**Conventional simulation:**

Total\_view = 1400

Gantry\_rotation\_time = 500

View\_increment = 28

If folder name has “HR”, it means the motion was generated on data with z=0.625mm. this applied to 2D\_spline, 3D\_spline, 3D\_spline\_6degrees, 3D\_spline\_new.

2D\_spline (DoF = 3): only tx, ty, rz. Each parameter <= 5

3D\_spline (DoF = 4): only tx, tz, rx, rz. Total t <= 5, rx <=5, rz <=5.

3D\_spline\_6degrees (DoF = 6): each parameter <=5, rx+ry <=7

3D\_spline\_new (mild): total t <=5, total<=angle <=3

All above, amplitude\_max = 5, displacement\_max = 3, change\_direction\_limit = 2

**New portable CT motion simulation (also use the newly collected data with brain diseases in most of cases)**

Total\_view = 1400

Gantry\_rotation\_time = 500

View\_increment = 28

Image resample to [1,1,1].

z-coverage = 1cm, so 10 / 1mm = 10 slices as one stack in one gantry rotation.

In each rotation, use the current 10 slices as well as 20 slices before and 20 slices after to run the simulation.

Total rotation number = total slice // 10

**v1:**

Amplitude\_max\_severe = 5, displacement\_max\_severe = 3.

Amplitude\_max\_mild = 2, displacement\_max\_mild = 1.

Translation: tx and ty combine to have 5, tz in severe has amplitude max = 2, displacement\_max = 1 and in mild has 1,0.5.

Rotation: rx, ry, rz combine to have 5

Double skull:

Ty: 12, 8

Tx: 5, 3

Tz: 0,0

Rxryrz: 3,1

The frequency of motion happening in one gantry rotation: Motion\_freq = 1.1

The frequency of the occurance of double skull: Double\_skull\_freq = 0.1

If this rotation has motion, the frequency of having severe motion = 0.3

**v2:**

Amplitude\_max\_severe = 10, displacement\_max\_severe = 6.

Amplitude\_max\_mild = 5, displacement\_max\_mild = 3.

Translation: tx and ty combine to have 10/5, tz in severe has amplitude max = 3, displacement\_max = 1.5 and in mild has 2,1.

Rotation: rx, ry, rz combine to have 5 (always use mild)

Double skull:

Ty: 12, 8

Tx: 5, 3

Tz: 0,0

Rxryrz: 3,1

The frequency of motion happening in one gantry rotation: Motion\_freq = 0.5

The frequency of the occurance of double skull: Double\_skull\_freq = 0.25

If this rotation has motion, the frequency of having severe motion = 0.4

main\_folder = '/mnt/camca\_NAS/Portable\_CT\_data'

motion\_type = 'simulated\_all\_motion\_v1' # each gantry rotation has motion except the first one

amplitude\_max\_severe = 5 #8

displacement\_max\_severe = 3

amplitude\_max\_mild = 2 #5

displacement\_max\_mild = 1

motion\_freq = 1.1 # 0.3 or 0.35

severe\_freq = 0.3 # 0.6 or 0.65

double\_skull\_freq = 0.1 # 0.3 or 0.35

change\_direction\_limit = 2

CP\_num = 5

geometry = 'fan'

total\_view = 1400 ### 2340 views by default

gantry\_rotation\_time = 500 #unit ms, 500ms by default

view\_increment = 28 # increment in gantry views

# define the patient list

patient\_sheet = pd.read\_excel(os.path.join(main\_folder,'Patient\_list', 'NEW\_CT\_concise\_collected\_fixed\_static.xlsx'),dtype={'Patient\_ID': str, 'Patient\_subID': str})

patient\_sheet['use'] = patient\_sheet['use'].fillna(0)

patient\_sheet = patient\_sheet[(patient\_sheet['use'] != 0) & (patient\_sheet['use'] != 'no')]

print('patient sheet len: ', len(patient\_sheet))

data\_folder = os.path.join(main\_folder, 'nii\_imgs\_202404', 'static')

save\_folder = os.path.join(main\_folder, 'simulations\_202404', motion\_type)

ff.make\_folder([save\_folder])

########### define patient list index and simulation index

L = np.arange(0,1)

patient\_index\_list = np.arange(0,patient\_sheet.shape[0])

for i in range(0,patient\_sheet.shape[0]):

row = patient\_sheet.iloc[i]

patient\_id = row['Patient\_ID']

patient\_subid = row['Patient\_subID']

print('\n',i, patient\_id, patient\_subid)

save\_folder\_patient = os.path.join(save\_folder, patient\_id, patient\_subid)

ff.make\_folder([os.path.join(save\_folder, patient\_id), save\_folder\_patient])

img\_file = ff.find\_all\_target\_files(['fixed/img\_1mm.nii.gz'],os.path.join(data\_folder, patient\_id, patient\_subid))

if len(img\_file) != 1:

ValueError('no raw data')

img,spacing,img\_affine = ct.basic\_image\_processing(img\_file[0])

print('nib image shape: ',img.shape, ' spacing: ',spacing)

# define projectors

img = img[np.newaxis, ...]

projector = ct.define\_forward\_projector(img,spacing,total\_view)

fbp\_projector = ct.backprojector(img,spacing)

# very important - make sure that the arrays are saved in C order

cp.cuda.Device(0).use()

ct\_projector.set\_device(0)

# load the static image as reference

if os.path.isfile(os.path.join(main\_folder, 'simulations\_202404','simulated\_all\_motion',patient\_id, patient\_subid, 'static','image\_data','recon.nii.gz')) == 0:

static\_ref = None

else:

static\_ref= nb.load(os.path.join(main\_folder, 'simulations\_202404','simulated\_all\_motion',patient\_id, patient\_subid, 'static','image\_data','recon.nii.gz')).get\_fdata()

static\_ref = np.rollaxis(static\_ref,2,0)

print('static ref shape: ', static\_ref.shape)

print('static\_ref is None? ', static\_ref is None)

for random\_i in L:

t = np.linspace(0, gantry\_rotation\_time, CP\_num, endpoint=True)

# create folder

if random\_i == 0:

random\_folder = os.path.join(save\_folder\_patient,'static')

else:

random\_folder = os.path.join(save\_folder\_patient,'random\_' +str(random\_i))

ff.make\_folder([random\_folder, os.path.join(random\_folder,'image\_data')])

print('\n',random\_i , 'random')

if os.path.isfile(os.path.join(random\_folder,'image\_data','recon\_resample.nii.gz')) == 1:

print('already done this motion')

continue

# gantry coverage = 1cm, which means it rotates once for every 1cm interval in the z-axis

# set a sga reference

sga\_list = []; sga\_reference = int(np.random.uniform(0,90))

# second, find out how many rotations we need

print('spacing: ', spacing, ' img shape: ', img.shape)

rotation\_num = int(spacing[0] \* img.shape[1] // 10)

slice\_coverage = 10

if random\_i == 0:

rotation\_num = 1

slice\_coverage = img.shape[1]

print('rotation\_num: ', rotation\_num, ' slice\_coverage: ', slice\_coverage)

# start to generate the motion for each rotation

# set whether static or motion for each rotation

if random\_i != 0:

while True:

motion\_status = [np.random.uniform(0,1) <motion\_freq for k in range(rotation\_num-2)]

if motion\_freq <=0:

break

if np.sum(motion\_status) > 0:

break

motion\_status = [False] + motion\_status + [False] # add two static rotations at the beginning and end

if random\_i == 0: # static

motion\_status = [False]

print('motion\_status: ', motion\_status)

amplitude\_collect = np.zeros([rotation\_num, CP\_num, 6])

recon = np.zeros([slice\_coverage \* rotation\_num,img.shape[2],img.shape[3]])

projection = np.zeros([slice\_coverage \* rotation\_num, total\_view, 1 , projector.nu])

for rot\_n in range(0,rotation\_num):

# first set the SGA:

sga = sga\_reference + int(np.random.uniform(-5,5))

sga = max(0, sga); sga = min(90, sga)

sga\_list.append(sga)

print('rotation: ', rot\_n, 'sga: ', sga)

# set the partial image:

slice\_start = rot\_n \* slice\_coverage

slice\_end = (rot\_n + 1) \* slice\_coverage

slice\_prior = max(0, slice\_start - slice\_coverage \* 2)

slice\_post = min(img.shape[1], slice\_end + slice\_coverage \* 2)

img\_partial = img[:,slice\_prior:slice\_post,:]

# set the motion

# first: determine whether in this rotation we have motion or static, and whether it's severe or mild

if motion\_status[rot\_n]:

# amplitude\_max\_tem\_r = amplitude\_max\_mild # rotation always max = 5 degree

# displacement\_max\_tem\_r = displacement\_max\_mild

if np.random.uniform(0,1) < severe\_freq: # severe motion

severe\_motion = True

amplitude\_max\_tem\_t, amplitude\_max\_tem\_r = amplitude\_max\_severe, amplitude\_max\_severe

displacement\_max\_tem\_t, displacement\_max\_tem\_r = displacement\_max\_severe, displacement\_max\_severe

else: # mild motion

severe\_motion = False

amplitude\_max\_tem\_t, amplitude\_max\_tem\_r = amplitude\_max\_mild, amplitude\_max\_mild

displacement\_max\_tem\_t, displacement\_max\_tem\_r = displacement\_max\_mild, displacement\_max\_mild

else:

severe\_motion = False

amplitude\_max\_tem\_t, displacement\_max\_tem\_t, amplitude\_max\_tem\_r, displacement\_max\_tem\_r = 0, 0, 0, 0

print('severe motion: ', severe\_motion)

# second: determine whether the initial pose has been changed compared to last rotation

offset\_values = [0] \* 6

# third: generate the motion

# translation

amplitude\_txty\_mm = transform.motion\_control\_point\_generation(2, CP\_num, amplitude\_max = amplitude\_max\_tem\_t, displacement\_max = displacement\_max\_tem\_t, change\_direction\_limit = change\_direction\_limit, offset\_value = offset\_values[0:2], print\_result =False)

amplitude\_tx\_mm = amplitude\_txty\_mm[:,0]

amplitude\_ty\_mm = amplitude\_txty\_mm[:,1]

if severe\_motion:

amplitude\_tz\_mm = transform.motion\_control\_point\_generation(1, CP\_num, amplitude\_max = 2, displacement\_max = 1, change\_direction\_limit = change\_direction\_limit, offset\_value = offset\_values[2], print\_result =False)[:,0]

elif not severe\_motion and motion\_status[rot\_n]:

amplitude\_tz\_mm = transform.motion\_control\_point\_generation(1, CP\_num, amplitude\_max = 1, displacement\_max = 0.5, change\_direction\_limit = change\_direction\_limit, offset\_value = offset\_values[2], print\_result =False)[:,0]

else:

amplitude\_tz\_mm = transform.motion\_control\_point\_generation(1, CP\_num, amplitude\_max = 0, displacement\_max = 0, change\_direction\_limit = change\_direction\_limit, offset\_value = offset\_values[2], print\_result =False)[:,0]

# rotations

while True:

amplitude\_rxryrz\_degree = transform.motion\_control\_point\_generation(3, CP\_num, amplitude\_max = amplitude\_max\_tem\_r, displacement\_max = displacement\_max\_tem\_r, change\_direction\_limit = change\_direction\_limit, offset\_value = offset\_values[3:6], print\_result =False)

amplitude\_rx\_degree = amplitude\_rxryrz\_degree[:,0]

amplitude\_ry\_degree = amplitude\_rxryrz\_degree[:,1]

amplitude\_rz\_degree = amplitude\_rxryrz\_degree[:,2]

if np.max(abs(amplitude\_rx\_degree - amplitude\_rx\_degree[0]))+ np.max(abs(amplitude\_ry\_degree - amplitude\_ry\_degree[0])) <= 5:

break

# let's also consider the double skull artifacts, espeically in the occipital bone

if np.random.uniform(0,1) < double\_skull\_freq and motion\_status[rot\_n]:

print('yes we have double skull')

while True:

amplitude\_ty\_mm = transform.motion\_control\_point\_generation(1, CP\_num, amplitude\_max = 12, displacement\_max = 8, change\_direction\_limit = change\_direction\_limit, offset\_value = offset\_values[1], print\_result =False)[:,0]

amplitude\_tx\_mm = transform.motion\_control\_point\_generation(1, CP\_num, amplitude\_max = 5, displacement\_max = 3, change\_direction\_limit = change\_direction\_limit, offset\_value = offset\_values[0], print\_result =False)[:,0]

amplitude\_tz\_mm = transform.motion\_control\_point\_generation(1, CP\_num, amplitude\_max = 0, displacement\_max = 0, change\_direction\_limit = change\_direction\_limit, offset\_value = offset\_values[2], print\_result =False)[:,0]

# amplitude\_tz\_mm = transform.motion\_control\_point\_generation(1, CP\_num, amplitude\_max = 1, displacement\_max = 0.5, change\_direction\_limit = change\_direction\_limit, offset\_value = offset\_values[2], print\_result =False)[:,0]

amplitude\_rxryrz\_degree = transform.motion\_control\_point\_generation(3, CP\_num, amplitude\_max = 3, displacement\_max = 1, change\_direction\_limit = change\_direction\_limit, offset\_value = offset\_values[3:6], print\_result =False)

amplitude\_rx\_degree = amplitude\_rxryrz\_degree[:,0]

amplitude\_ry\_degree = amplitude\_rxryrz\_degree[:,1]

amplitude\_rz\_degree = amplitude\_rxryrz\_degree[:,2]

if np.max(abs(amplitude\_ty\_mm - amplitude\_ty\_mm[0])) >= 7:

break

print('amplitude\_tx\_mm: ', amplitude\_tx\_mm)

print('amplitude\_ty\_mm: ', amplitude\_ty\_mm)

print('amplitude\_tz\_mm: ', amplitude\_tz\_mm)

print('amplitude\_rx\_degree: ', amplitude\_rx\_degree)

print('amplitude\_ry\_degree: ', amplitude\_ry\_degree)

print('amplitude\_rz\_degree: ', amplitude\_rz\_degree)

# save the motion parameters

parameter\_file = os.path.join(random\_folder,'motion\_parameters.txt')

if rot\_n == 0:

ff.txt\_writer(parameter\_file, True, [t.tolist(),amplitude\_tx\_mm, amplitude\_ty\_mm, amplitude\_tz\_mm, amplitude\_rx\_degree, amplitude\_ry\_degree, amplitude\_rz\_degree, [sga],[total\_view],[gantry\_rotation\_time]],['time\_points','translation\_x\_CP','translation\_y\_CP','translation\_z\_CP', 'rotation\_x\_CP', 'rotation\_y\_CP','rotation\_z\_CP','starting\_gantry\_angle', 'total\_projection\_view','gantry\_rotation\_time(ms)'])

else:

ff.txt\_writer(parameter\_file, False, [t.tolist(),amplitude\_tx\_mm, amplitude\_ty\_mm, amplitude\_tz\_mm, amplitude\_rx\_degree, amplitude\_ry\_degree, amplitude\_rz\_degree, [sga],[total\_view],[gantry\_rotation\_time]],['time\_points','translation\_x\_CP','translation\_y\_CP','translation\_z\_CP', 'rotation\_x\_CP', 'rotation\_y\_CP','rotation\_z\_CP','starting\_gantry\_angle', 'total\_projection\_view','gantry\_rotation\_time(ms)'])

collect = np.stack([amplitude\_tx\_mm, amplitude\_ty\_mm, amplitude\_tz\_mm, amplitude\_rx\_degree, amplitude\_ry\_degree, amplitude\_rz\_degree], axis = 1)

amplitude\_collect[rot\_n,...] = collect

# prepare spline fit

spline\_tx = transform.interp\_func(t, np.asarray([i/spacing[1] for i in amplitude\_tx\_mm]))

spline\_ty = transform.interp\_func(t, np.asarray([i/spacing[2] for i in amplitude\_ty\_mm]))

spline\_tz = transform.interp\_func(t, np.asarray([i/spacing[0] for i in amplitude\_tz\_mm]))

spline\_rx = transform.interp\_func(t,np.asarray([i / 180 \* np.pi for i in amplitude\_rx\_degree]))

spline\_ry = transform.interp\_func(t,np.asarray([i / 180 \* np.pi for i in amplitude\_ry\_degree]))

spline\_rz = transform.interp\_func(t,np.asarray([i / 180 \* np.pi for i in amplitude\_rz\_degree]))

angles = ff.get\_angles\_zc(total\_view, 360 ,sga)

# if static and has the static image ref, then no need to do the projection

if motion\_status[rot\_n] == False:

if static\_ref is not None:

recon[slice\_coverage \* (rot\_n) : slice\_coverage \* (rot\_n + 1),...] = static\_ref[slice\_coverage\*rot\_n:slice\_coverage\*(rot\_n+1),...]

continue

# generate forward projection

projection\_partial = ct.fp\_w\_spline\_motion\_model(img\_partial, projector, angles, spline\_tx, spline\_ty, spline\_tz, spline\_rx, spline\_ry, spline\_rz, geometry, total\_view = total\_view, gantry\_rotation\_time = gantry\_rotation\_time, slice\_num = None, increment = view\_increment, order = 3)

projection\_partial = projection\_partial[slice\_start - slice\_prior : slice\_end - slice\_prior,...]

# generate backprojection

recon\_partial = ct.filtered\_backporjection(projection\_partial,angles,projector,fbp\_projector, geometry, back\_to\_original\_value=True)

recon[slice\_coverage \* (rot\_n) : slice\_coverage \* (rot\_n + 1),...] = recon\_partial

# save fp

# projection\_save\_version = nb.Nifti1Image(projection[:,:,0,:], img\_affine)

# nb.save(projection\_save\_version, os.path.join(random\_folder,'projection.nii.gz'))

parameter\_file = os.path.join(random\_folder,'motion\_parameters.npy')

np.save(parameter\_file, np.array([[amplitude\_collect],[sga\_list], [t], [total\_view], [gantry\_rotation\_time]], dtype=object))

# generate backprojection

# recon = ct.filtered\_backporjection(projection,angles,projector,fbp\_projector, geometry, back\_to\_original\_value=True)

# save recon

recon\_nb\_image = np.rollaxis(recon,0,3)

print('under 1mm, recon shape: ', recon\_nb\_image.shape)

nb.save(nb.Nifti1Image(recon\_nb\_image,img\_affine), os.path.join(random\_folder,'image\_data','recon.nii.gz'))

# resample recon

recon\_1mm = nb.load(os.path.join(random\_folder,'image\_data','recon.nii.gz'))

new\_dim = [1,1,2.5]

recon\_resample = ff.resample\_nifti(recon\_1mm, order=3, mode = 'nearest', cval = np.min(recon\_1mm.get\_fdata()), in\_plane\_resolution\_mm=new\_dim[0], slice\_thickness\_mm=new\_dim[-1])

recon\_resample = nb.Nifti1Image(recon\_resample.get\_fdata(), affine=recon\_resample.affine, header=recon\_resample.header)

print('after 1mm, recon shape: ', recon\_resample.get\_fdata().shape)

nb.save(recon\_resample, os.path.join(random\_folder,'image\_data','recon\_resample.nii.gz'))

# # # monitor the process

# # max\_value = np.max(recon\_nb\_image)

# # min\_value = np.min(recon\_nb\_image)

# # print('max: ', max\_value, 'min: ', min\_value)

# # # check\_list.append([patient\_id, patient\_subid,random\_i, max\_value, min\_value, with\_double\_skull])

# # # df = pd.DataFrame(check\_list, columns = ['patient\_id', 'patient\_subid','random' ,'max', 'min', 'with\_double\_skull'])

# # # df.to\_excel(os.path.join(main\_save\_folder,'check\_list\_static.xlsx'))

**Portable CT motion simulation:**

Total\_view = 1400

Gantry\_rotation\_time = 500

View\_increment = 56

z-coverage = 1cm, so 10 / 0.625 = 16 slices as one stack in one gantry rotation.

In each rotation, use the current 16 slices as well as 32 slices before and 32 slices after to run the simulation.

Total rotation number = total slice // 16

Amplitude\_max\_severe = 10, displacement\_max\_severe = 6.

Amplitude\_max\_mild = 5, displacement\_max\_mild = 3.

Severe motion: translation using severe amplitude and displacement

Mild motion: translation using mild

Rotation always use mild in the bottom head, but changes to larger values in mid/top head when severe

Remove the “offset”, which is the motion happening between two rotations. The reason is that doing so will diminish the 1-to-1 correspondence of motion-free image in the simulation vs. static ground truth (as the object changes the pose), making the model training harder.

The frequency of motion happening in one gantry rotation: Motion\_freq = 0.3

The frequency of the occurance of double skull: Double\_skull\_freq = 0.3

If this rotation has motion, the frequency of having severe motion = 0.6.

If this rotation has double skull artifacts, the amplitude in ty will be larger while in tx and tz will be smaller.

If there is one motion in this rotation, then just copy the slices from the static reference.

New code:

cg = Defaults.Parameters()

motion\_cases = [['MO101701M000006', 'MO001A000073'],['MO101701M000006', 'MO001A000073\_2'],

['MO101701M000006', 'MO001A000128'] ,['MO101701M000010', 'MO001A000123'],

['MO101701M000021','MO001A000048'], ['MO101701M000014','MO001A000047'],['MO101701M000014','MO001A000112']

]

# define motion range:

motion\_type = 'simulated\_data\_portable\_CT\_new'

amplitude\_max\_severe = 10

displacement\_max\_severe = 6

amplitude\_max\_mild = 5

displacement\_max\_mild = 3

# offset\_max = 5

motion\_freq = 0.3

severe\_freq = 0.6

double\_skull\_freq = 0.3

change\_direction\_limit = 2

CP\_num = 5

geometry = 'fan'

total\_view = 1400 ### 2340 views by default

gantry\_rotation\_time = 500 #unit ms, 500ms by default

view\_increment = 56 # increment in gantry views

# define the patient list

data\_folder = os.path.join(cg.data\_dir,'raw\_data/nii-images/thin\_slice')

main\_save\_folder = os.path.join(cg.data\_dir,motion\_type)

folder\_name = 'random\_'

patient\_list= ff.find\_all\_target\_files(['\*/\*'],data\_folder)[0:1]

print(len(patient\_list))

check\_list = []

for p in patient\_list:

patient\_subid = os.path.basename(p)

patient\_id = os.path.basename(os.path.dirname(p))

if [patient\_id, patient\_subid] in motion\_cases:

print('already have motion. only recon original image'); L = [0]

continue

print('patient: ',patient\_id, patient\_subid)

save\_folder = os.path.join(main\_save\_folder,patient\_id,patient\_subid)

ff.make\_folder([os.path.dirname(save\_folder),save\_folder])

img\_file = ff.find\_all\_target\_files(['img-nii-0.625/img.nii.gz'],p)

if len(img\_file) != 1:

ValueError('no raw data')

img,spacing,img\_affine = ct.basic\_image\_processing(img\_file[0])

print('nib image shape: ',img.shape, ' spacing: ',spacing)

# define projectors

img = img[np.newaxis, ...]

projector = ct.define\_forward\_projector(img,spacing,total\_view)

fbp\_projector = ct.backprojector(img,spacing)

# very important - make sure that the arrays are saved in C order

cp.cuda.Device(0).use()

ct\_projector.set\_device(0)

# load the static image as reference

if os.path.isfile(os.path.join(cg.data\_dir, 'simulated\_data\_portable\_CT', patient\_id, patient\_subid, 'static','image\_data','recon.nii.gz')) == 0:

static\_ref = None

else:

static\_ref= nb.load(os.path.join(cg.data\_dir, 'simulated\_data\_portable\_CT', patient\_id, patient\_subid, 'static','image\_data','recon.nii.gz')).get\_fdata()

static\_ref = np.rollaxis(static\_ref,2,0)

print('static ref shape: ', static\_ref.shape)

print('static\_ref is None? ', static\_ref is None)

# do simulation

L = np.arange(1,6)

for random\_i in L:

t = np.linspace(0, gantry\_rotation\_time, CP\_num, endpoint=True)

# create folder

if random\_i == 0:

random\_folder = os.path.join(save\_folder,'static')

else:

random\_folder = os.path.join(save\_folder,folder\_name +str(random\_i))

ff.make\_folder([random\_folder, os.path.join(random\_folder,'image\_data')])

print('\n',random\_i , 'random')

if os.path.isfile(os.path.join(random\_folder,'image\_data','recon.nii.gz')) == 1:

print('already done this motion')

continue

# gantry coverage = 1cm, which means it rotates once for every 1cm interval in the z-axis

# first set the starting gantry angle (SGA)

sga = int(np.random.uniform(0,90))

# second, find out how many rotations we need

rotation\_num = int(spacing[0] \* img.shape[1] // 10)

slice\_coverage = int(10 // 0.625) # slice coverage = 1cm

# start to generate the motion for each rotation

# set whether static or motion for each rotation

while True:

motion\_status = [np.random.uniform(0,1) <motion\_freq for k in range(rotation\_num-3)]

if motion\_freq <=0:

break

if np.sum(motion\_status) > 0:

break

motion\_status = [False, False] + motion\_status + [False] # add two static rotations at the beginning and end

print('motion\_status: ', motion\_status)

amplitude\_collect = np.zeros([rotation\_num, CP\_num, 6])

recon = np.zeros([slice\_coverage \* rotation\_num,img.shape[2],img.shape[3]])

# projection = np.zeros([slice\_coverage \* rotation\_num, total\_view, 1 , projector.nu])

for rot\_n in range(0,rotation\_num):

# print('rotation: ', rot\_n)

# set the partial image:

slice\_start = rot\_n \* slice\_coverage

slice\_end = (rot\_n + 1) \* slice\_coverage

slice\_prior = max(0, slice\_start - slice\_coverage \* 2)

slice\_post = min(img.shape[1], slice\_end + slice\_coverage \* 2)

img\_partial = img[:,slice\_prior:slice\_post,:]

# set the motion

# first: determine whether in this rotation we have motion or static, and whether it's severe or mild

if motion\_status[rot\_n]:

amplitude\_max\_tem\_r = amplitude\_max\_mild # rotation always max = 5 degree

displacement\_max\_tem\_r = displacement\_max\_mild

if np.random.uniform(0,1) < severe\_freq: # severe motion

severe\_motion = True

amplitude\_max\_tem\_t = amplitude\_max\_severe

displacement\_max\_tem\_t = displacement\_max\_severe

else: # mild motion

severe\_motion = False

amplitude\_max\_tem\_t = amplitude\_max\_mild

displacement\_max\_tem\_t = displacement\_max\_mild

else:

severe\_motion = False

amplitude\_max\_tem\_t, displacement\_max\_tem\_t, amplitude\_max\_tem\_r, displacement\_max\_tem\_r = 0, 0, 0, 0

# second: determine whether the initial pose has been changed compared to last rotation

offset\_values = [0] \* 6

# third: generate the motion

# translation

amplitude\_txty\_mm = transform.motion\_control\_point\_generation(2, CP\_num, amplitude\_max = amplitude\_max\_tem\_t, displacement\_max = displacement\_max\_tem\_t, change\_direction\_limit = change\_direction\_limit, offset\_value = offset\_values[0:2], print\_result =False)

amplitude\_tx\_mm = amplitude\_txty\_mm[:,0]

amplitude\_ty\_mm = amplitude\_txty\_mm[:,1]

if severe\_motion:

amplitude\_tz\_mm = transform.motion\_control\_point\_generation(1, CP\_num, amplitude\_max = 3, displacement\_max = 1.5, change\_direction\_limit = change\_direction\_limit, offset\_value = offset\_values[2], print\_result =False)[:,0]

elif not severe\_motion and motion\_status[rot\_n]:

amplitude\_tz\_mm = transform.motion\_control\_point\_generation(1, CP\_num, amplitude\_max = 2, displacement\_max = 1, change\_direction\_limit = change\_direction\_limit, offset\_value = offset\_values[2], print\_result =False)[:,0]

else:

amplitude\_tz\_mm = transform.motion\_control\_point\_generation(1, CP\_num, amplitude\_max = 0, displacement\_max = 0, change\_direction\_limit = change\_direction\_limit, offset\_value = offset\_values[2], print\_result =False)[:,0]

# rotations

while True:

amplitude\_rxryrz\_degree = transform.motion\_control\_point\_generation(3, CP\_num, amplitude\_max = amplitude\_max\_tem\_r, displacement\_max = displacement\_max\_tem\_r, change\_direction\_limit = change\_direction\_limit, offset\_value = offset\_values[3:6], print\_result =False)

amplitude\_rx\_degree = amplitude\_rxryrz\_degree[:,0]

amplitude\_ry\_degree = amplitude\_rxryrz\_degree[:,1]

amplitude\_rz\_degree = amplitude\_rxryrz\_degree[:,2]

if np.max(abs(amplitude\_rx\_degree - amplitude\_rx\_degree[0]))+ np.max(abs(amplitude\_ry\_degree - amplitude\_ry\_degree[0])) <= 6:

break

# let's also consider the double skull artifacts, eespeically in the occipital bone

if np.random.uniform(0,1) < double\_skull\_freq and motion\_status[rot\_n]:

print('yes we have double skull')

amplitude\_ty\_mm = transform.motion\_control\_point\_generation(1, CP\_num, amplitude\_max = 12, displacement\_max = 8, change\_direction\_limit = change\_direction\_limit, offset\_value = offset\_values[1], print\_result =False)[:,0]

amplitude\_tx\_mm = transform.motion\_control\_point\_generation(1, CP\_num, amplitude\_max = 5, displacement\_max = 3, change\_direction\_limit = change\_direction\_limit, offset\_value = offset\_values[0], print\_result =False)[:,0]

amplitude\_tz\_mm = transform.motion\_control\_point\_generation(1, CP\_num, amplitude\_max = 1, displacement\_max = 0.5, change\_direction\_limit = change\_direction\_limit, offset\_value = offset\_values[2], print\_result =False)[:,0]

# print('amplitude\_tx\_mm: ', amplitude\_tx\_mm)

# print('amplitude\_ty\_mm: ', amplitude\_ty\_mm)

# print('amplitude\_tz\_mm: ', amplitude\_tz\_mm)

# print('amplitude\_rx\_degree: ', amplitude\_rx\_degree)

# print('amplitude\_ry\_degree: ', amplitude\_ry\_degree)

# print('amplitude\_rz\_degree: ', amplitude\_rz\_degree)

# save the motion parameters

parameter\_file = os.path.join(random\_folder,'motion\_parameters.txt')

if rot\_n == 0:

ff.txt\_writer(parameter\_file, True, [t.tolist(),amplitude\_tx\_mm, amplitude\_ty\_mm, amplitude\_tz\_mm, amplitude\_rx\_degree, amplitude\_ry\_degree, amplitude\_rz\_degree, [sga],[total\_view],[gantry\_rotation\_time]],['time\_points','translation\_x\_CP','translation\_y\_CP','translation\_z\_CP', 'rotation\_x\_CP', 'rotation\_y\_CP','rotation\_z\_CP','starting\_gantry\_angle', 'total\_projection\_view','gantry\_rotation\_time(ms)'])

else:

ff.txt\_writer(parameter\_file, False, [t.tolist(),amplitude\_tx\_mm, amplitude\_ty\_mm, amplitude\_tz\_mm, amplitude\_rx\_degree, amplitude\_ry\_degree, amplitude\_rz\_degree, [sga],[total\_view],[gantry\_rotation\_time]],['time\_points','translation\_x\_CP','translation\_y\_CP','translation\_z\_CP', 'rotation\_x\_CP', 'rotation\_y\_CP','rotation\_z\_CP','starting\_gantry\_angle', 'total\_projection\_view','gantry\_rotation\_time(ms)'])

collect = np.stack([amplitude\_tx\_mm, amplitude\_ty\_mm, amplitude\_tz\_mm, amplitude\_rx\_degree, amplitude\_ry\_degree, amplitude\_rz\_degree], axis = 1)

amplitude\_collect[rot\_n,...] = collect

# prepare spline fit

spline\_tx = transform.interp\_func(t, np.asarray([i/spacing[1] for i in amplitude\_tx\_mm]))

spline\_ty = transform.interp\_func(t, np.asarray([i/spacing[2] for i in amplitude\_ty\_mm]))

spline\_tz = transform.interp\_func(t, np.asarray([i/spacing[0] for i in amplitude\_tz\_mm]))

spline\_rx = transform.interp\_func(t,np.asarray([i / 180 \* np.pi for i in amplitude\_rx\_degree]))

spline\_ry = transform.interp\_func(t,np.asarray([i / 180 \* np.pi for i in amplitude\_ry\_degree]))

spline\_rz = transform.interp\_func(t,np.asarray([i / 180 \* np.pi for i in amplitude\_rz\_degree]))

angles = ff.get\_angles\_zc(total\_view, 360 ,sga)

# if static and has the static image ref, then no need to do the projection

if motion\_status[rot\_n] == False:

if static\_ref is not None:

recon[slice\_coverage \* (rot\_n) : slice\_coverage \* (rot\_n + 1),...] = static\_ref[slice\_coverage\*rot\_n:slice\_coverage\*(rot\_n+1),...]

continue

# generate forward projection

projection\_partial = ct.fp\_w\_spline\_motion\_model(img\_partial, projector, angles, spline\_tx, spline\_ty, spline\_tz, spline\_rx, spline\_ry, spline\_rz, geometry, total\_view = total\_view, gantry\_rotation\_time = gantry\_rotation\_time, slice\_num = None, increment = view\_increment, order = 3)

projection\_partial = projection\_partial[slice\_start - slice\_prior : slice\_end - slice\_prior,...]

# generate backprojection

recon\_partial = ct.filtered\_backporjection(projection\_partial,angles,projector,fbp\_projector, geometry, back\_to\_original\_value=True)

recon[slice\_coverage \* (rot\_n) : slice\_coverage \* (rot\_n + 1),...] = recon\_partial

# save fp

# projection\_save\_version = nb.Nifti1Image(projection[:,:,0,:], img\_affine)

# nb.save(projection\_save\_version, os.path.join(random\_folder,'projection.nii.gz'))

parameter\_file = os.path.join(random\_folder,'motion\_parameters.npy')

np.save(parameter\_file, np.array([[amplitude\_collect],[sga], [t], [total\_view], [gantry\_rotation\_time]], dtype=object))

# generate backprojection

# recon = ct.filtered\_backporjection(projection,angles,projector,fbp\_projector, geometry, back\_to\_original\_value=True)

# save recon

recon\_nb\_image = np.rollaxis(recon,0,3)

print(recon\_nb\_image.shape)

nb.save(nb.Nifti1Image(recon\_nb\_image,img\_affine), os.path.join(random\_folder,'image\_data','recon.nii.gz'))

# # monitor the process

# max\_value = np.max(recon\_nb\_image)

# min\_value = np.min(recon\_nb\_image)

# print('max: ', max\_value, 'min: ', min\_value)

# # check\_list.append([patient\_id, patient\_subid,random\_i, max\_value, min\_value, with\_double\_skull])

# # df = pd.DataFrame(check\_list, columns = ['patient\_id', 'patient\_subid','random' ,'max', 'min', 'with\_double\_skull'])

# # df.to\_excel(os.path.join(main\_save\_folder,'check\_list\_static.xlsx'))

Previous code:

if rot\_n == 0:

offset\_values = [0] \* 6

else:

if np.random.uniform(0,1) < offset\_freq:

offset\_values = [np.random.uniform(-offset\_max,offset\_max), np.random.uniform(0), np.random.uniform(0), np.random.uniform(-3,3), np.random.uniform(-3,3), np.random.uniform(-3,3)]

previous = [amplitude\_tx\_mm[-1], amplitude\_ty\_mm[-1], amplitude\_tz\_mm[-1], amplitude\_rx\_degree[-1], amplitude\_ry\_degree[-1], amplitude\_rz\_degree[-1]]

offset\_values = [previous[m] + offset\_values[m] for m in range(6)]

else:

offset\_values = [amplitude\_tx\_mm[-1], amplitude\_ty\_mm[-1], amplitude\_tz\_mm[-1], amplitude\_rx\_degree[-1], amplitude\_ry\_degree[-1], amplitude\_rz\_degree[-1]]

# third: generate the motion

# tx and ty, in-plane translation

amplitude\_txty\_mm = transform.motion\_control\_point\_generation(2, CP\_num, amplitude\_max = amplitude\_max\_tem, displacement\_max = displacement\_max\_tem, change\_direction\_limit = change\_direction\_limit, offset\_value = offset\_values[0:2], print\_result =False)

amplitude\_tx\_mm = amplitude\_txty\_mm[:,0]

amplitude\_ty\_mm = amplitude\_txty\_mm[:,1]

# tz: through-plane translation

amplitude\_tz\_mm = transform.motion\_control\_point\_generation(1, CP\_num, amplitude\_max = amplitude\_max\_tem, displacement\_max = displacement\_max\_tem, change\_direction\_limit = change\_direction\_limit, offset\_value = offset\_values[2], print\_result =False)[:,0]

if severe\_motion:

amplitude\_tz\_mm = transform.motion\_control\_point\_generation(1, CP\_num, amplitude\_max = 5, displacement\_max = 3, change\_direction\_limit = change\_direction\_limit, offset\_value = offset\_values[2], print\_result =False)[:,0]

elif not severe\_motion and motion\_status[rot\_n]:

amplitude\_tz\_mm = transform.motion\_control\_point\_generation(1, CP\_num, amplitude\_max = 3, displacement\_max = 2, change\_direction\_limit = change\_direction\_limit, offset\_value = offset\_values[2], print\_result =False)[:,0]

# rotations

while True:

amplitude\_rxryrz\_degree = transform.motion\_control\_point\_generation(3, CP\_num, amplitude\_max = amplitude\_max\_tem, displacement\_max = displacement\_max\_tem, change\_direction\_limit = change\_direction\_limit, offset\_value = offset\_values[3:6], print\_result =False)

amplitude\_rx\_degree = amplitude\_rxryrz\_degree[:,0]

amplitude\_ry\_degree = amplitude\_rxryrz\_degree[:,1]

amplitude\_rz\_degree = amplitude\_rxryrz\_degree[:,2]

if np.max(abs(amplitude\_rx\_degree - amplitude\_rx\_degree[0]))+ np.max(abs(amplitude\_ry\_degree - amplitude\_ry\_degree[0])) <= 7:

break

# let's also consider the double skull artifacts, eespeically in the occipital bone

if np.random.uniform(0,1) < double\_skull\_freq and motion\_status[rot\_n]:

print('yes we have double skull')

if rot\_n <= rotation\_num / 3 \* 2: # not top layer

amplitude\_ty\_mm = transform.motion\_control\_point\_generation(1, CP\_num, amplitude\_max = 12, displacement\_max = 8, change\_direction\_limit = change\_direction\_limit, offset\_value = offset\_values[1], print\_result =False)[:,0]

else:

amplitude\_ty\_mm = transform.motion\_control\_point\_generation(1, CP\_num, amplitude\_max = 15, displacement\_max = 10, change\_direction\_limit = change\_direction\_limit, offset\_value = offset\_values[1], print\_result =False)[:,0]