Berkay Mengunogul 150119934

CSE 4084 Multimedia Systems
Homework 3 Report

Problem 1

This problem is about inverse filtering. You should review the corresponding slides to refresh your memory before attempting this problem. To help you understand how inverse filter is implemented and applied a MATLAB script has been provided (inverse_filtering.m).

Once you open the script using MATLAB, you will see on Line 8 the statement “T=1e-1”. This defines the threshold value used in the inverse filter. The script simulates the blur due to motion and applies inverse filtering for its removal.

Problem 1.1: Use the image “original_cameraman.jpg” for this problem and try different values of the threshold and see how it effects the performance of the inverse filter. Find the ISNR value when the threshold is set to 0.5 (up to two decimal digits).

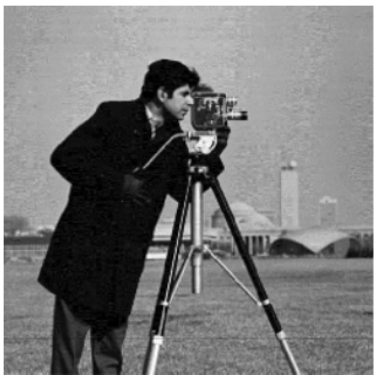


Figure 1: Original Cameraman Image

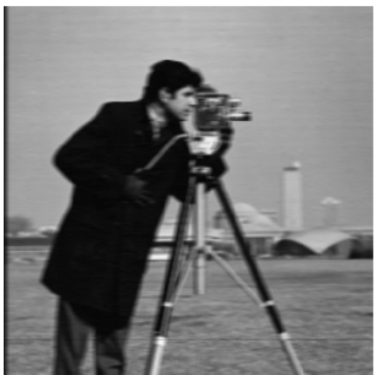


Figure 2: Blurred Cameraman Image

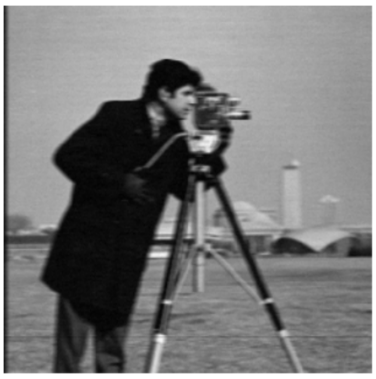


Figure 3: Noisy Cameraman Image

Solution 1.1: Figure 1, shows the original image. Figure 2 and Figure 3 show the outputs of motion blur simulation as Blurred and Noisy Image. As it is asked threshold value is changed to 0.5 in code and the output image is shown in Figure 4. As a result, it can be observed that the blurry effect still remains and there are ringing artifacts that occurred.

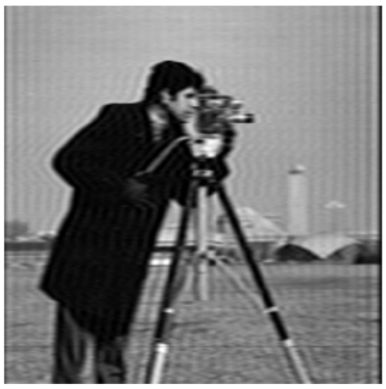


Figure 4: Restored Image when T=0.5

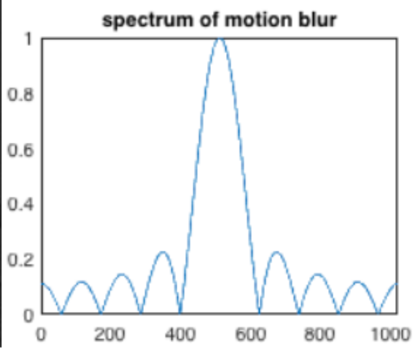


Figure 5: Restored Image when T=1

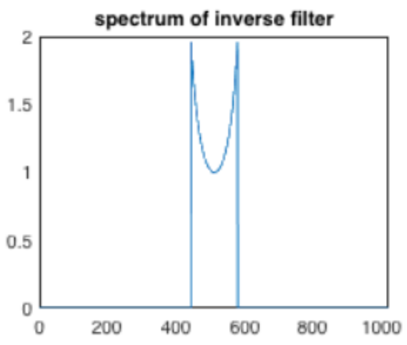
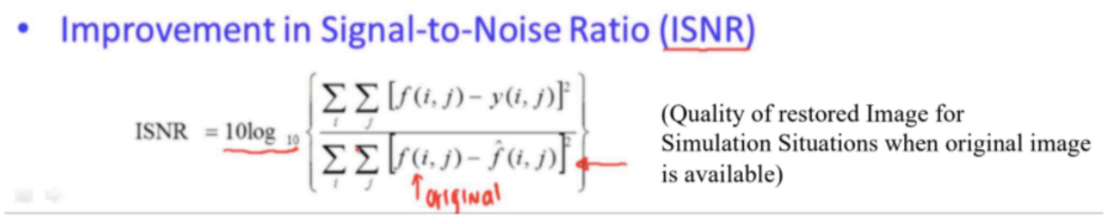


Figure 6: Restored Image when T=1



• Improvement in Signal-to-Noise Ratio (ISNR)

$$ISNR = 10\log_{10}\left\{\frac{\sum_i\sum_j [f(i,j)-y(i,j)]^2}{\sum_i\sum_j [f(i,j)-\hat{f}(i,j)]^2}\right\}$$

original

(Quality of restored Image for Simulation Situations when original image is available)

Equation 1: ISNR Formula

From the lecture notes, considering the ISNR formula in Equation 1, the **ISNR value is calculated as 2.86**.

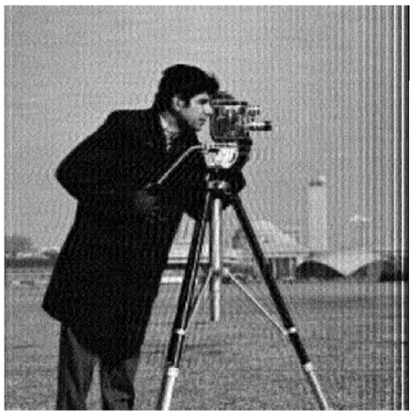


Figure 7: Restored Image when T=0.1

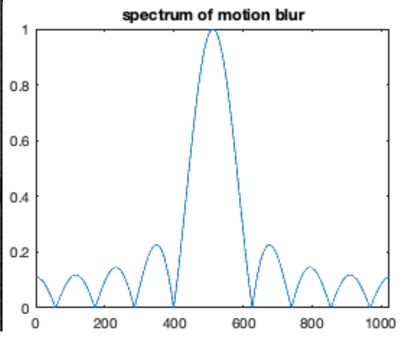


Figure 8: Restored Image when T=0.5

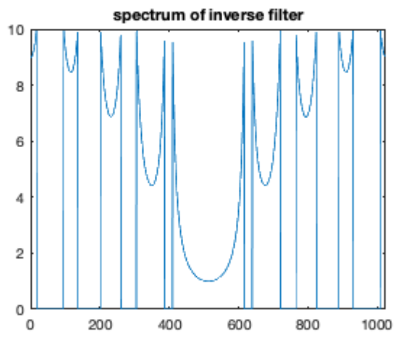


Figure 9: Restored Image when T=0.5

When the threshold is adjusted to 0.1, the resulting image is displayed in Figure 7. It is more clear and more sharp compared to Figure 4. However, vertical lines appeared as noise in this image.

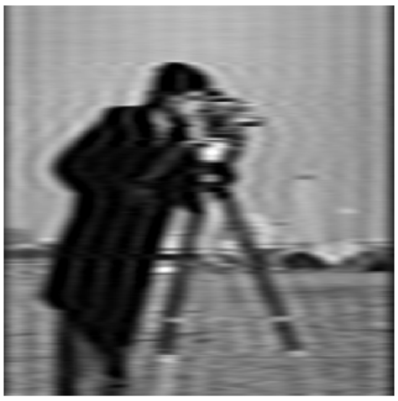


Figure 10: Restored Image when T=0.9

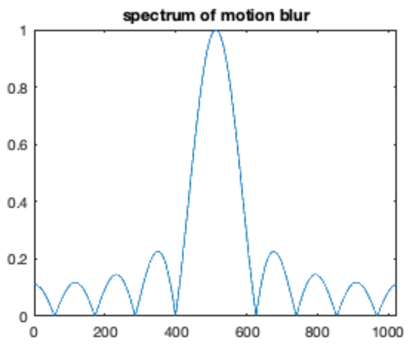


Figure 11: Restored Image when T=0.9

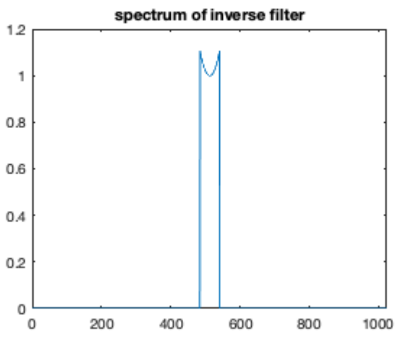


Figure 12: Restored Image when T=0.9

When the threshold is adjusted to 0.9, the resulting image is displayed in Figure 10. It is way more blurry and ringing artifacts are thicker compared to Figure 4. The result of this process is not acceptable.

Adjusted codes are provided at Code 1.

Problem 2

In this problem, you will implement the Constrained Least Squares (CLS) filter and examine its performance when the regularization parameter is set at different values. The original image (Cameraman256.bmp) and a set of MATLAB files have been provided for this problem (cls_restoration.m, next2pow.m, wrapper.m). Follow the instructions below to finish this problem.

Problem 2.1: Place the original image and all the provided MATLAB files in the same directory.

Solution 2.1: Given files are moved to the same directory.

Problem 2.2: The file “wrapper.m” is the entry or the “main” code. It loads the original image, applies a motion blur to it, and degrades the image by adding noise. The 17th line in “wrapper.m” sets the value of the regularization parameter “alpha”.

Problem 2.3: The MATLAB file “cls_restoration.m” has an incomplete implementation of the CLS filter. You need to uncomment line 24 in the “cls_restoration.m” and complete the implementation of the CLS filter.

Solution 2.3: The frequency response of the CLS filter is provided in the lecture notes and it is shown in Equation 2.

$$\left| \frac{H^*(u,v)}{\left|H(u,v)\right|^2 + .1 \cdot \left|C(u,v)\right|^2} \right|$$

Equation 2: Frequency Response of CLS Filter

Considering Equation 2, R has been implemented and it is written in Code 2.

Problem 2.4: After you complete the implementation of the CLS filter, you should run “wrapper.m” with different values of alpha: (0.0001, 0.001, 0.1, 1, 10, 100). For each value of alpha, compute the improvement in SNR (ISNR and put your results in a table. Note that the computation of ISNR involves three images: the original image, the blurred and noisy image, and the restored image.

Solution 2.4:

Alpha	ISNR of Noisy and Blurred Image
0,0001	-5.92
0,001	-1.53
0,1	04.30
1	02.15
10	00.49
100	-0.93

Table 1: ISNR Results of Noisy and Blurred Image

ISNR calculation is already implemented in Problem 1 beforehand considering Equation 1. So calculated results are displayed in Table 1 for each alpha value.

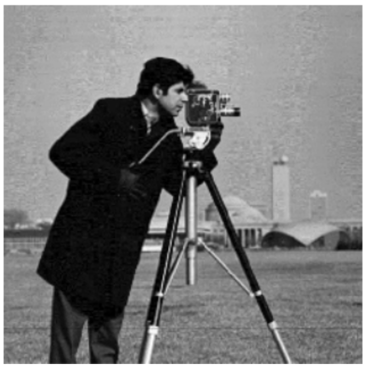


Figure 13: Original Image

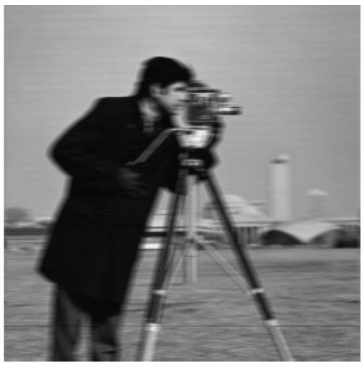


Figure 14: Blurred Image

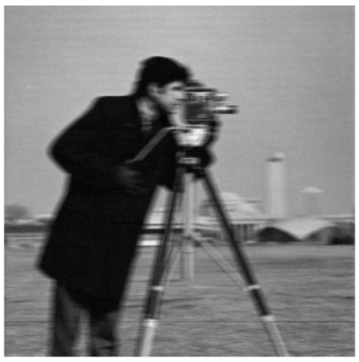


Figure 15: Noisy and Blurred Image

Original Image is displayed in Figure 13, image after the blurred effect is displayed in Figure 14, noise added to the blurred image and displayed in Figure 15.

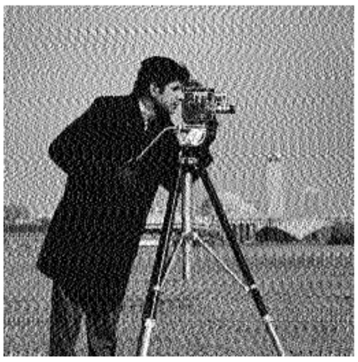


Figure 16: CLS restoration with alpha = 0.0001

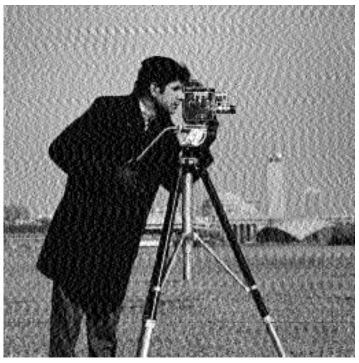


Figure 17: CLS restoration with alpha = 0.001

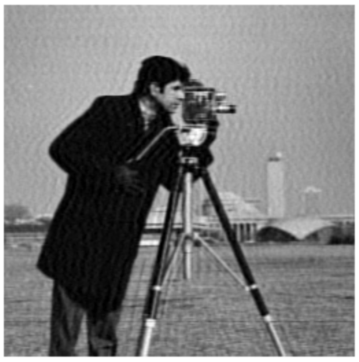


Figure 18: CLS restoration with alpha = 0.1

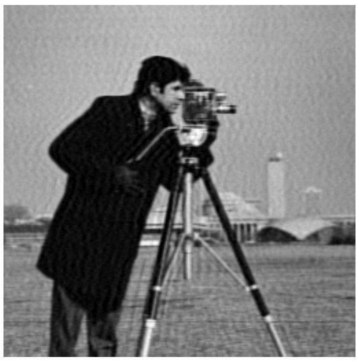


Figure 19: CLS restoration with alpha = 1

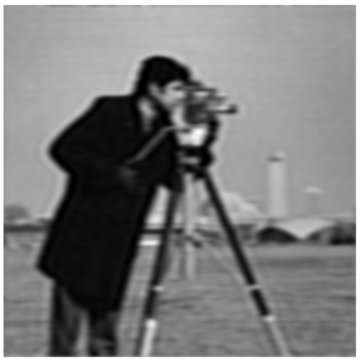


Figure 20: CLS restoration with alpha = 10

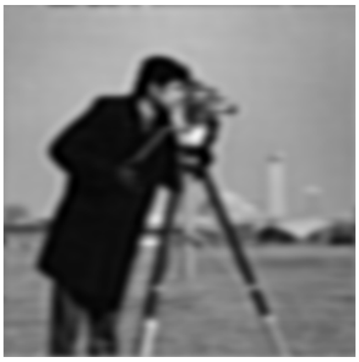


Figure 21: CLS restoration with alpha = 100

Output results of the CLS restoration are displayed starting from Figure 16 to Figure 21 considering values in Table 1 with the same order. As it can be observed, when alpha goes to -infinity, the image became sharper but noisier. When alpha goes to +infinity, the image becomes more blurry. The most acceptable results are within the tested values are both alpha = 0.1 and alpha = 1.

Adjusted codes are provided at Code 2.

Problem 3

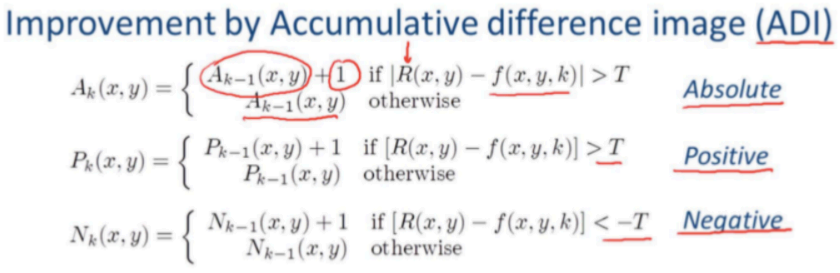
In this problem, you will use Accumulative Difference Image (ADI) to calculate the motion of an object. The object is a bright rectangle moving with a constant speed in a dark background.

Your task is to find the speed of the object in the horizontal direction (x-direction) and in the vertical direction (y-direction), as well as the total space this object occupied while moving. The total space is defined as the total number of pixels that this object occupies at least once during its movement.

Use the code “motion_ADI.m” for this problem. The code has detailed comments regarding each functioning part. Basically, the code generates the reference frame and 10 consecutive frames containing the moving object.

Problem 3.1: All you need to do is decide on the appropriate threshold T in line 23 in the code and implement the three equations for ADI in the lecture notes regarding motion-based segmentation. Start your code flowing line 37 and finish it before the end of the for-loop. The rest of the code will calculate the speed of the moving object and the total space it occupies for you. Report speed_X_Direction and speed_Y_Direction.

Solution 3.1: Since the provided images contain a plain white rectangle with a black background, any difference between frames will be easy to observe. Therefore, the threshold is adjusted to 0 for keeping all the information visible.



Improvement by Accumulative difference image (ADI)

$$A_k(x,y) = \begin{cases} A_{k-1}(x,y) + 1 & \text{if } |R(x,y) - f(x,y,k)| > T \\ A_{k-1}(x,y) & \text{otherwise} \end{cases} \quad \text{Absolute}$$

$$P_k(x,y) = \begin{cases} P_{k-1}(x,y) + 1 & \text{if } [R(x,y) - f(x,y,k)] > T \\ P_{k-1}(x,y) & \text{otherwise} \end{cases} \quad \text{Positive}$$

$$N_k(x,y) = \begin{cases} N_{k-1}(x,y) + 1 & \text{if } [R(x,y) - f(x,y,k)] < -T \\ N_{k-1}(x,y) & \text{otherwise} \end{cases} \quad \text{Negative}$$

Equation 3: Formula of Absolute ADI, Positive ADI and Negative ADI

Considering the formulas provided in the lectures in Equation 3, equations are implemented to related lines.

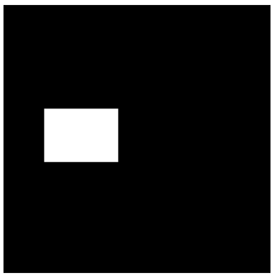


Figure 22: Visualize Object

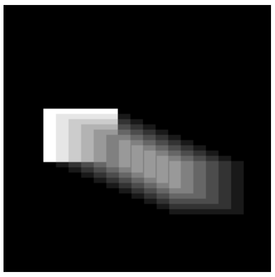


Figure 23: Absolute ADI

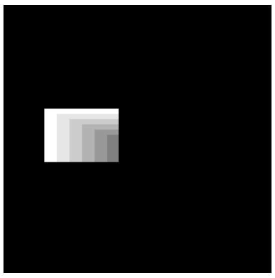


Figure 24: Positive ADI

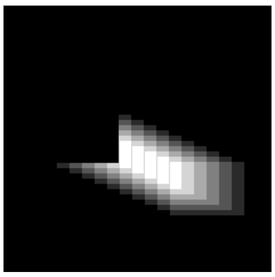


Figure 25: Negative ADI

Considering the formulas provided in the lectures in Equation 3, equations are implemented to related lines. Figure 22, displays the object and presents the frame R. Object moves along the x-axis and y-axis 10 times. The result of the Absolute ADI can be seen in Figure 23. In Figure 24, the Result of the Positive ADI is displayed and in Figure 25, the Result of the Negative ADI is displayed.

Adjusted codes are provided at Code 3 below.

Code 1:

inverse_filtering.m:

```
% inverse filter with thresholding


clear all

close all

clc


% specify the threshold T

T = 0.5;


%% read in the original, sharp and noise-free image

original = im2double(rgb2gray((imread('original_cameraman.jpg'))));

[H, W] = size(original);


%% generate the blurred and noise-corrupted image for experiment

motion_kernel = ones(1, 9) / 9;  % 1-D motion blur

motion_freq = fft2(motion_kernel, 1024, 1024);  % frequency response of motion
blur

original_freq = fft2(original, 1024, 1024);

blurred_freq = original_freq .* motion_freq;  % spectrum of blurred image

blurred = ifft2(blurred_freq);

blurred = blurred(1 : H, 1 : W);

blurred(blurred < 0) = 0;

blurred(blurred > 1) = 1;

noisy = imnoise(blurred, 'gaussian', 0, 1e-4);



%% Restoration from blurred and noise-corrupted image

% generate restoration filter in the frequency domain

inverse_freq = zeros(size(motion_freq));

inverse_freq(abs(motion_freq) < T) = 0;

inverse_freq(abs(motion_freq) >= T) = 1 ./ motion_freq(abs(motion_freq) >= T);
```

```
% spectrum of blurred and noisy-corrupted image (the input to restoration)

noisy_freq = fft2(noisy, 1024, 1024);

% restoration

restored_freq = noisy_freq .* inverse_freq;

restored = ifft2(restored_freq);

restored = restored(1 : H, 1 : W);

restored(restored < 0) = 0;

restored(restored > 1) = 1;


%% analysis of result

noisy_psnr = 10 * log10(1 / (norm(original - noisy, 'fro') ^ 2 / H / W));

restored_psnr = 10 * log10(1 / (norm(original - restored, 'fro') ^ 2 / H / W));


%% visualization

figure; imshow(original, 'border', 'tight');

figure; imshow(blurred, 'border', 'tight');

figure; imshow(noisy, 'border', 'tight');

figure; imshow(restored, 'border', 'tight');

figure; plot(abs(fftshift(motion_freq(1, :)))); title('spectrum of motion
blur'); xlim([0 1024]);

figure; plot(abs(fftshift(inverse_freq(1, :)))); title('spectrum of inverse
filter'); xlim([0 1024]);

%% ISNR

noisy_snr = sum((original(:) - noisy(:)).^ 2, "all");

restored_snr = sum((original(:) - restored(:)).^ 2, "all");

isnr = num2str(10*log10(noisy_snr/restored_snr), '%05.2f')
```

Code 2:

wrapper.m:

```
clear all

close all


%% Simulate 1-D blur and noise
```

```
image_original = im2double(imread('Cameraman256.bmp', 'bmp'));

[H, W] = size(image_original);

blur_impulse = fspecial('motion', 7, 0);

image_blurred = imfilter(image_original, blur_impulse, 'conv', 'circular');

noise_power = 1e-4;

randn('seed', 1);

noise = sqrt(noise_power) * randn(H, W);

image_noisy = image_blurred + noise;


figure; imshow(image_original, 'border', 'tight');

figure; imshow(image_blurred, 'border', 'tight');

figure; imshow(image_noisy, 'border', 'tight');


%% CLS restoration

alpha = 100;  % you should try different values of alpha

image_cls_restored = cls_restoration(image_noisy, blur_impulse, alpha);

figure; imshow(image_cls_restored, 'border', 'tight');


%% computation of ISNR


noisy_snr = sum((image_original(:) - image_noisy(:)).^ 2, "all");

restored_snr = sum((image_original(:) - image_cls_restored(:)).^ 2, "all");

noisy_isnr = num2str(10*log10(noisy_snr/restored_snr), '%05.2f')
```

next2pow.m:

```
function result = next2pow(input)

if input <= 0

    fprintf('Error: input must be positive!\n');

    result = -1;

else

    index = 0;

    while 2 ^ index < input

        index = index + 1;
    end
    result = 2 ^ index;
end
```

cls_restoration.m:

```
function image_restored = cls_restoration(image_noisy, psf, alpha)


%% find proper dimension for frequency-domain processing
[image_height, image_width] = size(image_noisy);
[psf_height, psf_width] = size(psf);
dim = max([image_width, image_height, psf_width, psf_height]);
dim = next2pow(dim);


%% frequency-domain representation of degradation
psf = padarray(psf, [dim - psf_height, dim - psf_width], 'post');
psf = circshift(psf, [-(psf_height - 1) / 2, -(psf_width - 1) / 2]);
H = fft2(psf, dim, dim);


%% frequency-domain representation of Laplace operator
Laplace = [0, -0.25, 0; -0.25, 1, -0.25; 0, -0.25, 0];
Laplace = padarray(Laplace, [dim - 3, dim - 3], 'post');
Laplace = circshift(Laplace, [-1, -1]);
C = fft2(Laplace, dim, dim);


%% Frequency response of the CLS filter
% Refer to the lecture for frequency response of CLS filter
% Complete the implementation of the CLS filter by uncommenting the
% following line and adding appropriate content
% Formula of R = ||H|*(u, v)/ (|H(u, v)|^2 + alpha*|C(u,v)|^2)|
R = conj(H)./(abs(H.^2) + alpha*abs(C.^2));


%% CLS filtering
Y = fft2(image_noisy, dim, dim);
```

```
image_restored_frequency = R .* Y;
image_restored = ifft2(image_restored_frequency);
image_restored = image_restored(1 : image_height, 1 : image_width);
```

Code 3:

motion_ADI.m:

```
clear all
close all



A = zeros(256,256); % initialize a 256*256 image


% initialize absolute ADI, positive ADI and Negative ADI
% all initialized to zero
% Note that all ADIs are of the same size with the image
% DO NOT change the name of the ADIs as they will be used later
ADI_abs = zeros(256,256);
ADI_pos = zeros(256,256);
ADI_neg = zeros(256,256);


% initialize the starting position of the moving object
% the moving object is a rectangle similar to the example in the lecture
% slides
start1 = 100;
start2 = 150;
start3 = 40;
start4 = 110;


%threshold T as in equations in the lecture slides regarding ADI
T = 0; % since we are working with black and white pixels, and our moving
       % object is plain white, catching any difference will be fine. So
       % that T = 0.
```

```matlab
%initialize the reference frame R
A(start1:start2, start3:start4) = 1;

%visualize the object and in the reference frame R
figure,imshow(A,[], 'border','tight');

j = 0;
for i = 5: 5 :50
        j = j + 12;
        A2 = zeros(256,256);
        A2(start1 + i: start2 + i, start3 + j: start4 + j) = 1;

        % You need to code up the follwing part that calculate the ADIs
        % Namely, the absolute ADI, the positive ADI and the negative ADI
        % Equations can be found in lecture slides regarding ADIs
        % You need to decide on the appropriate threshold T for this case
        % at line 23
        ADI_abs = ADI_abs + (abs(A-A2) > T); % Take the absoulte value and
                                             % add it.
        ADI_pos = ADI_pos + ((A-A2)> T); % Take the positive values and
                                         % add it .
        ADI_neg = ADI_neg + ((A-A2)< -T); % Take the negative values and
                                          % add it .
end

% The following part will calculate the moving speed
% and the total space(in pixel number) occupied by the moving object
[row, col] = find(ADI_neg > 0);
speed_X_Direction = (max(col) - start4) / 10
speed_Y_Direction = (max(row) - start2) / 10
total_space_occupied = sum(sum(ADI_abs > 0))

% The following part helps you to visualize the ADIs you compute
% compare them with the example shown in lecture

% You should be getting someting very similar

figure,imshow(ADI_abs,[], 'border','tight');

figure,imshow(ADI_pos,[], 'border','tight');

figure,imshow(ADI_neg,[], 'border','tight');
```