IMAGE RECONSTRUCTION UNDER VARIOUS CONDITIONS USING PARALLEL BEAMS

Emre Ataklı, Department of Electrical & Electronics Engineering, METU. e211294@metu.edu.tr

***Abstract*—Inverse Radon Transform based image reconstruction has main importance in biomedical engineering. In this study, some applications in back projection and filtered back projection (FBP) with various filters namely Ram-Lak (ramp), Cosine and Hanning filters will be shown. These techniques will be applied to a simple square image and the Shepp-Logan phantom. Best results as reconstructed image are obtained by FBP technique using Cosine filter.**

***Index Terms*— Back-Projection, Filtering, Projection**

I. INTRODUCTION

DR. Willem Roentgen has discovered X-rays in late 1895, in Wurtzburg, Germany, as a typical example of an accidental invention, seemingly, but an inspirational gift in reality. Roentgen was carrying out experiments with a Crookes tube, which is a lot common research tool at that time. When he applied large voltages to the tube to study the behavior of electrons emitted from the metal, he noticed that a piece of phosphorus substance shone. Upon this strange event, he started to try to understand what is really going on by doing a set of experiments. During these experiments he saw that the thing that causes the glow on phosphorus material can help take image of the human anatomy. After that, it had been understood that both light and X-rays are electromagnetic radiation whereas X-rays are different as they are high energy light. X-rays can penetrate through many objects because of being high energy light. But they penetrate differently through different materials according to their densities. For example, X-rays can penetrate through fat/muscle easier than bone. This is the basis in imaging the body with X-rays. After the imaging process, obtained X-ray data can be used to reconstruct medical images thanks to the fact that attenuation rate of X-rays in the body depends on tissue characteristics. [1]

In this study, some mathematical tools that are used in X-ray imaging will be implemented. Moreover, the effects of using different kind of filters, different number of beams and step sizes will be shown by comparing them.

II. THEORY

This study contains two parts as Projection and Image Reconstruction. (see Fig. 1,2, and 3)

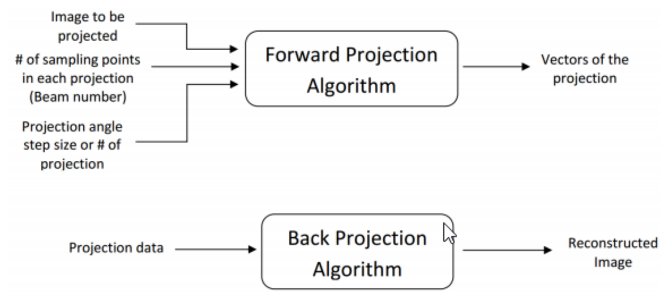

Fig 1: Basic schematic of the term project.

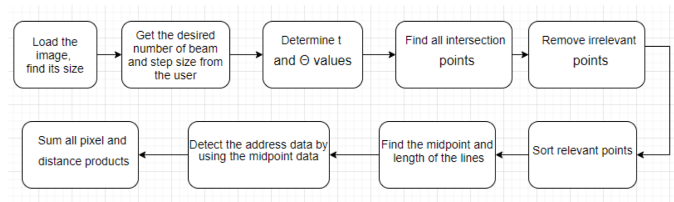

Fig 2: A brief flowchart of the projection algorithm.

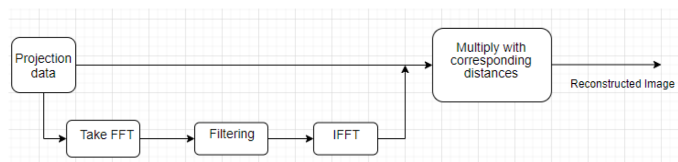

Fig 3: A brief flowchart of the back-projection algorithm.

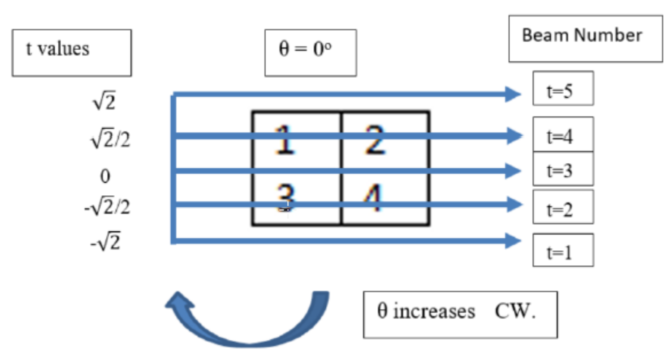

Fig 4: Beam and t values for the sample image when $\theta = 0^\circ$

A. Mathematical background

The Radon transform and its inverse provide the mathematical foundation for reconstructing the tomographic image from projection data. [2]

B. Filters

Ramp Filter. The ramp filter is a high pass filter that does not permit low frequencies that cause blurring in the image. This type of filter is used to reduce the star artifact.

The Ramp Filter is a compensatory filter because it eliminates the star artifact resulting from simple back projection. High pass filters sharpen the edges of the image and enhance the edge information. A serious disadvantage of high pass filtering is the amplification of random noise in the image. In order to reduce the amplification of high-frequencies the ramp filter is combined with a low-pass filter.

The common method to reduce or remove random noise in a SPECT image is the application of smoothing filters. These filters are low-pass filters. In this study, two of the low pass filters were utilized: Hanning and Cosine Filters. [3]

Hanning Filter. The Hanning filter is a simple low-pass filter. [4]

Cosine Filter. This type of filter is the standard response multiplied by a cosine shape. [5]

C. *Algorithm*

In this study, these following steps were applied to take projection of the input image:

- Input image, step size and number of beams were specified by the user.
- Θ values were determined according to the step size and t values were determined according to the number of beams and the size of the image.
- Intersection points for all beams for all projection angles were found using the line equation.
- The points that are irrelevant to the image were removed.
- The relevant points were sorted.
- The midpoints and length of each line segment were calculated.
- The address, i.e. row and column data were found by using the size and midpoint data.
- All pixel values and corresponding distance products. were summed (in other words, taking integral)

Secondly, these following steps were applied to take back projection of the input image:

- Discrete Fourier Transform of the projection data was obtained.
- Projection in the frequency domain was multiplied by a desired filter in order to get rid of low frequency components in the projection data.
- Inverse Fourier Transform was obtained.
- Finally, the filtered projection data was back-projected by multiplying the distance by the projection data.
- The resultant image was shown by normalizing it.

III. Results

With the help of this study, the effect of different type of filters, different number of beams and step sizes can be easily shown and compared each other. Below some of those results are included.

Python has been used for the implementation of the projection and back projection algorithms and simulation of their resultant images.

The images used for producing the projections for image reconstruction are given in the left-most side in Fig. 5 and Fig. 9 as a square image and Shepp-Logan phantom.

Results which act as quantitative evaluation measures are listed in tabular form in Table 1.

As seen from two different error measures, (in which MSE is Mean Squared Error), Cosine Filter is the best filter as compared to ramp filter and Hanning Filter.

IV. Conclusion

Although this study shows the effect of various filters, number of projections and step sizes on quality of the back projection image, this study can be extended with more images and filters to show their effects more clearly.

One of the limitations on the work is the language used in this work. It is Python. If C or another language similar in efficiency was used, algorithm would run fast.

Another limitation is the fact that this project and back-projection study was applied on an images, not the real human body.

Possible cause for the unexpected blurring in the unfiltered back-projection could not be found.

References

[1] https://sunnybrook.ca/research/content/?page=sri-groups-xray-info-3

[2] Shahzad Ahmad Qureshi, Sikander M. Mirza, M. Arif, "Inverse Radon Transform-Based Image Reconstruction Using Various Frequency Domain Filters In Parallel Beam Transmission Tomography"

[3] Maria Lyra and Agapi Ploussi, "Filtering in SPECT Image Reconstruction" International Journal of Biomedical Imaging, June 2011.

[4] M. N. Salihin Yusoff and A. Zakaria, "Determination of the optimum filter for qualitative and quantitative 99mTc myocardial SPECT imaging," Iranian Journal of Radiation Research, vol. 6, no. 4, pp. 173–181, 2009.

[5] https://octave.sourceforge.io/image/function/rho_filter.html

TABLE I
Errors And Elapsed Times Under Different Conditions of Projections

Shepp-Logan			
	Average Error	MSE	Elapsed Time
Ramp	0.148	0.043	19.2
Hanning	0.179	0.069	23.46
Cosine	-0.108	0.044	18.96
No filter	-2.953	9.500	18.18
Square			
	Average Error	MSE	Elapsed Time
Ramp	0.134	0.054	18.72
Hanning	0.053	0.034	19.14
Cosine	-0.076	0.026	18.54
No filter	-0.952	1.061	15.3

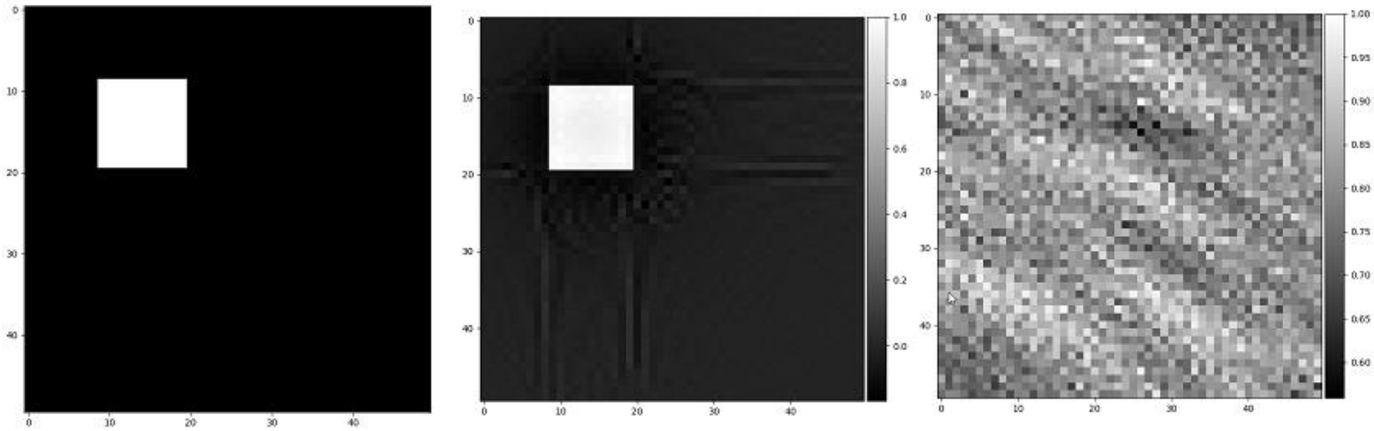

Fig 5: Images of original Square, reconstructed with and without filter, respectively. (Ramp filter, 180 fans and 100 detectors)

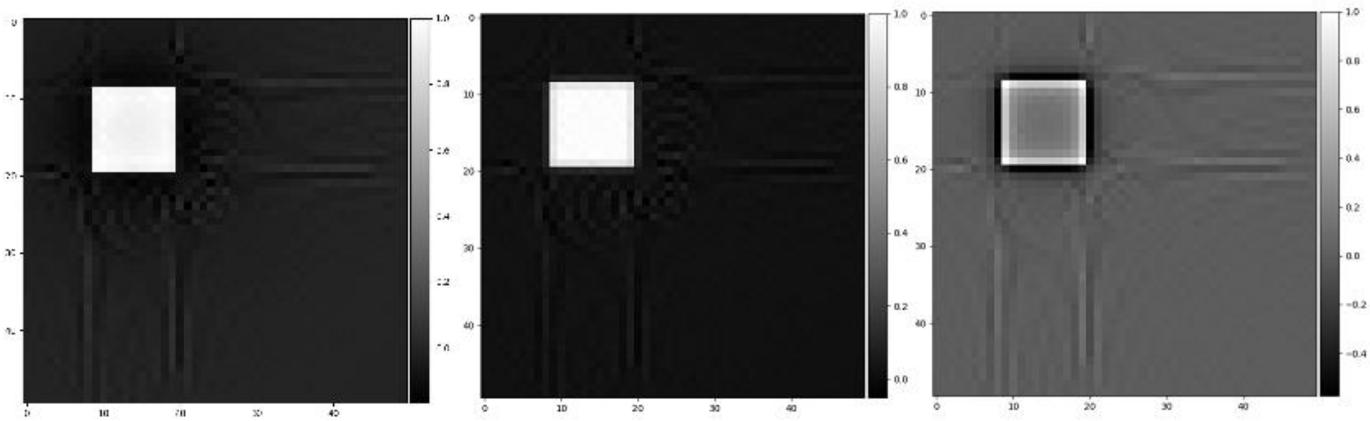

Fig 6: Images of the Square reconstructed with ramp, Hanning, Cosine filters, respectively. (180 fans and 100 detectors)

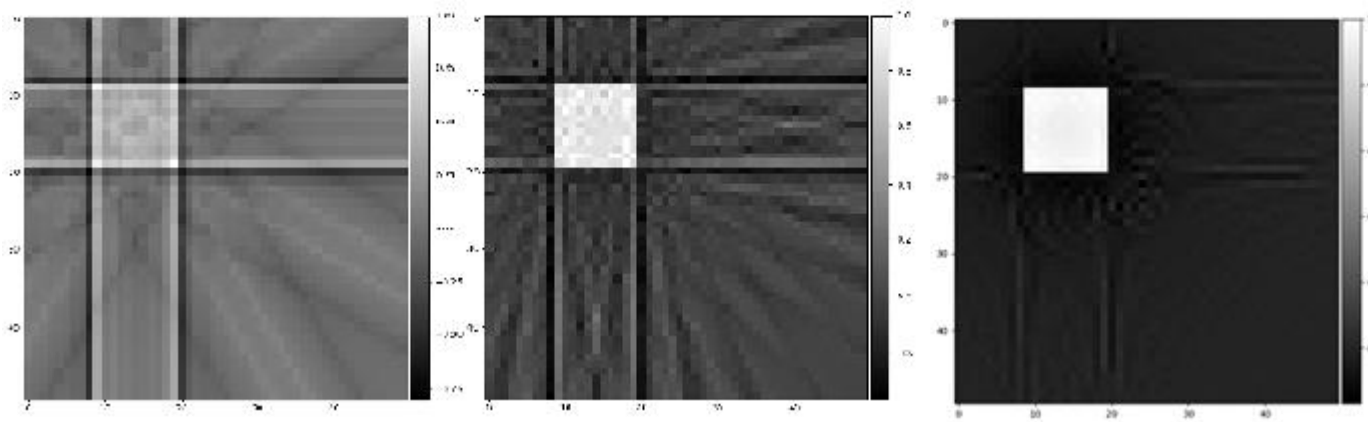

Fig 7: Images of the Square reconstructed with the step sizes 30,10, and 1 respectively. (Ramp filter, 100 detectors)

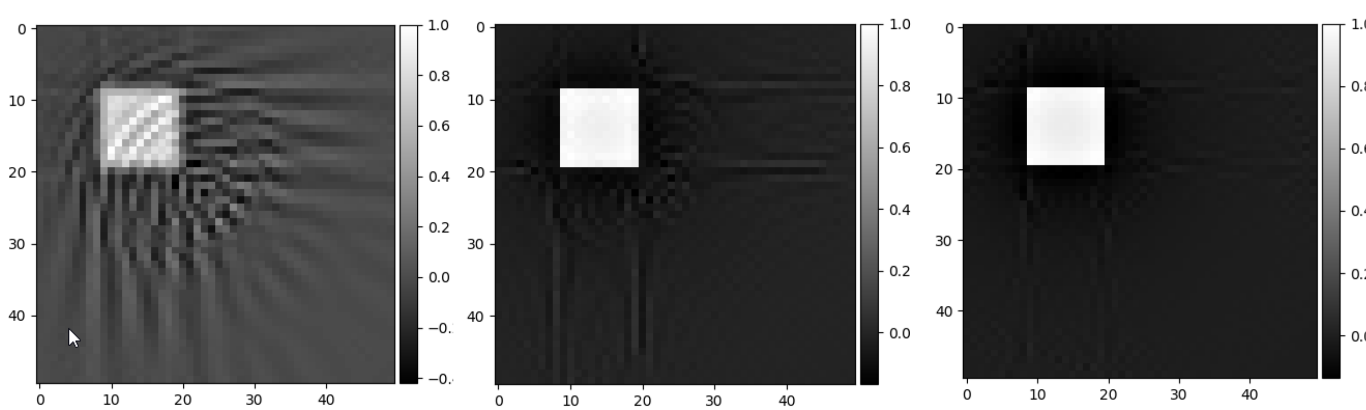

Fig 8: Images of the Square reconstructed with the number of beams 30, 100, 180 respectively. (Ramp filter, 180 fans)

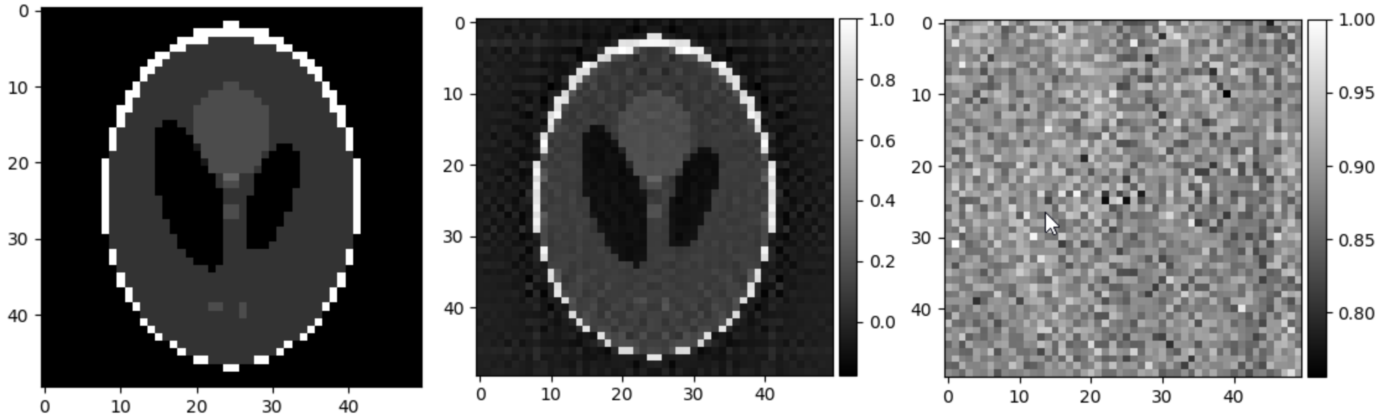

Fig 9: Original Shepp-Logan, reconstructed with and without filter, respectively. (Ramp filter, 180 fans and 100 detectors)

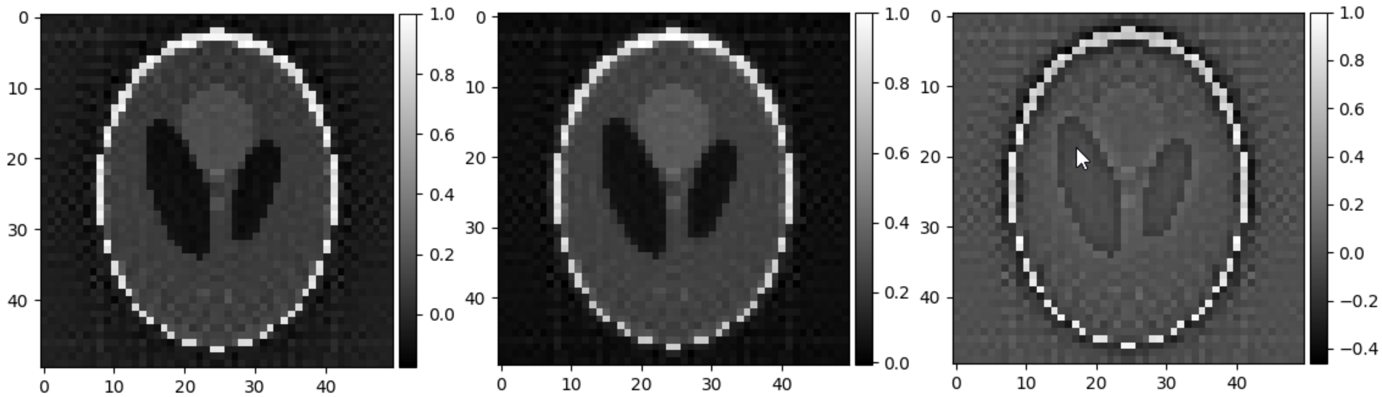

Fig 10: Shepp-Logan image reconstructed with ramp, Cosine and Hanning filters, respectively. (180 fans and 100 detectors)

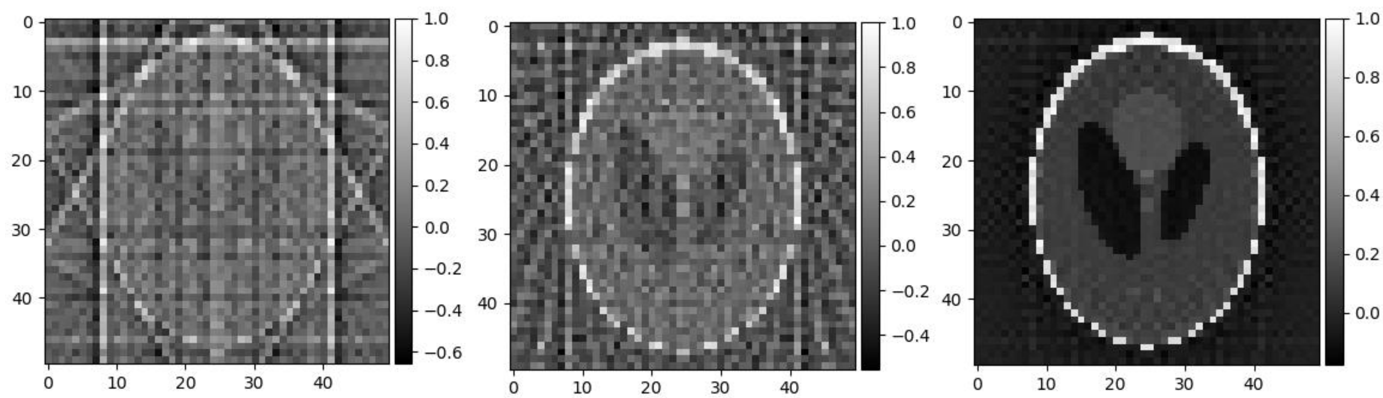

Fig 11: Images of the Shepp-Logan reconstructed with the step sizes 30,10, and 1 respectively. (Ramp filter, 100 detectors)

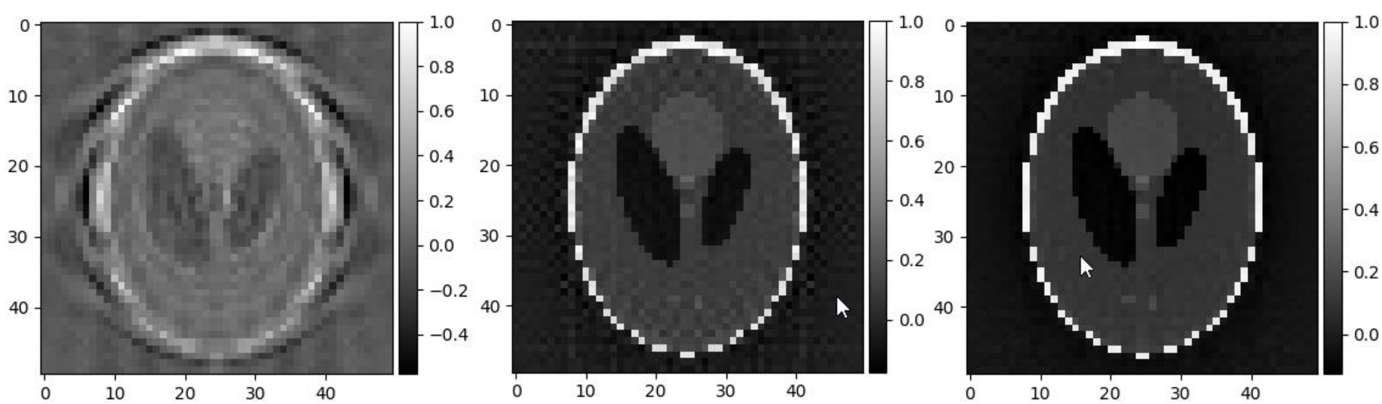

Fig 12: Images of the Shepp-Logan reconstructed with the number of beams 30, 100, 180 respectively. (Ramp filter, 180 fans)

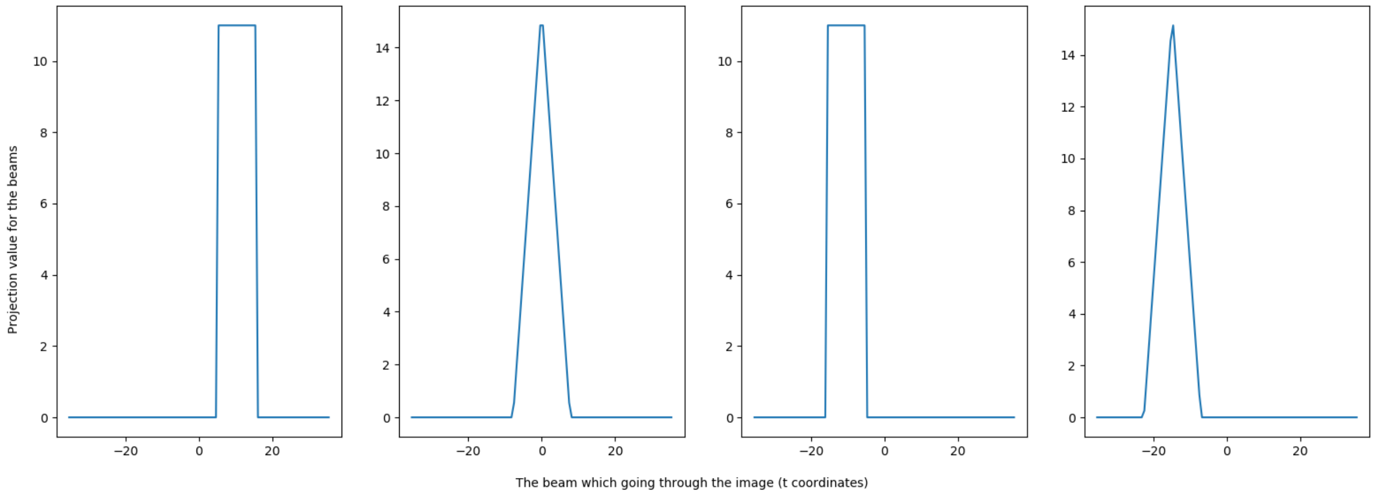

Fig 13: Projections of the square image at 0, 45, 90, 135 degrees, respectively. (See Fig 4.)

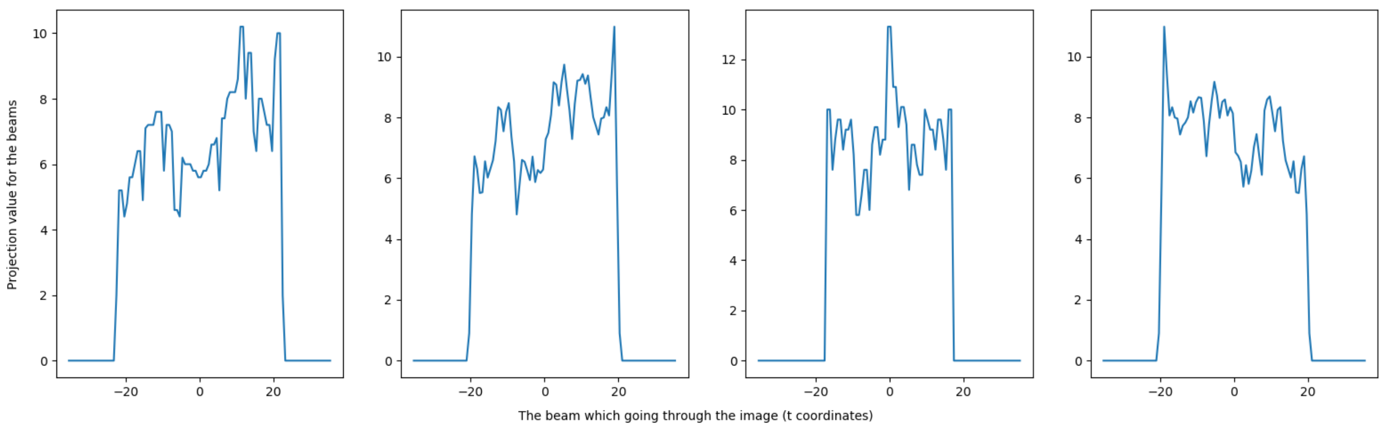

Fig 14: Projections of the Shepp-Logan image at 0, 45, 90, 135 degrees, respectively. (See Fig 4.)

APPENDIX A:

```
import PySimpleGUI as sg
import sys,pickle,time
from mplcursors import cursor
import scipy.io as sio
from scipy import signal
import numpy as np
import matplotlib.pyplot as plt
from numpy.fft import fft2,ifft2
from mpl_toolkits.axes_grid1 import make_axes_locatable
sg.change_look_and_feel('DefaultNoMoreNagging')
layout = [                                                  # Here's for the GUI window
        [sg.Text('Choose where you get the projection data from:')],
   [sg.Radio('From text file      ', "RADIO2"), sg.Radio('From mat file     ', "RADIO2"),
        sg.Radio('Do new projection              ', "RADIO2", default=True)],
        [sg.Text('Enter the number of beams:')],
        [sg.InputText('100')],
        [sg.Text('Enter the step size:')],
        [sg.InputText('30')],
        [sg.Text('kare_kosede_50ye50.mat is the default')],
        [sg.Listbox(values=['cameraman_256_256.mat','bird_472_472.mat', 'lena_256ya256.mat', 'horse_400_400.mat', 'Shepp-Logan.mat'],
        default_values=['kare_kosede_50ye50.mat'], size=(30, 5))],
        [sg.Text('Choose filter type:')],
        [sg.Radio('Ramp          ', "RADIO3", default=True), sg.Radio('Hanning     ', "RADIO3"),
        sg.Radio('Cosine     ', "RADIO3"), sg.Radio('No filter ', "RADIO3")],
        [sg.Checkbox('Do only projection', default=False),sg.Text(' '*15+'Enter the projection angle:'),
        sg.InputText(size=(5,1))],
        [sg.Submit(), sg.Cancel()]]
window  = sg.Window('Projection GUI', auto_size_text=True, default_element_size=(40, 1)).Layout(layout)
while True:
        event, values = window.Read()
        if event == 'Submit':
                window.Close()
                break
        elif event == 'Cancel':
                window.Close()
                sys.exit()
pi = np.pi
if values[6] == True:
        filter = 6
        filter_name = 'Ramp Filter'
elif values[7] == True:
        filter = 7
        filter_name = 'Hanning Filter'
elif values[8] == True:
        filter = 8
        filter_name = 'Cosine Filter'
elif values[9] == True:
        filter = 0
        filter_name = 'No Filter'
def project():
        pro_bas = time.time()
        y_values = x_values = np.arange(-size/2, size/2+1)
                                # determine x & y values on the image
        t = np.linspace(-size/pow(2,1/2), size/pow(2,1/2),number_of_beams)
        carp = size * np.sqrt(2)
        karsi_uz = np.where(teta <= 90,carp * np.cos((45-teta) * pi/180),carp * np.cos((135-teta) * pi/180))
        # 5. step: Find all intersection points for all beams for all projection angles using line equation:
        result=[]
        for aci in teta_degree:
                tan = np.tan(aci)
                cos = np.cos(aci)
                for t_degeri in t:
                        for x_degeri in x_values:
                                resulted_y_values = tan * x_degeri + t_degeri / cos #line equation

        result.append([aci,t_degeri,x_degeri,resulted_y_values])
        for aci in teta_degree:
                cos = np.cos(aci)
                sin = np.sin(aci)
                for t_degeri in t:
                        for y_degeri in y_values:
                                if aci==0 and y_degeri==t_degeri:
        # in case of 0 in the denominator
                                        for x_degeri in x_values:

        result.append([aci,t_degeri,x_degeri,y_degeri])
                                elif aci != 0:

        resulted_x_values = (y_degeri * cos - t_degeri)/sin # line equation

        result.append([aci,t_degeri,resulted_x_values,y_degeri])
        # Remove the repeated points:
        final_result = [list(t) for t in set(tuple(element) for element in result)]
        son = []
        # 6. Step: Remove the points which are irrelevant to the object:
        # Bu işlemle irrelevant noktaları attığımız için mesela 0 derece t=sqrt(-2) noktaları gitti
        for element in final_result:                    # 6.5 saniye
                if ((element[2]) <= (x_values[-1]) and (element[2]) >= (x_values[0]) and (element[3]) <= (y_values[-1]) and (element[3]) >= (y_values[0])):
                        son.append(element)
        son=sorted(son)
                # 7. Step: Sort the relevant points
# Below, I grouped the elements of 'son' variable with respect to their angle and t values while it had one row only before this işlem
        temp_aci_t_degeri = son[0][0:2]
        alt_liste=[son[0]]
        son_son=[]
        for i in son[1:]:
```

```
        if i[0:2] == temp_aci_t_degeri:
            alt_liste.append(i)
            temp_aci_t_degeri = i[0:2]
        else:
            son_son.append(alt_liste)
            alt_liste = []
            alt_liste.append(i)
            temp_aci_t_degeri = i[0:2]
    son_son.append(alt_liste)
    # 8. Find the midpoint and the length of line
segments:
    midX=[]
    midY=[]
    distance_son_son=[]
    for i in son_son:
        temp=i[0]
        distance=[]
        for j in i[1:]:
            temp_midX=((j[2]+temp[2])/2)
            temp_midY=((j[3]+temp[3])/2)
            dist_temp = pow((j[2]-
temp[2])*(j[2]-temp[2])+(j[3]-temp[3])*(j[3]-temp[3]),1/2)
            midX.append(temp_midX)
            midY.append(temp_midY)
            distance.append(dist_temp)
            temp = j
        distance_son_son.append(distance)

    # 9. Detect the address (row and column data) by
using the midpoint data.
    rowdata = (np.ceil(size/2 - np.floor(midY))-1)
    columndata = (np.ceil(size/2 - np.floor(midX))-1)
    # 10. Sum all pixel value and distance products
    say = 0
    projection = []
    for i in distance_son_son:
        toplam=0
        for j in i:
            toplam += (j *
img[int(rowdata[say])][int(columndata[say])])
            say=say+1
        projection.append(toplam)
    grup = []
    sa = 0
    for te in teta:
        if ( int(te) == 45 or int(te) == 135):
            grup.append(number_of_beams)
        else:
            k = 0
            for i in range(len(t)):
                if abs(t[i]) >
karsi_uz[sa]/2:
                    k+=1
                else:
                    break
            grup.append(number_of_beams-
k*2)
        sa += 1
    # açılara göre gruplu projection:
    Inputt = iter(projection)
    son_projection_with_zeros =
[list(__import__('itertools').islice(Inputt, elem)) for elem in
grup]
    # açılara göre gruplu distance:
    Inputt = iter(distance_son_son)
    son_distance_with_zeros =
[list(__import__('itertools').islice(Inputt, elem)) for elem in
grup]
    # pad the projection with 0s which occur when the
teta values other than 45 and 90 degrees
    son_projection_with_zeros_yeni = np.empty(0)
    for pro in son_projection_with_zeros:
        a = int( ( number_of_beams - len(pro) ) / 2)
        b = number_of_beams - a - len(pro)
        ekle = np.pad(pro,(a,b),'constant')
        son_projection_with_zeros_yeni =
np.append(son_projection_with_zeros_yeni,ekle)
    son_projection_with_zeros =
son_projection_with_zeros_yeni.reshape(number_of_projectio
ns,number_of_beams).tolist()
    grup_say=0
    for pro in son_distance_with_zeros:
        if (len(pro) < number_of_beams):
            for i in
range(int((number_of_beams - grup[grup_say])/2)):
                pro.insert(0,0)
                pro.insert(len(pro),0)
        grup_say+=1
    with open('projection_data.txt','w') as dosya_txt:

    dosya_txt.write(str(number_of_projections)+'\n'+str(
number_of_sampling_points)+'\n')
        for k in
range(len(son_projection_with_zeros)):
            dosya_txt.write(str(k+1)+'\n')
            for j in
son_projection_with_zeros[k]:
                dosya_txt.write(str(j)+'\n')
    mat_array=np.array(son_projection_with_zeros)
    #list to ndarray conversion
    column_array=np.array(columndata)
    row_array=np.array(rowdata)
    with open('distance_list.obj','wb') as dist:
        pickle.dump(son_distance_with_zeros,dist)
    sio.savemat('projection_datas/'+values[5][0][:-
4]+'_projection_data.mat',
    mdict={ 'projection':
mat_array,'columndata':column_array,'rowdata':row_array,'siz
e':size, 'original':img     })
    print('projection time: ',time.time() - pro_bas)
    if values[10] == True:
            # If we do projection only

    plot_projection(t,son_projection_with_zeros,number
_of_sampling_points,step_size)
    return
son_projection_with_zeros,son_distance_with_zeros,rowdata,
columndata
def
plot_projection(t,projection,number_of_sampling_points,step_
size):
    if values[11] == '':
    # plot the projection of only an angle
        fig, axs = plt.subplots(1,4)
        sayyy = 0
        for i in axs.flatten():
            i.plot(t.round(2),projection[sayyy])
            sayyy += 1
```

```
            fig.text(0.5, 0.02, 'The beam which going through the image (t coordinates)',ha='center')
            fig.text(0.1, 0.3, 'Projection value for the beams',ha='center',rotation='vertical')
            plt.suptitle('Projections for '+'\nNumber of sampling points: '+str(number_of_sampling_points)+'\n'+' Step size: '+str(step_size))
            plt.figure()
            plt.imshow(img,cmap='gray')
            plt.title('Original image')
        elif values[11] == 'all':                    # plot sinogram
            fig, axes = plt.subplots(1,2)

        axes[1].imshow(np.array(projection).T,cmap='gray')
            axes[1].set_ylabel('The beam which going through the image (t coordinates)')
            axes[1].set_xlabel('Angle')
            axes[1].set_title('Sinogram for '+'\nNumber of sampling points: '+str(number_of_sampling_points)+'\n'+' Step size: '+str(step_size))
            axes[0].set_title('Original')
            axes[0].imshow(img,cmap='gray')
        else:
            # plot the projection of only an angle
            cizdirilecek_aci = float(values[11])
            cizdirilecek_acinin_indexi = np.where(teta==cizdirilecek_aci)[0][0]
            fig, axes = plt.subplots(1,2)

        axes[1].plot(t.round(2),projection[cizdirilecek_acinin_indexi],'ro')
            axes[1].set_xlabel('The beam which going through the image (t coordinates)')
            axes[1].set_ylabel('Projection value for the beams')
            axes[1].set_title('Projections for '+str(cizdirilecek_aci)+'$^\circ$'+'\nNumber of sampling points: '+str(number_of_sampling_points)+'\n'+' Step size: '+str(step_size))
            axes[0].set_title('Original')
            axes[0].imshow(img,cmap='gray')
        cursor(multiple=True)
        plt.subplots_adjust(left=0.125, bottom=0.1, right=0.9 , top=0.85 , wspace=0.4, hspace=0.2)
        plt.show()
def ramp_filter():
        t = np.linspace(0, 1, number_of_sampling_points)
        return abs(abs(signal.sawtooth(2 * pi * t))-1)
def filter_it(filter_type=None,high_pass_filter=None):
        fft_of_projection = fft2(image_to_be_reconstructed)
        if high_pass_filter is None:
            high_pass_filter = filter_type(number_of_sampling_points)
        filtered_fft_of_projection = fft_of_projection * high_pass_filter
        return ifft2(filtered_fft_of_projection).tolist()
def filterla():
        if filter == 6:
            back_projection(filter_it (high_pass_filter=ramp_filter() ))
        elif filter == 7:
            back_projection(filter_it ( eval('np.hanning') ) )
        elif filter == 8:
            back_projection(filter_it ( eval('signal.cosine') ))
        elif filter == 0:
                        # no filter
            back_projection()
def back_projection(getir=None):
        back_pro_bas = time.time()
        if getir == None:
            getir = image_to_be_reconstructed
        # Multiply the filtered projection data with the distance:
        netice = []
        for i in getir:
            o = []
            for k in i:
                o.append(k * np.array(distance[getir.index(i)][i.index(k)]))
            netice.append(o)
        son_netice = []
        for i in netice:
            ara_netice=[]
            for k in i:
                if type(k) == np.ndarray:

        ara_netice.append(k.tolist())
                else:
                    ara_netice.append(k)
            son_netice.append(ara_netice)

        img_back = np.zeros((size,size))
        say = 0
        for i in son_netice:
            for j in i:
                if not j == 0:
                    for k in j:

        img_back[int(rowdata[say])][int(columndata[say])] += k.real
                        say += 1
        max_img = np.amax(img_back)
        img_normalized = img_back / max_img
        error_img = img - img_normalized
        # if you want error_img to be included, uncomment related parts
        img_normalized_er = error_img / np.amax(error_img)
        av_err = np.mean(img_normalized_er)
        mse = np.mean(np.square(img_normalized_er))
        print('back projection time: ',time.time() - back_pro_bas)
        print('av_err :',av_err)
        print('mse :',mse)
        fig,(original,back) = plt.subplots(1,2) #,error)
        plt.subplots_adjust(left=0.125, bottom=0.1, right=0.9 , top=0.9 , wspace=0.4, hspace=0.2)
        original.imshow(img,cmap='gray')
        # im_err = error.imshow(img_normalized_er,cmap='gray')
        # error related, comment/uncomment
        im_back = back.imshow(img_normalized,cmap='gray')
        divider_b = make_axes_locatable(back)
```

```
        # divider_e = make_axes_locatable(error)
                    # error related,
comment/uncomment
        cax1 = divider_b.append_axes("right", size="5%",
pad=0.05)
        # cax2 = divider_e.append_axes("right", size="5%",
pad=0.05)       # error related, comment/uncomment
        # original.set_title('Original image')
                    # error related,
comment/uncomment
        # back.set_title('Back projected image')
                    # error related,
comment/uncomment
        # error.set_title('Error')
                          # error
related, comment/uncomment
        fig.colorbar(im_back,cax=cax1)
        # fig.colorbar(im_err,cax=cax2)
                          # error
related, comment/uncomment
        fig_name = "number_of_sampling_points:
"+str(number_of_sampling_points)+" step_size:
"+str(step_size)+" "+filter_name+".png"
        plt.suptitle('number_of_sampling_points:
'+str(number_of_sampling_points)+'\n'+' step_size:
'+str(step_size)+'\n'+filter_name)
        plt.show()

if values[2] == True:

        # If "Do new projection" is chosen
        if values[5] == []:
                mat =
sio.loadmat('matlar/'+'kare_kosede_50ye50.mat')
        # 1. step:      load the default image
                values[5] = ['kare_kosede_50ye50.mat']
        else:

                        # or other image
                mat = sio.loadmat('matlar/'+values[5][0])
        img = list(mat.values())[3]
        size = img.shape[0]

                # 2. step:      determine the size of the
image
        number_of_sampling_points = number_of_beams =
int(values[3])          # 3. step:      get number of
beams
        step_size = float(values[4])

        #               get step_size
        teta = np.arange(0,180,step_size)
                          #
specify angle values according to the step size
        teta_degree = teta * pi / 180
        number_of_projections = teta_adedi = teta.shape[0]
        if values[10] == True:

        # Do only projection
                project()
        else:

        image_to_be_reconstructed,distance,rowdata,column
data = project()
                filterla()
else:

        # Use ready projection data (txt or mat)
        if values[0] == True:
                                # from txt
                with open('projection_data.txt') as
dosya_txt:
                        # data_from_txt = dosya_txt.read()
                        lines_from_txt =
dosya_txt.readlines()
                        number_of_projections =
int(lines_from_txt[0])
                        number_of_sampling_points =
int(lines_from_txt[1])
                image_to_be_reconstructed =
image_to_be_reconstructed.tolist()
                step_size = 180/number_of_projections
                size = mat_liste[6][0][0]
                columndata = mat_liste[4].tolist()[0]
                rowdata = mat_liste[5].tolist()[0]
                img = mat_liste[7]
                with open('distance_list.obj','rb') as dist:
                        distance = pickle.load(dist)
        elif values[1] == True:

        # from mat
                if values[5] == []:
                        values[5] =
['kare_kosede_50ye50.mat']
                mat =
sio.loadmat('projection_datas/'+values[5][0][:-
4]+'_projection_data.mat')
                mat_liste = list(mat.values())
                image_to_be_reconstructed = mat_liste[3]
                number_of_projections =
image_to_be_reconstructed.shape[0]
                number_of_sampling_points =
number_of_beams = image_to_be_reconstructed.shape[1]
                image_to_be_reconstructed =
image_to_be_reconstructed.tolist()
                step_size = 180/number_of_projections
                size = mat_liste[6][0][0]
                columndata = mat_liste[4].tolist()[0]
                rowdata = mat_liste[5].tolist()[0]
                img = mat_liste[7]
                with open('distance_list.obj','rb') as dist:
                        distance = pickle.load(dist)
        filterla()
```

APPENDIX B:

- Be sure that you are in the project code main folder. Required modules of Python are installed using this command in the command prompt: “pip install –r requirements.txt”
- The program is run by writing “python projection.py” at the command prompt. Then a graphical user interface window opens. User chooses the required options and enters required fields, but is not interested in unrelated options and fields. Then after a while, resultant images are shown.