# MATLAB Implementation of Computed Tomography

Muhammed Saadeddin Koçak
Electrical & Electronics Enginnering Department
Middle East Technical University
Ankara,Turkey
e223234@metu.edu.tr

***Abstract*—Computed Tomography is a widely used medical imaging modality. Computed Tomography utilizes ionizing radiation to display human anatomy. In this documentation a simple MATLAB implementation of computed tomography projection, back projection, and filtered back projection algorithm is demonstrated. Instead of using human body and ionizing radiation, random images and MATLAB is used to investigate Computed Tomography characteristics. Simulation of Computed Tomography in computer allowed applying different beam number, step number, and filter types without harming any living organism. Improvements on Computed Tomography may and must be tested on such simulations before clinical experiments.**

***Keywords—computed tomography, projection, back projection, filtered back projection, MATLAB, imaging, filtering,***

## I. INTRODUCTION

In December 22, 1895 Wilhelm K. Rontgen was probably not aware of the fact that he invented a groundbreaking medical imaging modality. However, he was aware that the X-rays had some interesting properties. Lighting the first fire Roentgen allowed other researchers to develop X-ray imaging modalities by publishing "A New Kind of Rays". After this publication handful of researchers started to develop X-rays both for medical and commercial use. Dr Edwin Frost was the first one using X-ray radiography for diagnostic purposes. E. Haschek and O. Lindenthal developed further diagnostic approaches such as contrast enhanced vein imaging using X-rays. On the other hand, researchers like Mihran Kassabian were observing the effects of X-rays on human tissues.

World War II caused a significant acceleration in both scientific and experimental works. With the help of mathematical background provided by Johann Radon, in 1967 Godfrey Hounsfield invented the first commercially viable Computed Tomography [1]. After this invention medical imaging reseacrhes have gained further acceleration and resulted in invention of modalities such as SPECT, PET and so on.

In this project, the aim was to implement a software-based simulation of Computed Tomography. Motivation behind this aim was allow a researcher to make development on Computed Tomography without harming any living organism. Furthermore, characteristics and limitations of Computed Tomography can be examined through a software-based simulation. With the help of MATLAB, projection, back projection, and filtered back projection algorithms are implemented and resulting projection data and back projected images are discussed in detail. Changing crucial variables such as beam number is experimented and results are noted.

## II. THEORY AND ALGORITHM

Here implemented algorithm will be explained using pseudocode and related background will be presented. Projection and back projection algorithms are explained separately although they are very similar and closely related because projection and back projection can be run independently. Indeed, back projection algorithm should obtain only projection matrix where projection algorithm takes only image data.

### *A. Projection*

Projection Program Starts

Load the image

%Here image that is to be projected is loaded and transformed to a matrix.

Obtain the size

Obtain beam and step number

%Beam and step number are decided by user. Indeed, these parameters are crucial for several reasons and explained in Discussion and Conclusion subsection

Create t, theta, x, and y coordinate arrays

%Mathematically taking a projection means calculating a line integral. In Computed Tomography this projection naturally occurs through X-ray beams. In the case of this project mathematical calculations should be done through (1). However, conversion to cartesian coordinates yields (2). Through (2) projection is calculated for each t and theta value.

$$p_{\theta}(t) = \int_{(\theta,t)line} f(x,y)\,\mathrm{d}s \tag{1}$$

$$\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(x,y)\,\delta(x cos\theta + y sin\theta - t)\,\mathrm{d}x\mathrm{d}y \tag{2}$$

Open two for loops to calculate projection for each t and theta pair

Clear dynamic memory

%Intersection points of rays with image should be found to calculate total attenuation for that particular beam.

Find intersection points using (3) and (4)

$$y = (t/sin(teta) - (x * cot(teta)) \tag{3}$$

$$x = (t/cos(teta) - (x * tan(teta)) \tag{4}$$

%Here it is important to note that some irrelevant x and y coordinates may be present since calculation of above equation is done for all space. Also, there may be duplicate x and y pairs since to calculation from (3) and (4) may yield same point twice.

Determine unique and relevant x and y pairs

Sort x and y pairs

If there is not a relevant point equate total attenuation for that beam to zero

%Above means beam does not go through image

If there is only one relevant point equate total attenuation for that beam to zero

%Above means beam is tangential to image

Calculate distance and middle points through (5) and (6)

$$dist = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \tag{5}$$

$$mid = \frac{x_1 + x_2}{2}, \frac{y_1 + y_2}{2} \tag{6}$$

%Distance calculation is needed for attenuation calculation. On the other hand, middle point calculations provide address of image pixel.

Correct address information using middle point data

Calculate total attenuation for a beam by summing multiplication of image attenuation data and distance passed through each pixel

Assign calculated attenuation for specific beam and step number in projection matrix

Projection Program Ends

### B. *Back Projection*

Back Projection Program Starts

Load projection matrix and size of image

%From projection matrix beam and step number can be found. Indeed, in this algorithm number of rows yield beam number where number of columns yield step number.

Obtain beam and step number

Filtering Program Starts

%Back projection is calculated according to (7)

$$f_b(x, y) = \iint_{0\,-\infty}^{\pi\,\infty} p_\Theta(t)\delta(x\cos\Theta + y\sin\Theta - t)\,dt\,d\Theta \tag{7}$$

In mathematical case, back projection corresponds to summing all projection values for each pixel of image. However, issue with back projection is obvious when impulse response of back projection algorithm is calculated [2]. To correct this issue back projection should be calculated according to (8).

$$f_b(x, y) = f(x, y) ** \frac{1}{r} \tag{8}$$

At this point using Fourier Transforms (9) is obtained.

$$F^{-1}\{\, F\{p_\Theta(t)\}\,.\,|\rho|\,\} = p_\Theta(t) * F^{-1}\{|\rho|\} \tag{9}$$

Meaning of all above is that before back projecting image a triangular filter such as shown in Fig. 17 should be used. There are several crucial points here. Firstly, filter should have infinite response on the edges however, in practical case it is impossible. Secondly, filtering should be done in frequency domain for computational simplicity. If one tends to use (8) for filtering he needs to be aware of complexity of 2D convolution. Finally filtering should be applied before back projection.

Design desired filter in frequency domain

For each theta angle projection multiply filter with projection and then take inverse Fourier Transform

Filtering Program Ends

Create t, theta, x, and y coordinate arrays

Clear dynamic memory

Find intersection points using (3) and (4)

Determine unique and relevant x and y pairs

Sort x and y pairs

If there is not a relevant point do nothing

%Above means beam does not go through image

If there is only one relevant point do nothing

%Above means beam is tangential to image

Calculate distance and middle points through (5) and (6)

Correct address information using middle point data

Assign projection data for each beam according to distance and address values of each pixel.

Calculate summation of assigned pixel values

Normalize the image matrix

Display the back projected/filtered back projected image

Load original image for quantitative measurement

Calculate error by summing absolute differences for each pixel between original and back projected image

Normalize the error to avoid size dependent error calculation

## III. Results

In this subsection results will be presented. Since results mean different outputs for projection, back projection, filtered back projection, and quantitative error measurements, each will be presented separately.

Raeder should realize that for image figures horizontal axis is x and vertical axis is y coordinate conventionally.

### A. *Projection*

Projection algorithm provides the base of this project. Even though it may not seem important as it is, remembering only roentgen images should remind us the importance of single projection. Simple chest X-ray image might and had saved lots of lives. Having this understanding of importance of even single projection sets a serious motivation for observation of projection data.

Examining Fig. 1 one may make a very rough estimation about image structure. Visualizing X-ray beams from teta=0 degrees angle one may make a quite accurate estimation about the location of attenuation in the image. Furthermore having Fig. 2 as original image does not surprise anyone.

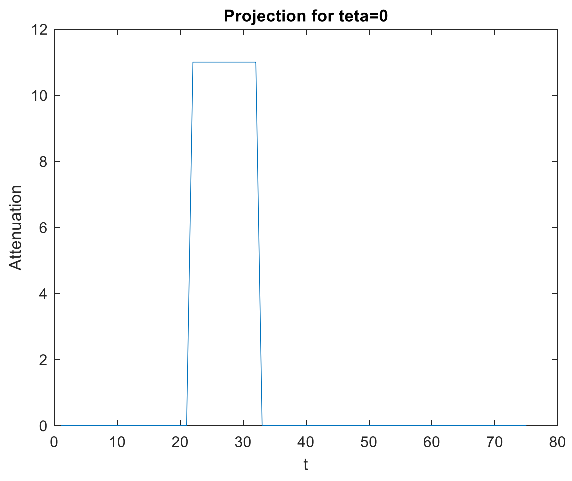

Figure 1: Projection of square.mat Image for theta=0 degrees

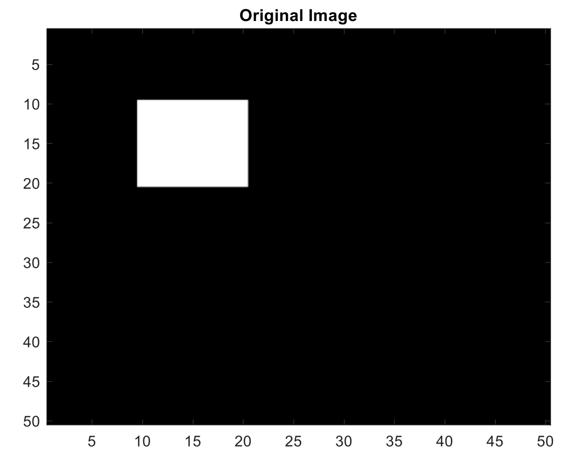

Figure 2: Original square.mat Image

Like previous case it is expected to obtain Fig. 3 having original image as sample image with attenuation values shown in Fig. 4.

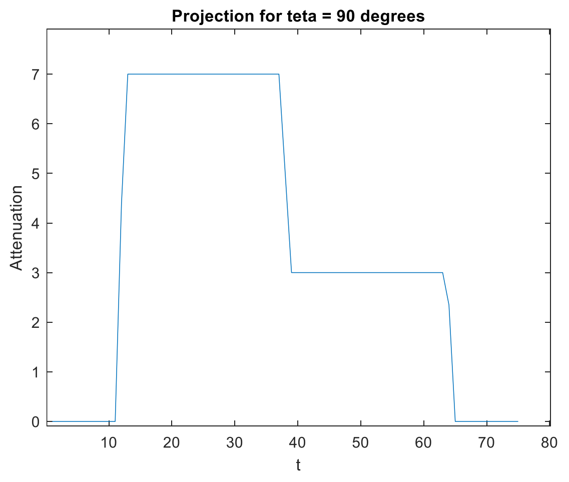

Figure 3: Projection of Sample Image Matrix for theta=90 degrees

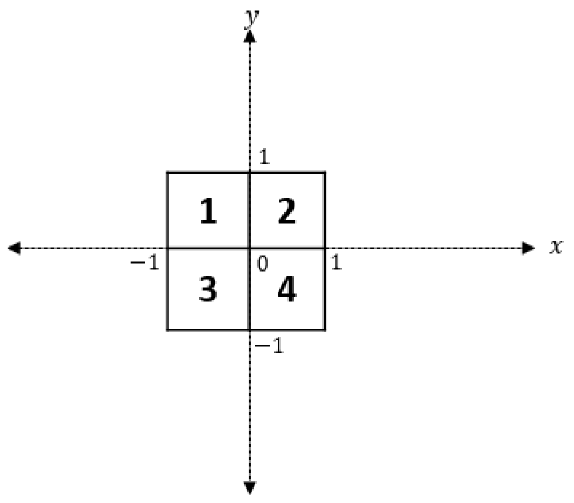

Figure 4: Sample Image Matrix

Unlike previous cases in an image with a lot of high frequency components it is hard to make meaning out of single projection data. Here correctness of algorithm may be checked using radon function of MATLAB.

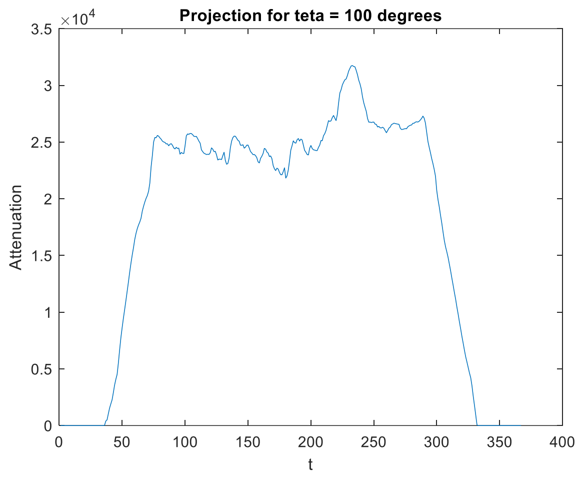

Figure 5: Projection of lena.mat Image with Beam Number 360

Examining Fig. 5 and Fig. 6 it is obvious that they are very similar indicating correctness of projection algorithm. However, in this case this single projection is not very meaningful especially for untrained eyes. Furthermore, effects of changing beam number may be observed comparing Fig. 5 and Fig. 7. In the Fig. 7 only 36 beams are used for projection.

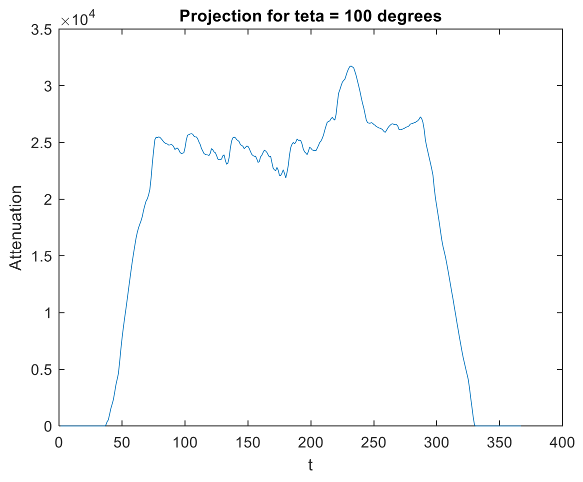

Figure 6: Projection of lena.mat using radon Function

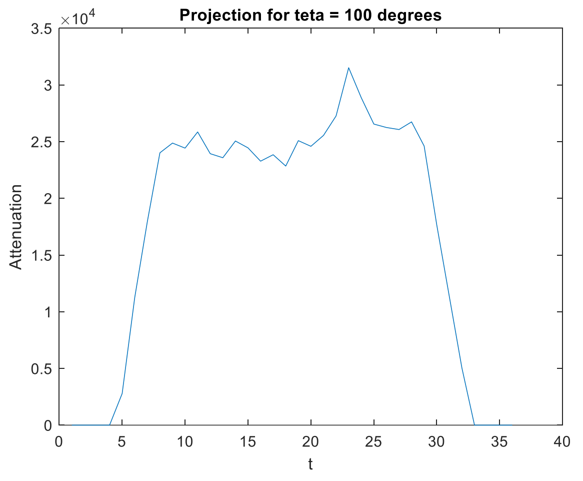

Figure 7: Projection of lena.mat Image with Bean Number 36

## B. *Back Projection and Filtered Back Projection*

Despite usefulness of single projection data, Computed Tomography arises from combining different projection data and obtaining a 3-D image. This combination process is called Back Projection. In this subsection various image back projection image will be presented. Moreover, filtered back projection images will also be presented.

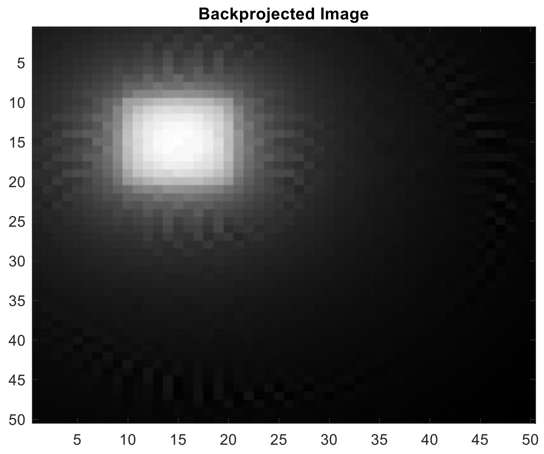

Figure 8: Reconstructed square.mat Image without Filtering with Beam Number 100 Step Number 90

In Fig. 8 and Fig. 9 back projected and filtered back projected images of square.mat image shown in Fig. 2 are displayed. It is important to note that triangular filter which is shown in Fig. 17 is used for all back projection images unless otherwise stated.

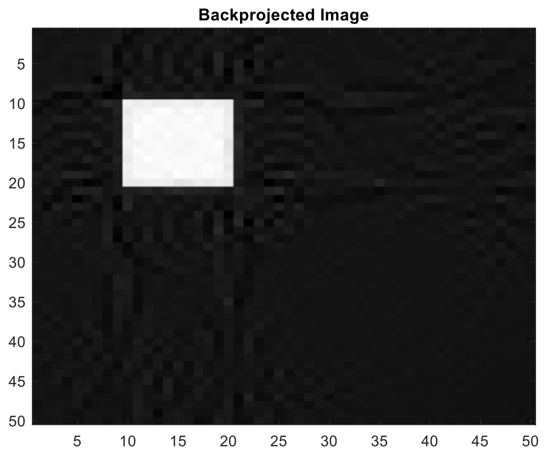

Figure 9: Reconstructed Image with Filtering with Beam Number 100 Step Number 90

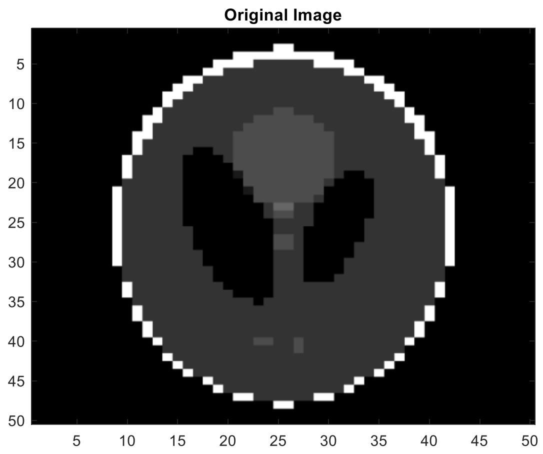

Figure 10: Original SheppLogan.mat Image

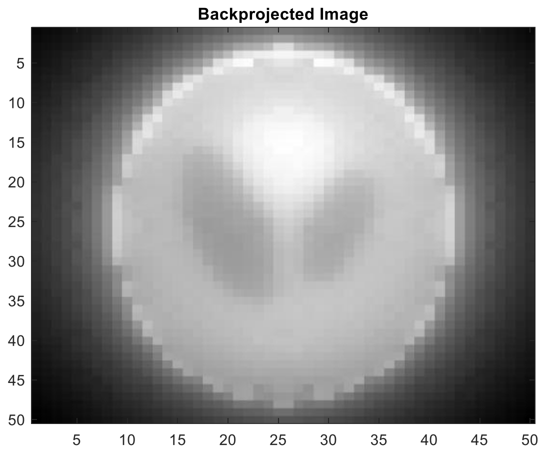

Figure 11: Reconstructed Image without Filtering with Beam Number 100 Step Number 90

In Fig. 11 and Fig. 12 back projected and filtered back projected images of SheppLogan.mat image shown in Fig. 10 are displayed.

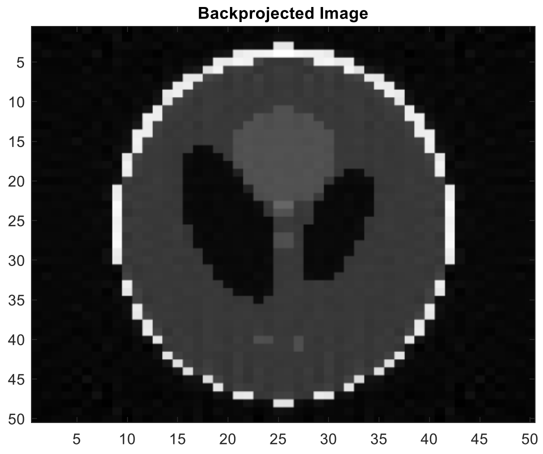

Figure 12: Reconstructed Image with Filtering with Beam Number 100 Step Number 90

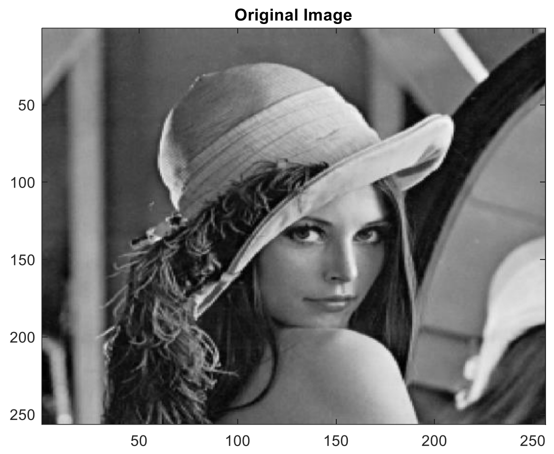

Figure 13: Original lena.mat Image

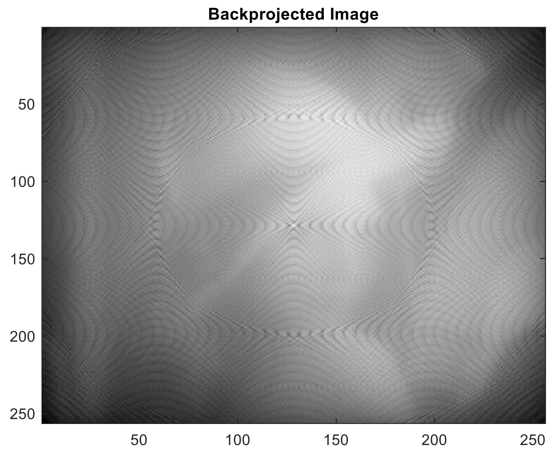

Figure 14: Reconstructed Image without Filtering with Beam Number 300 Step Number 180

In Fig. 14 and Fig. 15 back projected and filtered back projected images of lena.mat image shown in Fig. 13 are displayed.

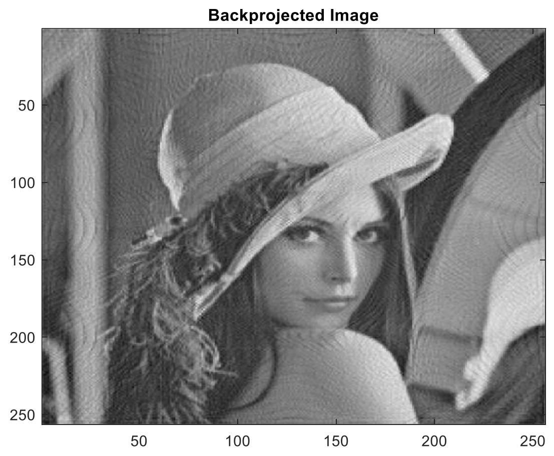

Figure 15: Reconstructed Image with Filtering with Beam Number 300 Step Number 180

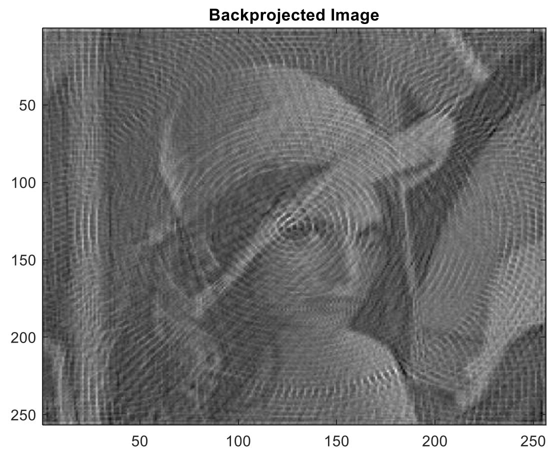

Figure 16: Reconstructed Image with Filtering with Beam Number 100 Step Number 90

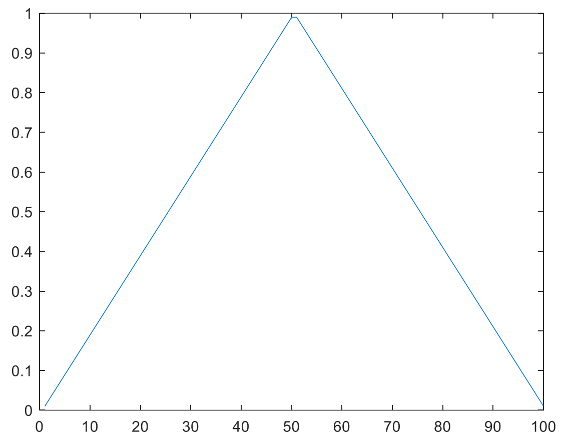

Figure 17: Triangular Filter

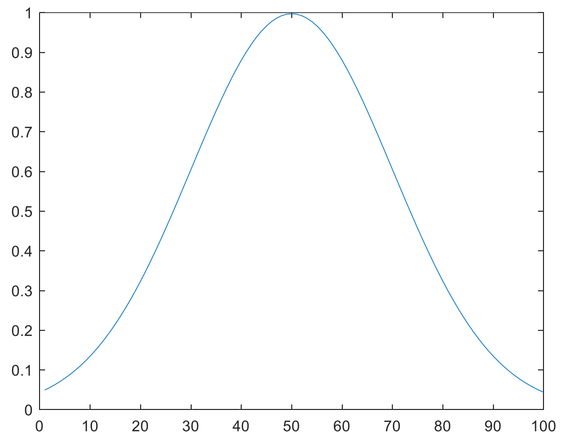

Figure 18: Gaussian Filter

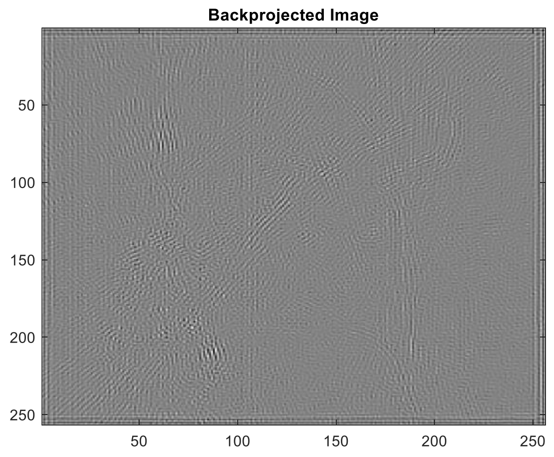

Figure 19: Reconstructed Image with Filtering with Beam Number 300 Step Number 180 with Gaussian Filter

### C. *Quantitative Error Measurements*

In Computed Tomography and actually in all imaging modalities it is crucial to know error rate. Therefore, in this project a unique error measurement algorithm is implemented. Resulting error rates are shown in Table I. Note that triangular filter is used for Table I.

TABLE I. ERROR MEASUREMENTS

| **Image** | **Error without Filter** | **Error with Filter** |
|---|---|---|
| square.mat | 0.1126 | 0.0807 |
| SheppLogan.mat | 0.2559 | 0.1130 |
| lena.mat | 98.3495 | 98.2022 |

## IV. Discussion and Conclusion

It is a very well-known fact that beam number, step number, filtering, filter type, image size, and original image characteristics have significant importance on back projected image quality. In this project all these criteria and their effects can be examined. Indeed, at the first aim of project was to detect and adjust parameters effecting back projected image quality. In this subsection all of the parameters described above, and their effects will be discussed in detail.

First focusing on beam number and step number it is known that increasing beam and step number allows for more detailed projection. Comparing Fig. 5 with Fig. 7 it is obvious that high beam number results in more detailed projection data. Moreover, since Computed Tomography algorithm combines different angle projections more projections mean more data to be combined resulting in sharper back projected images. Comparing Fig.15 and Fig.16 significance of beam and step number can be demonstrated. However, at this point it is extremely important to note increasing beam and step number means exposing patient to more ionizing radiation. Therefore, optimization of image quality versus ionizing radiation should be done by qualified experts. Also, it should be noted that high beam and step number results in more computation which causes longer runtime.

Comparing Fig. 8 with Fig. 9, Fig. 11 with Fig. 12, and Fig. 14 with Fig. 15 importance of filtering can be observed. It is necessary to stress that only difference between these figures is filtering. Despite longer runtime due to filtering calculations, proper filters increase image quality significantly. This claim is proved by comparing quantitative error measurements of filtered and not filtered images. For instance, SheppLogan.mat image has almost 50% error reduction when filter is applied. On the other hand, for lena.mat image it can be seen that error reduction is not that significant. The cause for this is size of image and spatial frequency components of image.

While talking about filters it is important the mention the differences between Gaussian and triangular filter. Although in general these filters provide similar high pass characteristics, Gaussian filter lacks a serious property. This property is completely filtering zero frequency components. As one can see from Fig. 18 even for zero frequency filter has small leakage. It may not seem important at first glance however, comparing Fig.15 with Fig. 19 one can easily see the serious difference between filtered back projected images. Deviations from ideal triangular filter will always cause worse image quality because of the reasons explained in Theory and Algorithm subsection.

Image size is another parameter effecting back projected image and computation time. As the size of image increases beams are separated to larger span. This condition requires higher beam number to obtain clear images. Higher beam number causes more computation and more exposure of ionizing radiation. Furthermore, spatial frequency spectrum of image effects back projected image quality. Since Computed Tomography algorithm has tendency to fail for high frequency components more than low frequency components as high frequency dominancy increases back projected image quality decreases. This can be observed by comparing Fig. 12 and Fig. 16. For lena.mat image high frequency components dominate and despite the usage of same beam and step number SheppLogan.mat image obtained is much clearer that lena.mat image. Of course, the size difference should also be noted.

To sum it all up, in this project simple Computed Tomography algorithm and effects of parameters used during algorithm implementation are discussed. A researcher should utilize such algorithms to observe and improve Computed Tomography. Simulation results are presented to make a solid understanding of effects of beam number, step number, filtering, filter type, image size, and original image characteristics on back projected image quality. Furthermore, quantitative error measurements are introduced to be more legit. At the end GUI and code implementations are given in the appendix.

## References

[1] C. Richmond, "Sir Godfrey Hounsfield," BMJ : British Medical Journal, 18-Sep-2004. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC517662/. [Accessed: 31-Dec-2019].

[2] K. K. Shung, M. Smith, and B. M. W. Tsui, *Principles of Medical Imaging*. Burlington: Elsevier Science, 2012.

APPENDIX A

PROJECTION CODE

```
%Muhammed Saadeddin Koçak 2232346

clear
%Loading Image
myimage=load("square.mat");
C=struct2cell(myimage);%Cell
A=cell2mat(C);%Matrix
% A=[1 2;3 4];
%Getting Size
sizeofmatrix=size(A);
sizeofimage=sizeofmatrix(1,1);%Assuming image is square
%Getting beam number and step size
numberofbeams=100;%Number of beams
stepsize=90;%Number of steps
%Defining t, teta, x, and y arrays according to beam number and
step size
t=linspace(-
sizeofimage*sqrt(2)/2,sizeofimage*sqrt(2)/2,numberofbeams);
teta=linspace(0,180-(180/stepsize),stepsize);
x=linspace(-sizeofimage/2,sizeofimage/2,sizeofimage+1);
y=linspace(-sizeofimage/2,sizeofimage/2,sizeofimage+1);
%Taking projections
 for i=1:(length(teta))
     for j=1:(length(t))
%Clearing dynamic memory
relevantmatrix=[];
rowdata=[]; columndata=[]; midpointx=[];
midpointy=[]; distance=[];
%Finding intersection points
    yfromx=(t(j)/sind(teta(i)))-(x.*cotd(teta(i)));
    xfromy=(t(j)/cosd(teta(i)))-(x.*tand(teta(i)));
%Detecting relevant x and y coordinates
m=1;
for g=1:(length(x))
    if (((-
sizeofimage/2)<=yfromx(g))&&(yfromx(g)<=sizeofimage/2))
        relevantmatrix(m,1)=x(1,g);
        relevantmatrix(m,2)=yfromx(1,g);
        m=m+1;
    else
    end
end
for k=1:(length(y))
```

```
    if (((-
sizeofimage/2)<=xfromy(k))&&(xfromy(k)<=sizeofimage/2))
        relevantmatrix(m,1)=xfromy(1,k);
        relevantmatrix(m,2)=y(1,k);
        m=m+1;
    else
    end
end
uniquerelevantmatrix=unique(relevantmatrix,'rows');%Deleting
same x and y points
sortedrelevantmatrix=sortrows(uniquerelevantmatrix);%Sorting
relevant x and y points
sizerelevant=size(uniquerelevantmatrix,1);%Detecting if there
are relevant points
if sizerelevant==0 %If no relevant points exist return total
attenuation
    totalattenuation=0;
    projectionmatrix(j,i)=totalattenuation;
else
[rowsizeofURM,columnsizeofURM]=size(uniquerelevantmatrix);
if (rowsizeofURM>1)%If there are more than one relevant point
then calculate attenuation
%Calculating distance and midpoints to use them for detecting
row and
%column data
for r=1:(rowsizeofURM-1)
    distance(r)=sqrt(((uniquerelevantmatrix(r,1)-
uniquerelevantmatrix(r+1,1))^2)+((uniquerelevantmatrix(r,2)-
uniquerelevantmatrix(r+1,2))^2));

midpointx(r)=(uniquerelevantmatrix(r,1)+uniquerelevantmatrix(r+1
,1))/2;

midpointy(r)=(uniquerelevantmatrix(r,2)+uniquerelevantmatrix(r+1
,2))/2;
end
%Finding corresponding column and row data
rowdata=(sizeofimage/2)-floor(midpointy);
columndata=(sizeofimage/2)+ceil(midpointx);
%Summing all attenuation
sizeofdistancematrix=size(distance);
sizeofdistancearray=sizeofdistancematrix(1,2);
totalattenuation=0;%Clearing dynamic data from previous
calculation
for h=1:sizeofdistancearray

totalattenuation=totalattenuation+(A(rowdata(1,h),columndata(1,h
))*distance(1,h));
```

```
end
projectionmatrix(j,i)=totalattenuation;%Assigning attenuation to
matrix
else%If there is only one relevant point beam is tangential,
then attenuation is zero
    totalattenuation=0;
    projectionmatrix(j,i)=totalattenuation;
end
end
    end
 end
 %Projection at a single angle
 figure
 plot(projectionmatrix(:,45));
```

## APPENDIX B

## BACK PROJECTION CODE

```
%Muhammed Saadeddin Koçak 2232346

clearvars -except projectionmatrix sizeofimage

[numberofbeams stepsize]=size(projectionmatrix);
imagematrix=zeros(sizeofimage);

%Loading Image
%Realize original image is loaded only for comparison. In
backprojection
%code A matrix is never used.
myimage=load("square.mat");
C=struct2cell(myimage);%Cell
A=cell2mat(C);%Matrix
figure
imagesc(A);
title ("Original Image");
colormap gray;
%Filtering
%Triangular Filter
triangfilter=triang(numberofbeams);%Designing filter in Fourier
Domain
for o=1:stepsize
filteredprojectionmatrix(:,o)=ifft2((triangfilter).*(fft2(projec
tionmatrix(:,o))));
end
projectionmatrix=filteredprojectionmatrix;
%Gaussian Filter
% inpargu=1:1:numberofbeams;
% gausfilter=50*normpdf(inpargu,numberofbeams/2,20);
% gausfilter=gausfilter';
% for o=1:stepsize
%
filteredprojectionmatrix(:,o)=ifft2((gausfilter).*(fft2(projecti
onmatrix(:,o))));
% end
% projectionmatrix=filteredprojectionmatrix;
%Plotting Filter
% figure
% plot(gausfilter);
%Defining t, teta, x, and y arrays according to beam number and
step size
t=linspace(-
sizeofimage*sqrt(2)/2,sizeofimage*sqrt(2)/2,numberofbeams);
teta=linspace(0,180-(180/stepsize),stepsize);
x=linspace(-sizeofimage/2,sizeofimage/2,sizeofimage+1);
y=linspace(-sizeofimage/2,sizeofimage/2,sizeofimage+1);
 for i=1:(length(teta))
     for j=1:(length(t))
%Clearing dynamic memory
relevantmatrix=[];
rowdata=[]; columndata=[]; midpointx=[];
midpointy=[]; distance=[];
%Finding intersection points
    yfromx=(t(j)/sind(teta(i)))-(x.*cotd(teta(i)));
    xfromy=(t(j)/cosd(teta(i)))-(x.*tand(teta(i)));
%Detecting relevant x and y coordinates
m=1;
for g=1:(length(x))
    if (((-
sizeofimage/2)<=yfromx(g))&&(yfromx(g)<=sizeofimage/2))
        relevantmatrix(m,1)=x(1,g);
        relevantmatrix(m,2)=yfromx(1,g);
        m=m+1;
    else
    end
end
for k=1:(length(y))
    if (((-
sizeofimage/2)<=xfromy(k))&&(xfromy(k)<=sizeofimage/2))
        relevantmatrix(m,1)=xfromy(1,k);
        relevantmatrix(m,2)=y(1,k);
        m=m+1;
    else
    end
end
uniquerelevantmatrix=unique(relevantmatrix,'rows');%Deleting
same x and y points
```

```
sortedrelevantmatrix=sortrows(uniquerelevantmatrix);%Sorting
relevant x and y points
sizerelevant=size(uniquerelevantmatrix,1);%Detecting if there
are relevant points
if sizerelevant==0 %If no relevant points do nothing
else
[rowsizeofURM,columnsizeofURM]=size(uniquerelevantmatrix);
if (rowsizeofURM>1)%If there are more than one relevant point
then calculate attenuation
%Calculating distance and midpoints to use them for detecting
row and
%column data
for r=1:(rowsizeofURM-1)
    distance(r)=sqrt(((uniquerelevantmatrix(r,1)-
uniquerelevantmatrix(r+1,1))^2)+((uniquerelevantmatrix(r,2)-
uniquerelevantmatrix(r+1,2))^2));

midpointx(r)=(uniquerelevantmatrix(r,1)+uniquerelevantmatrix(r+1
,1))/2;

midpointy(r)=(uniquerelevantmatrix(r,2)+uniquerelevantmatrix(r+1
,2))/2;
end
%Finding corresponding column and row data
rowdata=(sizeofimage/2)-floor(midpointy);
columndata=(sizeofimage/2)+ceil(midpointx);
%Backprojecting one ray beam
sizeofdistancematrix=size(distance);
sizeofdistancearray=sizeofdistancematrix(1,2);
for h=1:sizeofdistancearray

imagematrix(rowdata(1,h),columndata(1,h))=imagematrix(rowdata(1,
h),columndata(1,h))+(projectionmatrix(j,i)*distance(1,h));
end
else%If there is only one relevant point beam is tangential,
then do nothing
end
end
    end
 end
%Normalizing image matrix
imagematrix=real(imagematrix);
normimagematrix=(imagematrix-
min(min(imagematrix)))/(max(max(imagematrix)));
%Displaying Image
figure
imagesc(normimagematrix);
title("Backprojected Image");
```

```
colormap gray;
%Quantitative Comparison
for i=1:sizeofimage
    for j=1:sizeofimage
        difference(i,j)=abs(normimagematrix(i,j)-A(i,j));
    end
end
%Displaying Differences Between Images
figure
imagesc(difference);
title("Difference Between Images");
colormap gray;
%Calculating Error
error=sum(sum(difference));
errorratio=error/(sizeofimage^2);
```

## APPENDIX C

## GRAPHICAL USER INTERFACE CODE

```
classdef EE415ProjectGUI < MATLAB.apps.AppBase


    % Properties that correspond to app components
    properties (Access = public)
        UIFigure                    MATLAB.ui.Figure
        ProjectButton               MATLAB.ui.control.Button
        UIAxes                      MATLAB.ui.control.UIAxes
        UIAxes3                     MATLAB.ui.control.UIAxes
        BeamNumberEditFieldLabel    MATLAB.ui.control.Label
        BeamNumberEditField         MATLAB.ui.control.NumericEditField
        StepNumberEditFieldLabel    MATLAB.ui.control.Label
        StepNumberEditField         MATLAB.ui.control.NumericEditField
        ImageButtonGroup            MATLAB.ui.container.ButtonGroup
        SquareButton                MATLAB.ui.control.RadioButton
        SheppLoganButton            MATLAB.ui.control.RadioButton
        LenaButton                  MATLAB.ui.control.RadioButton
        FilterTypeButtonGroup       MATLAB.ui.container.ButtonGroup
        TriangularButton            MATLAB.ui.control.RadioButton
        GaussianButton              MATLAB.ui.control.RadioButton
        NoFilterButton              MATLAB.ui.control.RadioButton
    end


    % Callbacks that handle component events
    methods (Access = private)


        % Value changed function: BeamNumberEditField
        function BeamNumberEditFieldValueChanged(app, event)
            value = app.BeamNumberEditField.Value;
        end
```

```
% Value changed function: StepNumberEditField
function StepNumberEditFieldValueChanged(app, event)
    value = app.StepNumberEditField.Value;
end


% Selection changed function: ImageButtonGroup
function ImageButtonGroupSelectionChanged(app, event)
    selectedButton = app.ImageButtonGroup.SelectedObject;
end


% Selection changed function: FilterTypeButtonGroup
function FilterTypeButtonGroupSelectionChanged(app, event)
    selectedButton = app.FilterTypeButtonGroup.SelectedObject;
end


% Button pushed function: ProjectButton
function ProjectButtonPushed(app, event)
    numberofbeams=app.BeamNumberEditField.Value;
    stepsize=app.StepNumberEditField.Value;
    switch app.ImageButtonGroup.SelectedObject.Text
        case 'Square'
            myimage=load("square.mat");
        case 'SheppLogan'
            myimage=load("SheppLogan.mat");
        case 'Lena'
            myimage=load("lena.mat");
    end
    C=struct2cell(myimage);%Cell
    A=cell2mat(C);%Matrix
    %Getting Size
    sizeofmatrix=size(A);
    sizeofimage=sizeofmatrix(1,1);%Assuming image is square
    %Getting beam number and step size
    %Defining t, teta, x, and y arrays according to beam number and step size
    t=linspace(-sizeofimage*sqrt(2)/2,sizeofimage*sqrt(2)/2,numberofbeams);
    teta=linspace(0,180-(180/stepsize),stepsize);
    x=linspace(-sizeofimage/2,sizeofimage/2,sizeofimage+1);
    y=linspace(-sizeofimage/2,sizeofimage/2,sizeofimage+1);
    %Taking projections
     for i=1:(length(teta))
         for j=1:(length(t))
    %Clearing dynamic memory
    relevantmatrix=[];
    rowdata=[]; columndata=[]; midpointx=[];
    midpointy=[]; distance=[];
    %Finding intersection points
        yfromx=(t(j)/sind(teta(i)))-(x.*cotd(teta(i)));
        xfromy=(t(j)/cosd(teta(i)))-(x.*tand(teta(i)));
    %Detecting relevant x and y coordinates
    m=1;
    for g=1:(length(x))
        if (((-sizeofimage/2)<=yfromx(g))&&(yfromx(g)<=sizeofimage/2))
```

```
                    relevantmatrix(m,1)=x(1,g);
                    relevantmatrix(m,2)=yfromx(1,g);
                    m=m+1;
                else
                end
            end
            for k=1:(length(y))
                if (((-sizeofimage/2)<=xfromy(k))&&(xfromy(k)<=sizeofimage/2))
                    relevantmatrix(m,1)=xfromy(1,k);
                    relevantmatrix(m,2)=y(1,k);
                    m=m+1;
                else
                end
            end
            uniquerelevantmatrix=unique(relevantmatrix,'rows');%Deleting same x and y 
points
            sortedrelevantmatrix=sortrows(uniquerelevantmatrix);%Sorting relevant x and 
y points
            sizerelevant=size(uniquerelevantmatrix,1);%Detecting if there are relevant 
points
            if sizerelevant==0 %If no relevant points exist return total attenuation
                totalattenuation=0;
                projectionmatrix(j,i)=totalattenuation;
            else
            [rowsizeofURM,columnsizeofURM]=size(uniquerelevantmatrix);
            if (rowsizeofURM>1)%If there are more than one relevant point then 
calculate attenuation
            %Calculating distance and midpoints to use them for detecting row and
            %column data
            for r=1:(rowsizeofURM-1)
                distance(r)=sqrt(((uniquerelevantmatrix(r,1)-
uniquerelevantmatrix(r+1,1))^2)+((uniquerelevantmatrix(r,2)-
uniquerelevantmatrix(r+1,2))^2));
                midpointx(r)=(uniquerelevantmatrix(r,1)+uniquerelevantmatrix(r+1,1))/2;
                midpointy(r)=(uniquerelevantmatrix(r,2)+uniquerelevantmatrix(r+1,2))/2;
            end
            %Finding corresponding column and row data
            rowdata=(sizeofimage/2)-floor(midpointy);
            columndata=(sizeofimage/2)+ceil(midpointx);
            %Summing all attenuation
            sizeofdistancematrix=size(distance);
            sizeofdistancearray=sizeofdistancematrix(1,2);
            totalattenuation=0;%Clearing dynamic data from previous calculation
            for h=1:sizeofdistancearray
                
totalattenuation=totalattenuation+(A(rowdata(1,h),columndata(1,h))*distance(1,h));
            end
            projectionmatrix(j,i)=totalattenuation;%Assigning attenuation to matrix
            else%If there is only one relevant point beam is tangential, then 
attenuation is zero
                totalattenuation=0;
                projectionmatrix(j,i)=totalattenuation;
            end
            end
                end
```

```
             end
            colormap(app.UIAxes,'gray');
            imagesc(app.UIAxes,A);

            %Backprojection
            [numberofbeams stepsize]=size(projectionmatrix);
            imagematrix=zeros(sizeofimage);

            switch app.FilterTypeButtonGroup.SelectedObject.Text
                case 'Triangular'
                    triangfilter=triang(numberofbeams);%Designing filter in Fourier
Domain
                    for o=1:stepsize

filteredprojectionmatrix(:,o)=ifft2((triangfilter).*(fft2(projectionmatrix(:,o))));
                    end
                    projectionmatrix=filteredprojectionmatrix;
                case 'Gaussian'
                    inpargu=1:1:numberofbeams;
                    gausfilter=normpdf(inpargu,numberofbeams/2,20);
                    gausfilter=gausfilter';
                    for o=1:stepsize

filteredprojectionmatrix(:,o)=ifft2((gausfilter).*(fft2(projectionmatrix(:,o))));
                    end
                    projectionmatrix=filteredprojectionmatrix;
                case 'No Filter'

            end
            %Defining t, teta, x, and y arrays according to beam number and step size
            t=linspace(-sizeofimage*sqrt(2)/2,sizeofimage*sqrt(2)/2,numberofbeams);
            teta=linspace(0,180-(180/stepsize),stepsize);
            x=linspace(-sizeofimage/2,sizeofimage/2,sizeofimage+1);
            y=linspace(-sizeofimage/2,sizeofimage/2,sizeofimage+1);
             for i=1:(length(teta))
                 for j=1:(length(t))
            %Clearing dynamic memory
            relevantmatrix=[];
            rowdata=[]; columndata=[]; midpointx=[];
            midpointy=[]; distance=[];
            %Finding intersection points
                yfromx=(t(j)/sind(teta(i)))-(x.*cotd(teta(i)));
                xfromy=(t(j)/cosd(teta(i)))-(x.*tand(teta(i)));
            %Detecting relevant x and y coordinates
            m=1;
            for g=1:(length(x))
                if (((-sizeofimage/2)<=yfromx(g))&&(yfromx(g)<=sizeofimage/2))
                    relevantmatrix(m,1)=x(1,g);
                    relevantmatrix(m,2)=yfromx(1,g);
                    m=m+1;
                else
                end
            end
            for k=1:(length(y))
                if (((-sizeofimage/2)<=xfromy(k))&&(xfromy(k)<=sizeofimage/2))
```

```
                    relevantmatrix(m,1)=xfromy(1,k);
                    relevantmatrix(m,2)=y(1,k);
                    m=m+1;
                else
                end
            end
            uniquerelevantmatrix=unique(relevantmatrix,'rows');%Deleting same x and y
points
            sortedrelevantmatrix=sortrows(uniquerelevantmatrix);%Sorting relevant x and
y points
            sizerelevant=size(uniquerelevantmatrix,1);%Detecting if there are relevant
points
            if sizerelevant==0 %If no relevant points do nothing
            else
            [rowsizeofURM,columnsizeofURM]=size(uniquerelevantmatrix);
            if (rowsizeofURM>1)%If there are more than one relevant point then
calculate attenuation
            %Calculating distance and midpoints to use them for detecting row and
            %column data
            for r=1:(rowsizeofURM-1)
                distance(r)=sqrt(((uniquerelevantmatrix(r,1)-
uniquerelevantmatrix(r+1,1))^2)+((uniquerelevantmatrix(r,2)-
uniquerelevantmatrix(r+1,2))^2));
                midpointx(r)=(uniquerelevantmatrix(r,1)+uniquerelevantmatrix(r+1,1))/2;
                midpointy(r)=(uniquerelevantmatrix(r,2)+uniquerelevantmatrix(r+1,2))/2;
            end
            %Finding corresponding column and row data
            rowdata=(sizeofimage/2)-floor(midpointy);
            columndata=(sizeofimage/2)+ceil(midpointx);
            %Backprojecting one ray beam
            sizeofdistancematrix=size(distance);
            sizeofdistancearray=sizeofdistancematrix(1,2);
            for h=1:sizeofdistancearray

imagematrix(rowdata(1,h),columndata(1,h))=imagematrix(rowdata(1,h),columndata(1,h))+(pr
ojectionmatrix(j,i)*distance(1,h));
            end
            else%If there is only one relevant point beam is tangential, then do
nothing
            end
            end
                end
             end
            %Normalizing image matrix
            imagematrix=real(imagematrix);

            normimagematrix=(imagematrix-
min(min(imagematrix)))/(max(max(imagematrix)));
            colormap(app.UIAxes3,'gray');
            imagesc(app.UIAxes3,normimagematrix);
        end
    end


    % Component initialization
    methods (Access = private)

        % Create UIFigure and components
        function createComponents(app)

            % Create UIFigure and hide until all components are created
            app.UIFigure = uifigure('Visible', 'off');
            app.UIFigure.Position = [100 100 1106 561];
            app.UIFigure.Name = 'UI Figure';

            % Create ProjectButton
            app.ProjectButton = uibutton(app.UIFigure, 'push');
            app.ProjectButton.ButtonPushedFcn = createCallbackFcn(app,
@ProjectButtonPushed, true);
            app.ProjectButton.FontSize = 24;
            app.ProjectButton.Position = [898.5 79 170 92];
            app.ProjectButton.Text = 'Project';

            % Create UIAxes
            app.UIAxes = uiaxes(app.UIFigure);
            title(app.UIAxes, 'Original Image')
            xlabel(app.UIAxes, 'X')
            ylabel(app.UIAxes, 'Y')
            app.UIAxes.Position = [32 225 459 307];

            % Create UIAxes3
            app.UIAxes3 = uiaxes(app.UIFigure);
            title(app.UIAxes3, 'Filtered Back Projected Image')
            xlabel(app.UIAxes3, 'X')
            ylabel(app.UIAxes3, 'Y')
            app.UIAxes3.Position = [582 225 486 307];

            % Create BeamNumberEditFieldLabel
            app.BeamNumberEditFieldLabel = uilabel(app.UIFigure);
            app.BeamNumberEditFieldLabel.HorizontalAlignment = 'right';
            app.BeamNumberEditFieldLabel.Position = [231 174 83 22];
            app.BeamNumberEditFieldLabel.Text = 'Beam Number';

            % Create BeamNumberEditField
            app.BeamNumberEditField = uieditfield(app.UIFigure, 'numeric');
            app.BeamNumberEditField.ValueChangedFcn = createCallbackFcn(app,
@BeamNumberEditFieldValueChanged, true);
            app.BeamNumberEditField.Position = [329 174 100 22];

            % Create StepNumberEditFieldLabel
            app.StepNumberEditFieldLabel = uilabel(app.UIFigure);
            app.StepNumberEditFieldLabel.HorizontalAlignment = 'right';
            app.StepNumberEditFieldLabel.Position = [238 109 76 22];
            app.StepNumberEditFieldLabel.Text = 'Step Number';

            % Create StepNumberEditField
            app.StepNumberEditField = uieditfield(app.UIFigure, 'numeric');
            app.StepNumberEditField.ValueChangedFcn = createCallbackFcn(app,
@StepNumberEditFieldValueChanged, true);
            app.StepNumberEditField.Position = [329 109 100 22];

            % Create ImageButtonGroup
            app.ImageButtonGroup = uibuttongroup(app.UIFigure);
            app.ImageButtonGroup.SelectionChangedFcn = createCallbackFcn(app,
@ImageButtonGroupSelectionChanged, true);
            app.ImageButtonGroup.Title = 'Image';
            app.ImageButtonGroup.Position = [71 93 123 106];

            % Create SquareButton
            app.SquareButton = uiradiobutton(app.ImageButtonGroup);
            app.SquareButton.Text = 'Square';
            app.SquareButton.Position = [11 60 61 22];
            app.SquareButton.Value = true;

            % Create SheppLoganButton
            app.SheppLoganButton = uiradiobutton(app.ImageButtonGroup);
            app.SheppLoganButton.Text = 'SheppLogan';
            app.SheppLoganButton.Position = [11 38 90 22];

            % Create LenaButton
            app.LenaButton = uiradiobutton(app.ImageButtonGroup);
            app.LenaButton.Text = 'Lena';
            app.LenaButton.Position = [11 16 65 22];

            % Create FilterTypeButtonGroup
            app.FilterTypeButtonGroup = uibuttongroup(app.UIFigure);
            app.FilterTypeButtonGroup.SelectionChangedFcn = createCallbackFcn(app,
@FilterTypeButtonGroupSelectionChanged, true);
            app.FilterTypeButtonGroup.Title = 'Filter Type';
            app.FilterTypeButtonGroup.Position = [638 89 123 106];

            % Create TriangularButton
            app.TriangularButton = uiradiobutton(app.FilterTypeButtonGroup);
            app.TriangularButton.Text = 'Triangular';
            app.TriangularButton.Position = [11 60 76 22];
            app.TriangularButton.Value = true;

            % Create GaussianButton
            app.GaussianButton = uiradiobutton(app.FilterTypeButtonGroup);
            app.GaussianButton.Text = 'Gaussian';
            app.GaussianButton.Position = [11 38 73 22];

            % Create NoFilterButton
            app.NoFilterButton = uiradiobutton(app.FilterTypeButtonGroup);
            app.NoFilterButton.Text = 'No Filter';
            app.NoFilterButton.Position = [11 17 76 22];

            % Show the figure after all components are created
            app.UIFigure.Visible = 'on';
        end
    end

    % App creation and deletion
    methods (Access = public)

        % Construct app
        function app = EE415ProjectGUI

            % Create UIFigure and components
            createComponents(app)

            % Register the app with App Designer
            registerApp(app, app.UIFigure)

            if nargout == 0
                clear app
            end
        end

        % Code that executes before app deletion
        function delete(app)

            % Delete UIFigure when app is deleted
            delete(app.UIFigure)
        end
    end
end
```

## APPENDIX D

## USAGE OF CODE

Given MATLAB code in Appendix A, B, and C is implemented in MATLAB 2019a. Copying and pasting codes separately in compatible MATLAB should provide user to see running code if required images are in the same folder with .m file. Necessary comments are inserted in code where user should modify according to his configuration.