**1.colab**

스크린샷, 앉아있는, 모니터, 화면이(가) 표시된 사진

자동 생성된 설명

스크린샷이(가) 표시된 사진

자동 생성된 설명

**2. pcl을 사용한 유클라디안 클러스터링(개선중)**

import pcl

#!/usr/bin/env python

# Copyright (C) 2017 Udacity Inc.

#

# This file is part of Robotic Arm: Pick and Place project for Udacity

# Robotics nano-degree program

#

# All Rights Reserved.

# Author: Harsh Pandya

# Import modules

import pcl

import numpy as np

import ctypes

import struct

from random import randint

def random\_color\_gen():

""" Generates a random color

Args: None

Returns:

list: 3 elements, R, G, and B

"""

r = randint(0, 255)

g = randint(0, 255)

b = randint(0, 255)

return [r, g, b]

def XYZRGB\_to\_XYZ(XYZRGB\_cloud):

""" Converts a PCL XYZRGB point cloud to an XYZ point cloud (removes color info)

Args:

XYZRGB\_cloud (PointCloud\_PointXYZRGB): A PCL XYZRGB point cloud

Returns:

PointCloud\_PointXYZ: A PCL XYZ point cloud

"""

XYZ\_cloud = pcl.PointCloud()

points\_list = []

for data in XYZRGB\_cloud:

points\_list.append([data[0], data[1], data[2]])

XYZ\_cloud.from\_list(points\_list)

return XYZ\_cloud

def XYZ\_to\_XYZRGB(XYZ\_cloud, color):

""" Converts a PCL XYZ point cloud to a PCL XYZRGB point cloud

All returned points in the XYZRGB cloud will be the color indicated

by the color parameter.

Args:

XYZ\_cloud (PointCloud\_XYZ): A PCL XYZ point cloud

color (list): 3-element list of integers [0-255,0-255,0-255]

Returns:

PointCloud\_PointXYZRGB: A PCL XYZRGB point cloud

"""

XYZRGB\_cloud = pcl.PointCloud\_PointXYZRGB()

points\_list = []

float\_rgb = rgb\_to\_float(color)

for data in XYZ\_cloud:

points\_list.append([data[0], data[1], data[2], float\_rgb])

XYZRGB\_cloud.from\_list(points\_list)

return XYZRGB\_cloud

def rgb\_to\_float(color):

""" Converts an RGB list to the packed float format used by PCL

From the PCL docs:

"Due to historical reasons (PCL was first developed as a ROS package),

the RGB information is packed into an integer and casted to a float"

Args:

color (list): 3-element list of integers [0-255,0-255,0-255]

Returns:

float\_rgb: RGB value packed as a float

"""

hex\_r = (0xff & color[0]) << 16

hex\_g = (0xff & color[1]) << 8

hex\_b = (0xff & color[2])

hex\_rgb = hex\_r | hex\_g | hex\_b

float\_rgb = struct.unpack('f', struct.pack('i', hex\_rgb))[0]

return float\_rgb

def float\_to\_rgb(float\_rgb):

""" Converts a packed float RGB format to an RGB list

Args:

float\_rgb: RGB value packed as a float

Returns:

color (list): 3-element list of integers [0-255,0-255,0-255]

"""

s = struct.pack('>f', float\_rgb)

i = struct.unpack('>l', s)[0]

pack = ctypes.c\_uint32(i).value

r = (pack & 0x00FF0000) >> 16

g = (pack & 0x0000FF00) >> 8

b = (pack & 0x000000FF)

color = [r,g,b]

return color

def get\_color\_list(cluster\_count):

""" Returns a list of randomized colors

Args:

cluster\_count (int): Number of random colors to generate

Returns:

(list): List containing 3-element color lists

"""

if (cluster\_count > len(get\_color\_list.color\_list)):

for i in xrange(len(get\_color\_list.color\_list), cluster\_count):

get\_color\_list.color\_list.append(random\_color\_gen())

return get\_color\_list.color\_list

def do\_euclidean\_clustering(white\_cloud):

'''

:param cloud\_objects:

:return: cluster cloud and cluster indices

'''

tree = white\_cloud.make\_kdtree()

# Create Cluster-Mask Point Cloud to visualize each cluster separately

ec = white\_cloud.make\_EuclideanClusterExtraction()

ec.set\_ClusterTolerance(0.01)

ec.set\_MinClusterSize(20)

ec.set\_MaxClusterSize(40)

ec.set\_SearchMethod(tree)

cluster\_indices = ec.Extract()

cluster\_color = get\_color\_list(len(cluster\_indices))

color\_cluster\_point\_list = []

for j, indices in enumerate(cluster\_indices):

for i, indice in enumerate(indices):

color\_cluster\_point\_list.append([white\_cloud[indice][0],

white\_cloud[indice][1],

white\_cloud[indice][2],

rgb\_to\_float(cluster\_color[j])])

cluster\_cloud = pcl.PointCloud\_PointXYZRGB()

cluster\_cloud.from\_list(color\_cluster\_point\_list)

return cluster\_cloud,cluster\_indices

# Euclidean Clustering

get\_color\_list.color\_list = []

white\_cloud = pcl.load("/content/TEST3\_.pcd")

print(white\_cloud)

cluster\_cloud,cluster\_indices = do\_euclidean\_clustering(white\_cloud)

print(cluster\_cloud)

print(cluster\_indices)

**3.DBSCAN(작성중)**

import pandas as pd

from sklearn.cluster import DBSCAN

import matplotlib.pyplot as plt

import seaborn as sns

from mpl\_toolkits.mplot3d import Axes3D

import pcl

pc = pcl.load("지면제거data2.pcd") # "pc.from\_file" Deprecated

pc\_array = pc.to\_array() # pc to Numpy

#cloud = pcl.load\_XYZRGBA("tabletop.pcd")

# Read Tree Sample data

data = pd.DataFrame(pc\_array)

data.columns=['X','Y','Z']

print(data)

# create model and prediction

model = DBSCAN(min\_samples=6)

predict = pd.DataFrame(model.fit\_predict(data))

predict.columns=['predict']

# concatenate labels to df as a new column

r = pd.concat([data,predict],axis=1)

print(r)

# scatter plot

fig = plt.figure(figsize=(6,6))

ax = Axes3D(fig, rect=[0, 0, .95, 1], elev=48, azim=134)

ax.scatter(r['X'], r['Y'], r['Z'], c=r['predict'], alpha=0.5)

ax.set\_xlabel('X')

ax.set\_ylabel('Y')

ax.set\_zlabel('Z')

plt.show()

**4. OCTREE 탐색 -> 가까운 점 설정 후 반경 내의 포인트를 탐색, 정해 준 포인트 까지 점을 탐색하고 정해준 포인트까지의 거리를 구해줌**

**Octree 구조체와 PCL 점군 검색**

PCL에서는 Octree 구조체를 이용하여서 총 3가지의 포인트 검색 방법을 제공하고 있습니다.

* octree.voxelSearch : 사용자 정의 기준점이 속해 있는 Voxel내 모든 점군을 검색 합니다.
* octree.nearestKSearch: 사용자 정의 기준점에서 가까운 N(사용자 정의)개의 점들을 검색 합니다.
* octree.radiusSearch : 사용자 정의 기준 기준점에서 반경(사용자 정의)내 점들을 검색 합니다.

결과 값은 포인트의 x,y,z좌표가 아니라 인덱스 정보입니다. 그리고 nearestKSearch와 radiusSearch는 기준점까지의 거리 정보도 같이 반환 입니다.

# Octree 구조체

먼저 Octree 구조체에 대하여 살펴 보겠습니다. Octree(팔진트리)는 컴퓨터 과학분야에서 많이 사용되는 이진트리(binary tree)자료 구조를 3D 환경에 적용하기 위하여 확장한 개념입니다. 2개의 자식 노드를 가진은 이진트리와 비교하여 8개의 자식 노드를 가진다는 특징을 빼고는 기본 동작원리는 같습니다.

정의 : 옥트리는 3D데이터 처리를 위한 트리형태의 데이터 구조체이다.

|  |  |
| --- | --- |
| ​​ | ​ |

출처: <https://pcl.gitbook.io/tutorial/part-2/part02-chapter02>