[[1]](#footnote-1)

*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.*

**IMAGE RECONSTRUCTION**

**UNDER VARIOUS CONDITIONS USING PARALLEL BEAM*S***

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

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

# 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.

# THEORY

This study contains two parts as Projection and Image Reconstruction.

## 

Fig 1: Basic schematic of the term project.

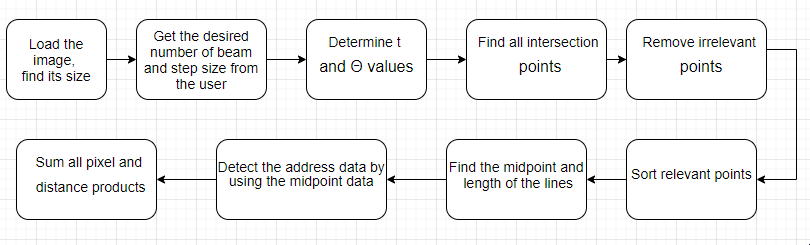


Fig 2: A brief flowchart of the projection algorithm.

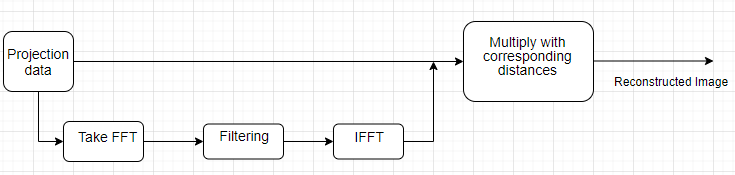


Fig 3: A brief flowchart of the backprojection algorithm.

## Mathematical background

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

## 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]

## 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.

# 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 Fig. 4(a) and Fig. 5(a) 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.

# 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.

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 Tımes Under

Dıfferent Condıtıons of Projectıons

|  |  |  |  |
| --- | --- | --- | --- |
| 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 |

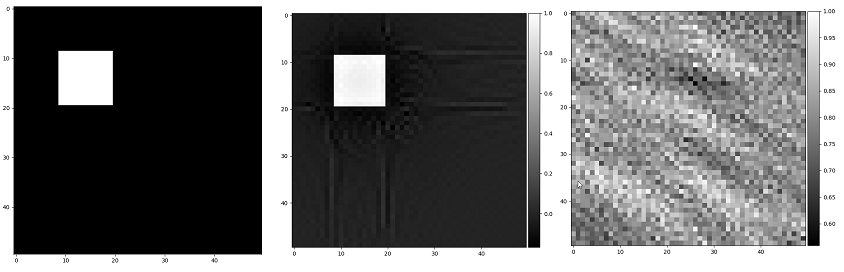


Fig 4: Images of Square, reconstructed with and without filter, respectively. (Filter type is ramp)

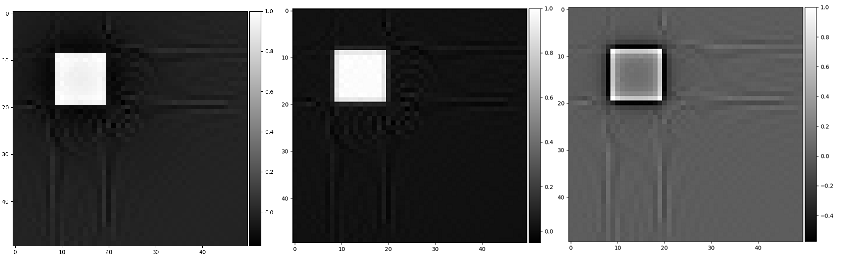


Fig 5: Images of the Square reconstructed with ramp, Hanning, Cosine filters, respectively.

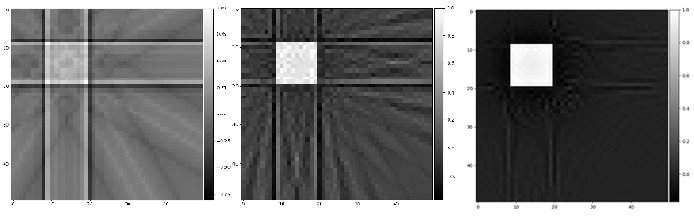


Fig 6: Images of the Square reconstructed with the step sizes 30,10, and 1 respectively.

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':

break

elif event == 'Cancel':

window.Close()

sys.exit()

if event == 'Submit':

window.Close()

elif event == 'Cancel':

sys.exit()

pi = np.pi

if values[6] == True:

filter = 6

filter\_name = 'Ramp Filter'

else:

if 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\_projections,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(values[5][0][:-4]+'\_projection\_data.mat', mdict={ 'projection': mat\_array,'columndata':column\_array,

'rowdata':row\_array,'size':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] == '':

fig, axs = plt.subplots(2,3)

sayyy = 0

for i in axs.flatten():

i.plot(t.round(2),projection[sayyy])

sayyy += 1

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':

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:

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,columndata = 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.

1. [↑](#footnote-ref-1)