Boot Camp Part 3

Objective

In the project, your will be tasked with finding and retrieving objects. Therefore, in this part, you will make a simple object detector.

Prerequisite

Install Linux and ROS following the instructions here (https://canvas.kth.se/courses/45204/pages/linux-installation) .

Detection

Create a package called **detection** and a node called **detection** in it:

```
cd ~/dd2419_ws/src
ros2 pkg create detection --build-type ament_python --node-name detection --dependencies python3
-numpy python3-open3d-pip rclpy sensor_msgs sensor_msgs_py --license MIT
```

Make sure all the dependencies are installed

```
cd ~/dd2419_ws
rosdep install --from-paths src -y --ignore-src --as-root pip:false
```

Build the new package

```
cd ~/dd2419_ws
colcon build --symlink-install
```

Source the workspace so the new package is included in your path

```
source ~/dd2419_ws/install/local_setup.bash
```

Edit the **detection** node by editing the file `~/dd2419_ws/src/detection/detection/detection.py:`

```
#!/usr/bin/env python

import math

import numpy as np

import rclpy
from rclpy.node import Node
```

```
from opensdi import c opensdi as osd

import ctypes
import struct

class Detection(Node):

    def __init__(self):
        super().__init__('detection')

        # Initialize the publisher
        self._pub = self.create_publisher(
            PointCloud2, '/camera/depth/color/ds_points', 10)

        # Subscribe to point cloud topic and call callback function on each recieved message
        self.create_subscription(
            PointCloud2, '/camera/depth/color/points', self.cloud_callback, 10)

    def cloud_callback(self, msg: PointCloud2):
        """Takes point cloud readings to detect objects.

        This function is called for every message that is published on the '/camera/depth/color/
points' topic.

        Your task is to use the point cloud data in 'msg' to detect objects. You are allowed to
add/change things outside this function.

        Keyword arguments:
        msg -- A point cloud ROS message. To see more information about it
        run 'ros2 interface show sensor_msgs/msg/PointCloud2' in a terminal.
        """

        # Convert ROS -> NumPy
        gen = pc2.read_points_numpy(msg, skip_nans=True)
        xyz = gen[:,:3]
        rgb = np.empty(xyz.shape, dtype=np.uint32)

        for idx, x in enumerate(gen):
            c = x[3]
            s = struct.pack('>f' , c)
            i = struct.unpack('>l', s)[0]
            pack = ctypes.c_uint32(i).value
            rgb[idx, 0] = np.asarray((pack >> 16) & 255, dtype=np.uint8)
            rgb[idx, 1] = np.asarray((pack >> 8) & 255, dtype=np.uint8)
            rgb[idx, 2] = np.asarray(pack & 255, dtype=np.uint8)

        rgb = rgb.astype(np.float32) / 255

        # Convert NumPy -> Open3D
        cloud = o3d.geometry.PointCloud()
        cloud.points = o3d.utility.Vector3dVector(xyz)
        cloud.colors = o3d.utility.Vector3dVector(rgb)

        # Downsample the point cloud to 5 cm
        ds_cloud = cloud.voxel_down_sample(voxel_size=0.05)

        # Convert Open3D -> NumPy
        points = np.asarray(ds_cloud.points)
        colors = np.asarray(ds_cloud.colors)




def main():
    rclpy.init()
    node = Detection()
```

```
        except KeyboardInterrupt:
            pass

    rclpy.shutdown()


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

In the above code, you see how to convert between ROS and Open3D. We also downsample the point cloud to 5 cm.

NOTE: The code above is just an example code for how to convert between ROS, NumPy, and Open3D. If you want to use something else, such as PCL, it is totally fine. If you are familiar with C++, I can recommend doing this part in C++ with PCL as PCL has nice functionality to accomplice this.

Running

You will need 4 terminals/"tabs", the order in which you run these are important.

Terminal 1

```
ros2 launch robp_boot_camp_launch boot_camp_part3_launch.xml
```

This will also run your odometry node from part 2.

Terminal 2

```
ros2 run tf2_ros static_transform_publisher --frame-id map --child-frame-id odom
```

To make sure you have a transform between **map** and **odom**.

Terminal 3

```
ros2 run detection detection
```

This will start your detection node.

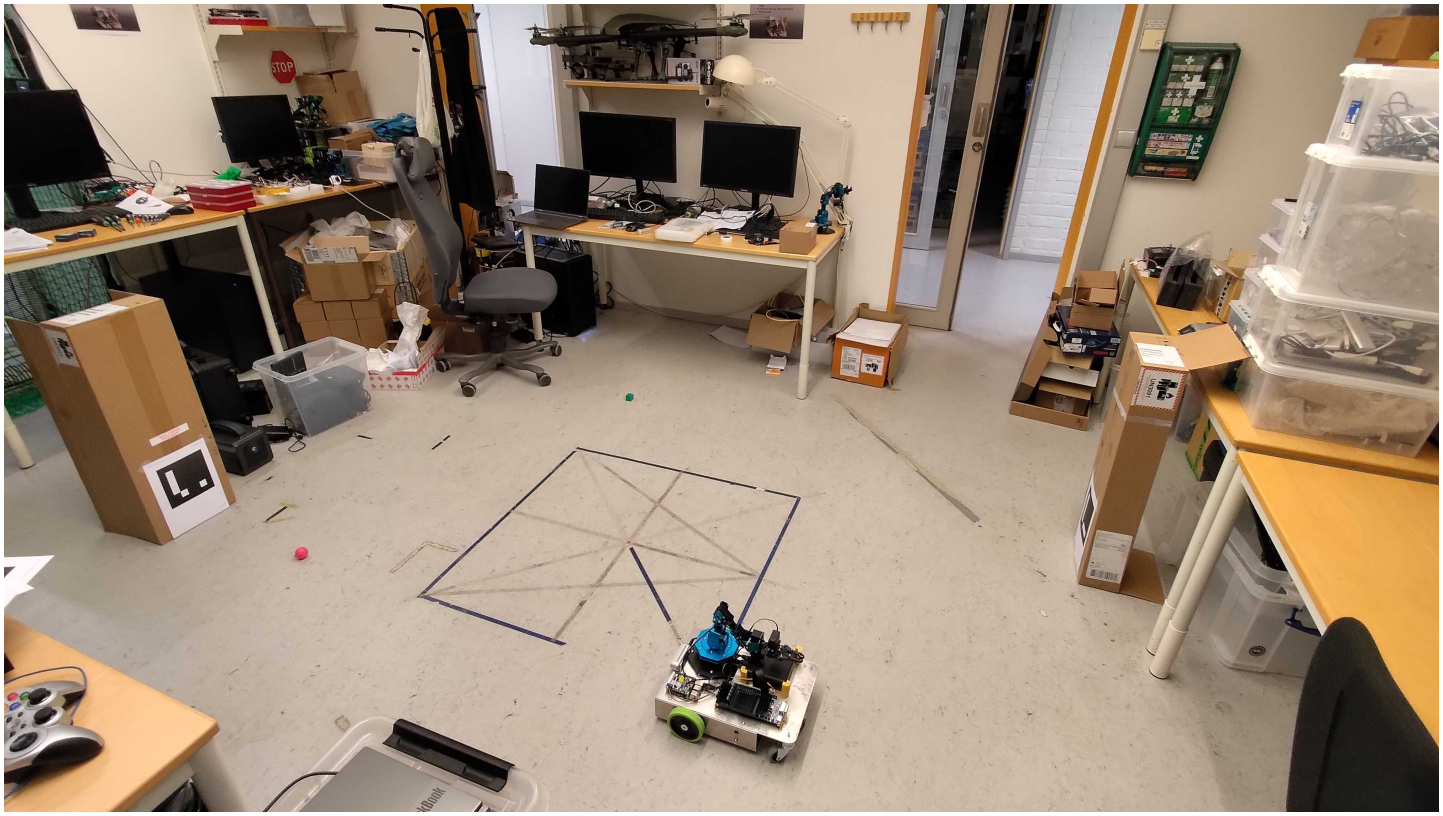
Terminal 4

```
ros2 bag play --read-ahead-queue-size 100 -l -r 1.0 --clock 100 --start-paused ~/dd2419_ws/bags/boot_camp
```

enables publishing of the clock time. *--start-paused* starts the rosbag in paused mode, this is to make sure that RViz and your nodes receive all the *static transforms* that are being published in the beginning. If you do not have these you will most likely have problems with timestamps. You press SPACE in this terminal to resume/pause the playback of the rosbag. To read what the optional arguments `--read-ahead-queue-size`, `-l`, `-r`, `--clock`, and `--start-paused`, do, you can run `ros2 bag play -h`.

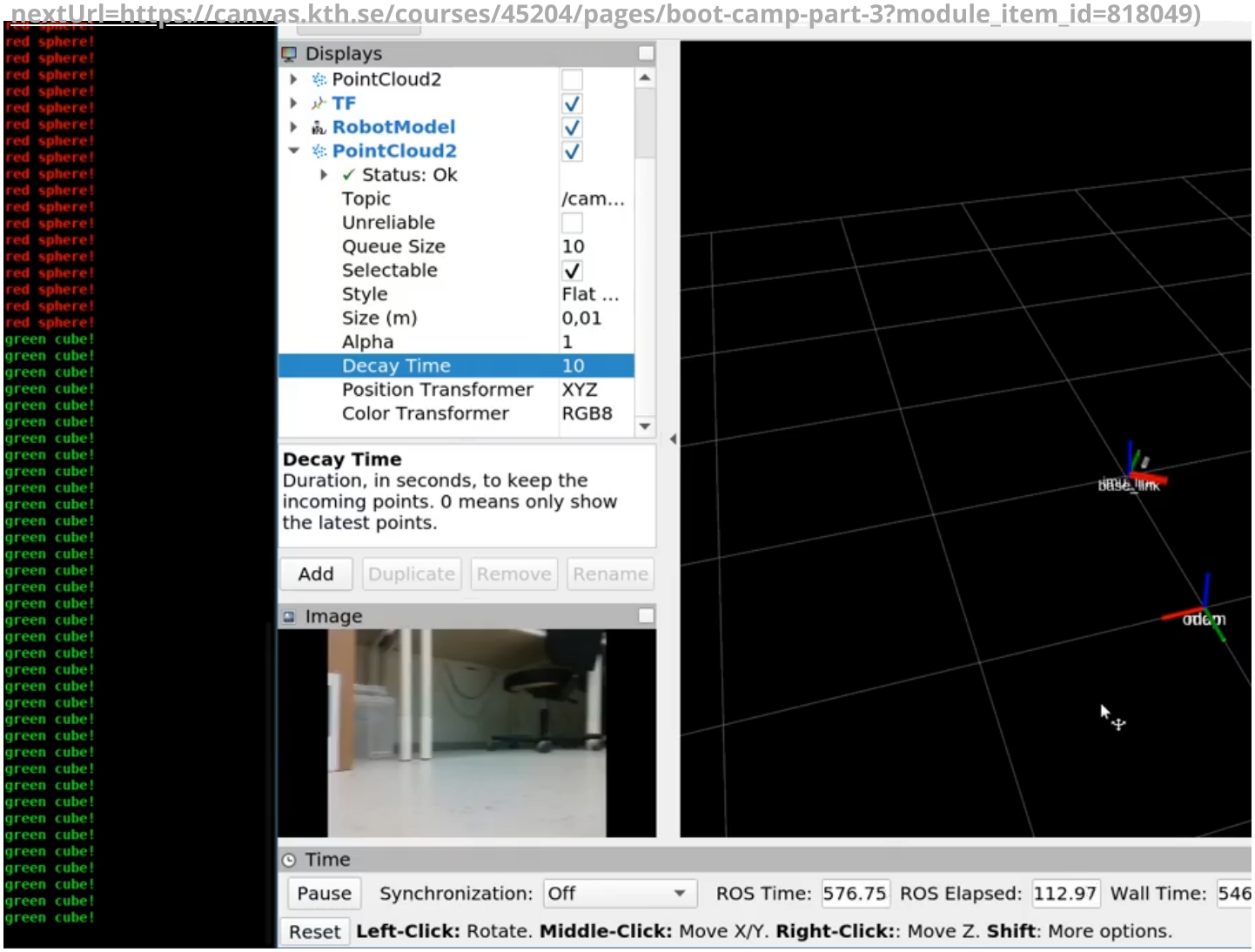
Task

In the environment, there are a red sphere and a green cube placed at (1.5, 0) and (1, -1.5) respectively, where (0, 0) is the robots starting position. You should be able to detect at least one of them and output when you see it/them. You have to use the point cloud to do this. It is cool if you are able to place the objects in the environment as well; this can be accomplished similarly to how you placed the ArUco markers.



Expected Output

In the video below you see both that we print something when we detect the red sphere and the green cube respectively, the minimum is that you should be able to do this for one object. But please do both if you can! If you want you can also send the points for visualization in RViz as you can see in the video as well, this may also help you when debugging.



Tips

- You can filter all points which are further away than 0.9m or are below ground, as the objects are the only objects in the environment that are closer than 1m from the robot and above ground.
- You can filter the points based on the color.