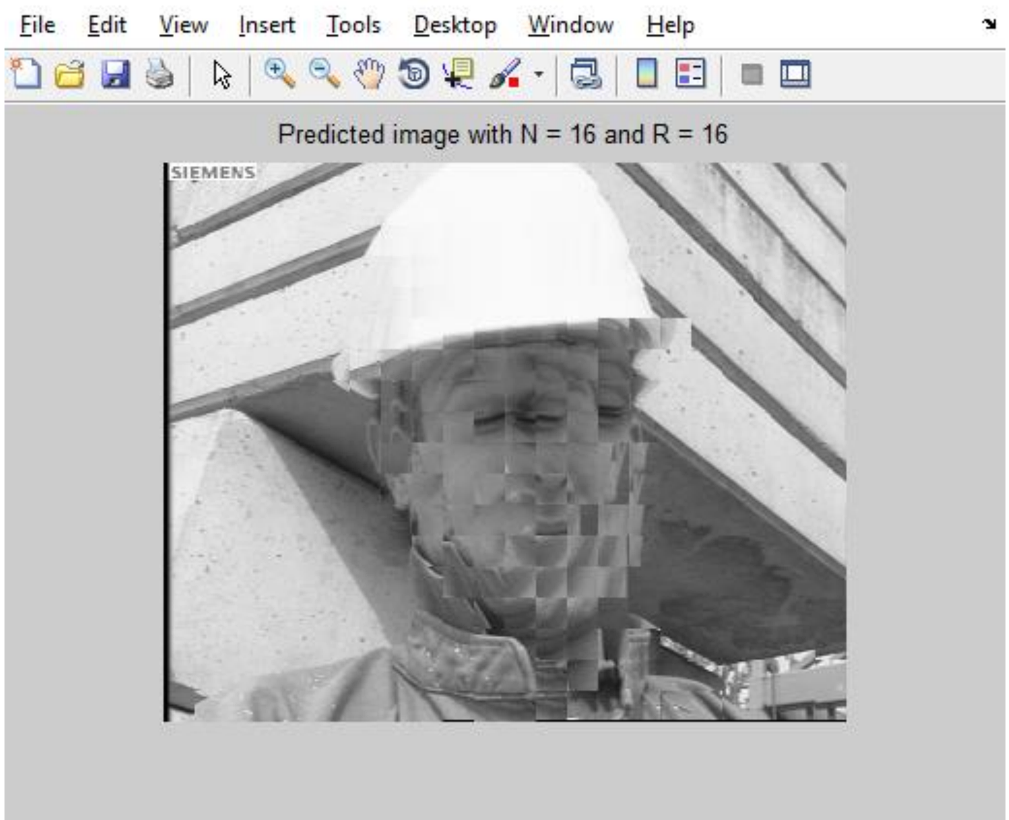
ECE508 Project 2

Video Communications

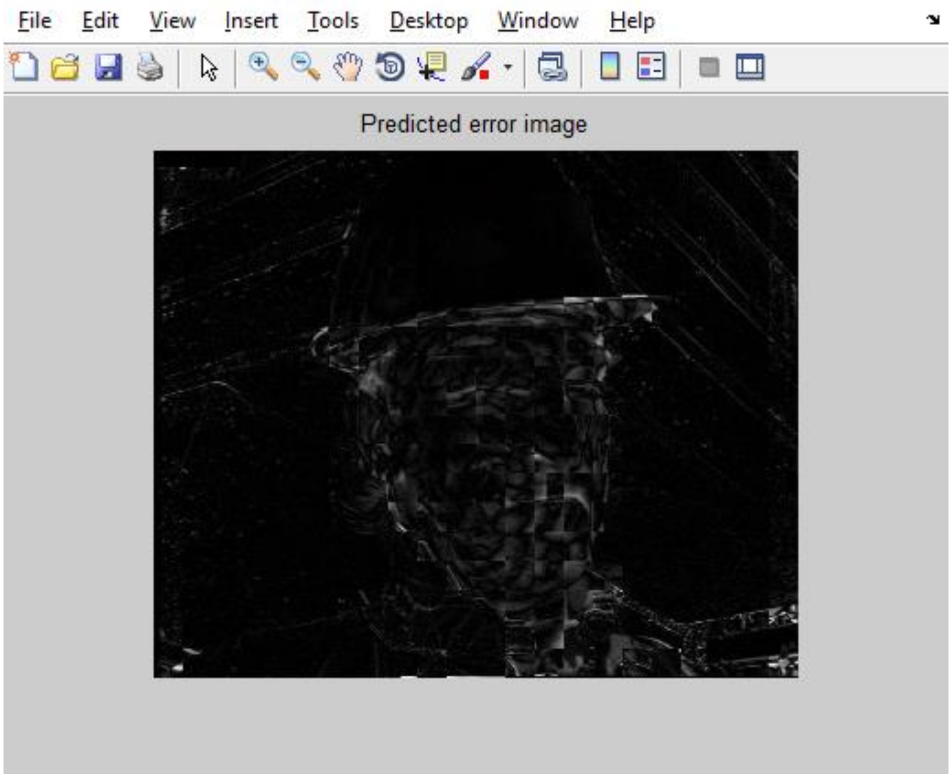
Name: **Krishna Pramod Kanakapura Umesh**

Student ID: **A20337195**

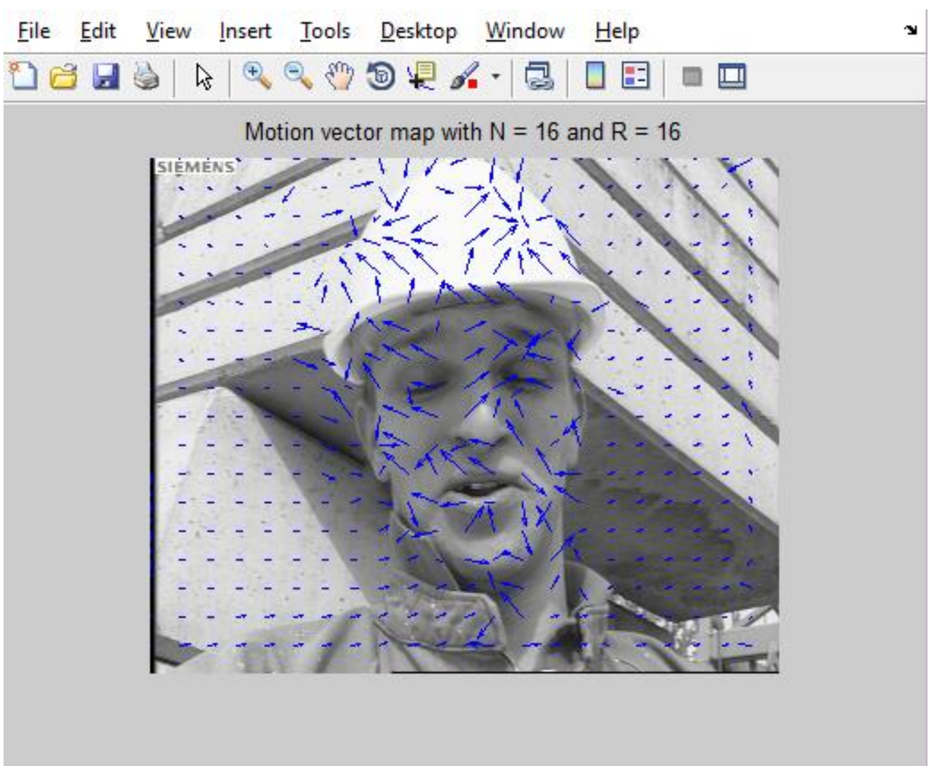
1. For Integer-Pel EBMA:
   a) Predicted Image:

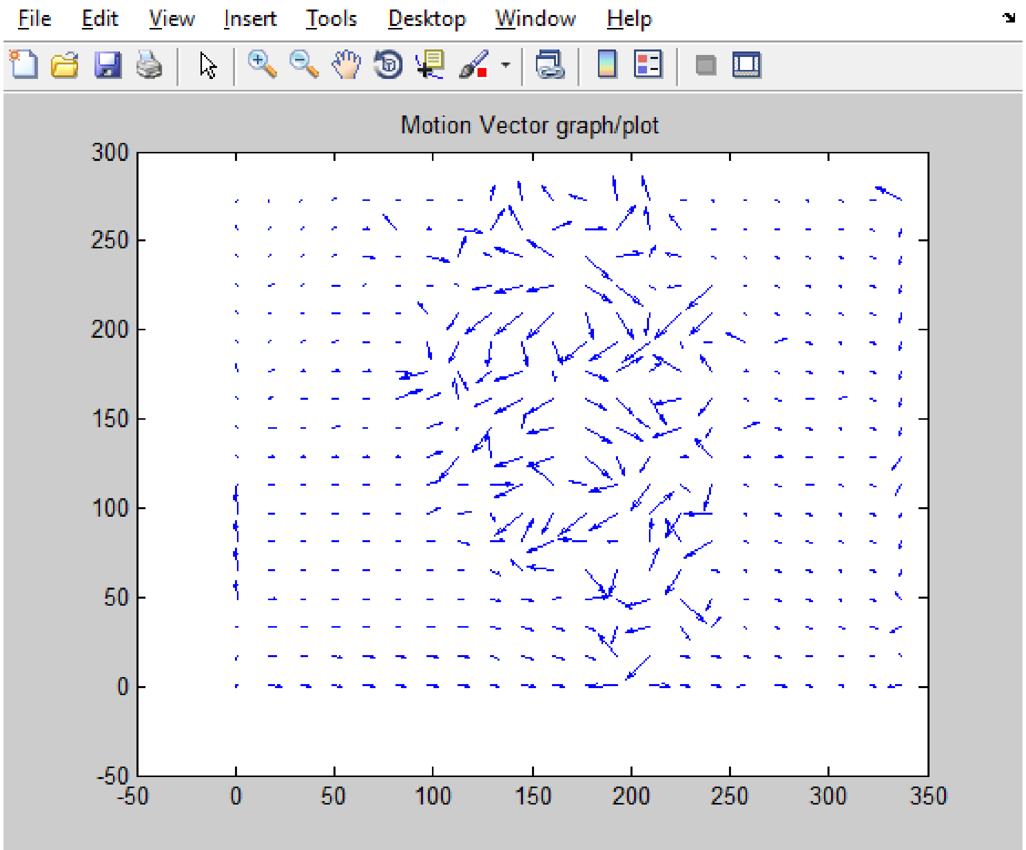


Prediction Error Image:



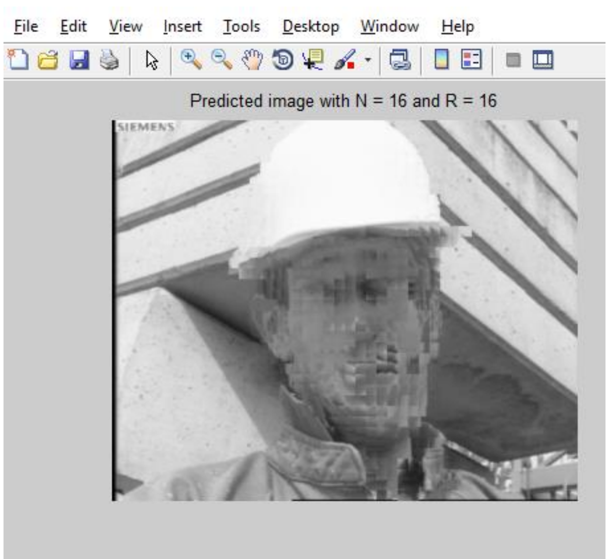
b) The estimated motion field is as shown in figure below:

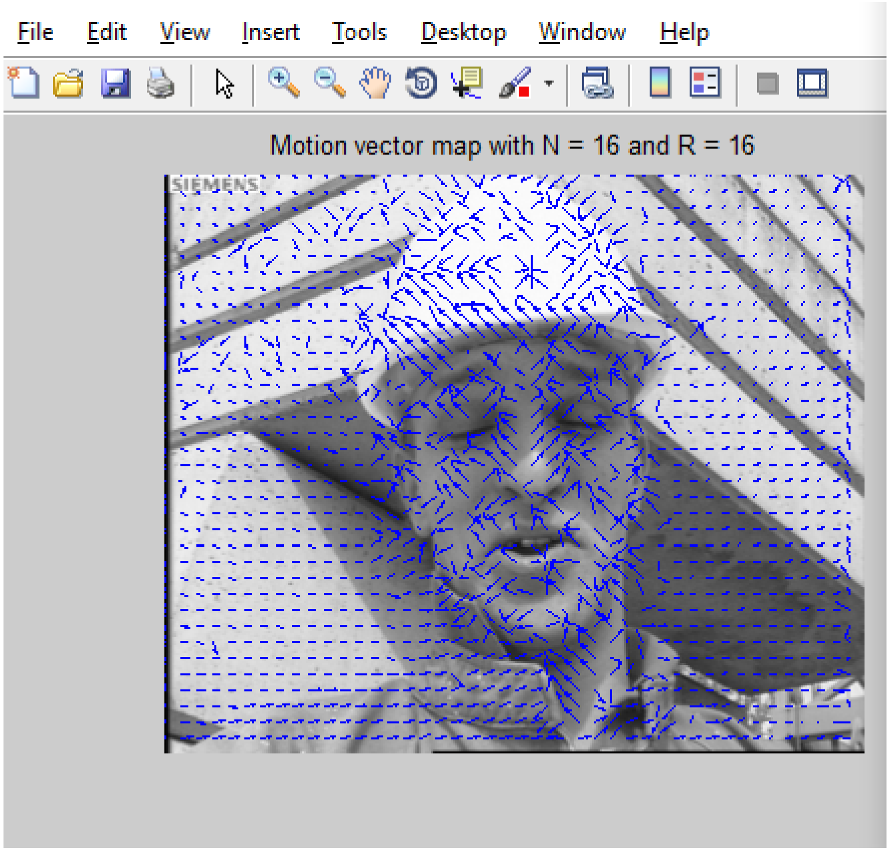


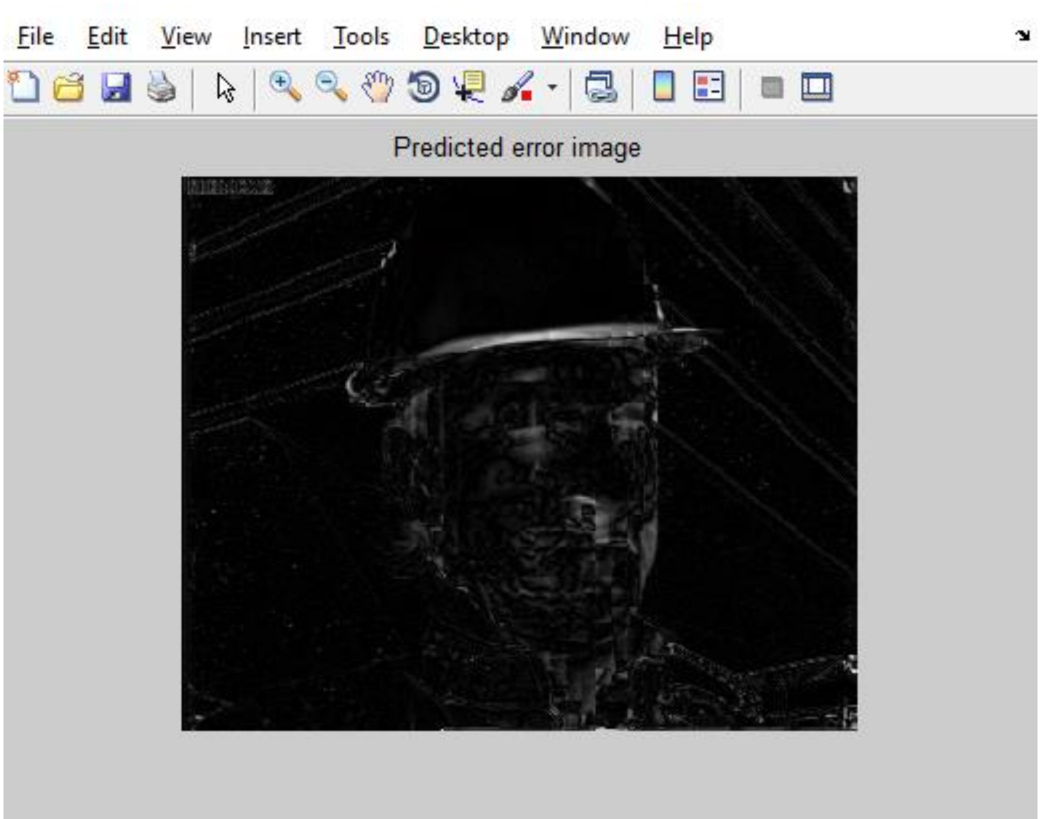


c) PSNR of the Predicted Frame = 27.5221 dB.

2. The PSNR of the predicted frame in Half-pel is 27.8215 dB.




File Edit View Insert Tools Desktop Window Help
Motion vector map with N = 16 and R = 16
SIEMENS


File Edit View Insert Tools Desktop Window Help
Predicted error image

Half-pel EBMA yields more accurate motion field and prediction, but it takes more computation time compared to Integer-Pel. Hence, in applications where time is a constraint, Integer-Pel can be used and Half-pel method can be used for EBMA where accuracy is important.

3. In case of dense motion field, the prediction accuracy reduces since the PSNR= 24.7609dB, but the computation time is faster when compared with both the Integer-Pel and Half-Pel.

Source Code:

```
function IntegerPel_EBMA(N, R, Fract, pix_flag)
%   For Integer Pel Fract = 1
%   N - Block size of the image
%   R - Search Range from which the user can select a particular range
%   Fract - Fractional Pel size
%   pix_flag - Interpolated Pixel based dense motion estimation flag.

clc;
%close all;
%clear all;
if isequal(Fract,1)
    pix = pix_flag;
else
    pix = 0;
    if pix_flag
        warning('Pixel-based dense motion estimation omitted')
    end
end

W=352; % Width of the given frame
H=288; % Height of the given frame

f2_org=int16(fread(fopen('foreman66.Y','r'),[W,H])');
f1_org=int16(fread(fopen('foreman72.Y','r'),[W,H])');
figure(1);
imshow((f2_org),[]); % Display the original image
title('Anchor Image')
figure(2);
imshow((f1_org),[]); % Display the target image
title('Target Image')
t = 0;
if Fract >1
    tic
    f1 = imresize(f1_org, Fract,'bilinear');
    f2 = imresize(f2_org, Fract,'bilinear');
    t = toc;
else
    f1 = f1_org;
```

```
    f2 = f2_org;
end
f2_extnd = wextend(2,'zpd',f2,R);

F1 = nonoverlap(f1, N);

F2 = overlap(f2_extnd, N);
dx = struct('val',[],'idx',[]);
dy = struct('val',[],'idx',[]);

%% EBMA Algorithm given in class lecture
d = -R:R;
tic
for m = 1:size(f1,1)/N
    for n = 1:size(f1,2)/N
        MAD_min = 256*N*N;
        search_lmt = 2*R ;
        for k = 1:search_lmt
            for l = 1:search_lmt
                MAD = sum(sum(abs(F1{m,n} - F2{(m-1)*N +k +1, (n-1)*N +l
+1})));
                if MAD<MAD_min
                   MAD_min = MAD;
                   dx.val = d(l);
                   dy.val = d(k);
                   dx.idx = l;
                   dy.idx = k;
                end
            end
        end
        FP{m,n} = F2{(m-1)*N +1 +dy.idx, (n-1)*N +1 +dx.idx};
        mvx{m,n} = int16(dx.val);
        mvy{m,n} = dy.val;
    end
end
t = toc +t;
fp = cell2mat(FP);
if Fract > 1
    disp(['Time taken by fractional-pel motion estimation = ' num2str(t)])
    fp = imresize(fp,1/Fract,'bilinear');
else
    disp(['Time taken by integer-pel motion estimation = ' num2str(t)])
end

figure(3);
imshow(f2,[]);
hold
[x1,x2] = meshgrid(1:N:size(f2,2), 1:N:size(f2,1));
quiver(x1,x2, cell2mat(mvx),cell2mat(mvy));
title(['Motion vector map with N = ' num2str(N) ' and R = ' num2str(R)])

% Predicted Image
figure(4);
imshow((fp),[]);
title(['Predicted image with N = ' num2str(N) ' and R = ' num2str(R)])
```

```matlab
% Motion Vector Graph/Plot
figure(5);
quiver(x1,x2,flipud(cell2mat(mvx)),flipud(cell2mat(mvy)));
title('Motion Vector graph/plot')

% Predicted Error Image
figure(6);
imshow((abs(f1_org-fp)),[]);
title('Predicted error image')

% PSNR calculation
PSNR = 10*log10(255*255/mean(mean((abs(f1_org-fp)).^2)));
disp(['The PSNR of the predicted frame = ' num2str(PSNR)])

if pix && (N >1)
    tic
    MVX = bilinear_func(int16(cell2mat(mvx)), N);
    MVY = bilinear_func(int16(cell2mat(mvy)), N);

    for m = 1:size(f1,1)
        for n = 1:size(f1,2)
            dx.val = MVX(m,n);
            dy.val = MVY(m,n);
            dx.idx = find(dx.val==d);
            dy.idx = find(dy.val==d);
            FP_pix{m,n} = f2_extnd((m-1) +1 +dy.idx, (n-1) +1 +dx.idx);
        end
    end
    fp_pix = cell2mat(FP_pix);
    t_p = toc;
    disp(['Time taken by interpolated pixel-based dense motion estimation = '
num2str(t_p)])
    figure(7);
    imshow(f2,[]);
    hold
    [x1,x2] = meshgrid(1:size(f2,2), 1:size(f2,1));
    quiver(x1,x2, (MVX),(MVY));
    title('Dense Pixel-based motion vector map')
    figure(8);
    imshow((fp_pix),[]);
    title('Dense Pixel-based predicted image')
    figure(9);
    imshow((abs(f1_org-fp_pix)),[]);
    title('Predicted error image')

    % PSNR calculation
    PSNR = 10*log10(255*255/mean(mean((abs(f1_org-fp_pix)).^2)));
    disp(['The PSNR of the pixel based predicted frame = ' num2str(PSNR)])

end
end

function F = nonoverlap(f, N)
%% Creating non overlaping NxN blocks
```

```
row_s = 1;
row_e = N;

blk_r = 1;
while (row_e <= size(f,1))
    col_s = 1;
    col_e = N;
    blk_c = 1;
    while (col_e <= size(f,2))
        F{blk_r, blk_c} = f(row_s:row_e, col_s:col_e);
        col_s = col_s +N;
        col_e = col_e +N;
        blk_c = blk_c +1;
    end
    row_s = row_s +N;
    row_e = row_e +N;
    blk_r = blk_r +1;
end
end

function F = overlap(f, N)
%% Creating overlaping RxR search region blocks

pel_inc = 1;
row_s = 1;
row_e = N;

blk_r = 1;
while (row_e <= size(f,1))
    col_s = 1;
    col_e = N;
    blk_c = 1;
    while (col_e <= size(f,2))
        F{blk_r, blk_c} = f(row_s:row_e, col_s:col_e);
        col_s = col_s +pel_inc;
        col_e = col_e +pel_inc;
        blk_c = blk_c +1;
    end
    row_s = row_s +pel_inc;
    row_e = row_e +pel_inc;
    blk_r = blk_r +1;
end
end

function Y = bilinear_func(X, N)
%% Bilinear Interpolation function

X = wextend('2','sym',X,1);
Y_temp = interp(X', N);

Y_temp = interp(Y_temp', N);

Y = Y_temp(N/2 +1:size(Y_temp,1) -N/2, N/2 +1:size(Y_temp,2) -N/2);
end
```

```
function Y = interp(X, N)
for m = 1:size(X,1)

    for n = 1:N:(size(X,2)*N -N)
        idx = floor((n-1)/N)+1;
        Y(m,n:(n+N -1)) = int16(double(X(m, idx +1) - X(m, idx))*((0:N -
1)/(N))) + X(m, idx);
    end
end
end
```

For Integer- Pel without dense motion:

```
IntegerPel_EBMA(16,16,1,0)
```

With Dense motion:
Pix_flag = 1

For Half-Pel without dense:

```
function HalfPel_EBMA(N, R, Fract, pix_flag)
%   For Half Pel Fract = 2
%   N - Block size of the image
%   R - Search Range from which the user can select a particular range
%   Fract - Fractional Pel size
%   pix_flag - Interpolated Pixel based dense motion estimation flag.

clc;
%close all;
%clear all;

if isequal(Fract,2)
    pix = pix_flag;
else
    pix = 0;
    if pix_flag
        warning('Pixel-based dense motion estimation omitted')
    end
end

W=352; % Width of the given frame
H=288; % Height of the given frame

f2_org=int16(fread(fopen('foreman66.Y','r'),[W,H])');
f1_org=int16(fread(fopen('foreman72.Y','r'),[W,H])');
figure(1);
imshow((f2_org),[]); % Display the original image
title('Anchor Image')
figure(2);
imshow((f1_org),[]); % Display the target image
title('Target Image')
t = 0;
if Fract >1
```

```
    tic
    f1 = imresize(f1_org, Fract,'bilinear');
    f2 = imresize(f2_org, Fract,'bilinear');
    t = toc;
else
    f1 = f1_org;
    f2 = f2_org;
end
f2_extnd = wextend(2,'zpd',f2,R);

F1 = nonoverlap(f1, N);

F2 = overlap(f2_extnd, N);
dx = struct('val',[],'idx',[]);
dy = struct('val',[],'idx',[]);

%% EBMA Algorithm given in class lecture
d = -R:R;
tic
for m = 1:size(f1,1)/N
    for n = 1:size(f1,2)/N
        MAD_min = 256*N*N;
        search_lmt = 2*R ;
        for k = 1:search_lmt
            for l = 1:search_lmt
                MAD = sum(sum(abs(F1{m,n} - F2{(m-1)*N +k +1, (n-1)*N +l
+1})));
                if MAD<MAD_min
                   MAD_min = MAD;
                   dx.val = d(l);
                   dy.val = d(k);
                   dx.idx = l;
                   dy.idx = k;
                end
            end
        end
        FP{m,n} = F2{(m-1)*N +1 +dy.idx, (n-1)*N +1 +dx.idx};
        mvx{m,n} = int16(dx.val);
        mvy{m,n} = dy.val;
    end
end
t = toc +t;
fp = cell2mat(FP);
if Fract > 1
    disp(['Time taken by fractional-pel motion estimation = ' num2str(t)])
    fp = imresize(fp,1/Fract,'bilinear');
else
    disp(['Time taken by integer-pel motion estimation = ' num2str(t)])
end

figure(3);
imshow(f2,[]);
hold
[x1,x2] = meshgrid(1:N:size(f2,2), 1:N:size(f2,1));
quiver(x1,x2, cell2mat(mvx),cell2mat(mvy));
title(['Motion vector map with N = ' num2str(N) ' and R = ' num2str(R)])
```

```
% Predicted Image
figure(4);
imshow((fp),[]);
title(['Predicted image with N = ' num2str(N) ' and R = ' num2str(R)])

% Motion Vector Graph/Plot
figure(5);
quiver(x1,x2,flipud(cell2mat(mvx)),flipud(cell2mat(mvy)));
title('Motion Vector graph/plot')

% Predicted Error Image
figure(6);
imshow((abs(f1_org-fp)),[]);
title('Predicted error image')

% PSNR calculation
PSNR = 10*log10(255*255/mean(mean((abs(f1_org-fp)).^2)));
disp(['The PSNR of the predicted frame = ' num2str(PSNR)])

if pix && (N >1)
    tic
    MVX = bilinear_func(int16(cell2mat(mvx)), N);
    MVY = bilinear_func(int16(cell2mat(mvy)), N);

    for m = 1:size(f1,1)
        for n = 1:size(f1,2)
            dx.val = MVX(m,n);
            dy.val = MVY(m,n);
            dx.idx = find(dx.val==d);
            dy.idx = find(dy.val==d);
            FP_pix{m,n} = f2_extnd((m-1) +1 +dy.idx, (n-1) +1 +dx.idx);
        end
    end
    fp_pix = cell2mat(FP_pix);
    t_p = toc;
    disp(['Time taken by interpolated pixel-based dense motion estimation = '
num2str(t_p)])
    figure(7);
    imshow(f2,[]);
    hold
    [x1,x2] = meshgrid(1:size(f2,2), 1:size(f2,1));
    quiver(x1,x2, (MVX),(MVY));
    title('Dense Pixel-based motion vector map')
    figure(8);
    imshow((fp_pix),[]);
    title('Dense Pixel-based predicted image')
    figure(9);
    %imshow(abs(f1_org-fp_pix),[]);
    title('Predicted error image')

    % PSNR calculation
    PSNR = 10*log10(255*255/mean(mean((abs(f1_org-fp_pix)).^2)));
    disp(['The PSNR of the pixel based predicted frame = ' num2str(PSNR)])
```

```
end
end

function F = nonoverlap(f, N)
%% Creating non overlaping NxN blocks

row_s = 1;
row_e = N;

blk_r = 1;
while (row_e <= size(f,1))
    col_s = 1;
    col_e = N;
    blk_c = 1;
    while (col_e <= size(f,2))
        F{blk_r, blk_c} = f(row_s:row_e, col_s:col_e);
        col_s = col_s +N;
        col_e = col_e +N;
        blk_c = blk_c +1;
    end
    row_s = row_s +N;
    row_e = row_e +N;
    blk_r = blk_r +1;
end
end

function F = overlap(f, N)
%% Creating overlaping RxR search region blocks

pel_inc = 1;
row_s = 1;
row_e = N;

blk_r = 1;
while (row_e <= size(f,1))
    col_s = 1;
    col_e = N;
    blk_c = 1;
    while (col_e <= size(f,2))
        F{blk_r, blk_c} = f(row_s:row_e, col_s:col_e);
        col_s = col_s +pel_inc;
        col_e = col_e +pel_inc;
        blk_c = blk_c +1;
    end
    row_s = row_s +pel_inc;
    row_e = row_e +pel_inc;
    blk_r = blk_r +1;
end
end

function Y = bilinear_func(X, N)
%% Bilinear Interpolation function

X = wextend('2','sym',X,1);
Y_temp = interp(X', N);
```

```
Y_temp = interp(Y_temp', N);

Y = Y_temp(N/2 +1:size(Y_temp,1) -N/2, N/2 +1:size(Y_temp,2) -N/2);
end

function Y = interp(X, N)
for m = 1:size(X,1)

    for n = 1:N:(size(X,2)*N -N)
        idx = floor((n-1)/N)+1;
        Y(m,n:(n+N -1)) = int16(double(X(m, idx +1) - X(m, idx))*((0:N -
1)/(N))) + X(m, idx);
    end
end
end
```

Without dense motion

```
HalfPel_EBMA(16,16,2,0)

With dense motion
HalfPel_EBMA(16,16,2,1)
```