Aligning robot components code:

# Transformations regarding the STL files for operational use in Reinforcement Learning algorithm

import numpy as np

# array containing all indivdual left handed transformation matricies

idv\_lh\_trans\_matrices = []

# array containing all individual right handed transformation matricies

idv\_rh\_trans\_matrices = []

# array containing all left handed transformation matrices

lh\_trans\_matrices = []

# array containing all right handed transformation matrices

rh\_trans\_matrices = []

# array conatining all left handed robot component meshes

lh\_meshes = []

# array containing all right handed robot component meshes

rh\_meshes = []

# Class for transformations matricies for STL file realignment

class TransformationMatrix():

# constructor initalising variables and calling matrix function

def \_\_init\_\_(self,alpha,theta,a,d):

# conversion to radians from degrees for alpha and theta angles

self.alpha = np.radians(alpha)

self.theta = np.radians(theta)

self.a = a

self.d = d

self.Transformation\_matrix = self.matrix\_generation()

# function for generating transformation matrix

def matrix\_generation(self):

matrix = np.array([

[np.cos(self.theta), -np.sin(self.theta) \* np.cos(self.alpha), np.sin(self.alpha) \* np.sin(self.theta), self.a \* np.cos(self.alpha)],

[np.sin(self.theta), np.cos(self.alpha) \* np.cos(self.theta), -np.sin(self.alpha) \* np.cos(self.theta), self.a \* np.sin(self.theta)],

[0.0, np.sin(self.alpha), np.cos(self.alpha), self.d],

[0.0, 0.0, 0.0, 1.0]

])

return matrix

# x = alpha. alpha: x-axis rotation. row 1 is left arm row 2 is right arm

x = [[90.0, 90.0, 0.0, 90.0, 0.0, 0.0, 90.0, 90.0, -90.0, -90.0, 90.0],

[90.0, 90.0, 0.0, 90.0, 0.0, 0.0, 90.0, 90.0, -90.0, -90.0, 90.0]]

# y = theta. theta: z-axis rotation (inital joint positions were used for y). row 1 is left arm row 2 is right arm

y = [[0.0, 90.0, 13.33, -13.33, 15.0, -65.0, -28.0, 90.0, 0.0, -90.0, -17.0],

[0.0, 90.0, 13.33, -13.33, -15.0, 65.0, 28.0, 90.0, 0.0, -90.0, 17.0]]

# a: x axis displacement

a = [[0.0, 0.0, 155.0, 75.0, 180.0, 100.0, 0.0, 0.0, 35.0, 55.1, 0.0],

[0.0, 0.0, 155.0, 75.0, 180.0, 100.0, 0.0, 0.0, 35.0, 55.1, 0.0]]

# d: y axis displacement

d = [[0.0, 400.0, 116.0, 0.0, -24.0, 24.0, 86.0, 0.0, 150.5, 0.0, 58.65],

[0.0, 400.0, -166.0, 0.0, -24.0, 24.0, 86.0, 0.0 , 150.5, 0.0, 58.65]]

for i in range(len(x[0])):

t\_1 = TransformationMatrix(x[0][i],y[0][i],a[0][i],d[0][i])

t\_2 = TransformationMatrix(x[1][i],y[1][i],a[1][i],d[1][i])

idv\_lh\_trans\_matrices.append(t\_1)

idv\_rh\_trans\_matrices.append(t\_2)

n = range(len(idv\_lh\_trans\_matrices))

# defining a recusive function to multiply all individual matrices for full alignment to 1st component

n = len(idv\_lh\_trans\_matrices) - 1

def transformation\_lh\_matrix\_gen(idv\_lh\_trans\_matrices,n,lh\_trans\_matrices):

if (n == 0):

lh\_trans\_matrices.append(idv\_lh\_trans\_matrices[n].Transformation\_matrix)

return idv\_lh\_trans\_matrices[n].Transformation\_matrix

else:

previous\_matrix = transformation\_lh\_matrix\_gen(idv\_lh\_trans\_matrices,n-1,lh\_trans\_matrices)

current\_matrix = np.dot(previous\_matrix,idv\_lh\_trans\_matrices[n].Transformation\_matrix)

lh\_trans\_matrices.append(current\_matrix)

return current\_matrix

def transformation\_rh\_matrix\_gen(idv\_lh\_trans\_matrices,n,rh\_trans\_matrices):

if (n == 0):

rh\_trans\_matrices.append(idv\_lh\_trans\_matrices[n].Transformation\_matrix)

return idv\_rh\_trans\_matrices[n].Transformation\_matrix

else:

previous\_matrix = transformation\_rh\_matrix\_gen(idv\_lh\_trans\_matrices,n-1,rh\_trans\_matrices)

current\_matrix = np.dot(previous\_matrix,idv\_rh\_trans\_matrices[n].Transformation\_matrix)

rh\_trans\_matrices.append(current\_matrix)

return current\_matrix

# Clear the lists before running the recursive functions

lh\_trans\_matrices.clear()

rh\_trans\_matrices.clear()

# Call recursive function for generating left transformation matrix

j = transformation\_lh\_matrix\_gen(idv\_lh\_trans\_matrices,n,lh\_trans\_matrices)

# Call recursive function for generating right trnasformation matrix

k = transformation\_rh\_matrix\_gen(idv\_rh\_trans\_matrices,n,rh\_trans\_matrices)

# deleting the matrix elements not needed for transformation

del lh\_trans\_matrices[0]

del lh\_trans\_matrices[9]

del rh\_trans\_matrices[0]

del rh\_trans\_matrices[9]

# adjustment matrix definition

lh\_adj\_matrix = [[-1.0,0.0,0.0,-100.0],[0.0,0.0,-1.0,0.0],[0.0,-1.0,0.0,0.0],[0.0,0.0,0.0,1.0]]

rh\_adj\_matrix = [[-1.0,0.0,0.0,187.0],[0.0,0.0,-1.0,0.0],[0.0,-1.0,0.0,0.0],[0.0,0.0,0.0,1.0]]

# premultiplying adjustment matrices

for i in range(len(lh\_trans\_matrices)):

lh\_trans\_matrices[i] = np.dot(lh\_adj\_matrix,lh\_trans\_matrices[i])

for i in range(len(rh\_trans\_matrices)):

rh\_trans\_matrices[i] = np.dot(rh\_adj\_matrix,rh\_trans\_matrices[i])

import os

from stl import mesh

from mpl\_toolkits import mplot3d

from matplotlib import pyplot

import numpy as np

folder\_path = r"C:\Users\whows\OneDrive - King's College London\Documents\BEng project\STL"

stl\_file\_names = ["Link02.stl","Link03.stl","Link04.stl","Link05.stl","Link06.stl","Link07.stl","Link08.stl","Link09.stl","Link10.stl",

"Link13.stl","Link14.stl","Link15.stl","Link16.stl","Link17.stl","Link18.stl","Link19.stl",

"Link20.stl","Link21.stl"]

stl\_file\_paths = [os.path.join(folder\_path, file\_name) for file\_name in stl\_file\_names]

# function for loading stl files

def load\_stl\_files(file\_paths):

meshes = [] # List to hold the mesh objects

for file\_path in file\_paths:

stl\_mesh = mesh.Mesh.from\_file(file\_path) # Loading the STL file

meshes.append(stl\_mesh) # Adding the mesh to the list

return meshes # Returning the list of meshes

# transforming meshes

def transformed\_meshes(lh\_meshes,rh\_meshes,lh\_trans\_matrices,rh\_trans\_matrices):

for i in range(len(lh\_meshes)):

lh\_meshes[i].transform(lh\_trans\_matrices[i])

rh\_meshes[i].transform(rh\_trans\_matrices[i])

# loading in the stl files

meshes = load\_stl\_files(stl\_file\_paths)

# Clear the lists before running the recursive functions

lh\_meshes.clear()

rh\_meshes.clear()

# seperating left and right hands

for i in range(len(meshes)):

if (i <= 8):

lh\_meshes.append(meshes[i])

else:

rh\_meshes.append(meshes[i])

# testing lengths of meshes generated

print("Length of lh\_trans\_matrices",len(lh\_trans\_matrices))

print("Length of rh\_trans\_matrices",len(rh\_trans\_matrices))

print("Length of meshes",len(meshes))

print("Length of lh\_meshes",len(lh\_meshes))

print("Length of rh\_meshes",len(rh\_meshes))

# transforming meshes for realginment

transformed\_meshes(lh\_meshes,rh\_meshes,lh\_trans\_matrices,rh\_trans\_matrices)

# Create a new plot

figure = pyplot.figure()

axes = figure.add\_subplot(projection='3d')

# Iterate through each left hand mesh and add it to the axes

for mesh in lh\_meshes:

axes.add\_collection3d(mplot3d.art3d.Poly3DCollection(mesh.vectors))

# iterate through each right hand mesh and add it to the axes

for mesh in rh\_meshes:

axes.add\_collection3d(mplot3d.art3d.Poly3DCollection(mesh.vectors))

# Auto scale to the combined mesh sizes

scale = np.concatenate([mesh.points.flatten() for mesh in lh\_meshes + rh\_meshes])

axes.auto\_scale\_xyz(scale, scale, scale)

# Show the plot to the screen

pyplot.show()