PG58: Tan Guan Yuan - 31062032
Nur Atiqah Safar - 30148839

Motion Compensation in Video coding

Introduction

Motion compensation (MC) has brought ease to video compression as it saves a lot of transmitted bits by predicting new video frames as accurately as possible based on previously transmitted or future frames (forward and backward prediction). There are a lot of redundancies inside digitized video such as statistical redundancy and subjective redundancy [1]. MC exploits the fact of this redundancy nature and just sends only the difference (the “prediction error”) between the compensated image and the actual new frame pixel data (as shown in Part 5) along with the motion vector (as shown in Part 6). An overview of the process is shown in figure 1.1 and a simplified version is shown in figure 1.2.

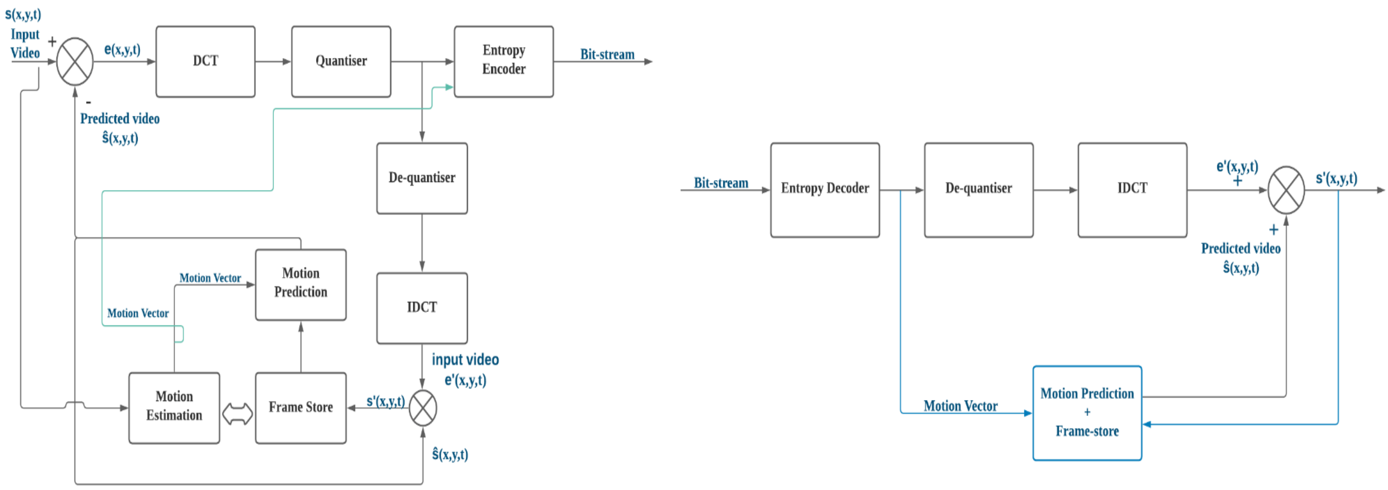


Figure 1.1 Overview of Video coding process flow. Left: Encoder, Right: Decoder.

Block Motion Compensation process

The idea is that there is a high correlation between each pixel and its neighbors and assigning a motion vector to a block of pixels is more useful than to an individual pixel. The process is to segment the current frame into *nxn* blocks, largest motion displacement of p pixels per frame, the current frame is matched with corresponding block in previous frame and finds the best suitable block matched and gives the displacement. Motion Equation that relates 3D motion to displacements is shown in *equation 1*. While representation of motion compensated prediction signal is shown in *equation 2*, and an illustrative example of the process is shown in *figure 1.3*.

Experiment

1. **Full Search/Exhaustive Search**

   Full search algorithm is the most computationally expensive among the other block matching algorithms. It calculates cost function at each possible location (total of (2P+1)*(2P+1) locations) within the search area and results in giving the best match. Has the highest PSNR amongst other block matching algorithms. An illustrative example is shown in *figure 1.4*.

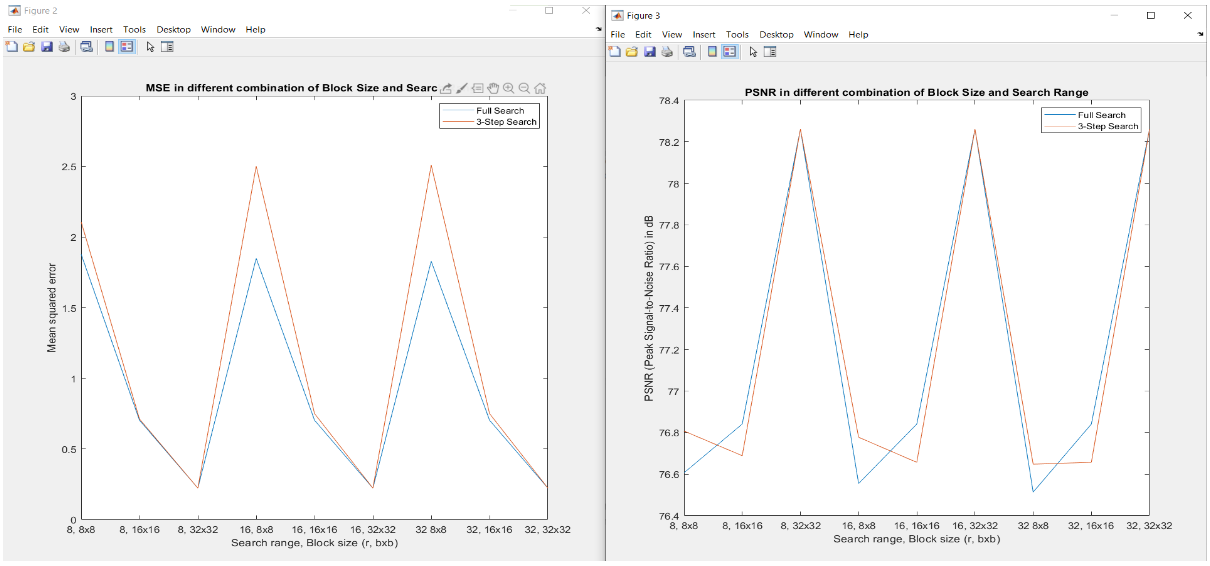
2. **Three-step search**

   First, initial step size is picked. P blocks at a distance of step size from centre (around centre block) are picked for comparison. From these searched points, pick the smallest cost and make it as the new origin. New step size is divided in half, and repeated until the step size is = 1, and so the block at that last position is the best match. An illustrative example is shown in *figure 1.5*.

Evaluation

Cost functions that are used are as follows: Mean Squared Error (MSE), Sum of Absolute Difference (SAD) and lastly PSNR, given by *equation (3), (4), (5)*.

Comparison between Full search method and Three step search method



Plot 1.1 MSE plot; Plot 1.2 PSNR plot

From the *plot 1.1* and *plot 1.2*, when the block size is highest (32x32), we obtain the highest PSNR and lowest MSE. While adjusting the search range only improves the performance by a little, the performance is best when the search range is highest, 32. SAD, MAD, PSNR (using SAD) is the highest expected when the block size is 32x32, as shown in *Plot 1.1.3* and *Plot 1.1.4* and *Plot 1.1.5*. As we are comparing the predicted image, using the largest block size, the predicted image is the most similar to the target image compared to the others using smaller block size, as shown in Part 4. We can also find out that using the Full Search method is far more computationally expensive than the 3-Step Search method no matter which block size or search range is chosen as shown in *plot 2.1*, *plot 3.1* and *plot 3.2.*

Conclusion

Although the Full Search algorithm gives us the best accuracy, when considering applying it in real-life video coding, it seems impractical as it is much more computationally expensive than the 3-Step Search algorithm which has similar performance with much lower time complexity. From the experiment, adjusting block size has more effect on performance than adjusting search range. In conclusion, according to the performance and efficiency, the best algorithm and parameters we can choose from is the 3-Step Search algorithm with a search range of 8 and block size of 32x32 with search range of 8.

References

[1] J.Feng, K.T.Lo."Motion Compensation for Video Compression," in Encyclopedia of Multimedia.

[2] T.Wiegand/B.Girod. "EE398A Image and Video Compression," https://web.stanford.edu/class/ee398a/handouts/lectures/EE398a_MotionEstimation_2012.pdf

[3] S.M.Kulkarni, D.S.Bormane, S.L.Nalbalwar "Coding of video Sequences using Three Step Search Algorithm" in Procedia Computer Science 49 (2015) 42 – 49

[4] "Motion Estimation Algorithms for Baseline Profile of H.264 Video Codec," in International Journal of Engineering Trends and Technology (IJETT) - Volume4Issue4- April 2013

Appendices

<u>Video compression process flow</u>

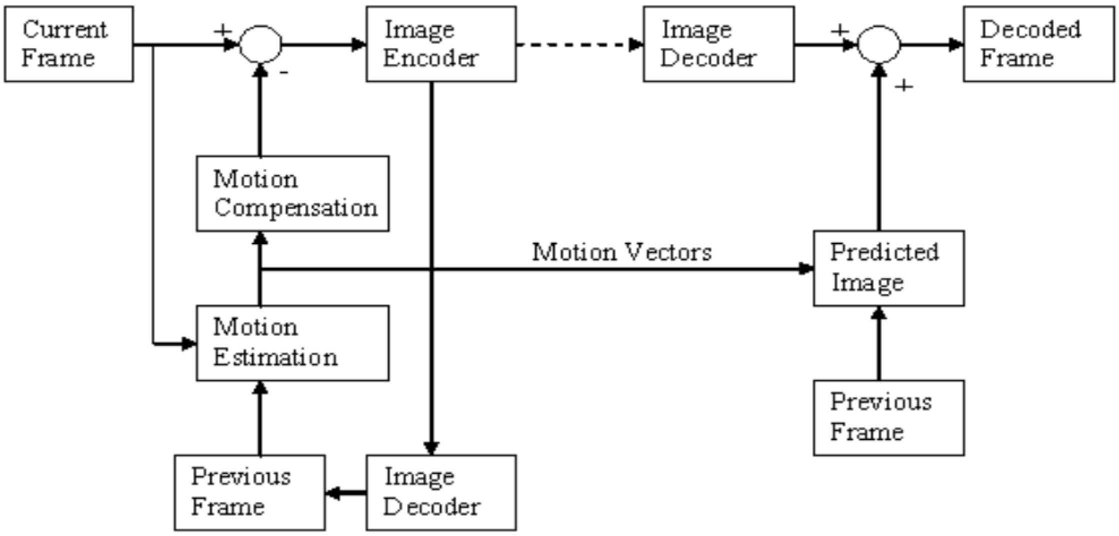


Figure 1.2 SImplified version of video compression process flow model [3]

<u>Block matching technique and search parameter</u>

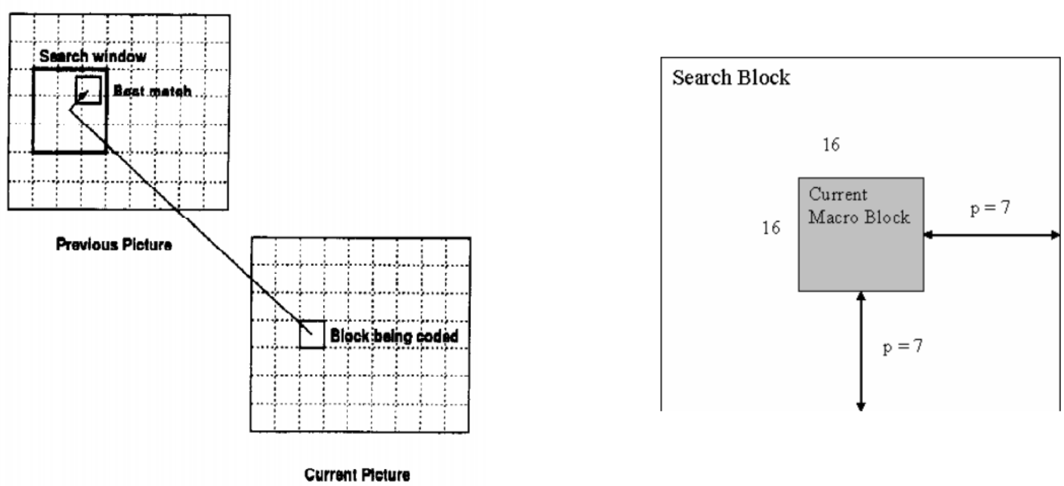


Figure 1.3 Block matching technique and search parameter [3]

Full search illustration

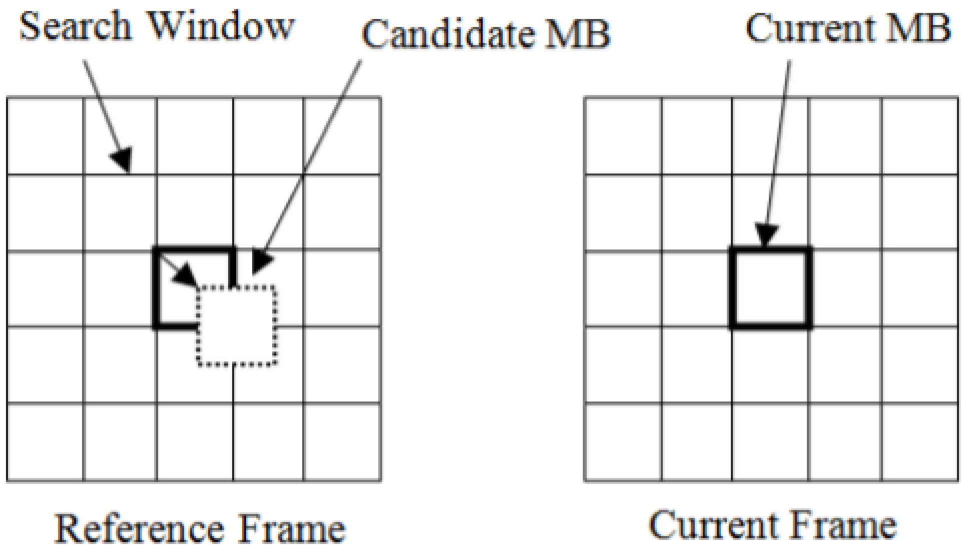


Figure 1.4 Full search method illustration [3]

Three step search Algorithm

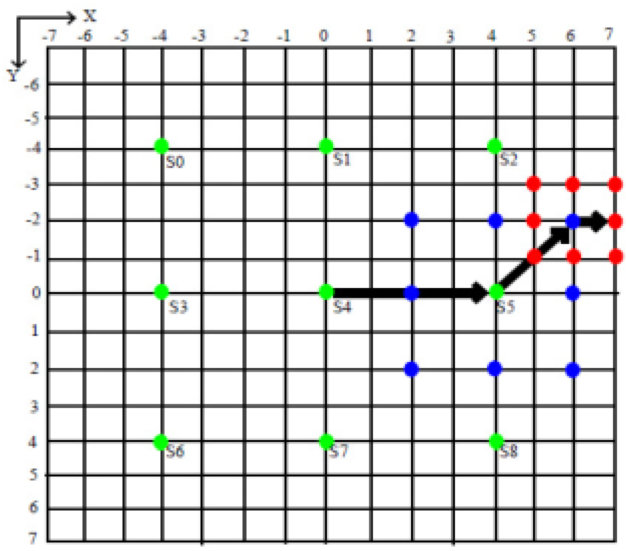


Figure 1.5 Three step search method illustration [3]

Motion Equation

$dx = x' - x;\ dy = y' - y$

$while\ \ x, y: location\ in\ previous\ image;\ x', y' = location\ in\ current\ image;\ dx. dy = displacement$

Equation 1: Motion Equation

Motion compensated prediction signal

$\hat{s}[x, y, t] = s'(x - dx,\ y - dy,\ t - \Delta t)$

Equation 2: Motion compensated prediction signal

1. Evaluation of the search method used

Mean Squared error (MSE)

$$MSE(d_x, d_y) = \sum_{y=1}^{By} \sum_{x=1}^{Bx} [s(x, y, t) - s'(x - d_x, y - d_y, t - \Delta t)]^2$$

Equation3: MSE (Mean Squared Error)

Sum of Absolute difference (SAD)

$$SAD(d_x, d_y) = \sum_{y=1}^{By} \sum_{x=1}^{Bx} |s(x, y, t) - s'(x - d_x, y - d_y, t - \Delta t)|$$

Equation 4: Sum of Absolute Difference (SAD)

Peak Signal to Noise Ratio (PSNR)

$$PSNR = 10\ log(2^n - 1)^2 / MSE$$

Equation 5: PSNR (Peak Signal to Noise Ratio)

PSNR

Methods	Search range, Block size								
	8, 8x8	8, 16x16	8, 32x32	16, 8x8	16, 16x16	16, 32x32	32, 8x8	32, 16x16	32, 32x32
Full Search	73.43	73.36	73.91	73.46	73.36	73.91	73.52	73.36	73.39
3 Step Method	73.57	73.28	73.91	73.68	73.32	73.91	73.52	73.32	73.39

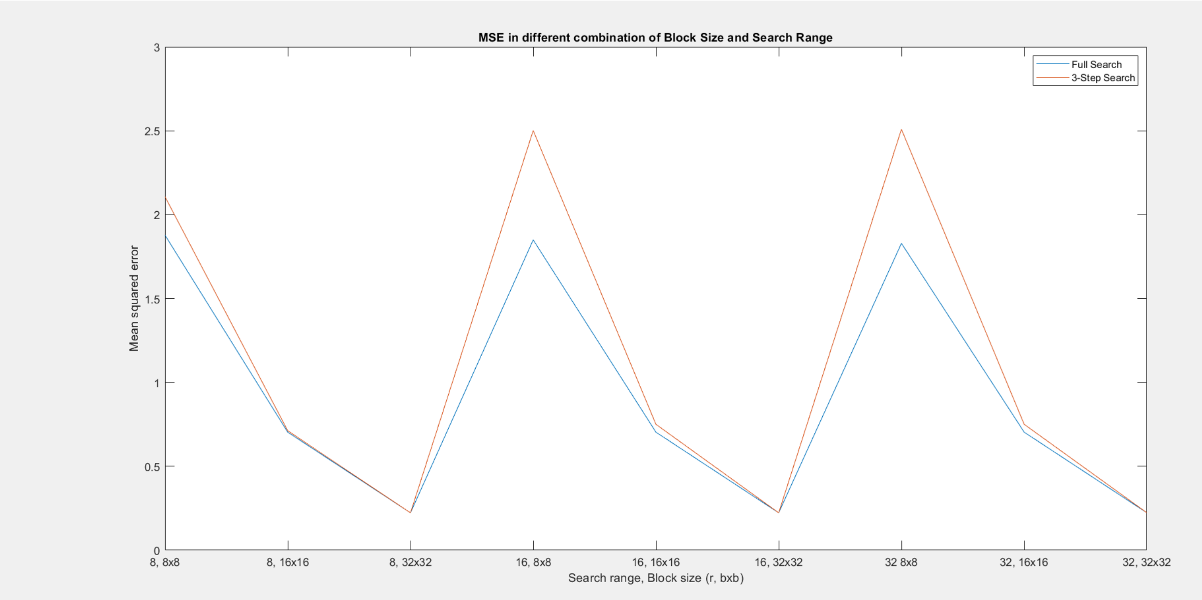
Table 1.1 PSNR result

MSE

Methods	Search range, Block size								
	8, 8x8	8, 16x16	8, 32x32	16, 8x8	16, 16x16	16, 32x32	32 8x8	32 16x16	32 32x32
Full Search	1.88	0.70	0.22	1.85	0.70	0.22	1.83	0.70	0.22
3 Step Method	2.11	0.71	0.22	2.50	0.75	0.22	2.51	0.75	0.23

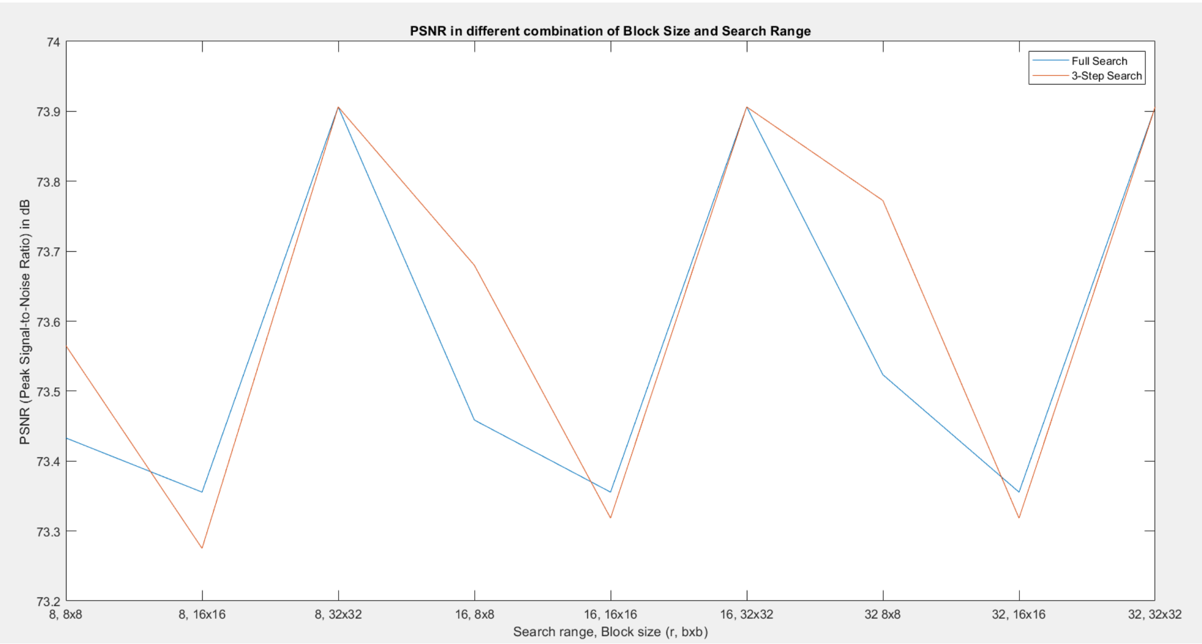
Table 1.2 MSE result

MSE Result



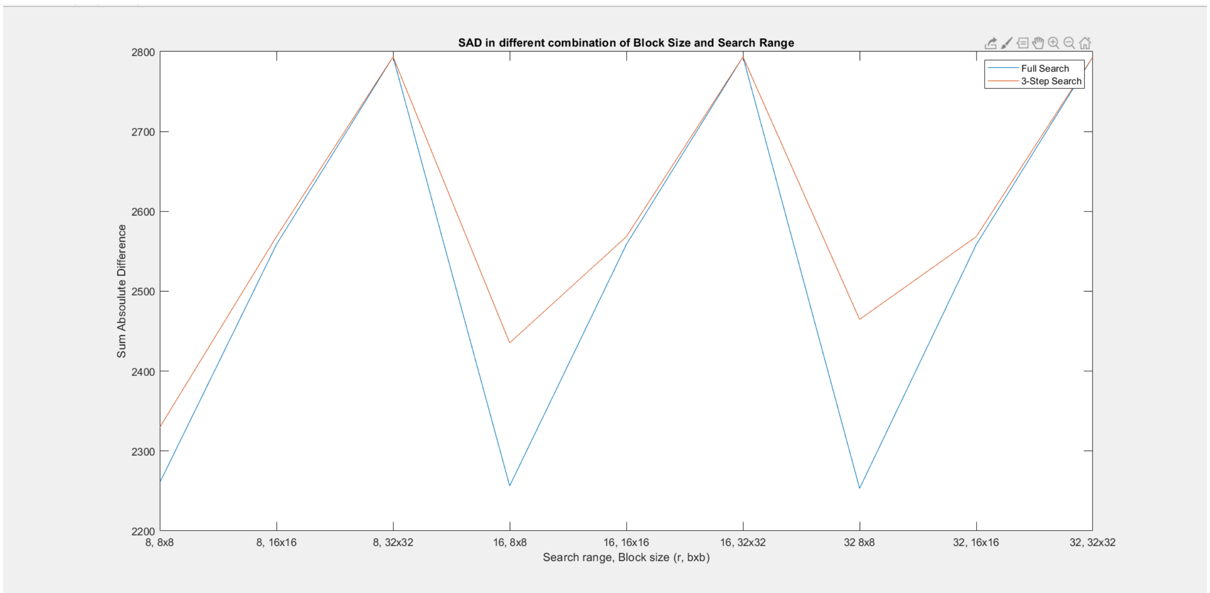
Plot1.1: MSE (Mean Squared Error) Result at different combinations of Block size and Search range

PSNR Result



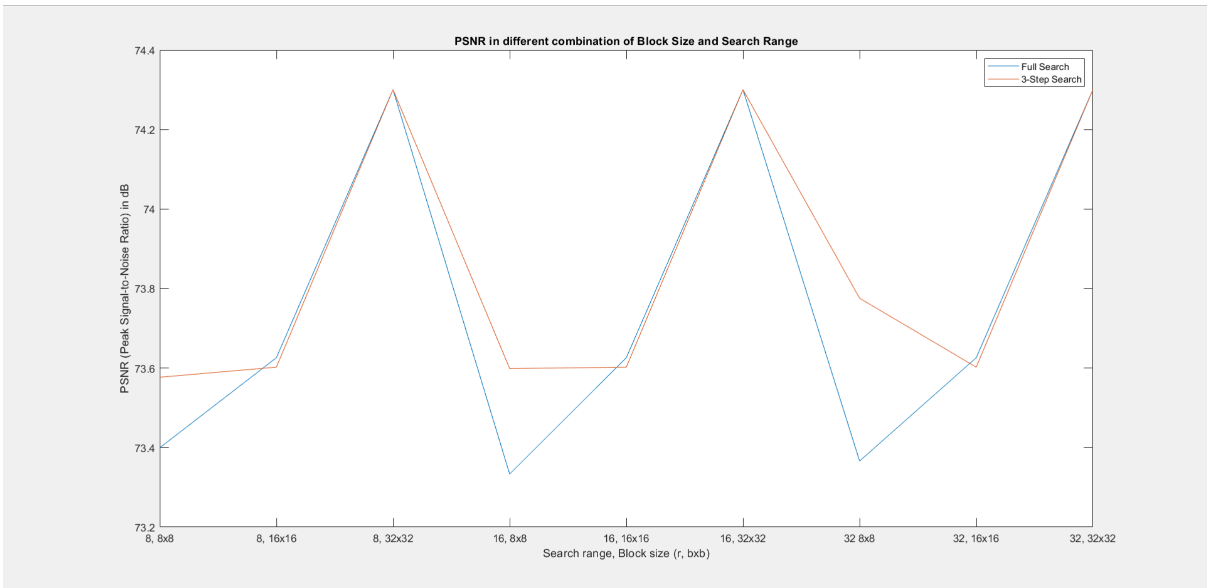
Plot 1.2: PSNR (Peak Signal Noise Ratio) Result at different combinations of Block size and Search range

SAD



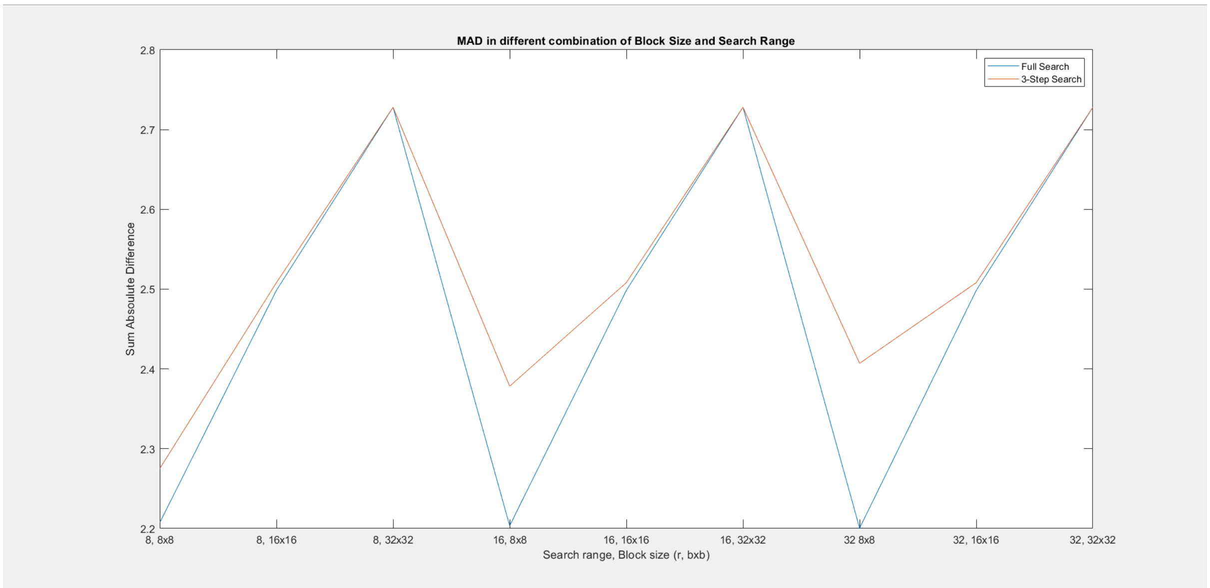
Plot1.1.3: SAD (Sum of Absolute Difference) Result at different combinations of Block size and Search range

PSNR using SAD



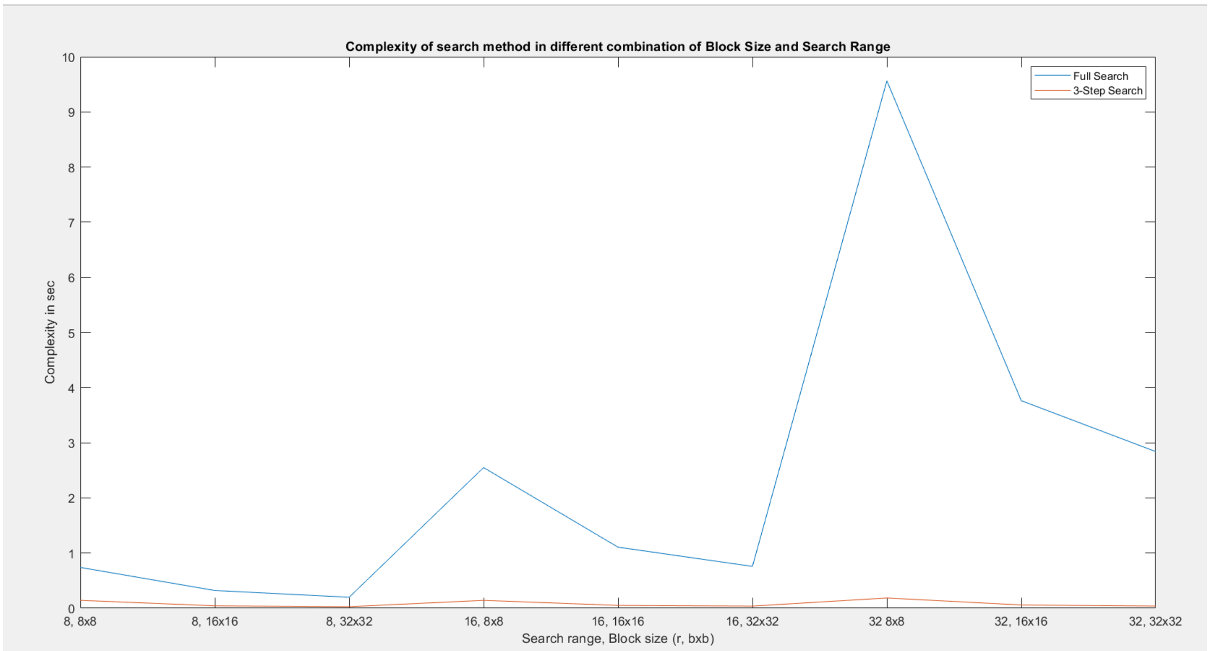
Plot1.1.4: PSNR result in SAD (Sum of Absolute Difference) as evaluation method at different combinations of Block size and Search range

MAD



Plot1.1.5: MAD (Mean of Absolute Difference) Result at different combinations of Block size and Search range

2. Complexity of Different Search Method



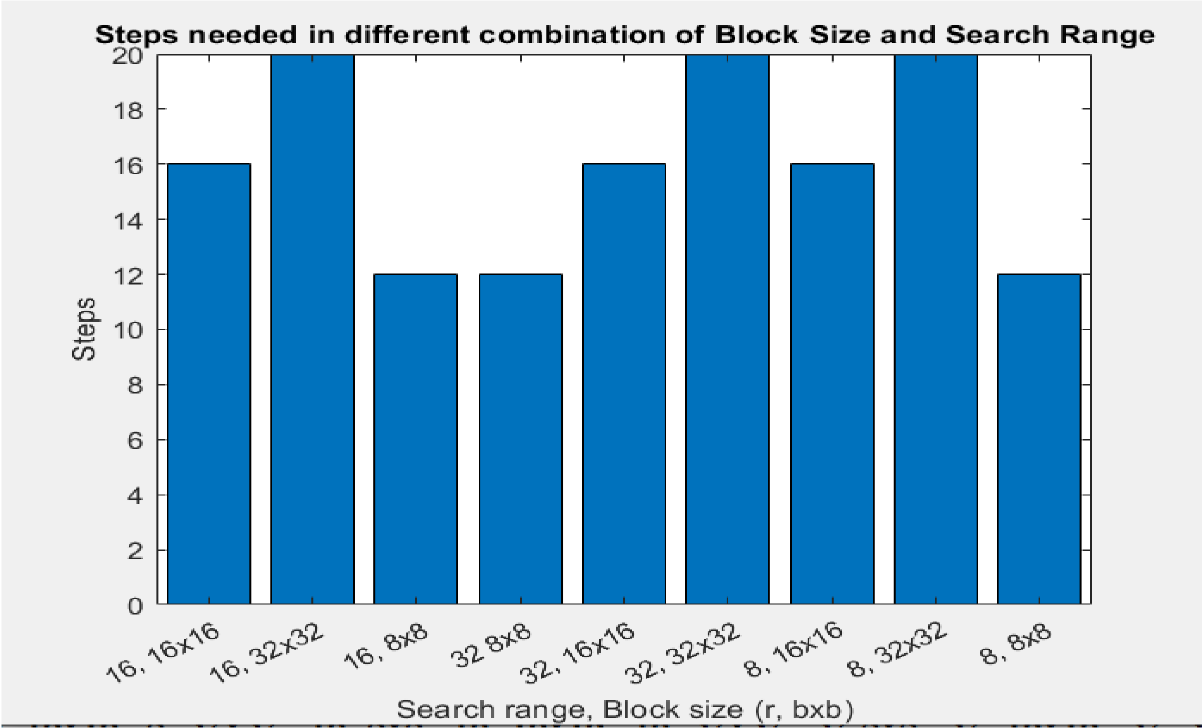
Plot2.1: Complexity result at different combinations of Block size and Search range

Time complexity for each method with different combination of search range and block size (sec)									
Methods	Search range, Block size (R, bxb)								
	8, 8x8	8, 16x16	8, 32x32	16, 8x8	16, 16x16	16, 32x32	32, 8x8	32, 16x16	32, 32x32
Full Search	1.02	0.39	0.23	2.74	1.25	0.71	9.63	4.03	2.70
3 Step Method	0.25	0.09	0.02	0.16	0.06	0.03	0.17	0.06	0.05

Table 2.1: Complexity result

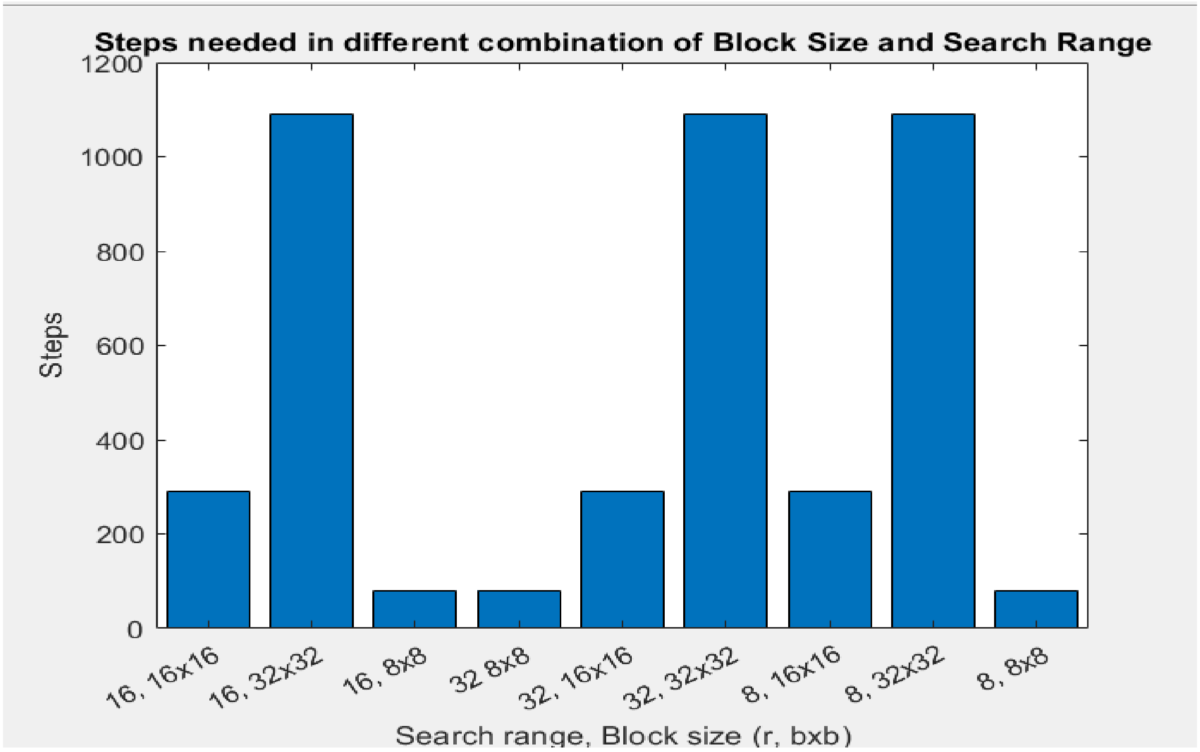
3. Steps Taken for each search method

1. 3-Step search



Plot3.1: Step needed at different combinations of Block size and Search range using 3-step search method.

2. Full search



Plot3.2: Step needed at different combinations of Block size and Search range using full search method.

Steps taken for each method with different combination of search range and block size									
Methods	Search range, Block size								
	8, 8x8	8, 16x16	8, 32x32	16, 8x8	16, 16x16	16, 32x32	32, 8x8	32, 16x16	32, 32x32
3 Step search	12	16	20	12	16	20	12	16	20
Full search	81	289	1089	81	289	1089	81	289	1089

Table 3.1: Steps needed at different combinations of Block size and Search range using 3-step search method and full search method.

4. Predicted Image using different search method

Reference Image	Target Image
	
Figure 4.1: Reference Image	*Figure 4.2: Target Image*

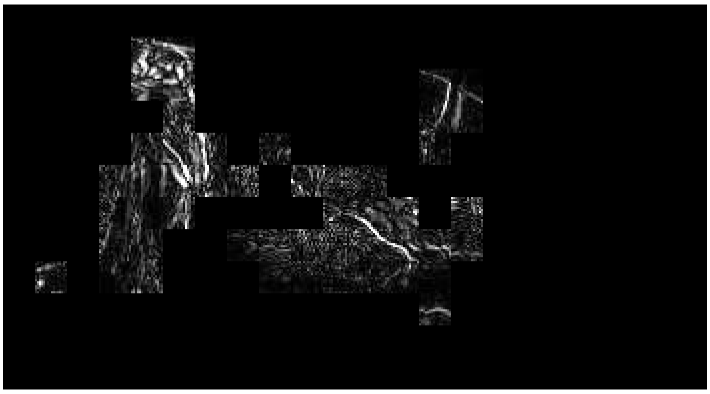
Examples of predicted images

Search range = 8, Block size = 8x8	
	
Three-Step Search	Full Search

Search Range = 8, Block size = 16x16	
	
Three-Step Search	Full Search

Search Range = 8, Block size = 32*32	
	
Three-Step Search	Full Search

5. Example of Residual Image

3-Step Search	
	Search range = 8 Block size = 8*8
	Search range = 8 Block size = 16*16
	Search range = 8 Block size = 32*32

6. Example of Motion Vector Graph

3-Step Search	
	Search range = 8 Block size = 8*8
	Search range = 8 Block size = 16*16
	Search range = 8 Block size = 32*32

Main Code

```
clear all; close all;clc;

%%Image Processing
ref_img = imread('Image67.jpg'); %read image %as reference image
target_img = imread('Image68.jpg'); %as target image
ref_img = im2double(ref_img); %modelling the image
target_img = im2double(target_img);
[height, width, channels] = size(ref_img); %size of image



%Experimenting with different parameter
strategy_list = {'fullsearch','3stepsearch'}; %search method
search_range = [8 16 32];  %search range
block_size = [8 16 32]; %try different block size used for matching  [8x8 16x16]
MSE = zeros(length(strategy_list), length(search_range), length(block_size));  %Mean↙
Squared Errir to evaluate the motion vector
psnr = zeros(length(strategy_list), length(search_range), length(block_size)); %for↙
every combination of strategy, search range and block size
computations1=0;
computations2=0;
computations3=0;
%%MAIN
for index_r = 1:length(search_range) %[8 16 32]
   R = search_range(index_r);
   for index_b = 1:length(block_size) %[8 16 32]
      blockSize = block_size(index_b);
      for index_s = 1:length(strategy_list)
         searchAlgo = strategy_list(index_s); %Search Algorithm
         motionVector = zeros(height/blockSize, width/blockSize, 2); %Define motion↙
Vector
         predicted_img = zeros(height, width, channels); %Define predicted image
         totalMSE = 0; %store total MSE
         tic;
            for h = 1:blockSize:height %1:9:192 %each location
               for w = 1:blockSize:width%1:9:352
                  % Motion estimation in each block
                  if strcmp(searchAlgo, 'fullsearch')
                     [predicted_block, blockMotionVector, r_MSE,computations1] = ↙
FullSearch(ref_img, target_img, h, w, width, height, R, blockSize);
```

```
            elseif strcmp(searchAlgo, '3stepsearch')
                [predicted_block, blockMotionVector, r_MSE,computations2] = ↙
ThreeStepSearch(ref_img, target_img, h, w, width, height, R, blockSize);
            end
            cost_fullsearch(index_r,index_b)=computations1; %steps required
            cost_3stepsearch(index_r,index_b)=computations2; %steps required
            predicted_img(h:h+blockSize-1, w:w+blockSize-1, :) = predicted_block; ↙
%predicted image
            blockIndex = [(h-1)/blockSize+1 (w-1)/blockSize+1];
            motionVector(blockIndex(1), blockIndex(2), :) = blockMotionVector; %↙
store into motionVector
            totalMSE = totalMSE + r_MSE;
        end
    end
    toc;
    complexity(index_r,index_b,index_s)=toc; %complexity for each case

  %%SAVING IMAGE(RESULT)
  imwrite(predicted_img, sprintf('as_range%d_b_size%d_%s_predict.jpg', R,↙
blockSize, searchAlgo{1})); %write and save predicted image
  imshow(target_img,'InitialMagnification','fit'); hold on;
  xidx = 1:blockSize:width;
  yidx = 1:blockSize:height;
  [X,Y] = meshgrid(xidx,yidx); %X(Value of xidx and size of (rowxcol) length(y)↙
xlength(x))
  u = squeeze(motionVector(:,:,1)); %squeeze into 2D array
  v = squeeze(motionVector(:,:,2));
  u = fliplr(v);
  %Showing motion vector
   %vectors X and Y represent the location of the base of each arrow, and U and V ↙
represent the directional components of each arrow.
  quiver(X, Y, u, v);
  hold off;
  F = getframe;% Capture the axes and return the image data
  RGB = frame2im(F);%Return image data associated with movie frame
  imwrite(RGB, sprintf('as_range%d_b_size%d_%s_quiver.jpg', R, blockSize,↙
searchAlgo{1})); %save motion vector

  residual_image = sum(abs(predicted_img-target_img), 3); %show the residual↙
image
```

```
        imshow(residual_image);
        imwrite(residual_image, sprintf('as_range%d_b_size%d_%s_residual.jpg', R,↙
blockSize, searchAlgo{1})); %residual image
        MSE(index_s, index_r, index_b) = totalMSE;
        psnr(index_s, index_r, index_b) = PSNR(predicted_img, target_img);
      end
  end
end
error_sq=(ref_img-target_img).^2;
MSE(3,:,:)=mean(error_sq(:))/(blockSize^2);
figure;
MSEplot(MSE); %plot MSE
figure;
PSNRplot(psnr); %plot PSNR
figure;
complexityplot(complexity); %plot Complexity
```

Three Step Search Function

```
function [predicted_block, motionVector, finalMSE,computations] = ThreeStepSearch ↙
(ref_img, target_img, h, w, width, height, R, blockSize)
   %R=search range
   h_origin = h;
   w_origin = w;
   computations=0;
   %with each time step store the motion vector until the end of 'R'
   motionVector = zeros(log2(R), 2); %xy 2 components ->2D array with log2(p)
   %segmenting the ref_img into block that is comparable with our target
   ref_block = ref_img(h:h+(blockSize-1), w:w+(blockSize-1), :);
   %in range of h to h+blockSize-1

   f = @(n) (2.^n);
   step_size = f(log2(R)-1:-1:0); % [4 2 1] or [8 4 2 1]  %step size

   for index_s = 1:length(step_size)   %find motion vector with least cost every time ↙
step
      finalMSE =999999999.99999999;% realmax;
      stepSize = step_size(index_s); %4 2 1

      for dh = (-1:1)*stepSize %[-4 0 4],[-2 0 2], [-1 0 1]
         if ((dh+h)>=1) && ((h+blockSize-1)+dh<=height) %after adding small step is also ↙
within the height and stepsize>=1

            for dw = (-1:1)*stepSize %[-4 0 4],[-2 0 2], [-1 0 1]
               if ((dw+w)>=1) && ((w+blockSize-1)+dw<=width)%within the width
                  target_block = target_img(h+dh:(h+dh+(blockSize-1)), w+dw:(w+dw+ ↙
(blockSize-1)), :);
                  %comparing with the previous block
                  sqr_error=(target_block-ref_block).^2;
                  MSE=sum(sqr_error(:))/(blockSize^2);
                  computations=computations+1;

                     if MSE <= finalMSE %decide which point to choose (choose with the ↙
least cost(MSE in this case))
                     finalMSE = MSE;
                     motionVector(index_s, :) = [dh dw]; %add the motion vector only if the ↙
cost is the least
                  end
               end
```

```
            end

        end
    end

    h = h + motionVector(index_s, 1);  %move every step 4->2->1
    w = w + motionVector(index_s, 2);
  end

  %predicted_block = ref_img(h:h+blockSize-1, w:w+blockSize-1, :);
  predicted_block = target_img(h:h+blockSize-1, w:w+blockSize-1, :); %end when the
  motionVector = sum(motionVector, 1); %resultant motionVector

end
```

```
function [predicted_block, motionVector, finalMSE,computations] = FullSearch↙
(ref_img, target_img, h, w, width, height, p, blockSize)
   finalMSE = realmax; %make it as large number
   motionVector = zeros(1, 2);
   ref_block = ref_img(h:h+blockSize-1, w:w+blockSize-1, :);
   computations=0;
   for dh = -p:p %every single pixel
      if (h+dh>=1)&&(((h+dh)+blockSize-1)<=height)
         for dw = -p:p
            if (w+dw>=1)&&(((w+dw)+blockSize-1)<=width)
               target_block = target_img(h+dh:((h+dh)+blockSize-1), w+dw:((w+dw)↙
+blockSize-1), :);
               sqr_error=(target_block-ref_block).^2;
               MSE=sum(sqr_error(:))/(blockSize^2);
               computations=computations+1;

               if MSE <= finalMSE %Calculate MSE to define next origin
                  finalMSE = MSE;
                  motionVector = [dh dw];
               end
            end
         end
      end
   end
   dh = motionVector(1);
   dw = motionVector(2);
   predicted_block = target_img(h+dh:((h+dh)+blockSize-1), w+dw:((w+dw)+blockSize-↙
1), :);
end
```

```
function [result] = PSNR(img_1, img_2)

   S_error = (double(img_1) - double(img_2)).^2; % convert to double so that it matches built-in-function
   MSE = mean(S_error(:)); %first, we find MSE(Mean-square error)
   result = 10 * log10((255^2)/(MSE)); %find PSNR
end
```

```
function MSEplot (MSE)
%%Data Processing
    fullSearch_MSE = reshape(transpose(squeeze(MSE(1,:,:))), 1,9);
    threeStepSearch_MSE = reshape(transpose(squeeze(MSE(2,:,:))), 1,9);
%%Plot
    plot(fullSearch_MSE), hold on;
    plot(threeStepSearch_MSE),
    hold off;
    legend('Full Search', '3-Step Search');
    xticks([1 2 3 4 5 6 7 8 9]);
    xticklabels({'8, 8x8','8, 16x16','8, 32x32','16, 8x8','16, 16x16','16, 32x32','32 8x8','32, ↙
16x16','32, 32x32'});
    title('MSE in different combination of Block Size and Search Range');
    xlabel('Search range, Block size (r, bxb)');
    ylabel('Mean squared error');
```

PSNRplot Function

```
function PSNRplot (PSNR)
%%Processing data
   fullSearch_PSNR = reshape(transpose(squeeze(PSNR(1,:,:))), [1,9]);
   threeStepSearch_PSNR = reshape(transpose(squeeze(PSNR(2,:,:))), [1,9]);
%%Plot
   plot(fullSearch_PSNR), hold on;
   plot(threeStepSearch_PSNR)
   legend('Full Search', '3-Step Search');
   xticks([1 2 3 4 5 6 7 8 9]);
   xticklabels({'8, 8x8','8, 16x16','8, 32x32','16, 8x8','16, 16x16','16, 32x32','32 8x8','32,↙
16x16','32, 32x32'});
   title('PSNR in different combination of Block Size and Search Range');
   xlabel('Search range, Block size (r, bxb)');
   ylabel('PSNR (Peak Signal-to-Noise Ratio) in dB');
end
```

ComplexityPlot Function

```
function complexityplot(complexity)
%%Data Processing
 fullSearch_complexity = reshape(transpose(squeeze(complexity(:,:,1))), [1,9]);
 threeStepSearch_complexity = reshape(transpose(squeeze(complexity(:,:,2))), [1,9]);
 %Plot
 plot(fullSearch_complexity), hold on;
 plot(threeStepSearch_complexity), hold off;
 legend('Full Search', '3-Step Search');
 xticks([1 2 3 4 5 6 7 8 9]);
 xticklabels({'8, 8x8','8, 16x16','8, 32x32','16, 8x8','16, 16x16','16, 32x32','32 8x8','32,
16x16','32, 32x32'});
 title('Complexity of search method in different combination of Block Size and
Search Range');
 xlabel('Search range, Block size (r, bxb)');
 ylabel('Complexity in sec');
end
```

SAD and MAD plot Function

```
function SMADplot (SAD,blockSize)
%%Data Processing
    fullSearch_SAD = reshape(transpose(squeeze(SAD(1,:,:))), 1,9);
    threeStepSearch_SAD = reshape(transpose(squeeze(SAD(2,:,:))), 1,9);
    %%SAD
    plot(fullSearch_SAD), hold on;
    plot(threeStepSearch_SAD), hold off;
    legend('Full Search', '3-Step Search');
    xticks([1 2 3 4 5 6 7 8 9]);
    xticklabels({'8, 8x8','8, 16x16','8, 32x32','16, 8x8','16, 16x16','16, 32x32','32 8x8','32,↙
16x16','32, 32x32'});
    title('SAD in different combination of Block Size and Search Range');
    xlabel('Search range, Block size (r, bxb)');
    ylabel('Sum Absoulute Difference');
    figure;
    %%MAD
    fullSearch_MAD=fullSearch_SAD/(blockSize.^2);
    threeStepSearch_MAD=threeStepSearch_SAD/(blockSize.^2);
    plot(fullSearch_MAD), hold on;
    plot(threeStepSearch_MAD), hold off;
    legend('Full Search', '3-Step Search');
    xticks([1 2 3 4 5 6 7 8 9]);
    xticklabels({'8, 8x8','8, 16x16','8, 32x32','16, 8x8','16, 16x16','16, 32x32','32 8x8','32,↙
16x16','32, 32x32'});
    title('MAD in different combination of Block Size and Search Range');
    xlabel('Search range, Block size (r, bxb)');
    ylabel('Sum Absoulute Difference');
end
```