**Emre ATAKLI - 2112944 Date of Submission: 30.11.2019**

Duration to complete Step-1: 10 days

**EE415 Term Project Step 1-Forward Problem**

# Algorithm

**General structure of the algorithm:**

Get desired number of beams and step size from the ıuser

Remove irrelevant points

Sort relevant points

Find the midpoint and length of the lines

Sum all pixel and distance products

Detect the address data by using the midpoint data

Find all intersection points

Determine t and Θ values

Load the image, find its size

Figure 1: A brief flowchart of the overall algorithm.

**Relevant point calculation:**

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])

Since we are required to do the relevant point calculation for every x/y for every t for every degree, I used three nested for loops to calculate x and y values.

The only difference in the calculation of the relevant y points compared to x points is discarding the cases the angle is zero and y\_degeri==t\_degeri, so the denominator is zero.

# Results and Comments

It should be noted that in this algorithm, the beams used in projection are in the manner that is in the Figure 2. Moreover radon() function used to validate the output of the projection algorithm in this document is under transform module which is in skimage package in Python.

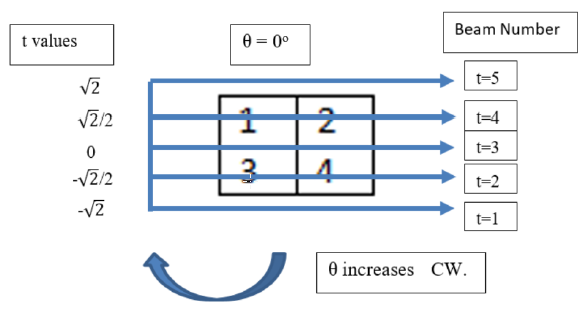


Figure 2: Beam number and t values for the sample image when Θ = 0°

**Output 1:**

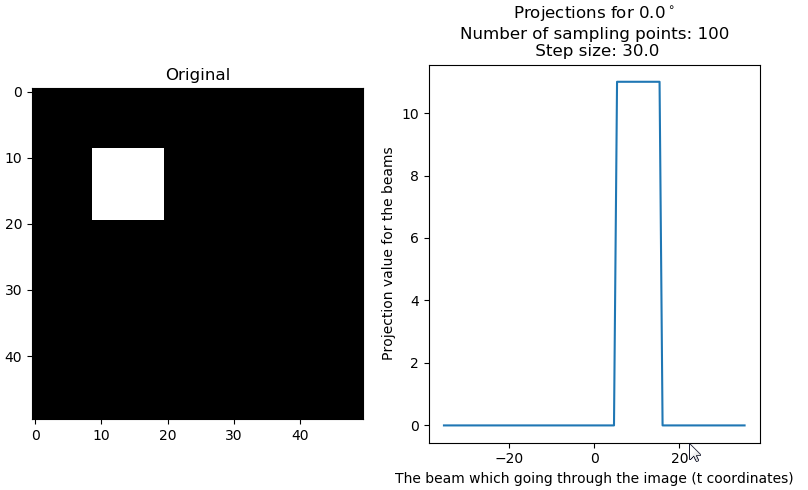
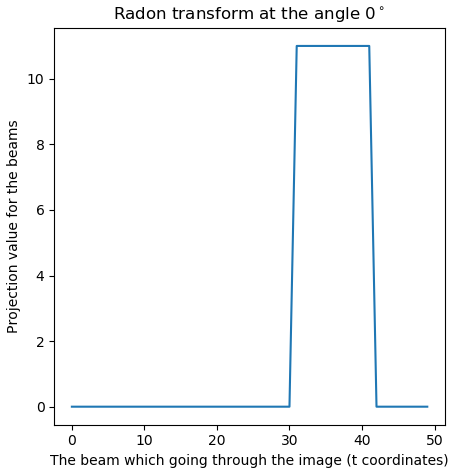
 

Figure 3: The square image (left), its corresponding projection at (middle), its validation (right).

**Comment-1:** The shape of the image is as expected because when the beams come through the small square , they coincide with it with 90°. The amplitude is also correct because all the coming beams encounter 11 pixels. Validation of the output is given in Figure 2.

**Output 2:**

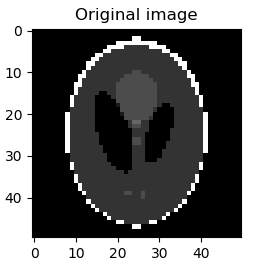
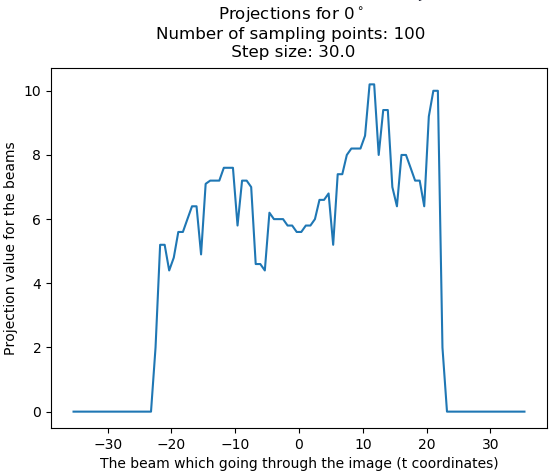
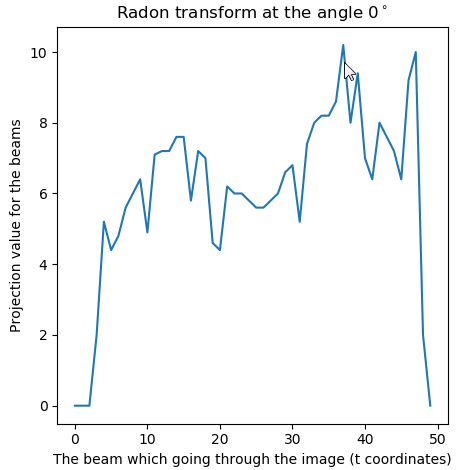
**** ****

Figure 4: The Shepp-Logan image (left), its corresponding projection at (middle), its validation (right).

**Comment-2:** This projection is as expected. Because the values at which t is positive should start higher than the side at which t is negative. Validation of the output is given in Figure 3.

**Output 3:**

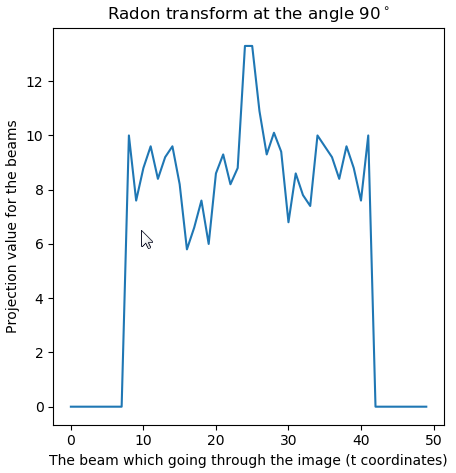
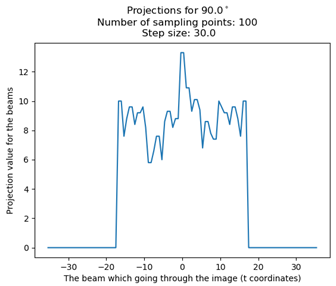
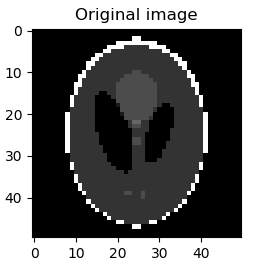
****

Figure 5: The Shepp-Logan image (left), its corresponding projection at (middle), its validation (right).

**Comment-3:** This projection is as expected since the image is seen symmetric from the angle 90°, the projection values should be seen almost symmetric. Furthermore, as expected, the projection value at t = 0 is maximum. Validation of the output is given in Figure 4.

**Output 4:**

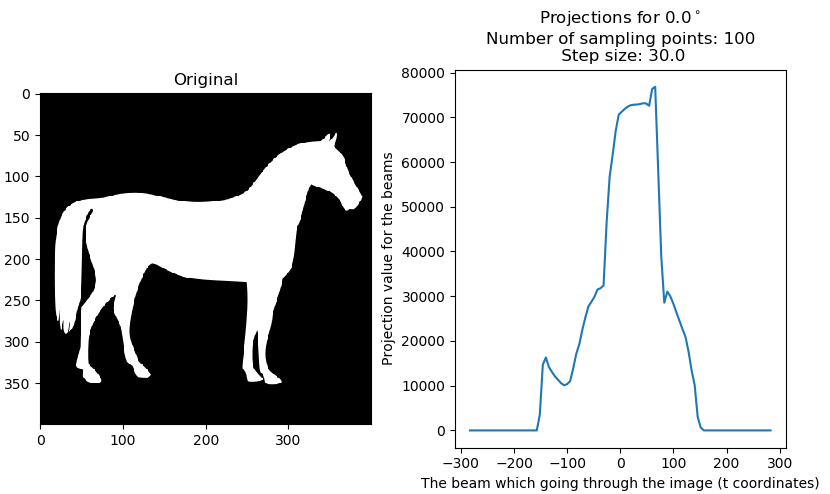
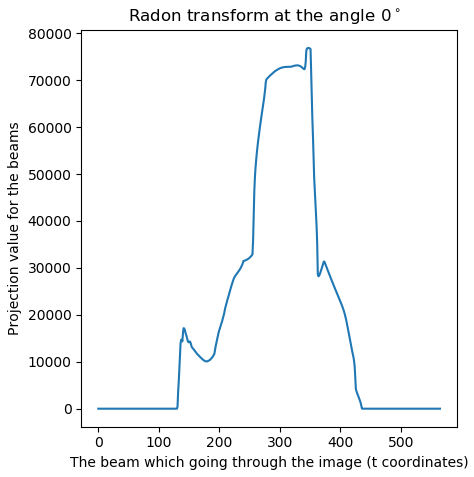
 

Figure 5: A horse image (left), its corresponding projection at (middle), its validation (right).

**Comment-4:** The maximum projection values of the horse image from the angle 0° is around t=0 as expected. Validation of the output is given in Figure 5.

# Appendix

**The code:**

import PySimpleGUI as sg

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=['bird\_472\_472.mat', 'lena\_256ya256.mat', 'horse\_400\_400.mat', 'Shepp-Logan.mat'],

default\_values=['kare\_kosede\_50ye50.mat'], size=(30, 3))],

[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=True),sg.Text(' Enter the projection angle:'),sg.InputText(size=(5,1))],#sg.Checkbox('Show Error Image')],

[sg.Submit(), sg.Cancel()]]

window = sg.Window('Projection GUI', auto\_size\_text=True, default\_element\_size=(40, 1)).Layout(layout)

import sys

import time

import pdb

from mplcursors import cursor

while True:

event, values = window.Read()

if event == 'Submit':

break

elif event == 'Cancel':

sys.exit()

# temp = values[3]

# values.pop(3)

# values.append(temp)

if event == 'Submit':

window.Close()

elif event == 'Cancel':

sys.exit()

import scipy.io as sio

from scipy import signal

import numpy as np

import pickle

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'

import matplotlib.pyplot as plt

from itertools import groupby

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)) # önce: 0.0156, şimdi: 0

# 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: # 9.25 -> 2.153

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])

# np.where(aci==0 and y\_values == t\_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:

bak = time.time()

# final\_result=[list(t) for t in set(tuple(element) for element in result)] # 13.84 saniye #5 sec @100-5 sec

# list unhashable olduğu için ilk önce tuple'a çeviriyorum, sonra aynı olan 'element'leri set ile tekliyorum:

final\_result1=[list(t) for t in set(tuple(np.round(element,2)) 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\_result1: # 6.5 saniye

if (float(element[2]) <= float(x\_values[-1]) and float(element[2]) >= float(x\_values[0]) and float(element[3]) <= float(y\_values[-1]) and float(element[3]) >= float(y\_values[0])):

son.append(element)

son=sorted(son) # 7. Step: Sort the relevant points (2.2 saniye) # 0.3 sec

# 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: # 3.32 saniye

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: # 2.24 saniye

toplam=0

for j in i:

toplam=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:

son\_projection=[]

say\_sırala = 0

for grup\_elemanı in grup:

ara\_projection=[]

for i in range(grup\_elemanı):

ara\_projection.append(projection[i+ say\_sırala])

say\_sırala = i+ say\_sırala + 1

son\_projection.append(ara\_projection)

# açılara göre gruplu distance:

say\_sırala = 0

son\_distance=[]

for grup\_elemanı in grup:

ara\_distance=[]

for i in range(grup\_elemanı):

ara\_distance.append(distance\_son\_son[i+ say\_sırala])

say\_sırala = i+ say\_sırala + 1

son\_distance.append(ara\_distance)

# make the projection with 0s which occur when the teta values other than 45 and 90 degrees

import copy

son\_projection\_with\_zeros = copy.deepcopy(son\_projection)

son\_distance\_with\_zeros = copy.deepcopy(son\_distance)

grup\_say=0

for pro in son\_projection\_with\_zeros: #4.26 saniye

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

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\_data.mat', mdict={ 'projection': mat\_array,'columndata':column\_array,

'rowdata':row\_array,'size':size, 'original':img })

print('projection time: ',time.time()-pro\_bas)

# pdb.set\_trace()

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')

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()

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])

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()

from numpy.fft import fft2,ifft2

from mpl\_toolkits.axes\_grid1 import make\_axes\_locatable

# elif values[1] == True:

# elif values[0] == True:

def ramp\_filter():

filter\_bas = time.time()

fft\_of\_projection = fft2(image\_to\_be\_reconstructed)

if number\_of\_sampling\_points % 2 == 0:

temp = number\_of\_sampling\_points/2

first\_half\_of\_filter = np.linspace(0,1/(temp-0.5)\*(temp-1),temp)

high\_pass\_filter = np.array(list(first\_half\_of\_filter) + list(first\_half\_of\_filter[::-1]))

else:

temp = np.floor(number\_of\_sampling\_points/2) + 1

first\_half\_of\_filter = np.linspace(0,1,temp)

high\_pass\_filter = np.array(list(first\_half\_of\_filter) + list(first\_half\_of\_filter[::-1][1:]))

filtered\_fft\_of\_projection = fft\_of\_projection \* high\_pass\_filter

ifft\_of\_projection = ifft2(filtered\_fft\_of\_projection)

# ifft\_of\_projection'ı array'den listeye çevir:

liste\_ifft\_of\_projection = []

for k in ifft\_of\_projection:

liste\_ifft\_of\_projection.append([i for i in k])

print('filtering time: ',time.time()-filter\_bas)

return liste\_ifft\_of\_projection

def hanning\_filter():

filter\_bas = time.time()

fft\_of\_projection = fft2(image\_to\_be\_reconstructed)

high\_pass\_filter = np.hanning(number\_of\_sampling\_points)

filtered\_fft\_of\_projection = fft\_of\_projection \* high\_pass\_filter

ifft\_of\_projection = ifft2(filtered\_fft\_of\_projection)

# ifft\_of\_projection'ı array'den listeye çevir:

liste\_ifft\_of\_projection = []

for k in ifft\_of\_projection:

liste\_ifft\_of\_projection.append([i for i in k])

print('filtering time: ',time.time()-filter\_bas)

return liste\_ifft\_of\_projection

def hamming\_filter():

filter\_bas = time.time()

fft\_of\_projection = fft2(image\_to\_be\_reconstructed)

high\_pass\_filter = np.hamming(number\_of\_sampling\_points)

filtered\_fft\_of\_projection = fft\_of\_projection \* high\_pass\_filter

ifft\_of\_projection = ifft2(filtered\_fft\_of\_projection)

# ifft\_of\_projection'ı array'den listeye çevir:

liste\_ifft\_of\_projection = []

for k in ifft\_of\_projection:

liste\_ifft\_of\_projection.append([i for i in k])

print('filtering time: ',time.time()-filter\_bas)

return liste\_ifft\_of\_projection

def cosine\_filter():

filter\_bas = time.time()

fft\_of\_projection = fft2(image\_to\_be\_reconstructed)

high\_pass\_filter = signal.cosine(number\_of\_sampling\_points)

filtered\_fft\_of\_projection = fft\_of\_projection \* high\_pass\_filter

ifft\_of\_projection = ifft2(filtered\_fft\_of\_projection)

# ifft\_of\_projection'ı array'den listeye çevir:

liste\_ifft\_of\_projection = []

for k in ifft\_of\_projection:

liste\_ifft\_of\_projection.append([i for i in k])

print('filtering time: ',time.time()-filter\_bas)

return liste\_ifft\_of\_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)

kl=np.array([1.6024768-0.52718694j, 1.6024768-0.52718694j])

tur = type(kl)

son\_netice=[]

for i in netice:

ara\_netice=[]

for k in i:

if type(k) == tur:

daha\_ara\_netice=[]

for j in k:

daha\_ara\_netice.append(j)

ara\_netice.append(daha\_ara\_netice)

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

max\_img\_er=np.amax(error\_img)

img\_normalized\_er=error\_img/max\_img\_er

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,error) = plt.subplots(1,3)

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)

# fig\_name = "number\_of\_sampling\_points: "+str(number\_of\_sampling\_points)+" step\_size: "+str(step\_size)+" "+filter\_name+".png"

# plt.savefig(fig\_name) # anlamadım hatayı

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('kare\_kosede\_50ye50.mat') # 1. step: load the default image

else: # or other image

mat = sio.loadmat(values[5][0])

img = list(mat.values())[3]#:][0]

# img1=np.zeros((50,50))

# img1[29:40,9:20] = img[9:20,9:20]

# img1[9:20,29:40] = img[9:20,9:20]

# img = img1

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()

if filter == 6:

back\_projection(ramp\_filter())

elif filter == 7:

back\_projection(hanning\_filter())

elif filter == 8:

back\_projection(cosine\_filter())

elif filter == 0: # no filter

back\_projection()

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()

# pdb.set\_trace()

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]

# print(type(columndata))

# print('size: ',columndata)

rowdata = mat\_liste[5].tolist()[0]

# distance\_arr = mat\_liste[6]

img = mat\_liste[7]

with open('distance\_list.obj','rb') as dist:

distance = pickle.load(dist)

elif values[1] == True: # from mat

mat = sio.loadmat('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]

# distance\_arr = mat\_liste[6]

img = mat\_liste[7]

with open('distance\_list.obj','rb') as dist:

distance = pickle.load(dist)

# pdb.set\_trace()

if filter == 6:

back\_projection(ramp\_filter())

elif filter == 7:

back\_projection(hanning\_filter())

elif filter == 8:

back\_projection(cosine\_filter())

elif filter == 0: # no filter

back\_projection()