19 AĞUSTOS 2022

MIDDLE EAST TECHNICAL UNIVERSITY

EE300 - SUMMER PRACTICE REPORT

STUDENT NAME: ABDULLAH EMİR GÖĞÜSDERE
STUDENT ID: 2443075

SP COMPANY: TÜBİTAK BİLGEM İLTAREN
COMPANY DIVISION: ELECTRO-OPTICS TECHNOLOGIES

SP DATES: 25.07.2022 – 19.08.2022

SP SUPERVISOR: AHMET BAŞARAN
SP SUPERVISOR EMAIL: AHMET.BASARAN@TUBITAK.GOV.TR

SP SUPERVISOR APPROVE SIGN: _____

Table of Contents

The Description of the SP Company

Advanced Technologies Research Institute (İLTAREN) is an institute that conducts research in the field of Electronic Warfare (EH) in the Ankara campus of the Joint Electronic Warfare Support Command Barracks (MEHDESKOM) within the Informatics and Information Security Advanced Technologies Research Center (BİLGEM).

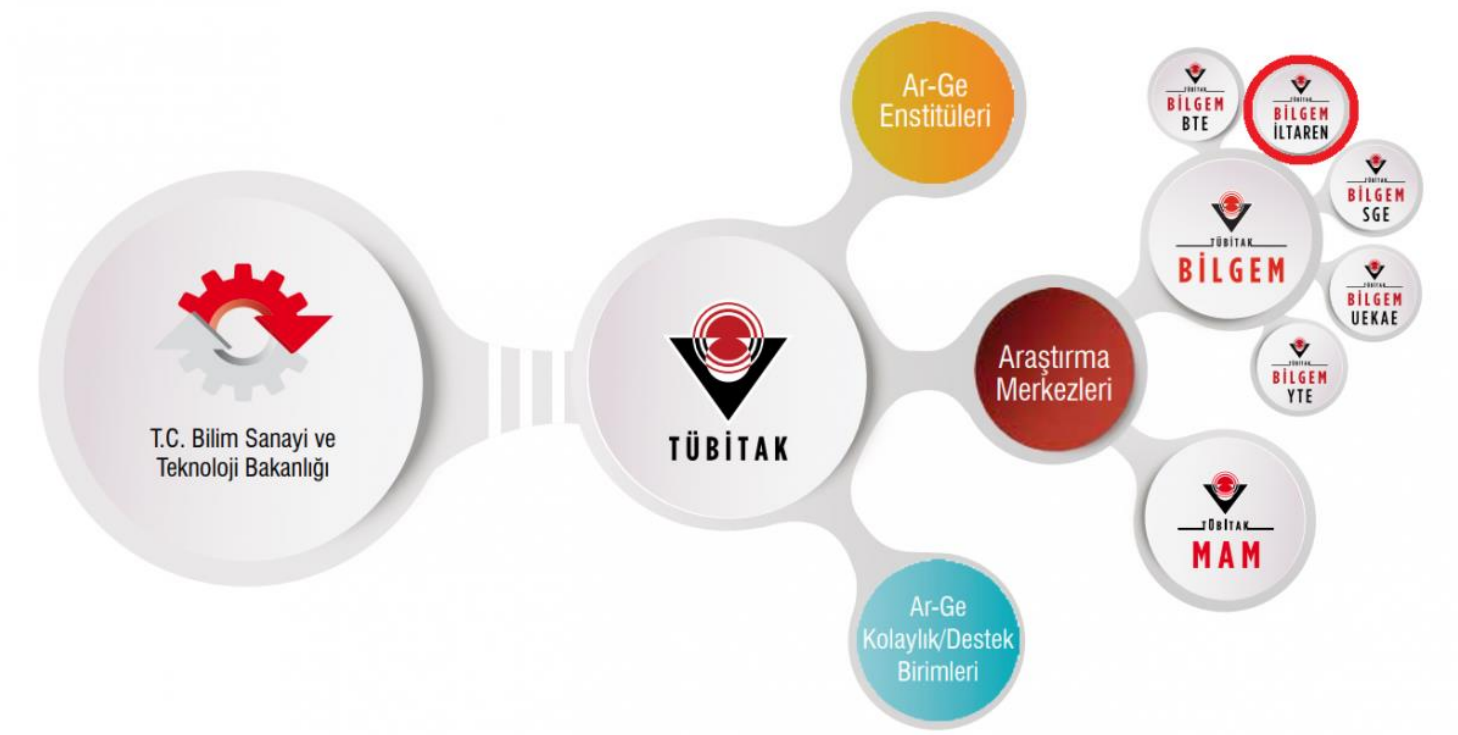

Figure 1. The hierarchy chart of İLTAREN

Advanced Technologies Research Institute (İLTAREN);

- Establishing infrastructures for test, measurement, analysis, and evaluation activities and carrying out these activities,
- Researching all types of software, hardware, and equipment in the field of electronic warfare, developing technologies and prototypes,
- Establishing methods and processes for the development and evaluation of electronic warfare techniques and tactics,
- It carries out the tasks of conducting scientific research in the relevant fields of activity.

Within İLTAREN, systems and infrastructures, including modeling and simulation software for decision support, are developed to assess the performance of various systems operating in radio frequency (RF), infrared (IR), and optical bands under varied environmental conditions.

In addition to software development research, lab structures are also created, including electronic warfare system prototypes and simulators with hardware components for these systems.

Systems for measurement, testing, and evaluation that operate in both laboratory and open-air settings are also being created to carry out all these procedures methodically using delicate and in-depth models backed by scientific evidence.

Approximately 500 employees work in İLTAREN. However, the information regarding how many engineers, technicians, administrators, and other employees work in İLTAREN is confidential.

In Figure 2, the Electro-Optics Technologies division where I worked in the company can be seen in the organization chart of İLTAREN.

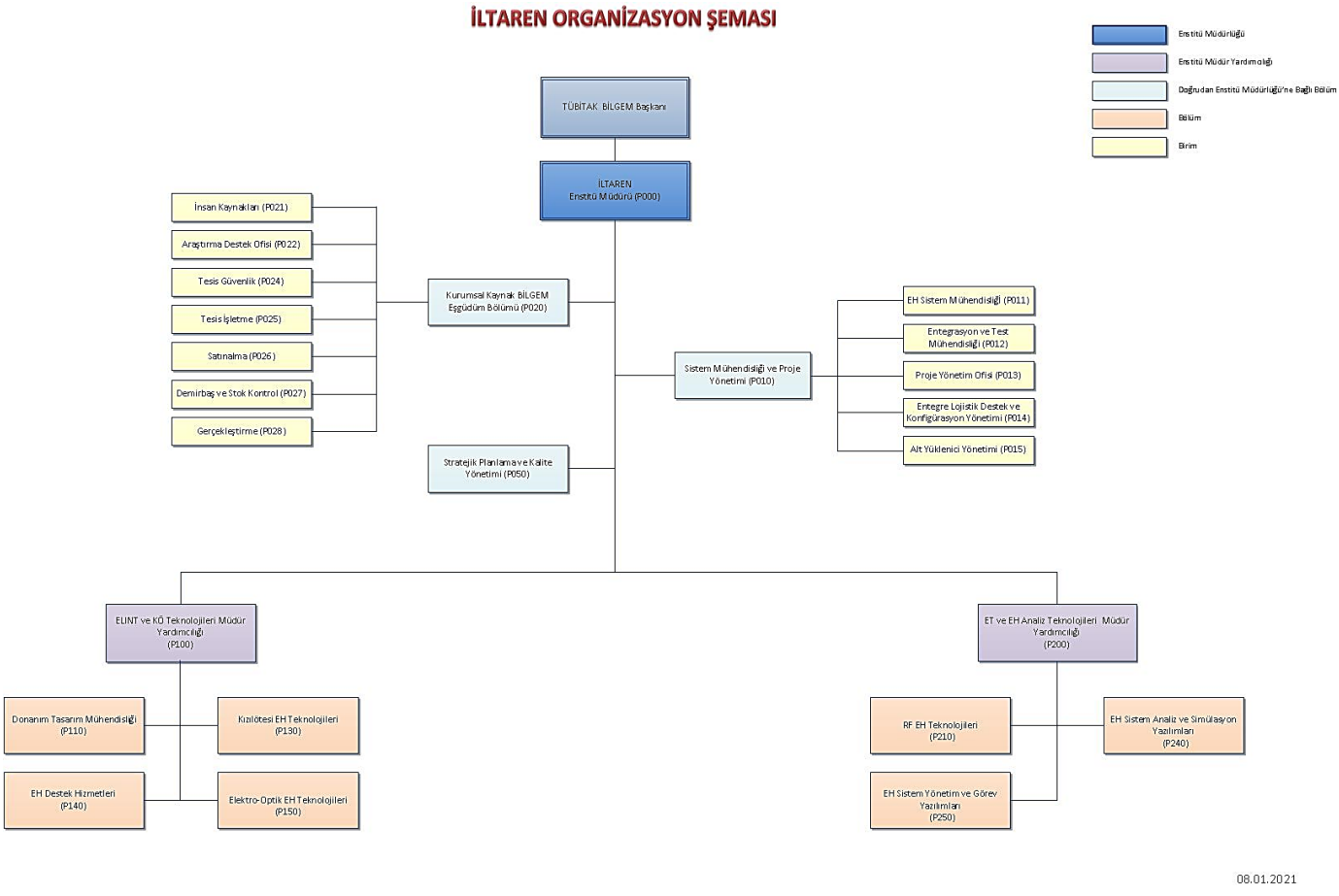

Figure 2. The organization chart of İLTAREN

A Brief History of the Company

The General Staff, Communication Electronics, and Information Systems Presidency (Gnkur. MEBS. Presidency) and TÜBİTAK came to an agreement on November 4, 1999, to work together to meet the needs of the Turkish Armed Forces (TSK) with scientific methods to have an effective defense capability, particularly by using electronic warfare technologies. To establish the Advanced Technologies Research Institute (İLTAREN), affiliated with TÜBİTAK, which will carry out the activities of determining the needs, producing original designs to satisfy the requirements, and developing methods, processes, and systems, a protocol defining the principles of cooperation was signed.

With this protocol, it is envisaged that İLTAREN's mission and vision, task analysis, organizational structure, and the preparation of the future master plan will be completed in cooperation, and the establishment of the institute is going to be completed.

In the TÜBİTAK Presidency building, İLTAREN was founded in its initial temporary site, and its core staff was assembled in 2000. With the signing of the second protocol on February 1, 2002, it was determined that the İLTAREN group will continue its operations under the framework of the National Electronics and Cryptology Research Institute (UEKAE).

The construction of the İLTAREN building, which was started on January 16, 2006, was completed on August 13, 2007. The İLTAREN building, where infrastructure works have been carried out since September 2007, was built by the Prime Minister, Mr. Recep Tayyip ERDOĞAN, and Chief of General Staff Gen.

Introduction

I began my summer practice in TÜBİTAK BİLGEM İLTAREN on 25th July 2022. I spent the first day of SP with the orientation program about the facilities working under TÜBİTAK BİLGEM, information security, and work safety. Afterward, we were designated to the company's associated department named Electro-Optics Technologies. After the designation, I was assigned by my supervisor to work on a project about the blind restoration of atmospherically degraded images [1]. After lots of researching and reading papers, articles, and examples, I managed to complete all the parts of the project. Moreover, I participated in seminars by TÜBİTAK during SP, which are

- Introduction of UEKAE (National Electronics and Cryptology Research Institute)
- Introduction of İLTAREN and Electronic Warfare
- Introduction of YTE (Software Technologies Research Institute) and Software Engineering Introduction Training
- Introduction of BTE (Institute of Information Technologies) and Artificial Intelligence

The Project

The first week

In the first week, I was expected to learn a basic understanding of the project. In other words, the first week was spent most of the time understanding the concepts of the project. Therefore, firstly, I try to grasp the main problem and solution of the project. The problem is the distortions of the blur, intensity scintillations, and spatio-temporal movements in the image caused by the atmospheric degradation sources such as aerosols and turbulence. In Figure 3, an example of an atmospherically degraded image is given.

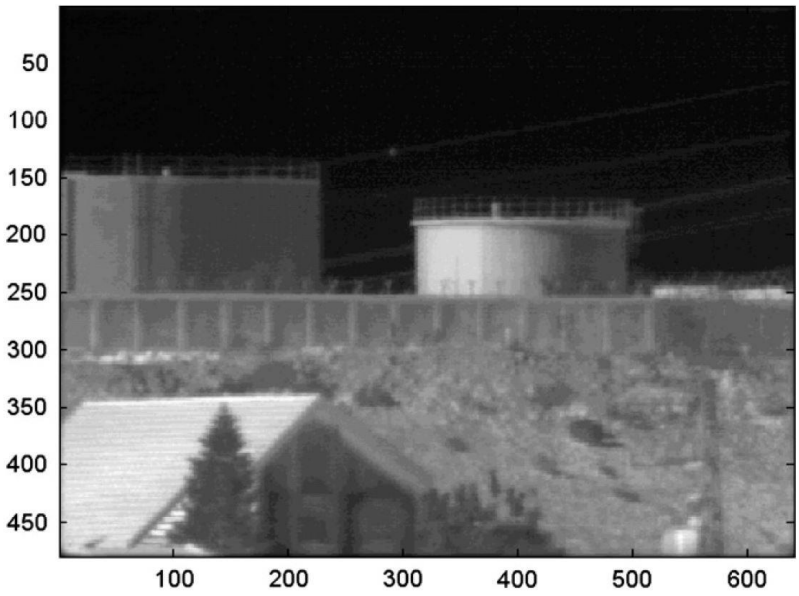

Figure 3. A real degraded image taken by a thermal camera in the 3–5 μm wavelength range [1]

Besides, it is vital to know that the technique for the restoration of atmospherically blurred images formulates the degraded image $G(u,v)$ as a convolution between the degradation function $H(u,v)$ (i.e., the atmospheric point spread function (PSF)) and the raw image $F(u,v)$ in the spatial domain [1].

$$G(u,v) = H(u,v) * F(u,v) \quad (1)$$

As a result, the solution is to recover from the distortions in the image using specific techniques such as a simple deconvolution method named Wiener filtering, which can be utilized in this case.

However, the main challenge is the need for reliable knowledge of the $H(u,v)$ since in most practical situations this data is not likely known. Typically, the only available data is the taken image itself. The process of image deconvolution, in this case, is called blind restoration or blind deconvolution [1]. To acquire information from the PSF, one can also use LSF (Line Spread Function).

Line Spread Function (LSF)

The profile of a line image (a function of one variable) is the line spread function (LSF). It is made from the summation of a line of overlapping PSFs. The point spread function (PSF) is defined as the response of an imaging system to a point object or source. If the system is isotropic (i.e., the system has the same physical properties in all directions), the LSF is not dependent on the orientation. In this case, the LSF holds all the information that the PSF does [2]. Figure 4 demonstrates the relationship between the PSF and the LSF for an imaging process.

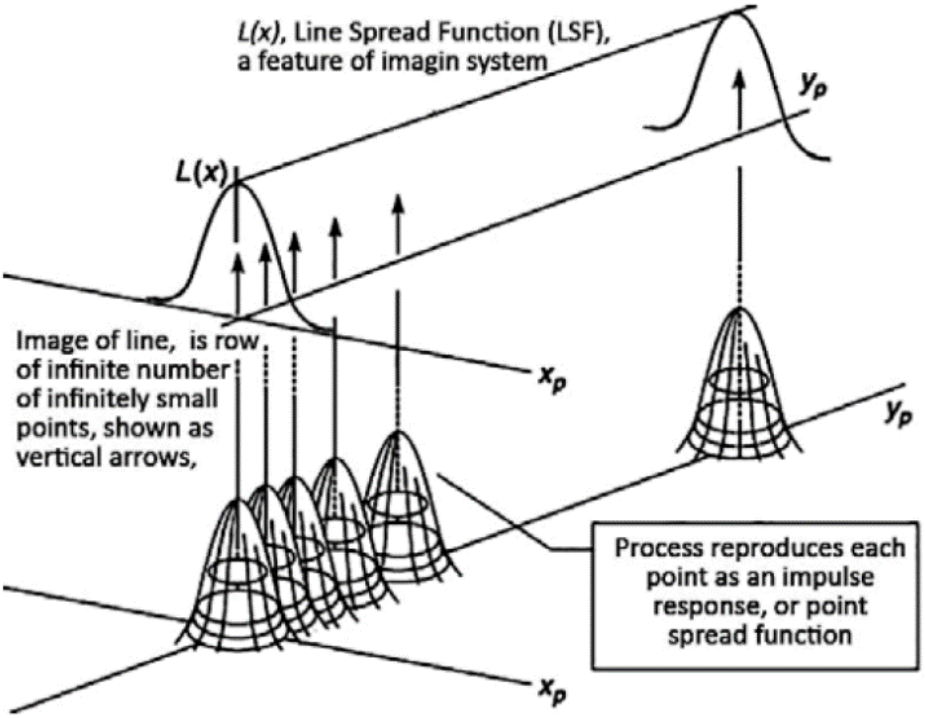

Figure 4. the LSF as a sum of PSFs along a line [2]

Consequently, to estimate (the atmospheric point spread function (PSF) in the spatial domain), we should find the LSF of the best step edge and take the FFT of the LSF, which is called MTF (Modulation Transfer Function). In the following pages, it is going to be explained what the best step edge is and how it is found in the image. Also, in this project, I am not tasked with finding the best step edge automatically. Therefore, I had to choose the best step edge manually.

Modulation Transfer Function (MTF)

The MTF of an imaging tool such as a lens measures the capability to transmit contrast at a specific resolution from the entity to the image [3]. In other words, MTF is a technique to possess contrast and resolution into only one feature. For the sake of simplicity, it should be enough to think of MTF as the Fourier transform of the LSF (Line Spread Function). Moreover, it should be known that as line spacing drops, the frequency increases, on the test target (Figure 5), it becomes more problematic for the imaging tool to transmit this drop on contrast efficiently; thus, the MTF decreases [3]. This occasion is shown in Figure 6.

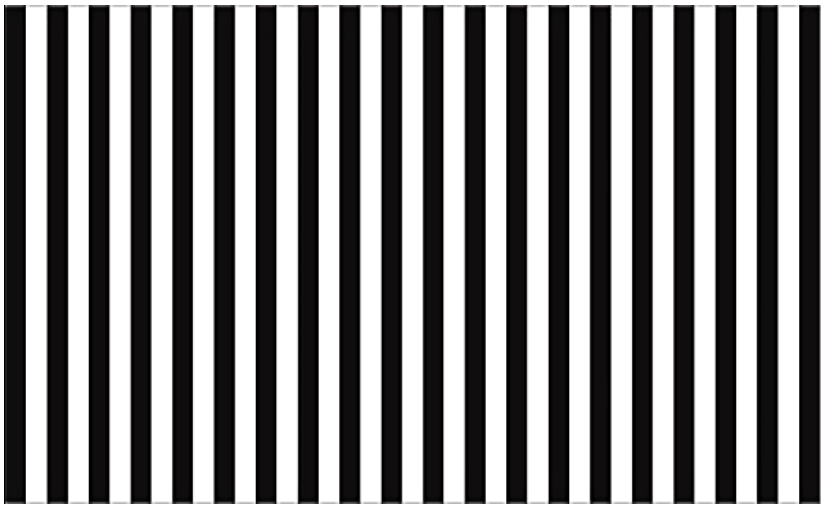

Figure 5. The test target

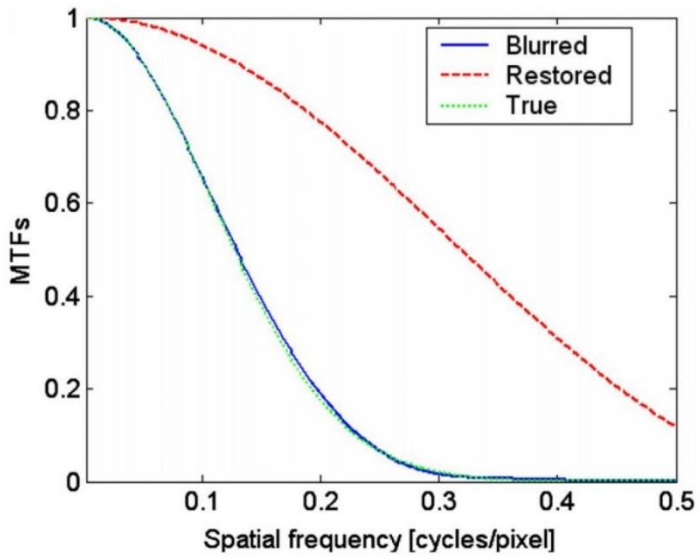

Figure 6. A comparison graph between the extracted MTFs before and after restoration and the true MTF [1]

Another concept that should be known is the best step edge. Most of the features of the image are extracted from the region where the best step edge is located.

The best step-edge

A step-edge includes a transition between two grey levels. By canny edge detection, sharp grey-level transitions in the image are located. Then, we search for a long high-contrast straight edge that is homogenous on both sides. A long edge will allow averaging over a large area to reduce noise artifacts. A high-contrast edge is desired since it yields a better SNR (Signal to Noise Ratio) in the LSF calculation. An example of the best step edge region in an image is given in Figure 7.

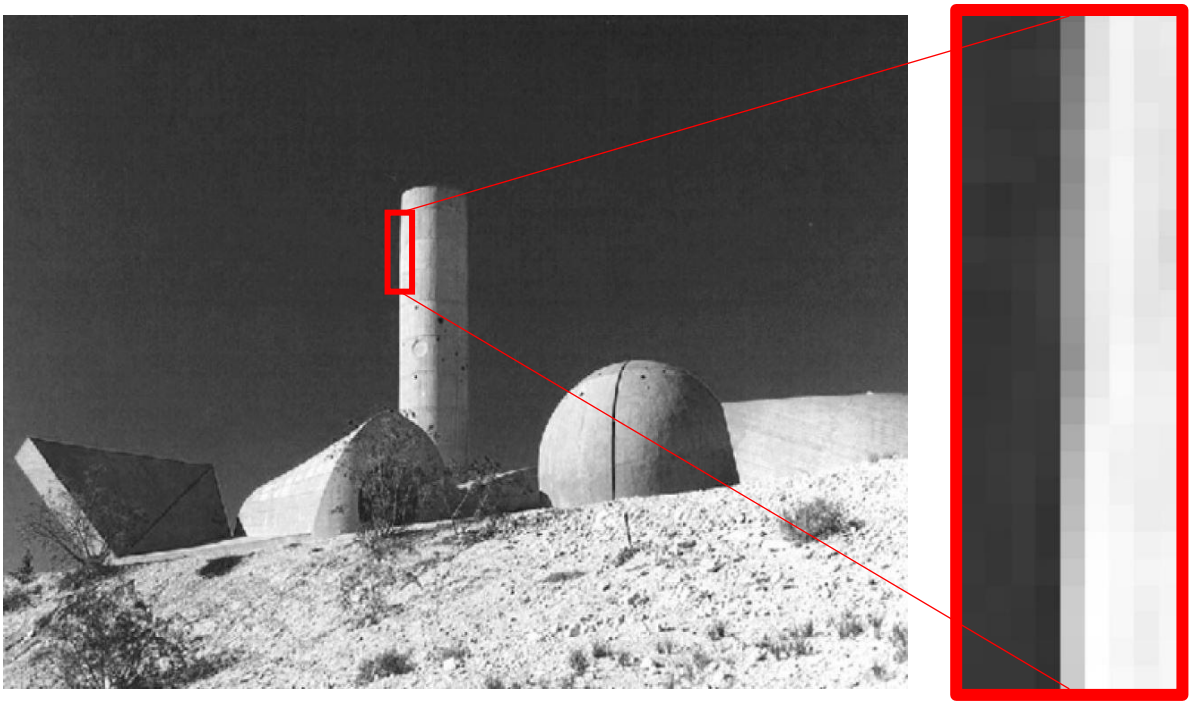

Figure 7. The best step-edge region [1]

As stated before, I am not assigned to find the best step edge automatically in this project. Hence, I always select the best step edge manually.

After picking the best step edge, the edge spread function (ESF) must be calculated to find the line spread function (LSF).

Edge Spread Function (ESF)

The edge spread function (ESF) is the image response to a high contrast edge. The derivative of the ESF becomes the line spread function (LSF), which is the image response to a high contrast line. The process of calculating the ESF starts with selecting an appropriate line profile, as shown in Figure 8. After picking a reference point, the distance of line profile pixels from the reference point is measured

and saved with their gray levels (i.e., intensities). The obtained data is then sorted for better plotting on the graph. In Figure 8, here is an illustration of the ESF.

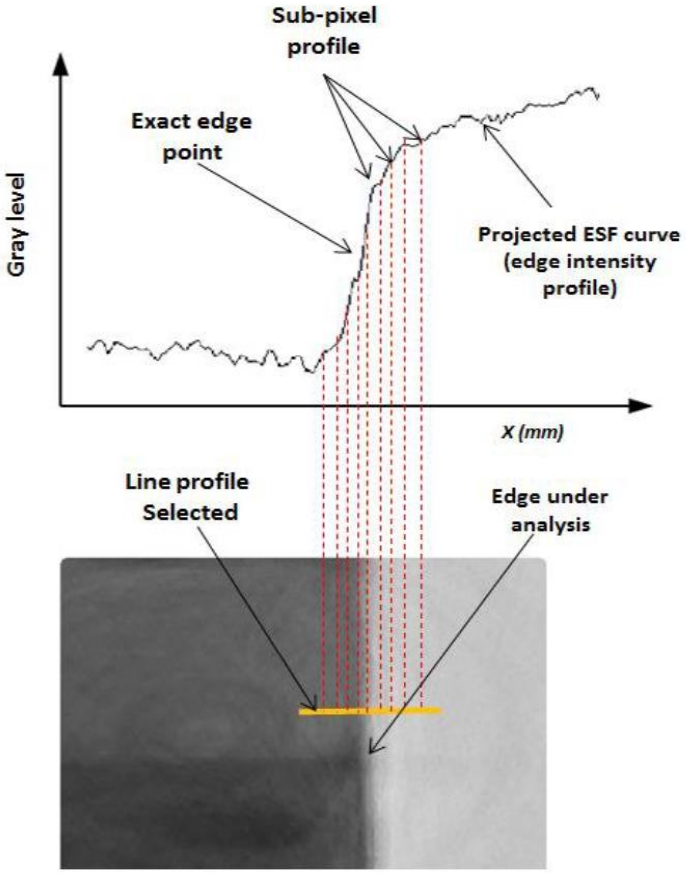

Figure 8. The calculation of the ESF [6]

Lastly, the rest part of the project is the restoration process. As stated earlier, Wiener filtering is a typical example of a simple deconvolution method to restore the image. Hence, the wiener filter is used for the restoration process.

Wiener filter

The Wiener filter is an MSE (Mean squared error) optimal stationary linear filter for images corrupted by blurring and noise. Calculating the Wiener filter requests that the noise and signal processes are 2nd order stationery [4]. Here is the formula for the Wiener filter used in this project.

$$Wiener(u,v) = \frac{1}{H(u,v)} \frac{|H(u,v)|^2}{|H(u,v)|^2 + \gamma} \quad (2)$$

In Eq. (2), γ is the ratio between the spectra of the noise and the original image, $H(u,v)$ is the MTF, and u and v are the spatial frequency coordinates. Later, the obtained Wiener filter is multiplied by the FFT of the degraded image $G(u,v)$, and the inverse FFT of the outcome is taken.

$$\hat{f}(m,n) = \mathcal{F}^{-1}(G(u,v) * Wiener(u,v)) \quad (3)$$

where $\hat{f}(m,n)$ is the restored image.

In summary, in my first week of SP, I tried to plan the steps of the project by skimming and researching the related articles, papers, and reports. Then, I came up with a procedure for the project about the blind restoration of atmospherically degraded images.

1. Find the best step-edge region in the degraded image
2. Calculate the ESF of the best step-edge region
3. Regulate the ESF data to reduce noise
4. Differentiate the ESF data to acquire the LSF
5. Take the FFT of the LSF to obtain the MTF
6. Substitute the MTF data into the Wiener filter
7. Multiply the FFT of the degraded image with the obtained Wiener filter
8. Take the inverse FFT of the outcome
9. Restore the image using the acquired result

In Figure 9, here is the illustration of the summary for the restoration of the atmospherically degraded images.

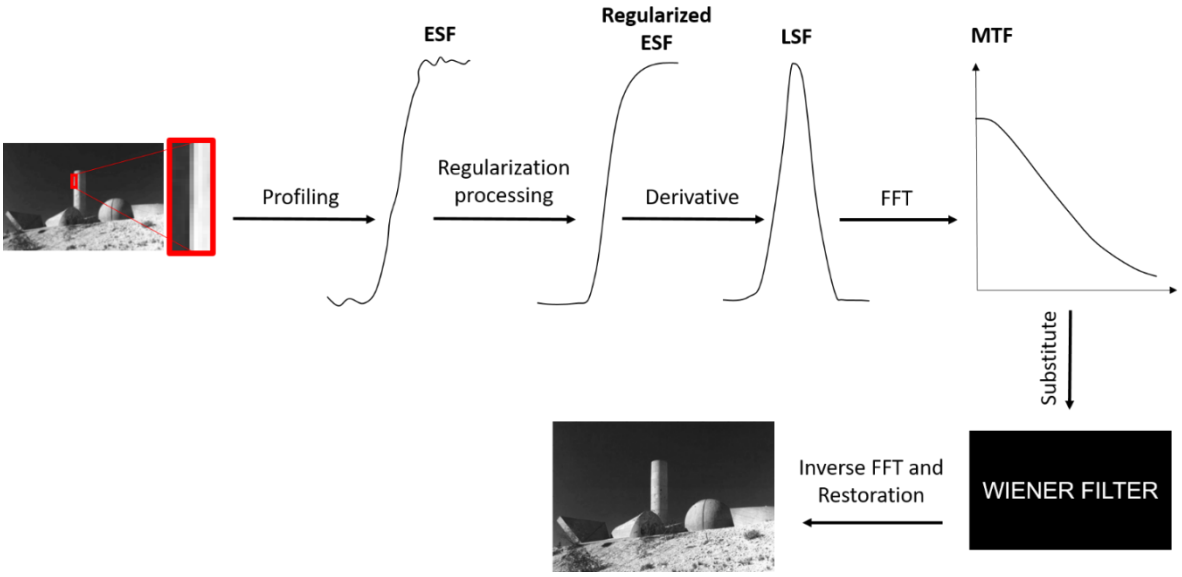

Figure 9. The planned steps of the project

The second week

In the second week of SP, I searched for papers and examples to learn how to obtain and regulate the ESF data from the best step-edge region using MATLAB. As I read the articles and examples, I found new techniques for noise reduction and getting more data from the image. Therefore, I decided to apply these techniques to the ESF process. The one used for getting more data from the image is named “the slanted-edge method.”

The slanted-edge method

Firstly, the projection line is defined as a line that crosses the edge line by 90 degrees. Then, the x-axis of the coordinate system for the method becomes the projection line, and the y-axis becomes the line where the edge is located. The method works best when the angle of the edge is between 3-5 degrees. Afterward, along the edge, the pixels’ x-coordinates are determined concerning the new coordinate system and saved to an array with their gray levels (intensities). The resulted data is the ESF. Next, the array is sorted regard to their x-coordinates from smallest to largest for plotting the ESF data. The advantage of this method is that the projection of the pixels onto the axis is not overlapping since the reference line for measuring distance is not vertical; thus, there are much more distinct data. In Figure 10, here is a comparison between the straight-edge method and the slanted-edge method.

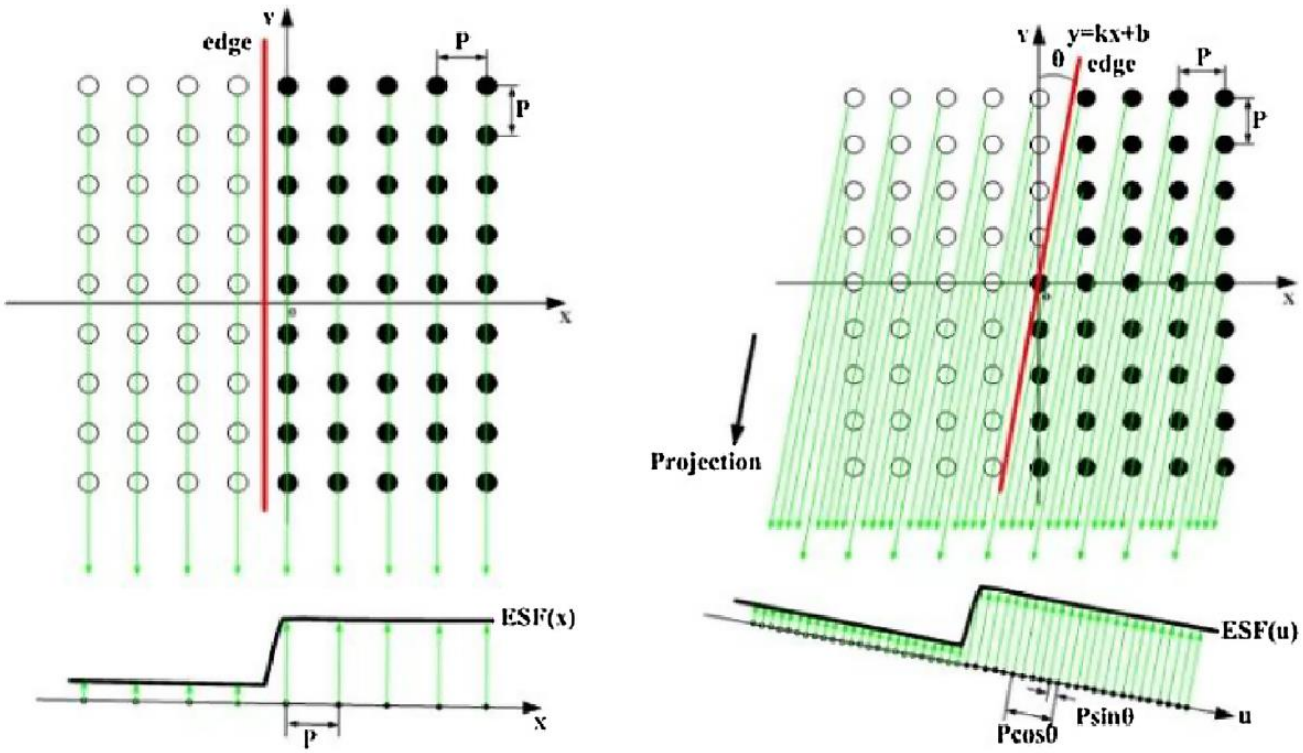

Figure 10. The comparison between the straight-edge method, and the slanted-edge method [5]

Also, the ones intended for noise reduction are named “the binning method” and “the curve fitting method.”

The binning method

First, the size of the bins should be selected between 0.03 and 0.15 since it provides a good trade-off between sampling uniformity and noise, especially 0.1 subpixel bin spacing. After that, the values are distributed to the bins according to their gray levels. In other words, a histogram regarding their gray values is created. Lastly, the means of the bins become the noise reduced ESF data.

The curve fitting method

In this method, I used the `polyfit()` function in MATLAB to fit the raw ESF data into a 9^{th}-degree polynomial equation since my supervisor advised me to do that.

Here is the plot of the ESF that used the slanted-edge method, the binning method, and the curve fitting method in Figure 11.

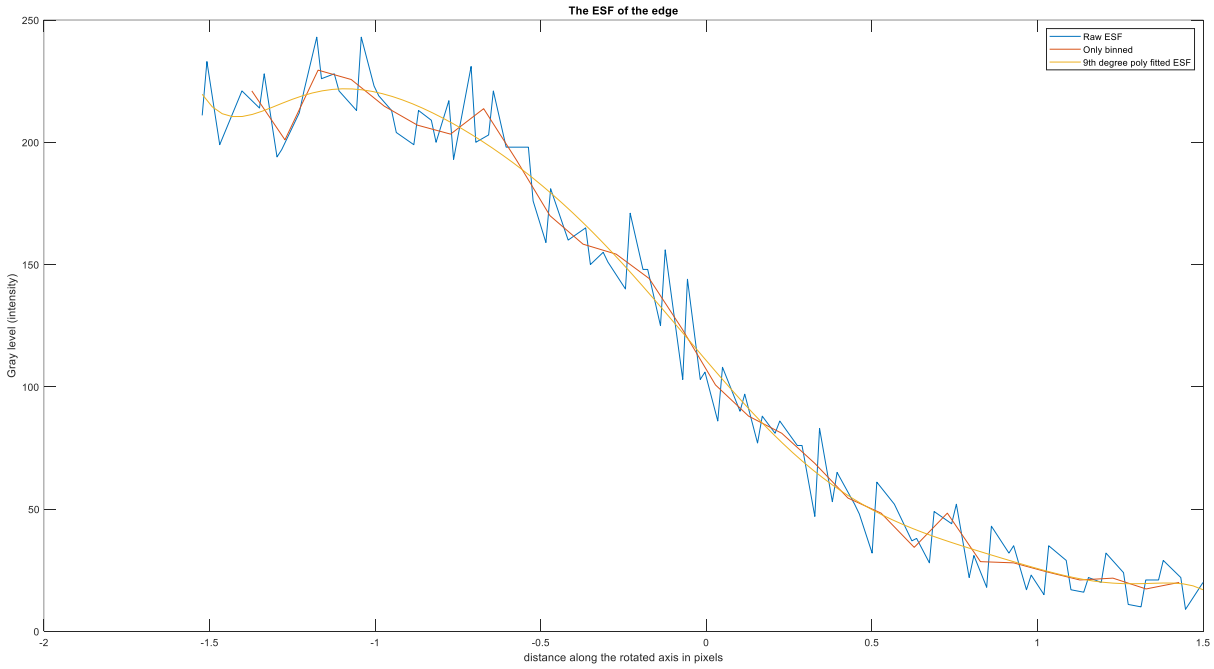

Figure 11. The ESF of the best step-edge region [5]

For more details, here is some of the MATLAB code I wrote for the slanted-edge part. Also, I benefited from the Slant Edge Script [5] written by Patrick Granton on September 2, 2010.

```
pixel_subdivision = 0.1;
% try to keep pixel_subdivision between 0.03 -> 0.15
% since it provides a good trade-off between sampling uniformity and noise,
% especially 0.1 subpixel bin spacing.
```

```
bin_pad = 0.0001;
% It adds a small space to include all values to the histogram
% of edge spread distances.
% transforming the coordinates to the new coordinate system
transformed_angle_radians = -(pi/2 - abs(angle_radians))*(angle_radians > 0) + ...
    (pi/2 - abs(angle_radians))*(angle_radians < 0);
transformed_edge_position = [x_column_pos*cos(transformed_angle_radians) +
y_row_pos*sin(transformed_angle_radians),...
                     x_column_pos*-sin(transformed_angle_radians) +
y_row_pos*cos(transformed_angle_radians)];

% offsetting and sorting the values regard to their x-coordinates
mean_trans_edge_position = mean(transformed_edge_position(:,1));
sorted_edge_position_plus_value = sortrows(cat(2, transformed_edge_position(:,1)-
mean_trans_edge_position, double(values)));

array_positions_of_edge = sorted_edge_position_plus_value(:,1);
array_values_of_edge = sorted_edge_position_plus_value(:,2);

%%% BINNING THE ESF %%%
% Determine bin spacing
topEdge = max(array_positions_of_edge) + bin_pad + pixel_subdivision;
botEdge = min(array_positions_of_edge) - bin_pad;
binEdges = botEdge:pixel_subdivision:topEdge;
numBins = length(binEdges) - 1;
binPositions = binEdges(1:end-1) + 1/2*pixel_subdivision;
binMean = zeros(1,length(numBins)); % preallocation

% enumarating the values according to their bins
which_bin = discretize(array_positions_of_edge, binEdges);

for i = 1:numBins
    flagBinMembers = (which_bin == i);
    binMembers = array_values_of_edge(flagBinMembers);
    binMean(i) = mean(binMembers);
end

ESF_raw = sorted_edge_position_plus_value(:,2);
xESF_raw = sorted_edge_position_plus_value(:,1);

ESF_bin = binMean(2:numBins - 1); % Eliminate first, second and last array
position
xESF_bin = binPositions(2:numBins - 1); % same as above comment

% removing NaN values
ESF_bin_nonan = ESF_bin(logical(1-isnan(ESF_bin)));
xESF_bin_nonan = xESF_bin(logical(1-isnan(ESF_bin)));

% filling the missing data
ESF_bin = interp1(xESF_bin_nonan,ESF_bin_nonan,xESF_bin,'pchip');
xESF_bin = xESF_bin_nonan;

% fitting the ESF
fit_ESF_P = polyfit(xESF_raw,ESF_raw,9);
fit_xESF = linspace(min(xESF_raw),max(xESF_raw),length(xESF_raw));
fit_ESF = polyval(fit_ESF_P,fit_xESF);
```

The third week

In the third week of SP, I continued to code to attain the LSF and MTF of the regulated ESF using MATLAB. In addition, with the help of my supervisor, I also regulated the LSF using Hamming window to obtain a better MTF in terms of data uniqueness.

To obtain the LSF, I need to take the derivative of the ESF data. To do that, I use the `diff()` function of MATLAB. Here is the code that I wrote for the differentiation of the ESF data.

```
% Differentiating the ESF to get LSF
LSF_raw = abs(diff(ESF_raw));
LSF_bin = abs(diff(ESF_bin));
LSF = abs(diff(fit_ESF));

% Removing the last element since diff() outputs in the one difference size of
% original array
xLSF_raw = xESF_raw(1:end-1);
xLSF_bin = xESF_bin(1:end-1);
xLSF = fit_xESF(1:end-1);
```

Then, I choose two points to define a region in the LSF data to apply the Hamming window method. Next, I multiply the hamming numbers with the values in the defined region.

```
% Choosing two points to determine the region for applying the Hamming window
f = figure("Name", "Binned LSF");
plot(xLSF_bin,LSF_bin);
[px,~] = getpts(f);
x1 = find(xLSF_bin > px(1)-0.1 & xLSF_bin < px(1)+0.1);
x2 = find(xLSF_bin > px(2)-0.1 & xLSF_bin < px(2)+0.1);
close(f);

temp = LSF_bin; % To save the original binned ESF data

% Applying hamming window method to the LSF that uses binned ESF
LSF_hamming = LSF_bin(x1:x2);
LSF_hamming = LSF_hamming .* hamming(length(LSF_hamming))';
LSF_ham = LSF_bin;
LSF_ham(x1:x2) = LSF_hamming;

LSF_bin = temp;
```

Later, I normalize the LSF not to possess big numbers in the MTF.

```
% Normalize the LSF
LSF = LSF/sum(LSF);
LSF_raw = LSF_raw/sum(LSF_raw);
LSF_bin = LSF_bin/sum(LSF_bin);
LSF_ham = LSF_ham/sum(LSF_ham);
```

Here is the plot of the LSF that used the raw ESF, the binned ESF, the hamming window, and the fitted ESF in Figure 12.

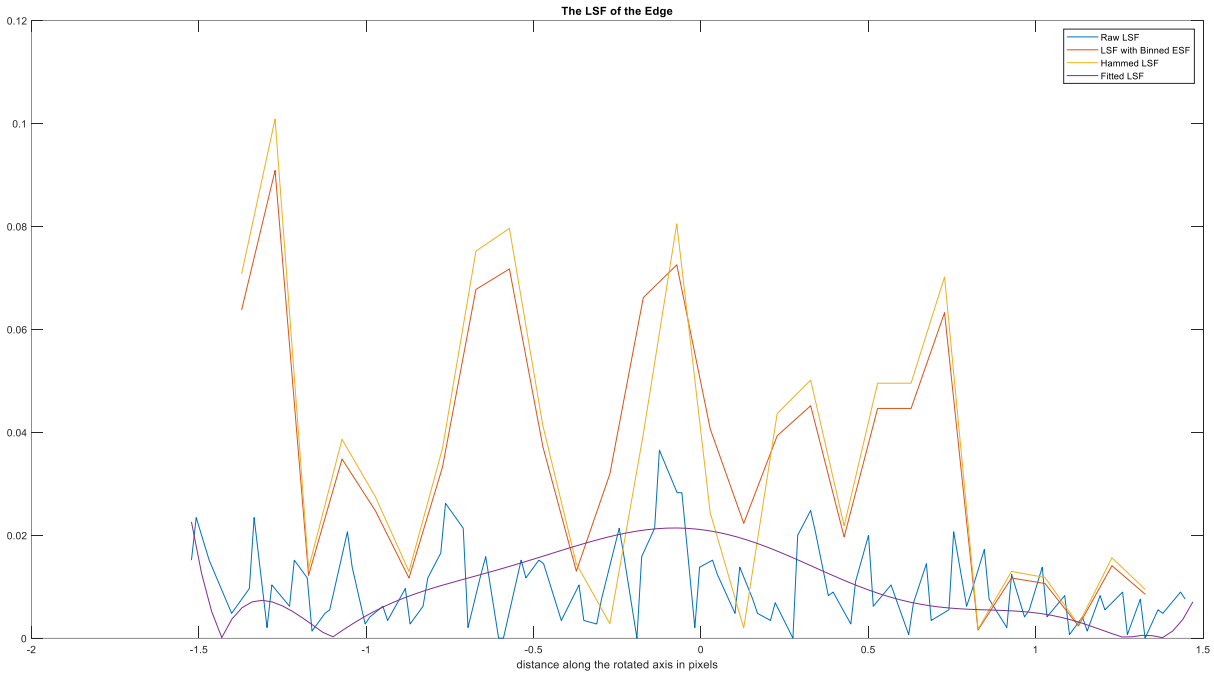

Figure 12. The LSF of the edge

To acquire the MTF, I need to take the FFT (Fast-Fourier Transform) of the LSF data. To do that, I use the `fft()` function of MATLAB. First, I defined the sampling of the FFT, and chose the sampling number N as 4096 to create smooth data.

```
N = 4096; % sample number

% generate the frequency axis
if mod(N,2)==0
    q = -N/2:N/2-1; % N even
else
    q = -(N-1)/2:(N-1)/2; % N odd
end

% Convert the spatial domain to frequency domain
Fs = 1/(pixel_subdivision*isotropicpixelspacing); % sampling rate in samples/mm
freq = q*Fs/N;

%%% Generating the MTF %%%
MTF_raw = abs(fftshift(fft(LSF_raw,N)))/max(max(abs(fftshift(fft(LSF_raw,N)))));
MTF_bin = abs(fftshift(fft(LSF_bin,N)))/max(max(abs(fftshift(fft(LSF_bin,N)))));
MTF_ham = abs(fftshift(fft(LSF_ham,N)))/max(max(abs(fftshift(fft(LSF_ham,N)))));
MTF = abs(fftshift(fft(LSF,N)))/max(max(abs(fftshift(fft(LSF,N)))));
```

Here is the plot of the MTF that used the raw LSF, the binned ESF, the hammed LSF, and the fitted ESF in Figure 13.

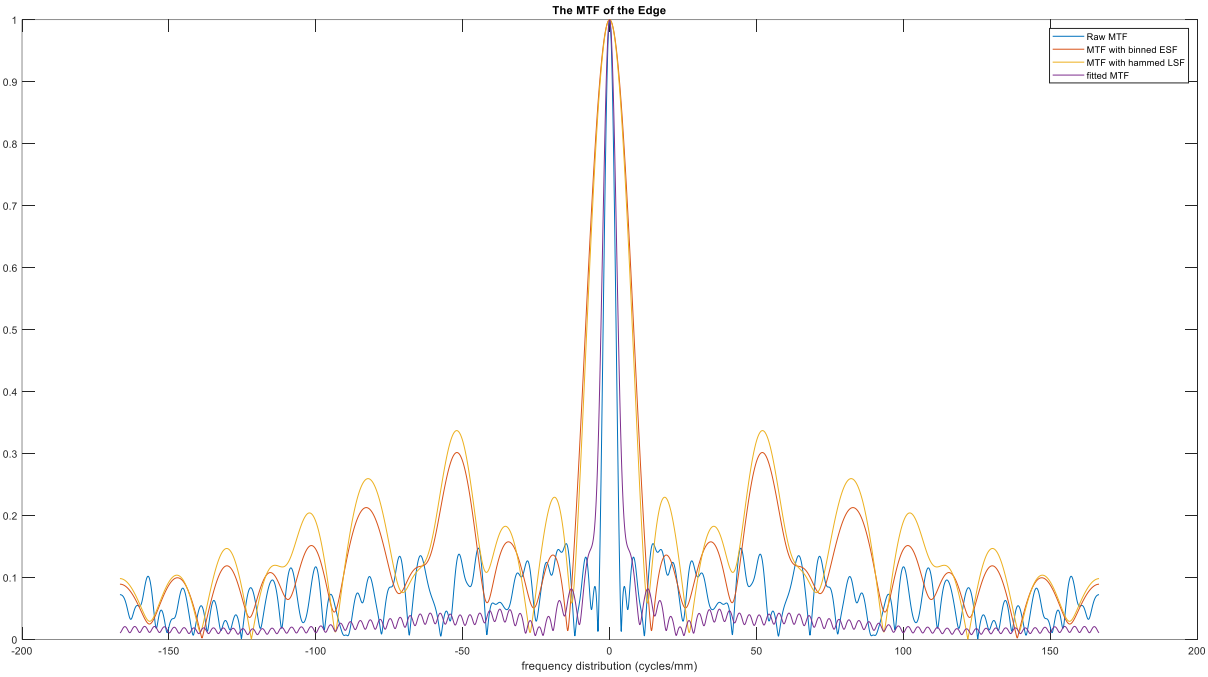

Figure 13. The MTF of the edge

In addition, here is the plot of the MTF that zoomed to the center frequency in Figure 14.

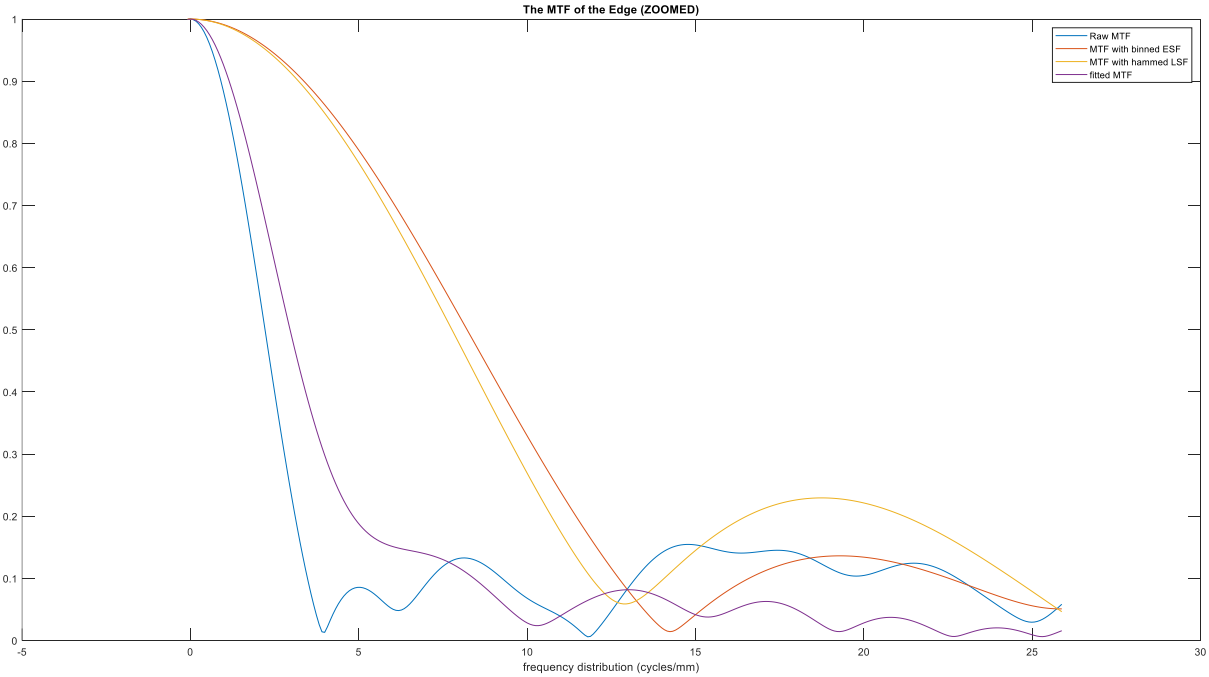

Figure 14. The MTF of the edge (Zoomed)

The fourth week

In the fourth week of SP, I began to code the restoration part of the project, which is taking the FFT of the degraded image and the MTF, then taking the inverse FFT of the multiplication of those two results using MATLAB. Here is a code for restoring the image by the Wiener filter.

```
gamma = 0.052; % manually selected

% Getting rid of the symmetric part of the MTF
shortaxis = round(length(MTF)/2):round(length(MTF)/2)+img_col-1;
MTF = MTF(shortaxis);

magnitude_MTF = MTF.*conj(MTF);
wiener = magnitude_MTF./(magnitude_MTF + gamma)./MTF;

% Applying the wiener filter to the all rows of the image
F = fftshift(fft2(image_full));
restored_image = ifft2(ifftshift(F.*wiener));
restored_image = sqrt(restored_image.*conj(restored_image));

min_value = double(min(min(image_full_raw)));
max_value = double(max(max(image_full_raw)));
restored_image = uint8(rescale(restored_image,min_value,max_value));
```

The restored images using Wiener filters with different MTFs are given in Figure 15-16-17-18. The MTFs used are named “Raw MTF” which uses raw ESF, no regularized LSF; “MTF with binned ESF” which uses binned ESF, no regularized LSF; “MTF with hammed LSF” which uses binned ESF, hammed LSF; “fitted MTF[1]” which uses fitted ESF, no regularized LSF. Additionally, the MTFs of the restored images are shown in Figure 19 for testing purposes.

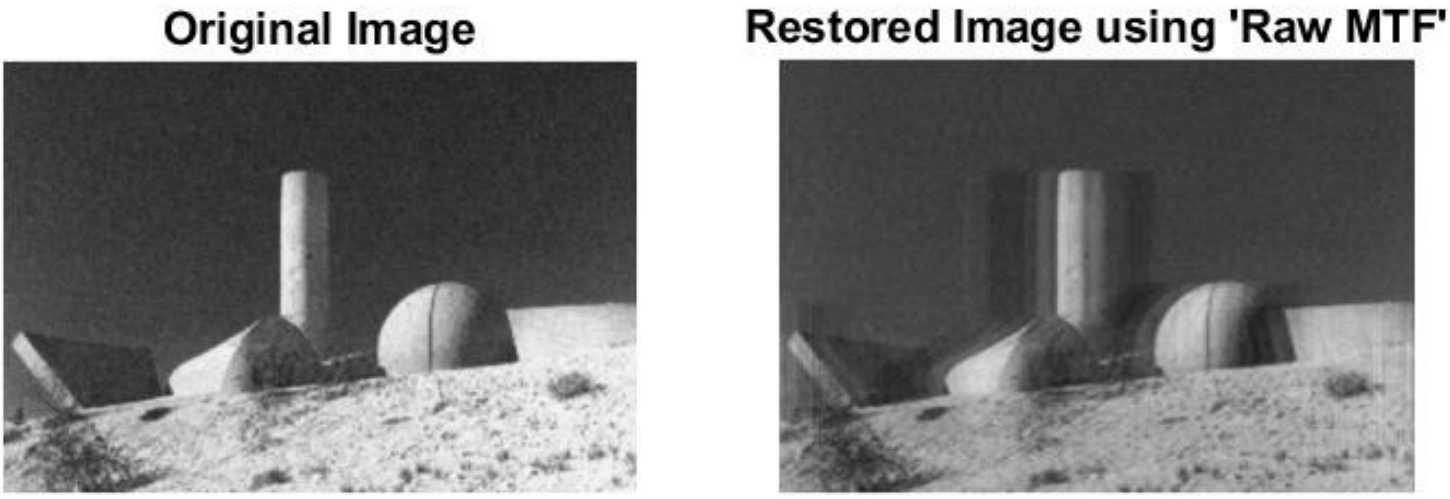

Figure 15. The Original Image vs. Restored Image using “Raw MTF”

[1] “fitted” word in “fitted MTF” does not mean that a curve fitting is applied to the MTF. It is for abbreviation purposes.

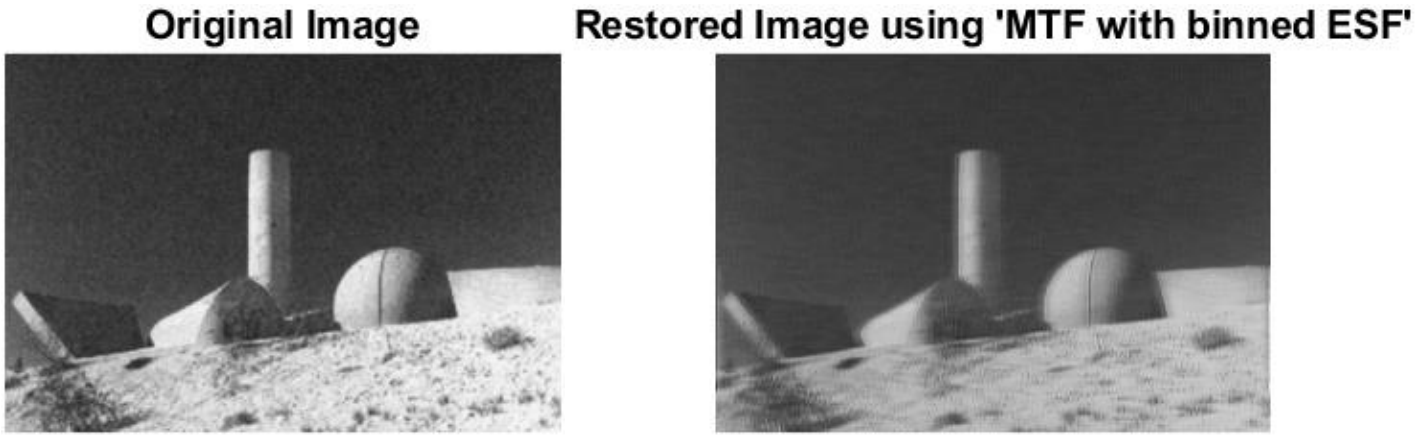

Figure 16. The Original Image vs. Restored Image using “MTF with binned ESF”

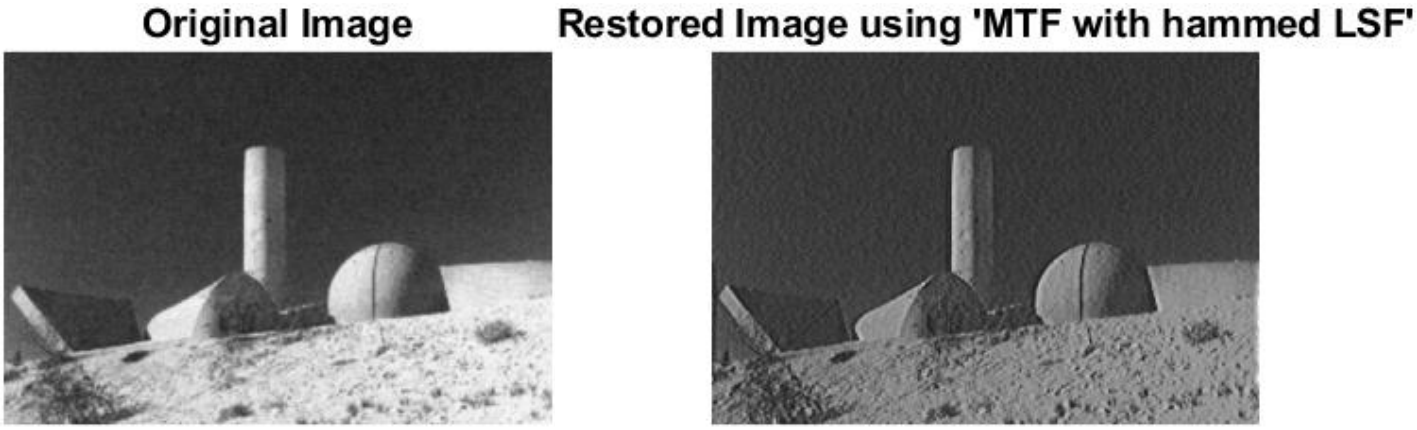

Figure 17. The Original Image vs. Restored Image using “MTF with hammed LSF”

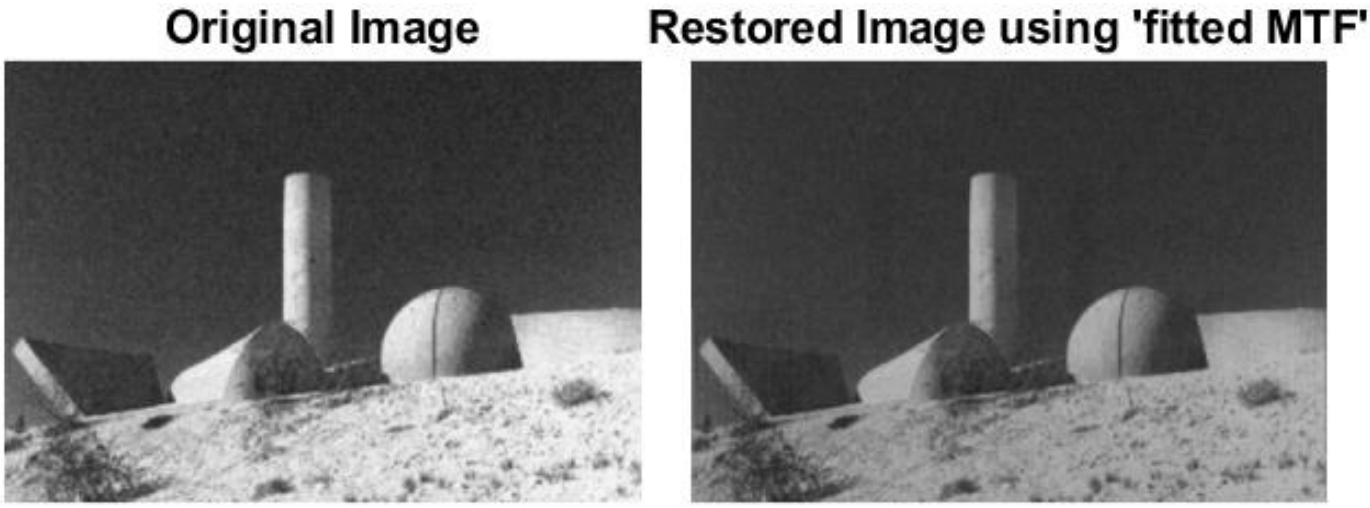

Figure 18. The Original Image vs. Restored Image using “fitted MTF”

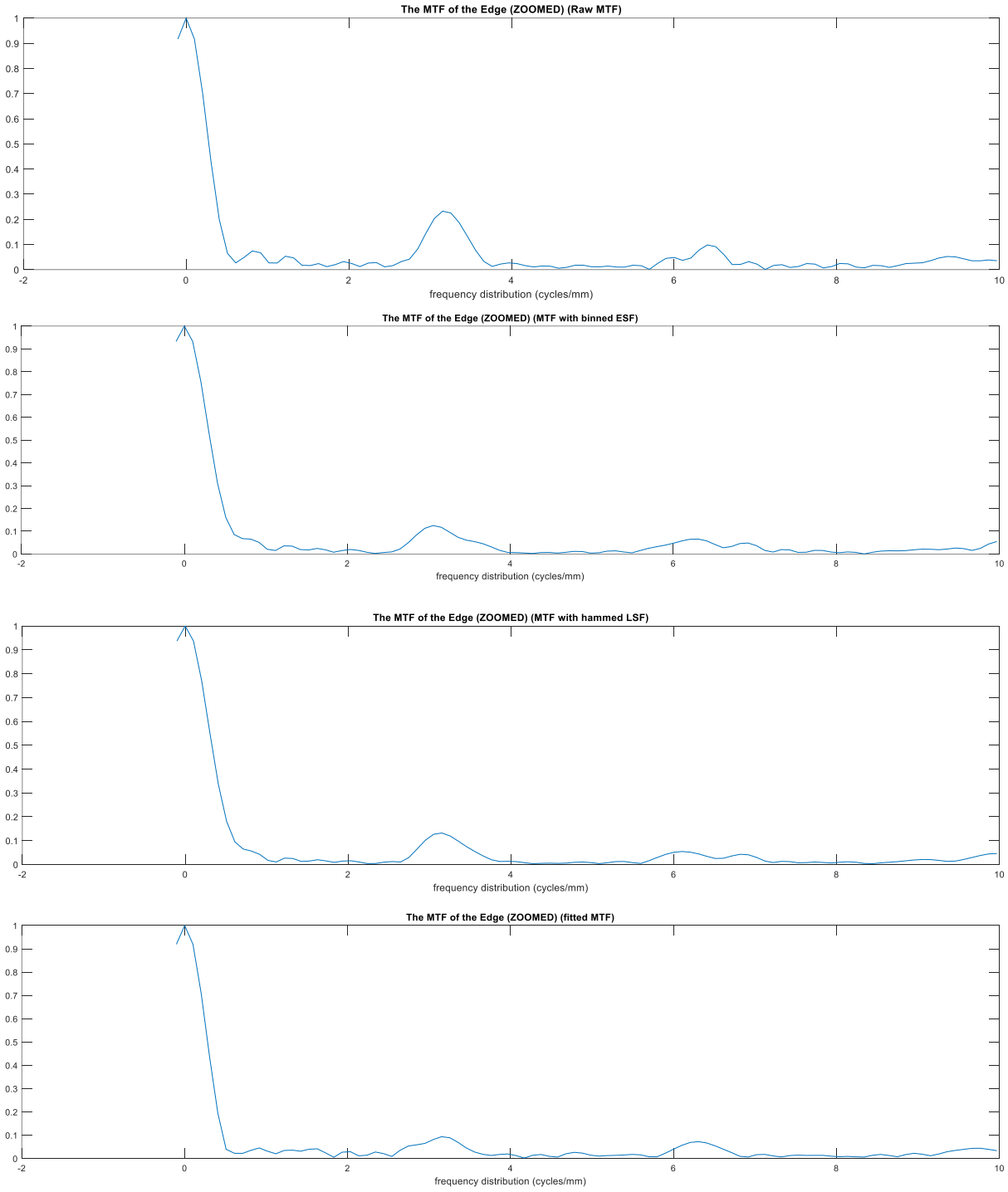

Figure 19. The MTF plots of all Wiener filters

There is not much of a difference between the MTF plots. The best ones in terms of the number of data that is not close to zero are the “MTF with hammed LSF” and “MTF with binned ESF” since they give much detail at higher spatial frequency. As you can see, the restored image using “MTF with hammed LSF” is the best-looking picture regarding image degradations.

Conclusions

In TÜBİTAK BİLGEM İLTAREN, I learned lots of technical knowledge about the theory and the use of MATLAB regarding image processing.

In the first week, I understood the basic concepts of the project. Such as, the MTF of an image can keep contrast and resolution in one data so that one can restore a picture utilizing a Wiener filter with the help of MTF. In the second week, I tried to obtain and regulate the ESF data from the best step-edge region with the slanted edge and binning method. In the third week, I tried to acquire the LSF and MTF of the regulated ESF and regulated the LSF using Hamming window to obtain a better MTF in terms of data uniqueness. Lastly, in the fourth week, I completed the restoration part of the project: taking the FFT of the degraded image and the MTF, then taking the inverse FFT of the multiplication of those two results using MATLAB.

Apart from technical knowledge, I did not gain considerable experience in leadership or teamwork since I was the only intern in that department.

About the company's quality and work discipline are not excellent or poor. The seminars were sufficiently prepared; however, the voice quality of the microphone was so poor that there was no word that could be comprehendible, and there was no progress on this occasion. Also, the number of employees who could set an example regarding working time discipline was few in the company.

I recommend this company if you are going to work in a software division. Although the company mainly focuses on the hardware area, the hardware team barely pays attention to interns.

References

[1] O. Shacham, O. Haik and Y. Yitzhaky, "Blind restoration of atmospherically degraded images by automatic," *Pattern Recognition Letters*, vol. 28, no. 15, pp. 2094-2103, 2007.

[2] G. BOSTAN, P. E. STERIAN, T. NECSOIU, A. P. BOBEI and C. D. SARAFOLEANU, "The slanted-edge method application in testing the optical resolution of a vision system," *Journal of Optoelectronics and Advanced Materials*, vol. 21, no. 1-2, pp. 22-34, 2019.

[3] "Introduction to modulation transfer function: Edmund optics," Edmund Optics Worldwide, [Online]. Available: https://www.edmundoptics.com/knowledge-center/application-notes/optics/introduction-to-modulation-transfer-function/. [Accessed 18 August 2022].

[4] T. Veldhuizen, "The Wiener filter," The University of Edinburgh, 16 January 1998. [Online]. Available: https://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/VELDHUIZEN. [Accessed 18 August 2022].

[5] P. Granton, "Slant Edge Script," MathWorks, 2 September 2010. [Online]. Available: https://www.mathworks.com/matlabcentral/mlc-downloads/downloads/submissions/28631/versions/1/previews/MATHWORKS_MTF/MTFscript.m/index.html. [Accessed 19 August 2022].

[6] A. M. Alshweikh, K. Kusminarto and G. B. Suparta, "An Improved Method of Measuring Spatial Resolution of the Computed Tomography from ESF based on CT phantom images," *Research India Publications.*, vol. 13, no. 15, pp. 12318-12325, 2018.

Appendix

The Project Code

```
%{
 Restoration Image Script
 File: restore_image_script.m
 Written by: Abdullah Emir Gogusdere,
 August 19, 2022
 Contact e-mail, abdullah.gogusdere@metu.edu.tr
%}

% This script benefits from the "Slant Edge Script" by Patrick Granton.
% For more details, refer to the reference at the end of the file.

close all; clearvars;

isotropicpixelspacing = 0.03;
% isotropic detector pixel spacing in mm, (i.e. pixel pitch).
% set this value to your detector

pixel_subdivision = 0.08;
% try to keep pixel_subdivision between 0.03 -> 0.15
% since it provides a good trade-off between sampling uniformity and noise,
% especially 0.1 subpixel bin spacing.

bin_pad = 0.0001;
% It adds a small space to include all values to the histogram
% of edge spread distances.

boundplusminus = 4;
% boundplusminus is a variable that is used to crop a small section of the
% edge in order to used to find the subpixel interpolated edge position.

boundplusminus_extra = 1;
% boundplusminus_extra incorperates addition pixel values near the edge to
% include in the binned histogram.

image_path = "test_image.JPG";
[fPath, fName, fExt] = fileparts(image_path);

switch lower(fExt)
  case '.tif'
    image = imread(image_path);
    image = image(:,:,1:3);
    image = rgb2gray(image);
%     For noise testing
%     image = imnoise(image,'gaussian');
%     image = imgaussfilt(image, 2);
    image_full = image;
    image_full_raw = image;
  case {'.jpg','.png'}
    image = rgb2gray(imread(image_path));
%     For noise testing
    image = imnoise(image,'gaussian');
    image = imgaussfilt(image, 2);
    image_full = image;
    image_full_raw = image;
  otherwise
    error('Unexpected file extension: %s', fExt);
end
```

```
% For selecting the best step-edge region interactively.
% figure("Name", "Select a rectangular region, and double click on it.");
% [image,rect] = imcrop(image);
% rect

rect = [434.5100  203.5100   15.9800  132.9800];
crop_col_leftpoint = round(rect(1)); crop_row_leftpoint = round(rect(2));
crop_collength = round(rect(3)); crop_rowlength = round(rect(4));

image = image_full(crop_row_leftpoint:crop_row_leftpoint+crop_rowlength...
    ,crop_col_leftpoint:crop_col_leftpoint+crop_collength);

[img_rowlength, img_columnlength] = size(image);

level = graythresh(image); % Determinig the threshold of the canny edge
detector

% Detect edge and orientation
BW_edge_raw = edge(double(image),'canny', level);

% Locate edge positions
[y_row_pos, x_column_pos] = find(BW_edge_raw==1);

% Fit edge positions
P = polyfit(x_column_pos,y_row_pos,1); % mx + b = y

% determine rough edge angle to determine orientation
angle_radians = atan(P(1));

if abs(angle_radians) > pi/4 % i.e. edge is vertical
    start_row = boundplusminus_extra;
    end_row = img_rowlength - boundplusminus_extra;
    BW_mask = false(img_rowlength,img_columnlength);

    roi = zeros(end_row-start_row+1, 3);
    counter = 1;

    for i = start_row:end_row
        [BW_y_row, BW_x_col] = find(BW_edge_raw(i,:)==1);

        index_start = min(BW_x_col)-boundplusminus;
        index_end = max(BW_x_col)+boundplusminus;
        if index_start <= 0
            index_start = 1;
        end
        if index_end > img_columnlength
            index_end = img_columnlength;
        end

        BW_mask(i,index_start:index_end) = 1;
        roi(counter,:) = [i,index_start,index_end];
        counter = counter + 1;
    end

    [y_row_pos, x_column_pos, values] = find(image.*uint8(BW_mask));

else % the edge is horizontal
    start_col = boundplusminus_extra;
    end_col = img_columnlength - boundplusminus_extra;
```

```
    BW_mask = false(img_rowlength,img_columnlength);

    roi = zeros(end_col-start_col+1, 3);
    counter = 1;
    for i = start_col:end_col
        [BW_y_row, BW_x_col] = find(BW_edge_raw(:,i)==1);
        index_start = min(BW_y_row)-boundplusminus;
        index_end = max(BW_y_row)+boundplusminus;
        if index_start <= 0
            index_start = 1;
        end
        if index_end > img_rowlength
            index_end = img_rowlength;
        end
        BW_mask(index_start:index_end,i) = 1;
        roi(counter,:) = [index_start,index_end,i];
        counter = counter + 1;
    end

    % Locate edge positions
    [y_row_pos, x_column_pos, values] = find(image.*uint8(BW_mask));

    % Fit edge positions
    P = polyfit(x_column_pos,y_row_pos,1);
    angle_radians = atan(P(1));
end

% Visualizing the area going to be processed
figure;
subplot(1,2,1);
imshow(image);
title("The image of the edge")
subplot(1,2,2);
imshow(uint8(BW_mask)*255);
title("the edge area going to be processed")

% transforming the coordinates to the new coordinate system
transformed_angle_radians = -(pi/2 - abs(angle_radians))*(angle_radians >
0) + ...
    (pi/2 - abs(angle_radians))*(angle_radians < 0);
transformed_edge_position = [x_column_pos*cos(transformed_angle_radians) +
y_row_pos*sin(transformed_angle_radians),...
                    x_column_pos*-sin(transformed_angle_radians) +
y_row_pos*cos(transformed_angle_radians)];

% offsetting and sorting the values regard to their x-coordinates
mean_trans_edge_position = mean(transformed_edge_position(:,1));
sorted_edge_position_plus_value = sortrows(cat(2,
transformed_edge_position(:,1)-mean_trans_edge_position, double(values)));

array_positions_of_edge = sorted_edge_position_plus_value(:,1);
array_values_of_edge = sorted_edge_position_plus_value(:,2);

% Determine bin spacing
topEdge = max(array_positions_of_edge) + bin_pad + pixel_subdivision;
botEdge = min(array_positions_of_edge) - bin_pad;
binEdges = botEdge:pixel_subdivision:topEdge;
numBins = length(binEdges) - 1;
binPositions = binEdges(1:end-1) + 1/2*pixel_subdivision;
binMean = zeros(1,length(numBins)); % preallocation
```

```
% enumarating the values according to their bins
which_bin = discretize(array_positions_of_edge, binEdges);

for i = 1:numBins
    flagBinMembers = (which_bin == i);
    binMembers = array_values_of_edge(flagBinMembers);
    binMean(i) = mean(binMembers);
end

ESF_raw = sorted_edge_position_plus_value(:,2);
xESF_raw = sorted_edge_position_plus_value(:,1);

ESF_bin = binMean(2:numBins - 1); % Eliminate first, second and last array
position
xESF_bin = binPositions(2:numBins - 1); % same as above comment

% removing NaN values
ESF_bin_nonan = ESF_bin(logical(1-isnan(ESF_bin)));
xESF_bin_nonan = xESF_bin(logical(1-isnan(ESF_bin)));

% filling the missing data
xESF_bin = xESF_bin_nonan;
ESF_bin = interp1(xESF_bin_nonan,ESF_bin_nonan,xESF_bin,'pchip');


fit_ESF_P = polyfit(xESF_raw,ESF_raw,9);
fit_xESF = linspace(min(xESF_raw),max(xESF_raw),length(xESF_raw));
fit_ESF = polyval(fit_ESF_P,fit_xESF);

figure;
plot(xESF_raw, ESF_raw, xESF_bin, ESF_bin, fit_xESF, fit_ESF);
title('The Binned ESF of the edge')
xlabel('distance along the rotated axis in pixels')
ylabel('Gray level (intensity)')
legend("Raw ESF", "Only binned", "9th degree poly fitted ESF")

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

% Differentiating the ESF to get LSF
LSF_raw = abs(diff(ESF_raw));
LSF_bin = abs(diff(ESF_bin));
LSF = abs(diff(fit_ESF));

% Removing the last element since diff() outputs in the one difference size
of
% original array
xLSF_raw = xESF_raw(1:end-1);
xLSF_bin = xESF_bin(1:end-1);
xLSF = fit_xESF(1:end-1);

% Choosing two points to determine the region for applying the Hamming
window
f = figure("Name", "Select two points on the graph of the Binned LSF");
plot(xLSF_bin,LSF_bin);
[px,~] = getpts(f);
x1 = find(xLSF_bin > px(1)-0.1 & xLSF_bin < px(1)+0.1);
x2 = find(xLSF_bin > px(2)-0.1 & xLSF_bin < px(2)+0.1);
close(f);
```

```
temp = LSF_bin; % To save the original binned ESF data

% Applying hamming window method to the LSF that uses binned ESF
LSF_hamming = LSF_bin(x1:x2);
LSF_hamming = LSF_hamming .* hamming(length(LSF_hamming))';
LSF_ham = LSF_bin;
LSF_ham(x1:x2) = LSF_hamming;

LSF_bin = temp;

% Normalize the LSF
LSF = LSF/sum(LSF);
LSF_raw = LSF_raw/sum(LSF_raw);
LSF_bin = LSF_bin/sum(LSF_bin);
LSF_ham = LSF_ham/sum(LSF_ham);

figure;
plot(xLSF_raw,LSF_raw,...
     xLSF_bin,LSF_bin,...
     xLSF_bin,LSF_ham,...
     xLSF,LSF);
title('The LSF of the Edge')
legend('Raw LSF', 'LSF with Binned ESF', 'Hammed LSF', 'Fitted LSF')
xlabel('distance along the rotated axis in pixels')

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

N = 4096; % sample number

% generate the frequency axis
if mod(N,2)==0
    q = -N/2:N/2-1; % N even
else
    q = -(N-1)/2:(N-1)/2; % N odd
end

% Convert the spatial domain to frequency domain
Fs = 1/(pixel_subdivision*isotropicpixelspacing); % sampling rate in
samples per mm
freq = q*Fs/N;

%%% Generating the MTF %%%
MTF_raw =
(abs(fftshift(fft(LSF_raw,N)))/max(max(abs(fftshift(fft(LSF_raw,N))))))';
MTF_bin =
abs(fftshift(fft(LSF_bin,N)))/max(max(abs(fftshift(fft(LSF_bin,N)))));
MTF_ham =
abs(fftshift(fft(LSF_ham,N)))/max(max(abs(fftshift(fft(LSF_ham,N)))));
MTF = abs(fftshift(fft(LSF,N)))/max(max(abs(fftshift(fft(LSF,N)))));

all_MTF = [MTF_raw;MTF_bin;MTF_ham;MTF];
str_MTF = ["Raw MTF", "MTF with binned ESF", "MTF with hammed LSF", "fitted
MTF"];

figure;
plot(freq,MTF_raw,freq,MTF_bin,freq,MTF_ham,freq,MTF);

title('The MTF of the Edge')
```

```
legend('Raw MTF', 'MTF with binned ESF', 'MTF with hammed LSF', 'fitted
MTF')
xlabel('frequency distribution (cycles/mm)')

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

[~, img_col] = size(image_full);

figure;
shortaxis = round(length(MTF)/2)+1:round(length(MTF)/2)+img_col;

plot(freq(shortaxis),MTF_raw(shortaxis),freq(shortaxis),MTF_bin(shortaxis),
...
    freq(shortaxis),MTF_ham(shortaxis),freq(shortaxis),MTF(shortaxis));

title('The MTF of the Edge (ZOOMED)')
legend('Raw MTF', 'MTF with binned ESF', 'MTF with hammed LSF', 'fitted
MTF')
xlabel('frequency distribution (cycles/mm)')

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
gamma = 0.052; % manually selected
image_array = [];

figure;
imshow(image_full_raw);
title("Original Image")

for i=1:size(all_MTF,1)

    image_full = image_full_raw; % Resetting image_full

    %%%% Preparing the wiener filter %%%%

    MTF = all_MTF(i,:);

    % Getting rid of the symmetric part of the MTF
    shortaxis = round(length(MTF)/2)+1:round(length(MTF)/2)+img_col;
    MTF = MTF(shortaxis);

    magnitude_MTF = MTF.*conj(MTF);
    wiener = magnitude_MTF./(magnitude_MTF + gamma)./MTF;

    % Applying the wiener filter to the all rows of the image
    F = fftshift(fft2(image_full));
    restored_image = ifft2(ifftshift(F.*wiener));
    restored_image = sqrt(restored_image.*conj(restored_image));

    min_value = double(min(min(image_full_raw)));
    max_value = double(max(max(image_full_raw)));
    restored_image = uint8(rescale(restored_image,min_value,max_value));
    image_full = restored_image;
    image_array = cat(3,image_array,image_full);
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%%%% Measuring the MTF of the restored images %%%%

all_MTF_reconstructed = zeros(size(image_array,3),N);

for x=1:size(image_array,3)
    image_full = image_array(:,:,x);

    image =
image_full(crop_row_leftpoint:crop_row_leftpoint+crop_rowlength...
        ,crop_col_leftpoint:crop_col_leftpoint+crop_collength);

    [img_rowlength, img_columnlength] = size(image);

    level = graythresh(image); % Determine the threshold of the canny edge
detector

    % Detect edge and orientation
    BW_edge_raw = edge(double(image),'canny', level);

    % Locate edge positions
    [y_row_pos, x_column_pos] = find(BW_edge_raw==1);

    % Fit edge positions
    P = polyfit(x_column_pos,y_row_pos,1); % mx + b = y

    % determine rough edge angle to determine orientation
    angle_radians = atan(P(1));

    if abs(angle_radians) > pi/4 % i.e. edge is vertical
        start_row = boundplusminus_extra;
        end_row = img_rowlength - boundplusminus_extra;
        BW_mask = false(img_rowlength,img_columnlength);

        roi = zeros(end_row-start_row+1, 3);
        counter = 1;

        for i = start_row:end_row
            [BW_y_row, BW_x_col] = find(BW_edge_raw(i,:)==1);

            index_start = min(BW_x_col)-boundplusminus;
            index_end = max(BW_x_col)+boundplusminus;
            if index_start <= 0
                index_start = 1;
            end
            if index_end > img_columnlength
                index_end = img_columnlength;
            end

            BW_mask(i,index_start:index_end) = 1;
            roi(counter,:) = [i,index_start,index_end];
            counter = counter + 1;
        end

        [y_row_pos, x_column_pos, values] = find(image.*uint8(BW_mask));

    else % the edge is horizontal
        start_col = boundplusminus_extra;
        end_col = img_columnlength - boundplusminus_extra;
        BW_mask = false(img_rowlength,img_columnlength);
```

```
        roi = zeros(end_col-start_col+1, 3);
        counter = 1;
        for i = start_col:end_col
            [BW_y_row, BW_x_col] = find(BW_edge_raw(:,i)==1);
            index_start = min(BW_y_row)-boundplusminus;
            index_end = max(BW_y_row)+boundplusminus;
            if index_start <= 0
                index_start = 1;
            end
            if index_end > img_rowlength
                index_end = img_rowlength;
            end
            BW_mask(index_start:index_end,i) = 1;
            roi(counter,:) = [index_start,index_end,i];
            counter = counter + 1;
        end

        % Locate edge positions
        [y_row_pos, x_column_pos, values] = find(image.*uint8(BW_mask));

        % Fit edge positions
        P = polyfit(x_column_pos,y_row_pos,1);
        angle_radians = atan(P(1));
    end
    % transforming the coordinates to the new coordinate system
    transformed_angle_radians = -(pi/2 - abs(angle_radians))*(angle_radians
> 0) + ...
        (pi/2 - abs(angle_radians))*(angle_radians < 0);
    transformed_edge_position =
[x_column_pos*cos(transformed_angle_radians) +
y_row_pos*sin(transformed_angle_radians),...
                        x_column_pos*-sin(transformed_angle_radians) +
y_row_pos*cos(transformed_angle_radians)];

    % offsetting and sorting the values regard to their x-coordinates
    mean_trans_edge_position = mean(transformed_edge_position(:,1));
    sorted_edge_position_plus_value = sortrows(cat(2,
transformed_edge_position(:,1)-mean_trans_edge_position, double(values)));

    array_positions_of_edge = sorted_edge_position_plus_value(:,1);
    array_values_of_edge = sorted_edge_position_plus_value(:,2);

    ESF_raw = sorted_edge_position_plus_value(:,2);
    xESF_raw = sorted_edge_position_plus_value(:,1);


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

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

    % Differentiating the ESF to get LSF
    LSF_raw = abs(diff(ESF_raw));

    % Removing the last element since diff() outputs in the one difference
size of
    % original array
    xLSF_raw = xESF_raw(1:end-1);

    % Normalize the LSF
    LSF_raw = LSF_raw/sum(LSF_raw);
```

```
    %%% Generating the MTF %%%
    MTF_raw =
(abs(fftshift(fft(LSF_raw,N)))/max(max(abs(fftshift(fft(LSF_raw,N))))))';
    all_MTF_reconstructed(x,:) = MTF_raw;
end

%%% Plotting the reconstructed images with their MTFs %%%%

[~, img_col] = size(image_full);
shortaxis = round(length(MTF_raw)/2)+1:round(length(MTF_raw)/2)+100;

for i=1:size(all_MTF_reconstructed,1)
    figure;
    subplot(1,2,1);
    imshow(image_full_raw);
    title("Original Image")
    subplot(1,2,2);
    imshow(image_array(:,:,i));
    title(sprintf("Restored Image using '%s'", str_MTF(i)))
end

for i=1:size(all_MTF_reconstructed,1)
    figure;
    plot(freq(shortaxis),all_MTF_reconstructed(i,shortaxis));
    title(sprintf('The MTF of the Edge (ZOOMED) (%s)',str_MTF(i)))
    xlabel('frequency distribution (cycles/mm)')
end

%{
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Reference:
P. Granton, "Slant Edge Script," MathWorks, 2 September 2010.
[Online]. Available: https://www.mathworks.com/matlabcentral/mlc-downloads
/downloads/submissions/28631/versions/1/previews/MATHWORKS_MTF/MTFscript.m
/index.html. [Accessed 19 August 2022].
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%}
```