

OpenCV , Open3D

ROKEY BOOT CAMP

ROS-2 심화학습 강의자료

Contents

- OpenCV와 ROS2
- OpenCV와 ROS2 실습
- Open3D와 ROS2
- Open3D와 ROS2 실습

OpenCV와 ROS2

OpenCV란?

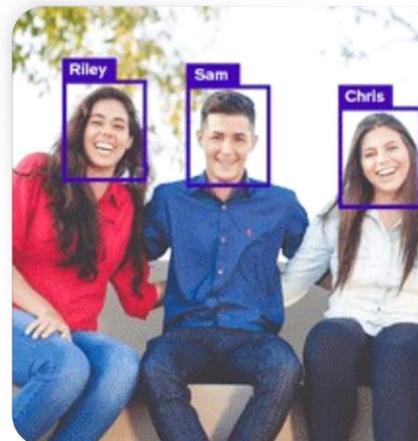
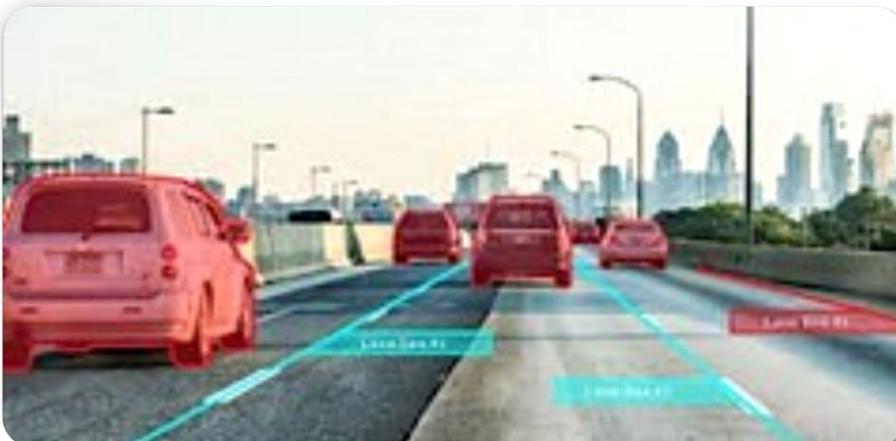
OpenCV

- **OpenCV**

- 컴퓨터 비전 작업을 위한 Python 라이브러리

- **컴퓨터비전**

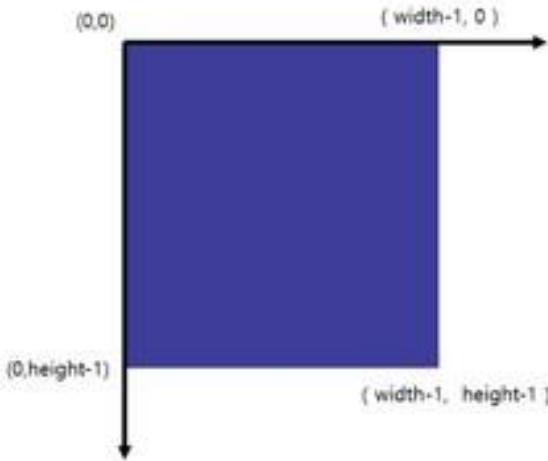
- 인간의 눈 ‘보다’와 인간의 뇌 ‘생각하다’를 처리하는 것
 - 카메라를 통해 이미지 데이터를 받아서 전처리/인식/분석 등을 수행함
 - 자율 주행, 얼굴 인식 등 다양한 분야에서 활용됨



OpenCV와 ROS2

OpenCV란?

■ OpenCV의 특징



OpenCV 좌표계



데이터 타입

이미지는 numpy 배열
형식으로 표현함

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

데이터 연산

주로 numpy 배열에 대한
행렬 연산을 수행함

OpenCV와 ROS2

실습 환경 구축

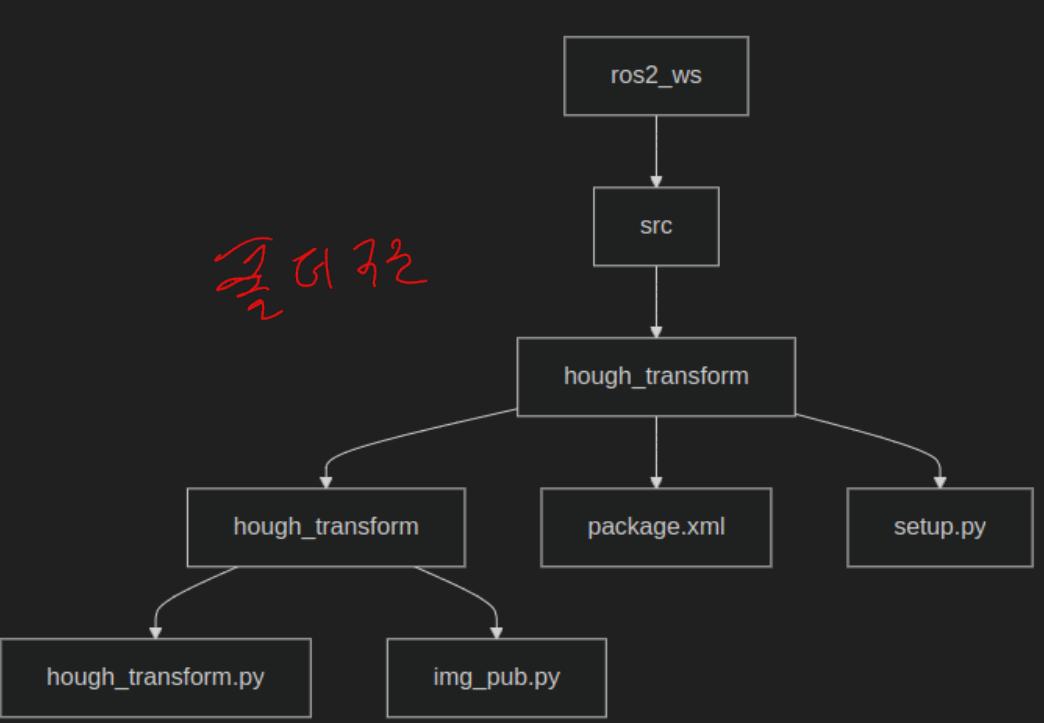
■ ros2_ws.zip 한눈에 보기

publisher와 subscriber 구조



ros2_ws의 하위 디렉토리

```
▽ ros2_ws
  > .vscode
  > build
  > img
  > install
  > log
  ▽ src/hough_transform
    > .vscode
    ▽ hough_transform
      > __pycache__
      & __init__.py
      & hough_transform.py
      & img_pub.py
    > resource
    > test
    & package.xml
    & setup.cfg
    & setup.py
```



colcon build

```
cd ros2_ws
cd src
colcon build
```

✓ 코드와 설명 : 실행은 ros2_ws에서 할 것

- `img_pub.py`

```
import rclpy
from rclpy.node import Node
from sensor_msgs.msg import Image
from cv_bridge import CvBridge
import cv2
```

- `Image`: sensor_msg 패키지에서 제공하는 이미지 데이터 타입
- `CvBridge`: ROS2의 이미지 데이터를 파이썬 사용 가능하게 변환해주는 클래스
- `cv2`: openCV 함수를 사용하기 위한 클래스

- `ImagePublisher` 클래스 정의

```
class ImagePublisher(Node):
    def __init__(self):
        super().__init__("image_publisher")
        self.declare_and_fetch_parameters()
        self.bridge = CvBridge()
        self.publisher = self.create_publisher(Image, "original_image", 10)
        self.setup_timer()
```

→ `Node` 클래스를 상속받아 ROS2 노드를 정의
각종 모듈 초기화
ROS2 퍼블리셔를 초기화하고
`self.setup_timer`로 퍼블리시 주기를 설정

✓ 코드 설명

```
def declare_and_fetch_parameters(self):
    """Declare and fetch the image path parameter."""
    self.declare_parameter("image_path", "")
    image_path_param = (
        self.get_parameter("image_path").get_parameter_value().string_value
    )
    if not image_path_param:
        self.get_logger().error(
            "No image path provided. Use '--ros-args -p image_path:=<path_to_image>'"
        )
        rclpy.shutdown()
    self.image_path = image_path_param

def setup_timer(self):
    """Set up a timer to publish images at regular intervals."""
    self.timer = self.create_timer(0.1, self.publish_image)

def publish_image(self):
    """Read an image from file and publish it."""
    cv_image = cv2.imread(self.image_path)
    if cv_image is None:
        self.get_logger().error(f"Failed to load image from {self.image_path}")
        return

    ros_image = self.bridge.cv2_to_imgmsg(cv_image, encoding="bgr8")
    self.publisher.publish(ros_image)
    self.get_logger().info(
        f"Published image: {self.image_path} with shape {cv_image.shape}"
    )
```

- **declare_and_fetch_parameters()**
 - 명령어의 인자로 전달된 이미지 경로를 저장함
 - 만일 명령어 인자가 없다면 오류를 출력하는 함수
- **setup_timer()**
 - self.create_timer(0.1, self.publish_image) 함수를 이용하여 publish_image함수가 0.1초마다 이미지를 publish하도록 설정하는 함수
- **publish_image()**
 - 이미지 파일을 불러온 후 imgmsg 형식으로 전환한 다음 publish하는 함수

✓ 코드 설명

▪ Hough_transform.py

```
import rclpy
from rclpy.node import Node
from sensor_msgs.msg import Image
from cv_bridge import CvBridge
import cv2
import numpy as np
```

- **numpy**: 파이썬에서 이미지 데이터를 처리하기 위한 행렬 관련 기능을 제공

```
class HoughTransform(Node):
    def __init__(self):
        super().__init__("hough_transform")
        self.method = (
            self.declare_parameter("method", "").get_parameter_value().string_value
        )
        self.bridge = CvBridge()
        self.subscriber = self.create_subscription(
            Image, "original_image", self.process_image, 10
        )
        self.publisher = self.create_publisher(Image, "hough_transform", 10)
        if not self.method:
            self.get_logger().error(
                "No method provided. Use '--ros-args -p method:=<method>'"
            )
        rclpy.shutdown()
```

→ Node 클래스를 상속받아 ROS2 노드를 정의
명령어의 인자로 받은 method(ex. line or circle) 저장
각종 모듈 초기화
ROS2 서브스크라이버와 퍼블리셔를 초기화 함

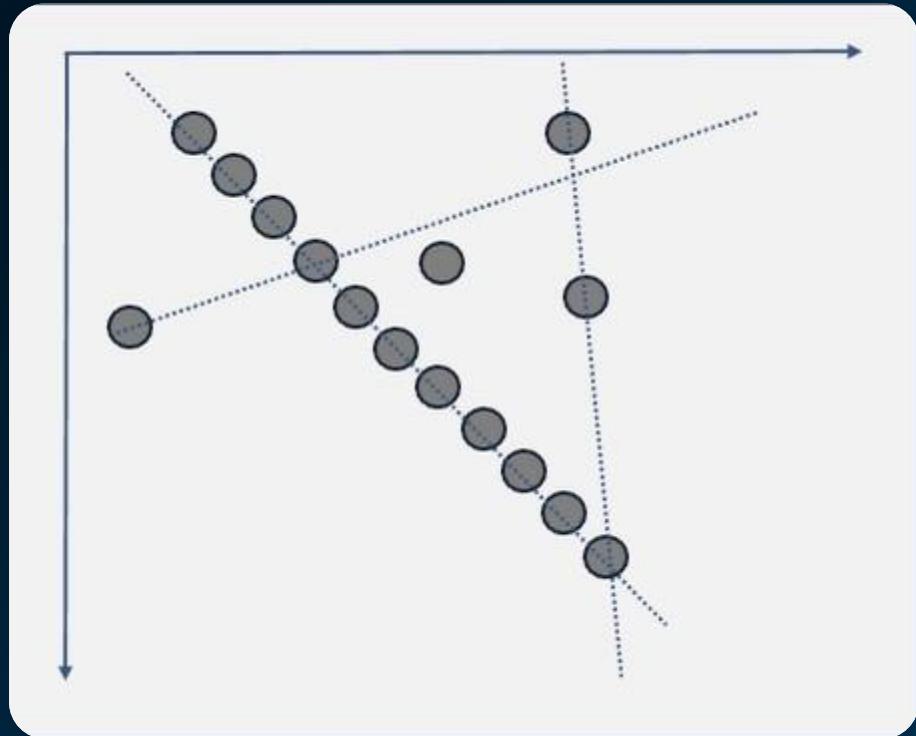
✓ 코드 설명

```
def process_image(self, msg):
    cv_image = self.bridge.imgmsg_to_cv2(msg, "bgr8")
    if self.method == "circle":
        processed_image = self.detect_circles(cv_image)
    elif self.method == "line":
        processed_image = self.detect_lines(cv_image)
    else:
        self.get_logger().error(f"Invalid method: {self.method}")
        return
    self.publisher.publish(
        self.bridge.cv2_to_imgmsg(processed_image, encoding="bgr8")
    )
```

▪ process_image()

subscribe node의 콜백 함수로, 이미지를 ros2 imgmsg형식에서 numpy array로 변환 후, method(line/circle)에 따라서 "허프 변환"을 수행 한 다음 다시 ros2 imgmsg로 바꾸어 publish하는 함수

✓ 허프 변환이란? Line



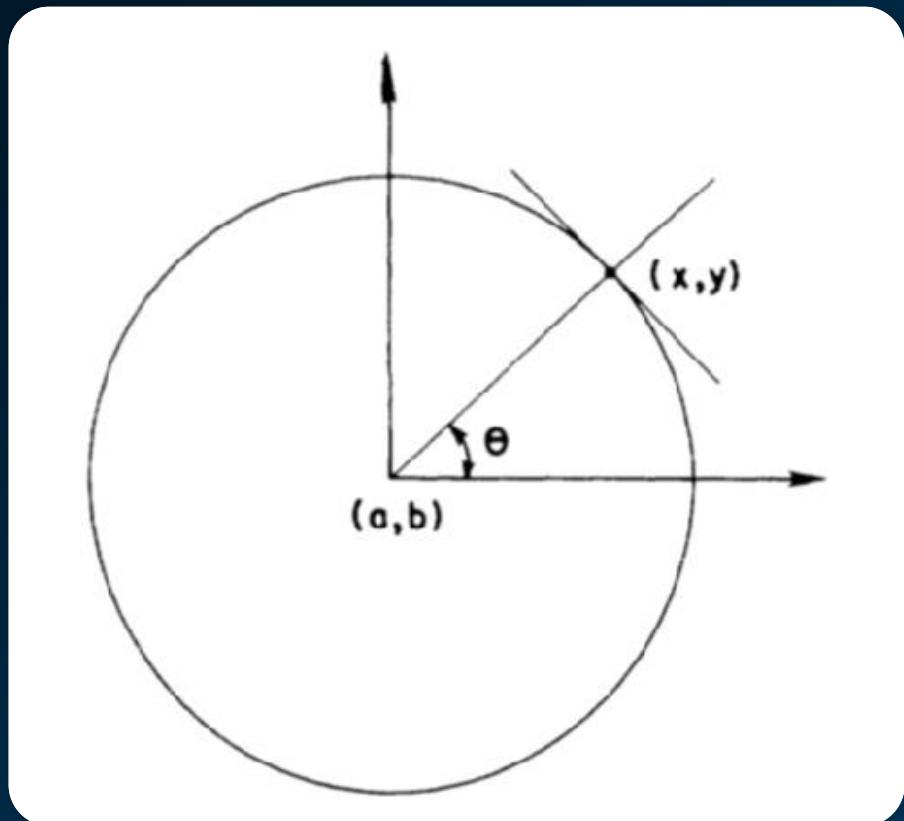
직선의 방정식을 만들어서 그 직선을 지나는 점의 개수가
Threshold(임계값)을 넘기면 직선으로 판단함

HoughLinesP(

```
image = "엣지를 검출한 대상 이미지",
rho = "단위 픽셀 거리마다 직선 검사",
theta = "단위 각도마다 직선 검사",
threshold = "임계값 설정",
minLineLength = "최소 n픽셀 이상 지속되어야 직선으로 판별",
maxLineGap = "단위 픽셀 이상 떨어진 경우에 직선으로 판별"
)
```

- **rho**와 **theta**는 이미지에 대한 직선을 얼마나 촘촘히 계산할 것인가를 설정하는 파라미터로, 값이 클수록 정확도는 증가하지만 연산량이 늘어남
- **threshold**는 직선이 되기 위한 점의 최소 개수를 정의하는 파라미터 threshold값이 작아질수록 더 많은 직선이 검출됨
- **minLineLength**는 직선 길이가 n이상일 때 직선으로 판별하는 파라미터 minLineLength값이 작아질수록 더 많은 직선이 검출됨
- **maxLineGap**은 각 픽셀 사이의 거리가 n이상일 때 직선으로 판별하는 파라미터로, 작을수록 뭉쳐있는 픽셀 덩어리를 직선으로 인식할 확률이 큼

✓ 허프 변환이란? Circle



- 이미지에서 edge를 검출하고 각 edge에 속한 점들에 대해 가능한 모든 원의 중심과 반지름을 계산한다. 원의 중심으로 검출된 수가 threshold를 넘기면 그 좌표에서의 값이 원의 중심과 반지름이 된다

✓ 허프 변환이란? Circle

```
HoughCircles(  
    image = "흑백 이미지",  
    method = "허프 변환 방법(일반적으로 cv2.HOUGH_GRADIENT 사용)",  
    dp = "허프 공간의 해상도 비율",  
    minDist = "원의 중심점들 사이의 최소 거리",  
    param1 = "Canny 에지 검출기의 상한 임계값",  
    param2 = "원 검출을 위한 임계값",  
    minRadius = "검출할 원의 최소 반지름",  
    maxRadius = "검출할 원의 최대 반지름"  
)
```

- **dp**는 허프 변환의 해상도 비율을 설정하는 파라미터로, 값이 작을수록 더 높은 해상도로 원을 탐지하여 정확도가 증가하지만, 연산량이 늘어남
- **minDist**는 검출된 원들 사이의 중심점 거리의 최소값을 설정하는 파라미터로, minDist값이 작을수록 서로 가까운 원들이 많이 검출될 수 있으며, 중복된 원이 검출될 가능성이 큼
- **param1**은 Canny edge 검출기의 상한 임계값을 설정하는 파라미터로, param1 값이 작을수록 더 약한 edge도 검출됨
- **param2**는 원 검출을 위한 허프 변환의 임계값을 설정하는 파라미터로, 값이 작을수록 원이 될 가능성이 있는 더 많은 후보가 검출되지만 노이즈가 많아질 수 있음
- **minRadius**와 **maxRadius**는 검출할 원의 최소 및 최대 반지름을 설정하는 파라미터로, 범위가 넓을수록 다양한 크기의 원이 검출됨

✓ CannyEdge란?



이미지에서 경계를 찾는 방법으로, 밝기 변화가 큰 부분을 찾아 edge로 감지하여 물체의 윤곽을 추출



Canny(

```
image = "엣지를 검출할 대상 이미지",
threshold1 = "에지로 간주할 경계의 하한값 (약한 에지 필터링에 사용)",
threshold2 = "에지로 간주할 경계의 상한값 (강한 에지 필터링에 사용")
```

- 밝기 변화가 threshold1보다 작으면 edge가 아닌 것으로 간주
- 밝기 변화가 threshold2보다 크면 강한 edge로 간주
- 밝기 변화가 threshold1보다 크고 threshold2보다 작으면 약한 edge로 간주
- 약한 edge는 주변에 강한 edge가 있으면 연결하고 그렇지 않으면 edge가 아닌 것으로 간주

✓ 코드 설명

```
def detect_circles(self, img):
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    circles = cv2.HoughCircles(
        cv2.GaussianBlur(gray, (9, 9), 2),
        cv2.HOUGH_GRADIENT,
        1,
        20,
        param1=50,
        param2=37,
        minRadius=80,
        maxRadius=100,
    )
    if circles is not None:
        for x, y, r in np.round(circles[0, :]).astype("int"):
            cv2.circle(img, (x, y), r, (0, 255, 0), 4)
            cv2.circle(img, (x, y), 2, (0, 0, 255), 12)
    return img

def detect_lines(self, img):
    edges = cv2.Canny(cv2.cvtColor(img, cv2.COLOR_BGR2GRAY), 50, 200)
    lines = cv2.HoughLinesP(edges, 1, np.pi / 180, 10, None, 20, 2)
    if lines is not None:
        for x1, y1, x2, y2 in lines[:, 0]:
            cv2.line(img, (x1, y1), (x2, y2), (0, 255, 0), 1)
    return img
```

■ detect_circles()

이미지를 컬러에서 흑백으로 변환한 후 cv2.HoughCircles() 함수를 이용하여 이미지에서 원을 탐지하여 원의 중심 좌표와 반지름 얻어내 이미지에 원을 그리는 함수

■ Detect_lines_image()

cv2.Canny() 함수를 사용하여 edge를 검출한 후 cv2.HoughLinesP() 함수를 통해 이미지에서 선분의 시작과 끝 좌표를 반환하여 이미지에 선분을 그리는 함수

■ Step 1.

ros2_ws.zip파일의 압축을 해제한다

■ Step 3. colcon build

```
colcon build
```

■ Step 2. ros2_ws 디렉토리로 이동한다

```
cd ros2_ws
```

■ Step 4. source install/setup.bash

```
source install/setup.bash
```

- **Step 5.** 각각의 터미널에 다음과 같은 명령어를 입력



```
ros2 run hough_transform image_publisher  
--ros-args -p image_path:=img/sudoku.png
```



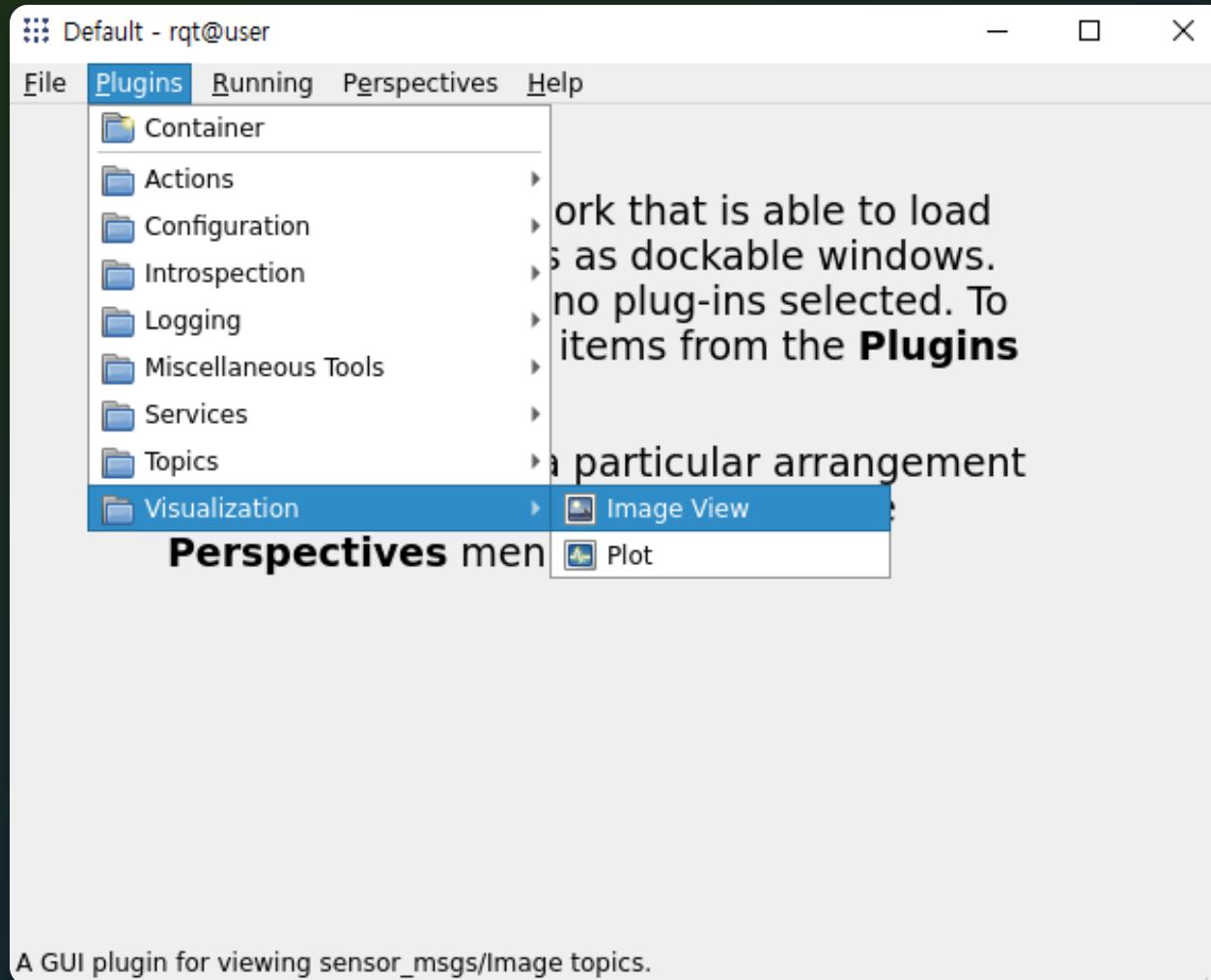
```
ros2 run hough_transform hough_transform  
--ros-args -p method:=line
```



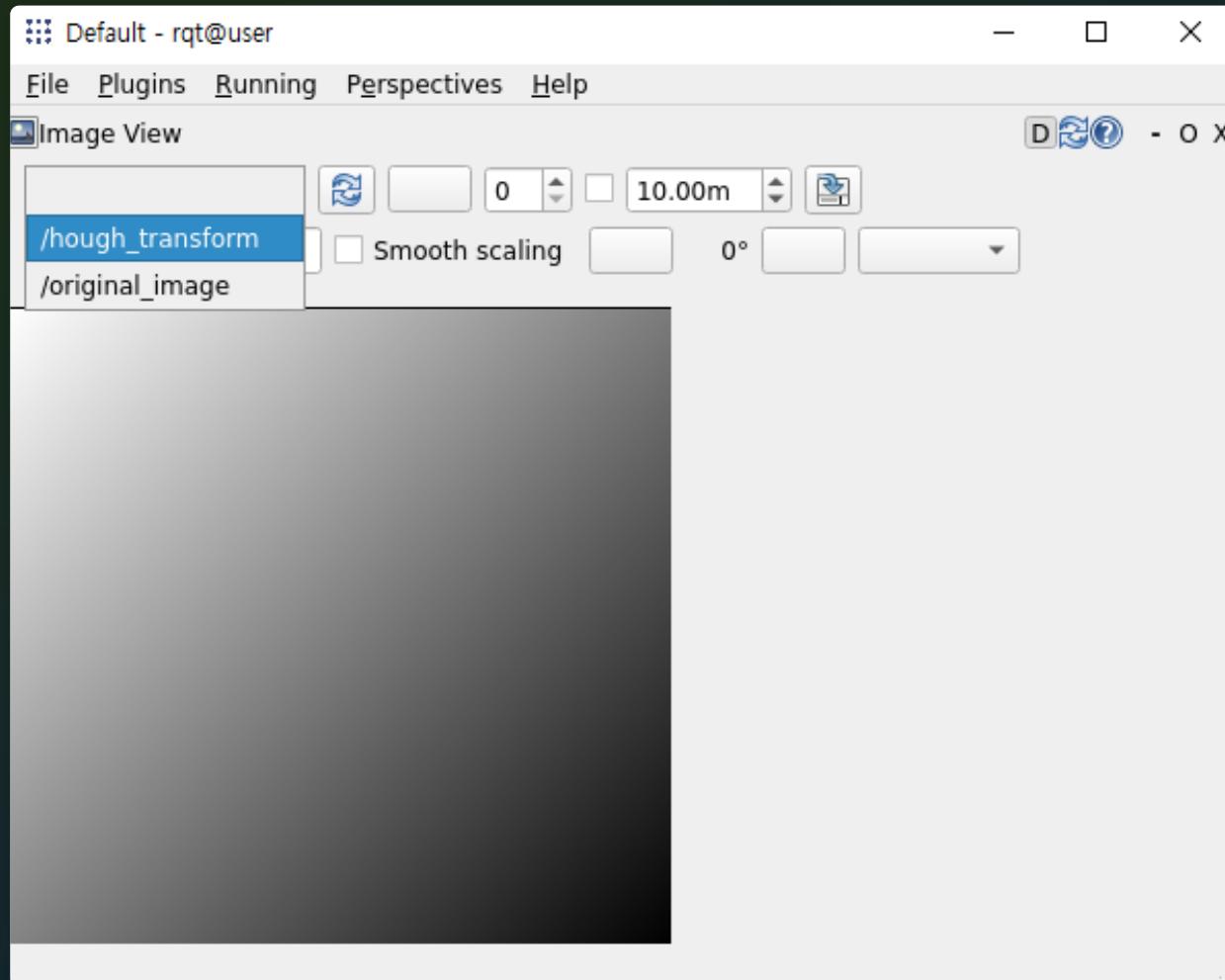
```
rqt
```

■ Step 6. rqt 세팅

Plugins > Visualization > Image View



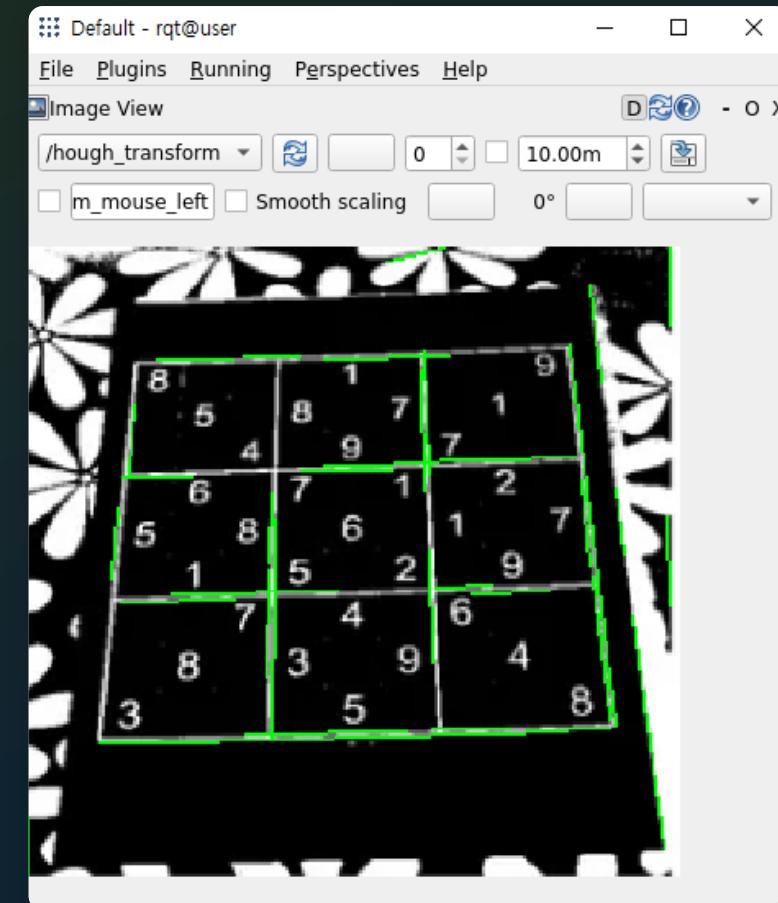
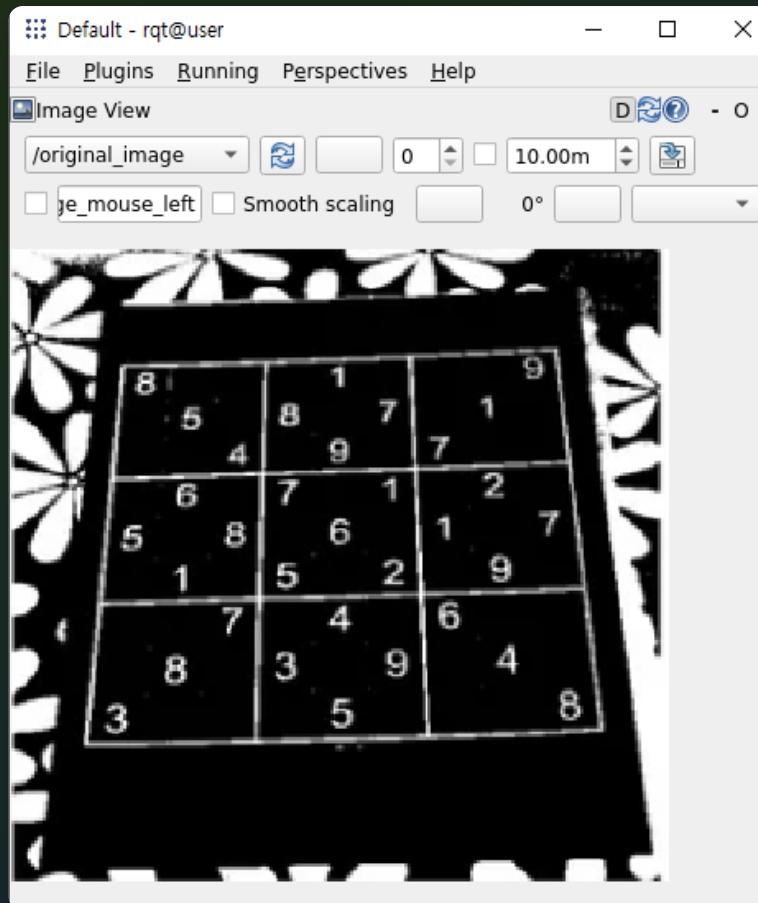
- Step 7. 토픽 선택
/hough_transform



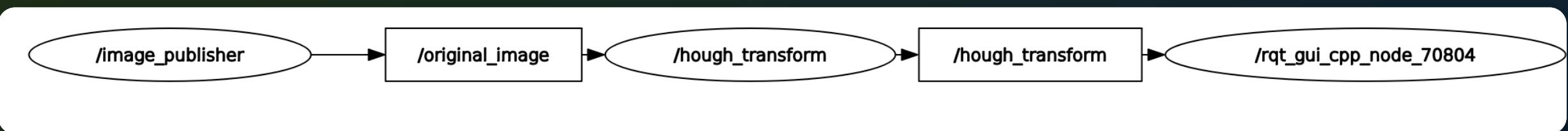
■ Step 8. 결과확인: 원본 이미지 vs 허프 변환 이미지

직선을 검출해서 초록색으로 표현함

HoughLinesP()의 파라미터를 조절하여 정확도를 높일 수 있음



■ Step 9. 결과확인: 노드 그래프

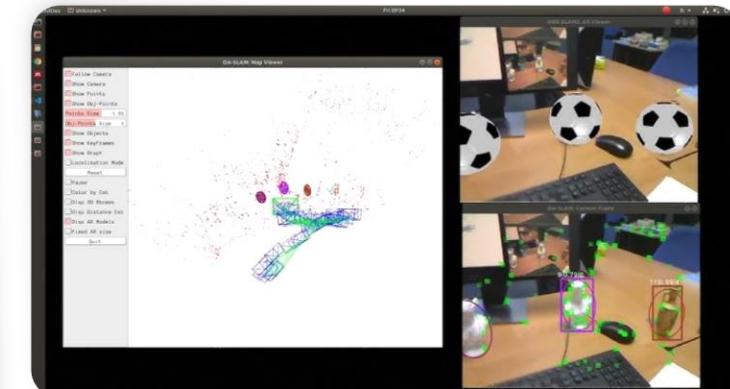
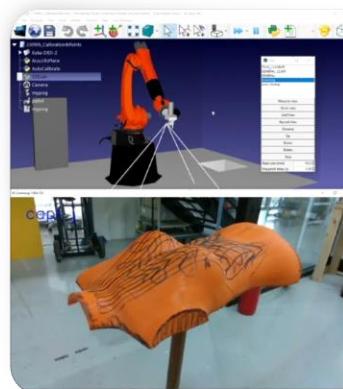


Open 3D

Open3D

- Open3D
 - 컴퓨터 그래픽과 3D 데이터 처리 작업을 위한 Python 라이브러리임
- 3D 데이터 처리
 - 인간이 물체를 '보는' 방식과 이를 '이해하고 분석하는' 과정을 모방함
 - 센서나 스캐너를 통해 3D 포인트 클라우드 데이터를 받아서 전처리/분석/시각화를 수행함
 - 자율 주행, 로봇 공학, AR/VR 등 다양한 분야에서 활용됨

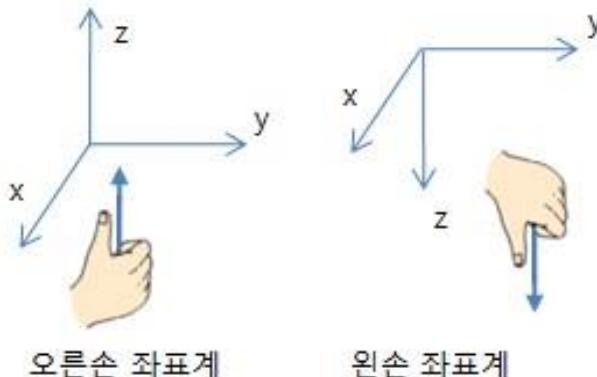
$$2d + \text{depth} = 30$$



Open3D와 ROS2

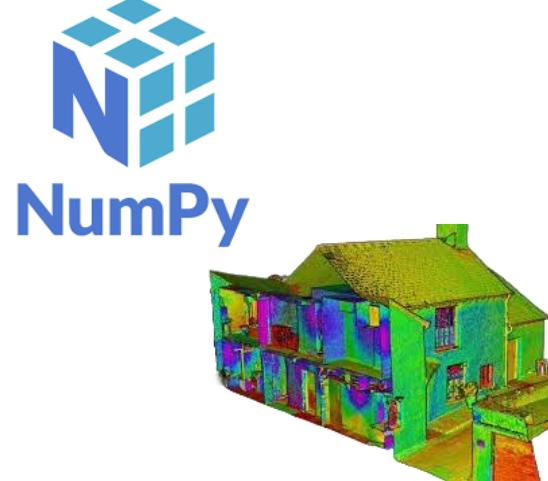
Open3D란?

▪ Open3D의 특징



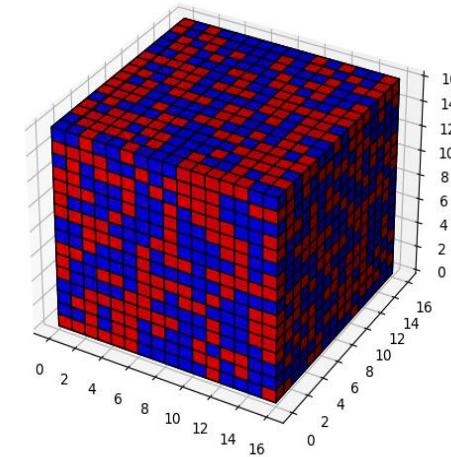
Open3D 좌표계

오른손 좌표계를 사용하며
원점은 (0,0,0)임



데이터 타입

이미지는 numpy 배열
혹은 PointCloud
형식으로 표현함



데이터 연산

3D 점들의 집합(포인트 클라우드)이나
모형에 대해 계산하고, 모양이나 위치를
분석하는 작업을 수행함

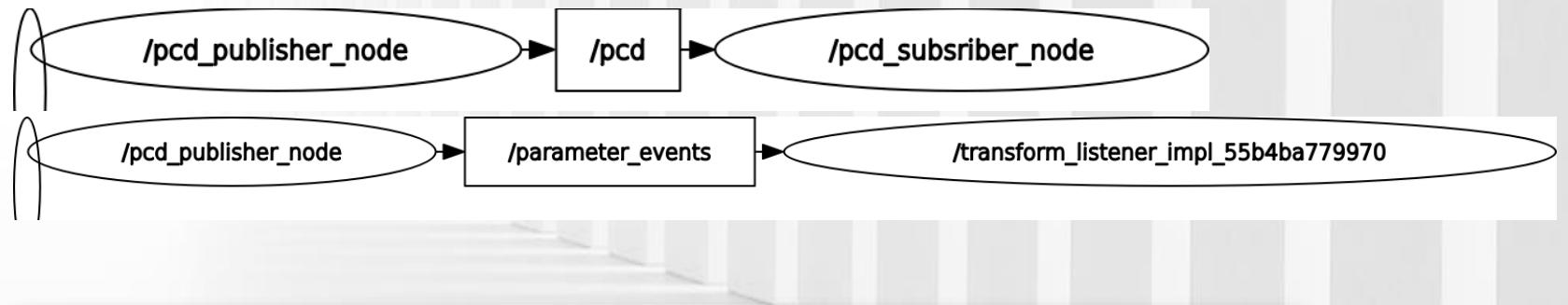
Open3D와 ROS2

실습 환경 구축

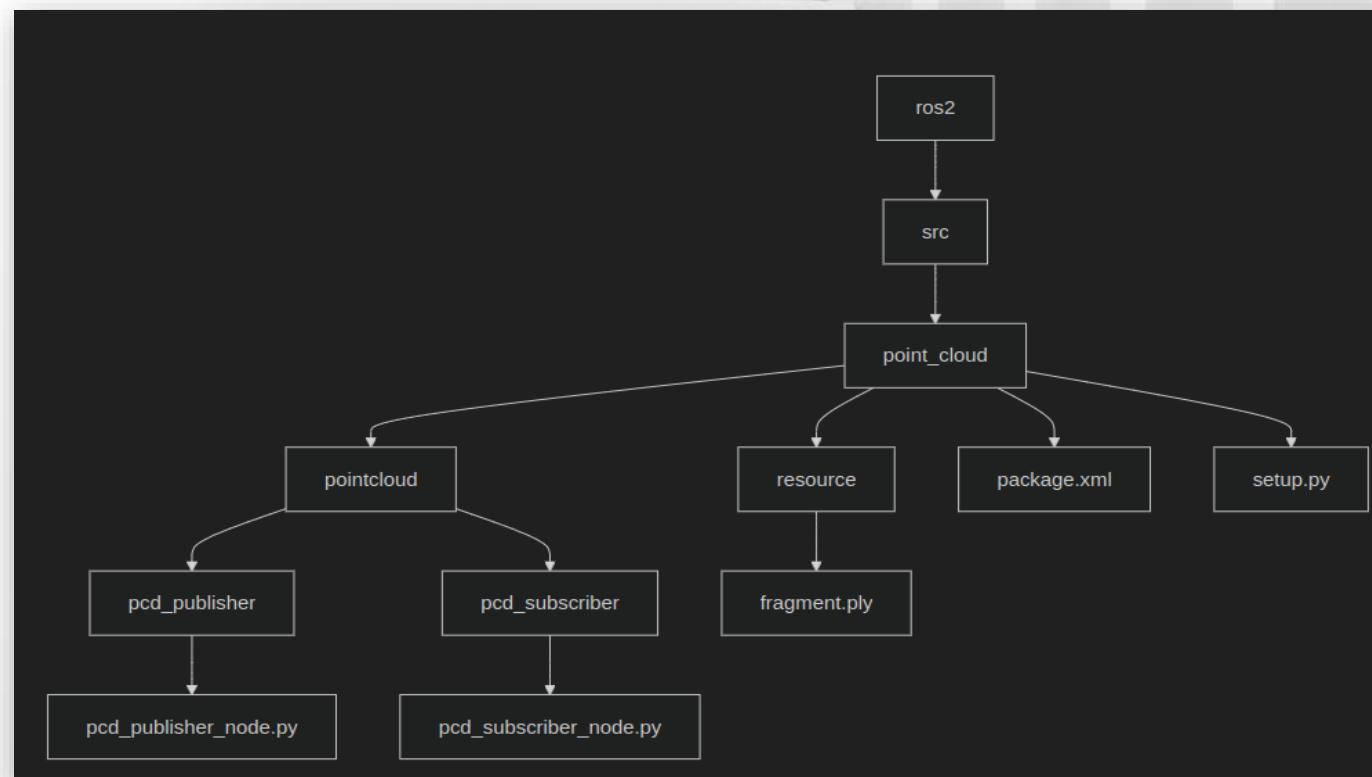
■ ros2_ws 한눈에 보기

publisher와 subscriber 구조

1. subscribe_node에 연결
2. Rviz에 연결



ros2_ws의 하위 디렉토리



✓ 코드와 설명

▪ pcd_publisher_node.py

- **struct**: Python에서 이진 데이터(숫자나 문자 같은 데이터)를 다룰 때 사용하는 라이브러리
- **open3d**: 3D 데이터를 처리하는 라이브러리



```
import sys
import os
import struct
import rclpy
import sensor_msgs.msg as sensor_msgs
import std_msgs.msg as std_msgs
import numpy as np
import open3d as o3d
from rclpy.node import Node
```

✓ 코드와 설명

▪ .ply파일 알아보기

- 3D 객체의 모양을 저장하는 파일 형식
- 점, 면 등의 정보를 포함해 3D 데이터를 저장
- 다음과 같은 코드를 통해 .ply파일을 numpy 배열로 바꿀 수 있음

```
● ● ●  
data = open3d.io.read_point_cloud("my_3d.ply")  
points = np.asarray(data.points)
```

```
● ● ●  
#.ply파일 내부에서의 정보  
0.5 0.3 0.7  
1.1 0.4 0.8  
-0.2 -1.0 0.5  
0.0 0.2 -0.6  
1.5 0.9 0.3
```

```
● ● ●  
#2차원 numpy배열로 변환된 정보  
array([[ 0.5,  0.3,  0.7],  
       [ 1.1,  0.4,  0.8],  
       [-0.2, -1.0,  0.5],  
       [ 0. ,  0.2, -0.6],  
       [ 1.5,  0.9,  0.3]])
```

✓ 코드와 설명

▪ pcd_publisher_node.py

- `self.load_point_cloud()` 포인트 클라우드 파일 불러오기
- `self.points = self.rotate_points_90(self.points)` 불러온 3D 점들을 90도 회전시킴
- `self.points[:, 2] += 2.5` 불러온 점들을 z축의 방향으로 이동시킴



```
class PCDPublisher(Node):
    #PCD 데이터를 퍼블리시하는 ROS2 노드
    def __init__(self, voxel_size, pcd_path):
        super().__init__('pcd_publisher_node')
        self.voxel_size = voxel_size
        self.pcd_path = pcd_path

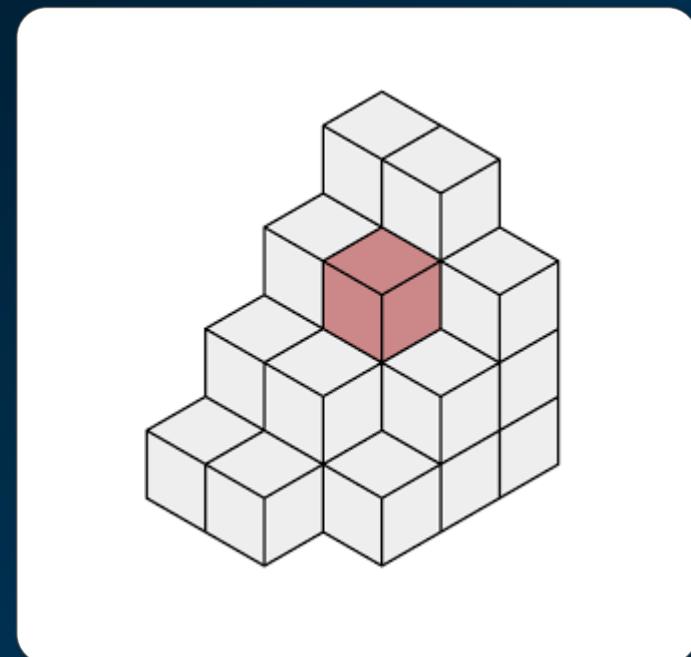
        self.load_point_cloud()
        self.points = self.rotate_points_90(self.points)
        self.points[:, 2] += 2.5

        self.pcd_publisher = self.create_publisher(sensor_msgs.PointCloud2, 'pcd', 10)
        self.timer = self.create_timer(1 / 30.0, self.timer_callback)
```

✓ 코드와 설명

▪ Voxel 알아보기

- 'Volume'과 'Pixel'의 합성어로, 3차원 공간에서의 '픽셀'을 의미
- 2D에서의 픽셀이 점으로 이미지를 표현한다면, 3D에서는 voxel이 작은 정육면체로 3D 물체를 표현
- 또한 Voxel의 크기는 필요에 따라 조절할 수 있음



Voxel의 크기가 작을 때	Voxel의 크기가 클 때
해상도 높음 처리 속도 느림	해상도 낮음 처리 속도 빠름

✓ 코드와 설명

```
● ● ●  
def load_point_cloud(self):  
    #포인트 클라우드 파일을 로드  
    if not os.path.exists(self.pcd_path):  
        raise FileNotFoundError("File doesn't exist.")  
  
    pcd = o3d.io.read_point_cloud(self.pcd_path)  
    if self.voxel_size > 0:  
        pcd = pcd.voxel_down_sample(self.voxel_size)  
  
    self.points = np.asarray(pcd.points)  
    self.colors = np.asarray(pcd.colors)  
  
def timer_callback(self):  
    #타이머 콜백 : points와 colors 데이터를 이용해 PointCloud 메시지를 생성  
    #좌표계 : map을 기준으로 함  
    pcd_msg = self.create_point_cloud_message(self.points, self.colors, 'map')  
    self.pcd_publisher.publish(pcd_msg)
```

- `o3d.io.read_point_cloud()`
 - 포인트 클라우드 데이터 불러오기
- `pcd.voxel_down_sample()`
 - 3D데이터의 해상도를 낮추는 함수
Voxel의 크기를 입력
 - down_sampling할 수록 처리속도가 빨라지지만 3D모델이 흐릿해짐
- `create_point_cloud_message(
 points,
 colors,
 'map'
)`
 - 3D점들의 위치(Points)와 색상(Colors)을 'map'이라는 좌표공간을 기준으로 ROS2 메시지를 만들

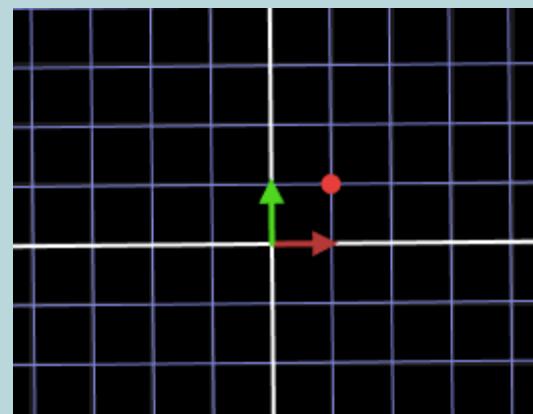
✓ 코드와 설명

▪ 행렬의 회전 알아보기

- 3차원 모델을 회전시킬 때 행렬의 곱셈을 통해 구현

좌표 평면 위의 (1,1) 점이 있음

$$\begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \times \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$



점에 대한 행렬에 다음과 같은
회전 변환 행렬을 곱하면
 θ 만큼 회전된 점의 값이 나옴

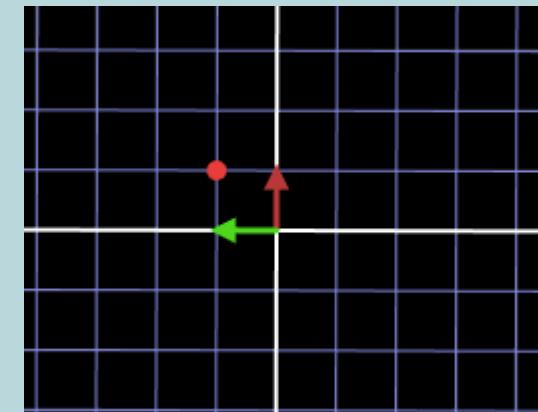
✓ 코드와 설명

▪ 행렬의 회전 알아보기

- 3차원 모델을 회전시킬 때 행렬의 곱셈을 통해 구현

90°만큼 회전시킬 때

$$\begin{bmatrix} 1 \\ 1 \end{bmatrix} \times \begin{bmatrix} \cos 90 & -\sin 90 \\ \sin 90 & \cos 90 \end{bmatrix} = \begin{bmatrix} 1 \\ -1 \end{bmatrix}$$



✓ 코드와 설명

▪ 행렬의 회전 알아보기

- 3차원 모델을 회전시킬 때 행렬의 곱셈을 통해 구현

3차원에서 회전 행렬

x축이 고정된 회전

$$R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\theta) & -\sin(\theta) \\ 0 & \sin(\theta) & \cos(\theta) \end{bmatrix}$$

y축이 고정된 회전

$$R_y(\theta) = \begin{bmatrix} \cos(\theta) & 0 & \sin(\theta) \\ 0 & 1 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) \end{bmatrix}$$

z축이 고정된 회전

$$R_z(\theta) = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

✓ 코드와 설명

■ rotate_points_90()

x축이 고정된 회전 행렬을 이용하여
3차원의 점을 90도 회전시킴

■ create_point_cloud_message()

포인트 클라우드 메시지를 생성함

```
def rotate_points_90(self, points):
    #포인트 클라우드를 X축 기준으로 90도 회전
    theta = np.radians(90)
    rotation_matrix = np.array([
        [1, 0, 0],
        [0, np.cos(theta), -np.sin(theta)],
        [0, np.sin(theta), np.cos(theta)]
    ])
    return np.dot(points, rotation_matrix)

def create_point_cloud_message(self, points, colors, parent_frame):
    #포인트 클라우드 메시지를 생성
    ros_dtype = sensor_msgs.PointField.FLOAT32
    data = [
        struct.pack('ffffI', x, y, z, (int(r * 255) << 16) | (int(g * 255) << 8) | int(b * 255))
        for (x, y, z), (r, g, b) in zip(points, colors)
    ]
    data = b''.join(data)

    fields = [
        sensor_msgs.PointField(name='x', offset=0, datatype=ros_dtype, count=1),
        sensor_msgs.PointField(name='y', offset=4, datatype=ros_dtype, count=1),
        sensor_msgs.PointField(name='z', offset=8, datatype=ros_dtype, count=1),
        sensor_msgs.PointField(name='rgb', offset=12, datatype=sensor_msgs.PointField.UINT32,
        count=1),
    ]

    header = std_msgs.Header(frame_id=parent_frame)
    return sensor_msgs.PointCloud2(
        header=header,
        height=1,
        width=points.shape[0],
        is_dense=False,
        is_bigendian=False,
        fields=fields,
        point_step=16,
        row_step=16 * points.shape[0],
        data=data
    )
```

✓ 코드와 설명

```
ros_dtype = sensor_msgs.PointField.FLOAT32
data = [
    struct.pack('fffI', x, y, z, (int(r * 255) << 16) | (int(g * 255) << 8) | int(b * 255))
    for (x, y, z), (r, g, b) in zip(points, colors)
]
data = b''.join(data)

fields = [
    sensor_msgs.PointField(name='x', offset=0, datatype=ros_dtype, count=1),
    sensor_msgs.PointField(name='y', offset=4, datatype=ros_dtype, count=1),
    sensor_msgs.PointField(name='z', offset=8, datatype=ros_dtype, count=1),
    sensor_msgs.PointField(name='rgb', offset=12, datatype=sensor_msgs.PointField.UINT32,
count=1),
]
```

- 이 코드는 좌표와 색상 데이터를 바이너리 형태로 변환하고 구조화 함

✓ 코드와 설명

```
● ○ ●  
points = [  
    [1.0, 2.0, 3.0],  
    [4.0, 5.0, 6.0]  
]  
colors = [  
    [1.0, 0.0, 0.0], # (Red: 255, 0, 0)  
    [0.0, 1.0, 0.0]  # (Green: 0, 255, 0)  
]
```

이와 같은 예시 데이터가 있을 때
x, y, z 좌표값은 float타입으로 바꾸고
colors는 int타입으로 바꾼다



```
ffffI : 1.0, 2.0, 3.0, 0xFF0000  
ffffI : 4.0, 5.0, 6.0, 0x00FF00
```



✓ 코드와 설명

RGB → 16진수 → 6진수



ffffI : 1.0, 2.0, 3.0, 0xFF0000

ffffI : 4.0, 5.0, 6.0, 0x00FF00



최종적으로 다음과 같이 바이너리 코드로 변환됨



x	y	z	rgb
\00\00\80\3f	\00\00\00\40	\00\00\40\40	\00\00\ff\00
\00\00\80\40	\00\00\40\40	\00\00\40\40	\00\ff\00\00

✓ 코드와 설명



```
def main(args=None):
    rclpy.init(args=args)

    if len(sys.argv) <= 1:
        raise ValueError("ply 파일이 필요합니다.")

    pcd_path = sys.argv[1]
    voxel_size = float(sys.argv[2]) if len(sys.argv) > 2 else
0.0

    pcd_publisher = PCDPublisher(voxel_size, pcd_path)
    rclpy.spin(pcd_publisher)
    pcd_publisher.destroy_node()
    rclpy.shutdown()

if __name__ == '__main__':
    main()
```

■ main()

voxel_size에 대한 인자와 .ply파일의 경로를 받아서 데이터를 publish함

✓ 코드와 설명

▪ pcd_subscriber_node.py

- `from sensor_msgs.msg import PointCloud2, PointField`

ROS2에서 3D 데이터를 처리하기 위해 사용되는 메시지 형식을 불러오는 부분

```
import struct
import math
import sys
import rclpy
from rclpy.node import Node
import sensor_msgs.msg as sensor_msgs
import numpy as np
import open3d as o3d
from sensor_msgs.msg import PointCloud2, PointField
```

✓ 코드와 설명

▪ pcd_subscriber_node.py



```
class PCDListener(Node):
    # ROS2 노드에서 포인트 클라우드 데이터를 수신하고 Open3D로 시각화
    def __init__(self):
        super().__init__('pcd_subscriber_node')
        self.vis = o3d.visualization.Visualizer()
        self.vis.create_window()
        self.o3d_pcd = o3d.geometry.PointCloud()
        self.pcd_subscriber = self.create_subscription(
            sensor_msgs.PointCloud2,
            'pcd',
            self.listener_callback,
            10
        )
```

self.vis = o3d.visualization.Visualizer()

- Open3D의 시각화 도구를 초기화

self.vis.create_window()

- Open3D 시각화 창을 생성

self.o3d_pcd = o3d.geometry.PointCloud()

- 비어있는 포인트 클라우드 객체를 생성
- ROS2를 통해 publish받은 데이터를 저장하기 위한 것

✓ 코드와 설명



```
def listener_callback(self, msg):
    # 포인트 클라우드 데이터를 처리하고 시각화
    points = read_points(msg, field_names=("x", "y", "z"),
skip_nans=True)
    pcd_as_numpy_array = np.array([point[:3] for point in
points])

    if pcd_as_numpy_array.shape[0] == 0:
        self.get_logger().error("Received empty point cloud")
        return

    self.vis.remove_geometry(self.o3d_pcd)
    self.o3d_pcd = o3d.geometry.PointCloud(
        o3d.utility.Vector3dVector(pcd_as_numpy_array))
    self.vis.add_geometry(self.o3d_pcd)
    self.vis.poll_events()
    self.vis.update_renderer() }
```

} #기타 시각화 및 업데이트 코드

■ `listener_callback()`

포인터 클라우드 데이터를 처리하고 시각화

- `points = read_points(msg, field_names=("x", "y", "z"), skip_nans=True)`

Point Cloud 메시지에서 x, y, z필드의 좌표 데이터만을 가져옴

- `pcd_as_numpy_array =`

`np.array([point[:3] for point in points])`

points데이터를 numpy배열로 바꾸는 코드

- `self.vis.remove_geometry(self.o3d_pcd)`

이전에 시각화된 포인트 클라우드를 제거 → 중복 렌더링을 방지

- `self.o3d_pcd = o3d.geometry.PointCloud(`

`o3d.utility.Vector3dVector(pcd_as_numpy_array)`

시각화를 위해 numPy 배열 데이터를

Open3D의 PointCloud 객체로 변환

✓ 코드와 설명



```
_DATATYPES = {  
    PointField.INT8: ('b', 1),  
    PointField.UINT8: ('B', 1),  
    PointField.INT16: ('h', 2),  
    PointField.UINT16: ('H', 2),  
    PointField.INT32: ('i', 4),  
    PointField.UINT32: ('I', 4),  
    PointField.FLOAT32: ('f', 4),  
    PointField.FLOAT64: ('d', 8)  
}
```

- 3D 데이터에 들어가는 각 점들의 정보
(예: 위치, 색상 등)가 어떤 형식으로 저장되는지를 나타낸 데이터 구조

```
def read_points(cloud, field_names=None):
    # PointCloud2 메시지에서 포인트 데이터를 추출
    assert isinstance(cloud, PointCloud2), 'cloud is not a sensor_msgs.msg.PointCloud2'
    fmt = _get_struct_fmt(cloud.is_bigendian, cloud.fields, field_names)

    width = cloud.width
    height = cloud.height
    point_step = cloud.point_step
    row_step = cloud.row_step
    data = cloud.data

    unpack_from = struct.Struct(fmt).unpack_from

    for v in range(height):
        offset = row_step * v
        for u in range(width):
            yield unpack_from(data, offset)
            offset += point_step
```

- **assert isinstance(cloud, PointCloud2)**

입력된 데이터가 PointCloud2 형식인지 확인하는 코드

- **fmt = _get_struct_fmt(cloud.is_bigendian, cloud.fields, field_names)**

3D 데이터를 저장할 '포맷'을 만듬. big endian, fields, field_name을 지정하여 데이터를 읽는 규칙, 데이터 내용 등을 설정

- **unpack_from = struct.Struct(fmt).unpack_from**

데이터를 어떤 포맷을 읽을 것인지 미리 정의해주는 역할

yield를 통해 함수 실행 중간에 unpack한 값을 return해줌

```
def _get_struct_fmt(is_bigendian, fields, field_names=None):
    # PointCloud2 필드 정보를 바탕으로 구조체 포맷 문자열 생성
    fmt = '>' if is_bigendian else '<'
    offset = 0
    fields_sorted = sorted(fields, key=lambda f: f.offset)
    for field in (f for f in fields_sorted if field_names is None or f.name in field_names):
        if offset < field.offset:
            fmt += 'x' * (field.offset - offset)
        offset = field.offset
        if field.datatype in _DATATYPES:
            datatype_fmt, datatype_length = _DATATYPES[field.datatype]
            fmt += field.count * datatype_fmt
            offset += field.count * datatype_length
        else:
            print(f'Skipping unknown PointField datatype [{field.datatype}]',
                  file=sys.stderr)
    return fmt
```

- **fmt = '>' if is_bigendian else '<'**
데이터를 읽을 방향 확인
- **offset = 0**
데이터를 읽기 시작할 위치를 0으로 설정

- **fields_sorted = sorted(fields, key=lambda f: f.offset)**
3D 데이터에 포함된 각 정보(x, y, z 좌표)를 offset 값으로 정렬
- **For문**
데이터 형식에 알맞게 포맷을 생성 후 return

✓ 코드와 설명



```
def main(args=None):
    # ROS2 노드를 초기화하고 실행
    rclpy.init(args=args)
    pcd_listener = PCDListener()
    rclpy.spin(pcd_listener)
    pcd_listener.destroy_node()
    rclpy.shutdown()
```

- **main()**

노드를 초기화하고 실행

✓ 코드와 설명

- Step 1. build하기



```
colcon build
```

- Step 2. open3d 라이브러리 설치



```
pip install open3d
```

- Step 3. 각각의 터미널에 명령어를 입력

Terminal1: fragment.ply파일의 경로와 voxel_size를 옵션으로 전달



```
ros2 run point_cloud pcd_publisher_node ~/ros2_ws/src/point_cloud/resource/fragment.ply 0
```

Terminal2

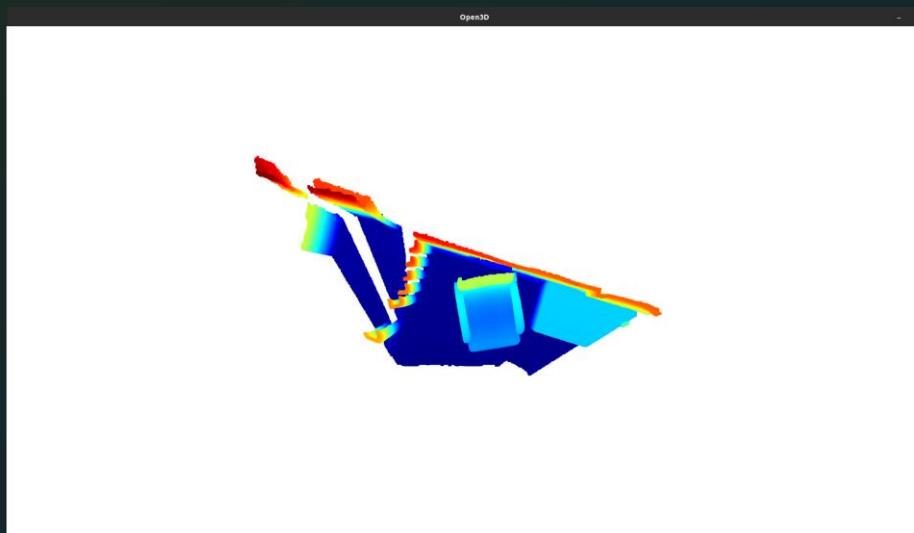


```
ros2 run point_cloud pcd_subscriber_node
```

✓ 코드와 설명

■ 실행결과

```
colcon build [1/1 done] [0 ongoing] 60x12
sjs ~ > ros2_ws > ros2 run point_cloud pcd_publisher_node
~/ros2_ws/src/point_cloud/resource/fragment.ply 0
/opt/ros/foxy/bin/ross2:6: DeprecationWarning: pkg_resources
is deprecated as an API. See https://setuptools.pypa.io/en/l
atest/pkg_resources.html
    from pkg_resources import load_entry_point
colcon build [1/1 done] [0 ongoing] 63x11
sjs ~ > ros2_ws > ros2 run point_cloud pcd_subscriber_node
/opt/ros/foxy/bin/ross2:6: DeprecationWarning: pkg_resources is
deprecated as an API. See https://setuptools.pypa.io/en/latest/
pkg_resources.html
    from pkg_resources import load_entry_point
```



→ 다음과 같이 Open3D 화면이 출력되면 성공

✓ 코드와 설명

- Rviz2로 시각화하기

Terminal1

fragment.ply파일의 경로와 voxel_size를 옵션으로 전달



```
ros2 run point_cloud pcd_publisher_node ~/ros2_ws/src/point_cloud/resource/fragment.ply 0
```

Terminal2

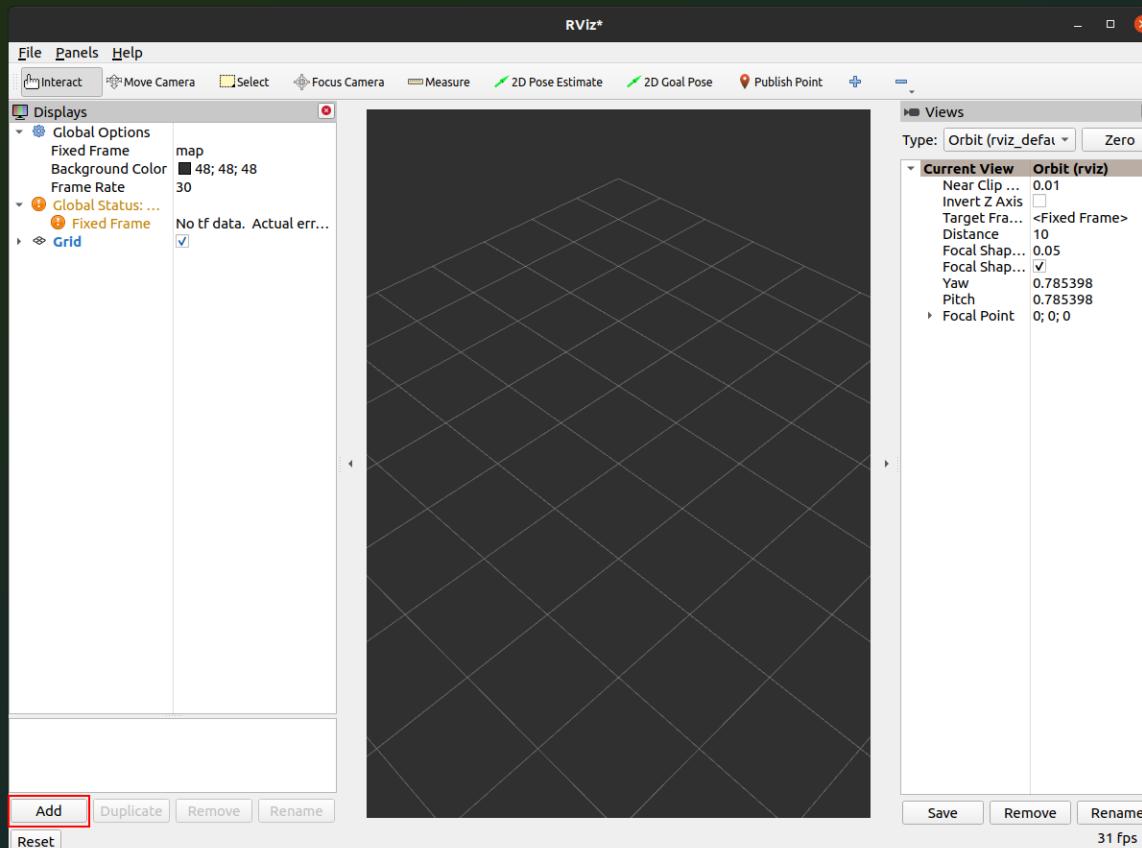


```
rviz2
```

✓ 코드와 설명

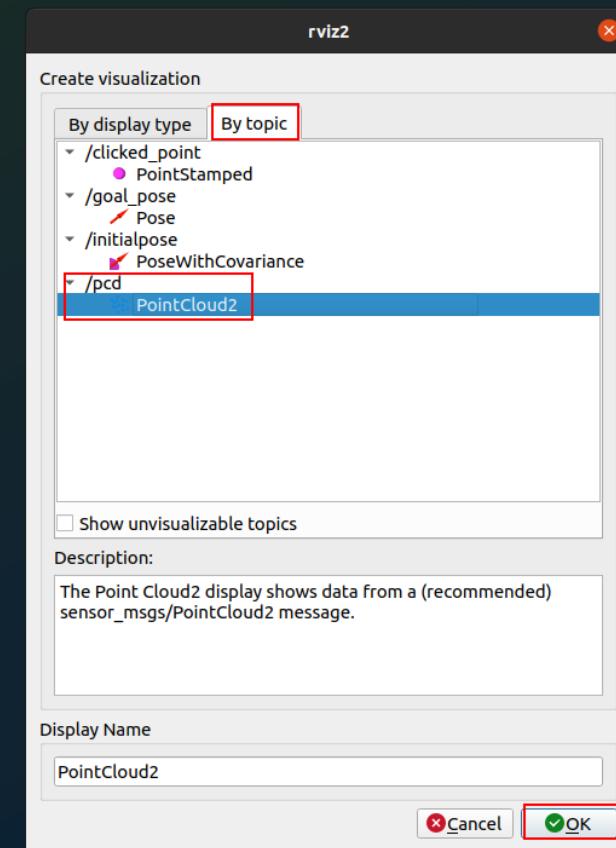
■ Rviz2로 시각화하기

Add버튼 클릭

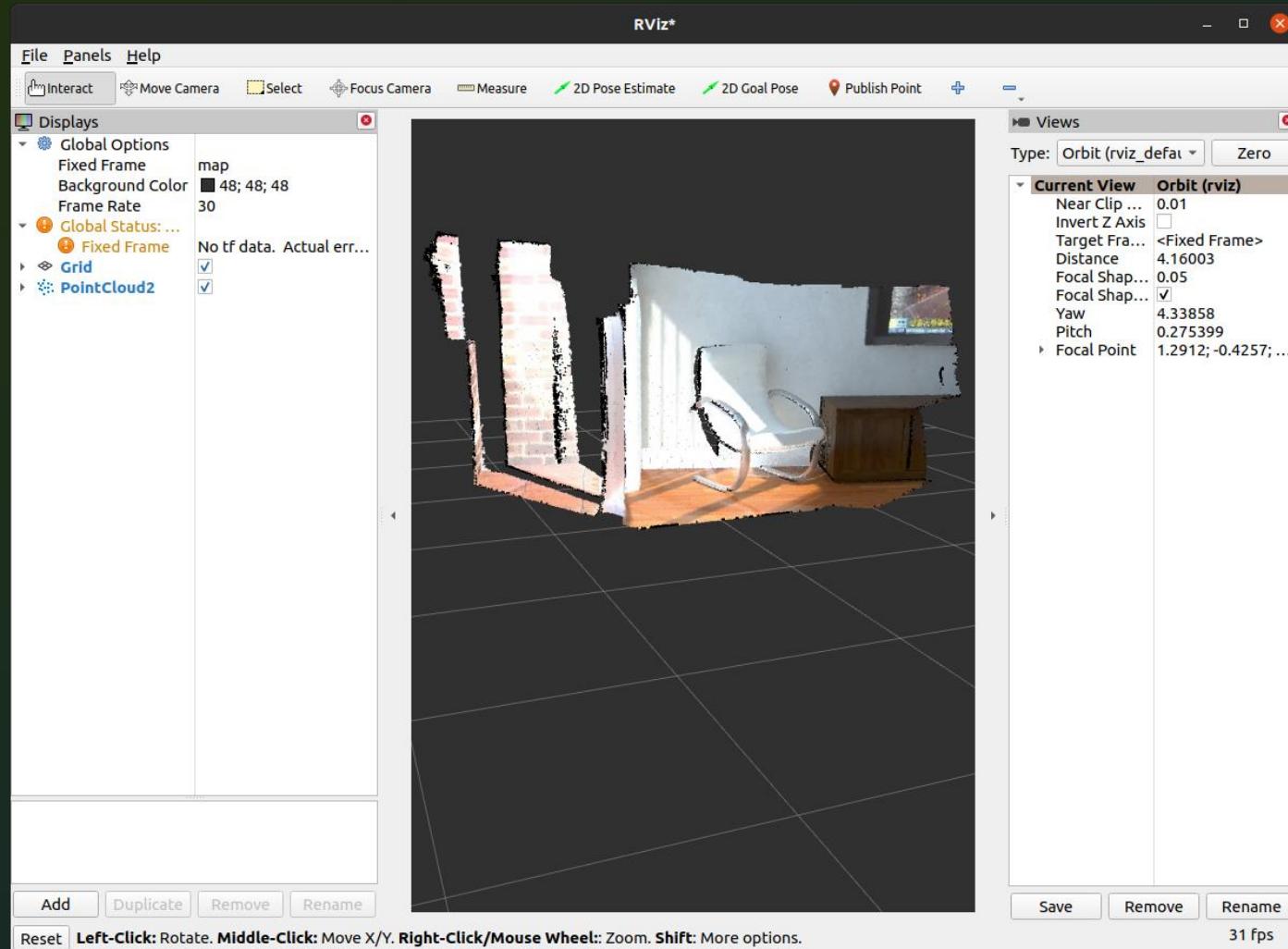


■ Rviz2로 시각화하기

By topic - PointCloud2선택 - OK



Rviz2로 시각화하기



좌 드래그	시점 회전
우 드래그	줌인/아웃
휠 스크롤	줌인/아웃
휠 클릭 드래그	카메라 위치 조절

✓ 코드와 설명

- Rviz2로 시각화하기



```
ros2 run point_cloud pcd_publisher_node ~/ros2_ws/src/point_cloud/resource/fragment.ply 0.05
```

- voxel_size를 0.05로 설정하면 듬성듬성 랜더링되는 3D모델을 볼 수 있다

✓ 코드와 설명

- Rviz2로 시각화하기

