.1249	0.1249	0.1249	0.1249	0.1249	0.1249	0.1240	0.1247
-0.0007	-0.0007	-0.0007	-0.0007	-0.0007	-0.0007	-0.0007	-0.0007	-0.0007	-0.0006
-0.0487	-0.0487	-0.0487	-0.0487	-0.0487	-0.0487	-0.0487	-0.0487	-0.0483	-0.0486
-0.0175	-0.0175	-0.0175	-0.0175	-0.0175	-0.0175	-0.0175	-0.0175	-0.0174	-0.0175

مشاهده می‌شود میزان تفاوت از ۰ برای حالتی که تنها ۲×۲ المان را نگه داشته ایم بیشتر از حالت دیگر است که این امر موجب افزایش خطا بیشتر در این حالت می‌گردد.

حال برای بهبود نتایج ۵ سطری که در ابتدا به انتهای تصویر اضافه کرده‌بودیم را حذف می‌کنیم. در زیر نتایج را برای حالتی که که ۵ سطر پایینی را حذف کرده‌ایم شاهد هستیم:

	۴×۴	۲×۲
rms خطا بعد از حذف سطرهای اضافی	٪۱٫۱۸	٪۱٫۲۳

مشاهده می‌شود همان‌طور که انتظار داشتیم در این حالت میزان خطا نسبت به حالت قبل کمتر شده‌است. و میزان اختلاف بین دو روش نیز با توجه به اینکه بیشتر خطا مربوط به ۵ سطر اخر بود به طور قابل توجهی کاهش یافته است.

در زیر شاهد کدهای مربوط به این قسمت هستیم:

```
clc;
clear;
close all;

%%%%%%%%%%%%%%%%%%
f= imread('baby2.jpg');
f= rgb2gray(f);
f = im2double(f);
[M,N] = size(f);

figure, imshow(f,[]),title('Original Images');

%%% zeropad
a_r = mod(M,8); a_c = mod(N,8);
if a_r ~= 0
   a_r = 8 - a_r;
end
if a_c ~= 0
    a_c = 8 - a_c;
end
temp = zeros(M+  a_r , N+a_c);
temp(1:M,1:N) = f;
ff= temp;
[M1,N1] = size(ff);


%%%%%%
f_4 = zeros(M1,N1);     f_2 = zeros(M1,N1);

for m=1:8:M1
    for n=1:8:N1
        DCT_4 = zeros(8,8); DCT_2 = zeros(8,8);
```

```
        DCT_temp = dct2(ff(m:m+7,n:n+7));
        DCT_4(1:4,1:4) =  DCT_temp(1:4,1:4); %%%% preserve only 4*4 left up
        DCT_2(1:2,1:2) =  DCT_temp(1:2,1:2); %%%% preserve only 2*2 left up

        f_4(m:m+7,n:n+7) = idct2(DCT_4);     %%%% reconstruct f
        f_2(m:m+7,n:n+7) = idct2(DCT_2);
    end
end

%%%%%% calulation normlized matrix  with padding
norm_ff = ff./norm(ff);
norm_f_2_pad = f_2./norm(f_2);
norm_f_4_pad  = f_4./norm(f_4);
%%%%%% caculation error rate
e_2_pad = norm(norm_ff - norm_f_2_pad )*100;
e_4_pad = norm(norm_ff - norm_f_4_pad )*100;

%%%%%% getting rid of padding
f_4 = f_4(1:M,1:N);
f_2 = f_2(1:M,1:N);
%%%%%% calulation normlized matrix  without padding
norm_f = f./norm(f);
norm_f_2 = f_2./norm(f_2);
norm_f_4 = f_4./norm(f_4);
%%%%%% caculation error rate
e_2 = norm(norm_f - norm_f_2)*100;
e_4 = norm(norm_f - norm_f_4)*100;


%%%%%% calulation compression rate
compression_rate_2 = 100*(1-(2^2/8^2));
compression_rate_4 = 100*(1-(4^2/8^2));
%%%%%% plotting Images
figure, imshow(f_4,[]), title(['using only 4*4 left up of DCT coefficients ,
compression rate is ',num2str(compression_rate_4),' %']);
figure, imshow(f_2,[]), title(['using only 2*2 left up of DCT coefficients,
compression rate is ',num2str(compression_rate_2),' %']);
```