Hello World II (Ouster, Blinker and Dataspeed: Input and Feedback)

Approximate Time Investment: 1-2 hours

Now that we covered Hello World I, we will move onto Hello World II where we will incorporate a feedback loop with input and output. We will create an "alarm" system where an input (motion detected by the Ouster) will trigger an output (blinkers flash when motion is detected by the Ouster).

We have done our sanity check that the Ouster OS2 is working using Ouster Studio before but now we will visualize the OS2's feed using ROS2 first. It is recommended to read through the documentation by Ouster in their official repository at the link below.

Official Ouster ROS2 Drivers: https://github.com/ouster-lidar/ouster-ros/tree/ros2

1. Clone the ouster-ros repository into our workspace's src folder by running the following commands in a new terminal.

```
cd ros2_ws/src
git clone -b ros2 --recurse-submodules https://github.com/ouster-lidar/ouster-ros.git
```

   a. Note that the clone command for the ouster-ros driver is different from the previous packages. As of 9/23/2025, the Ouster ROS driver repository supports a variety of ROS versions. The "-b ros2 --recurse-submodules" part of the clone command ensures that we are only cloning the ROS2 branch of the repository. It is good practice to look through the official documentation of the drivers in this documentation rather than relying too heavily on the launch commands given here.

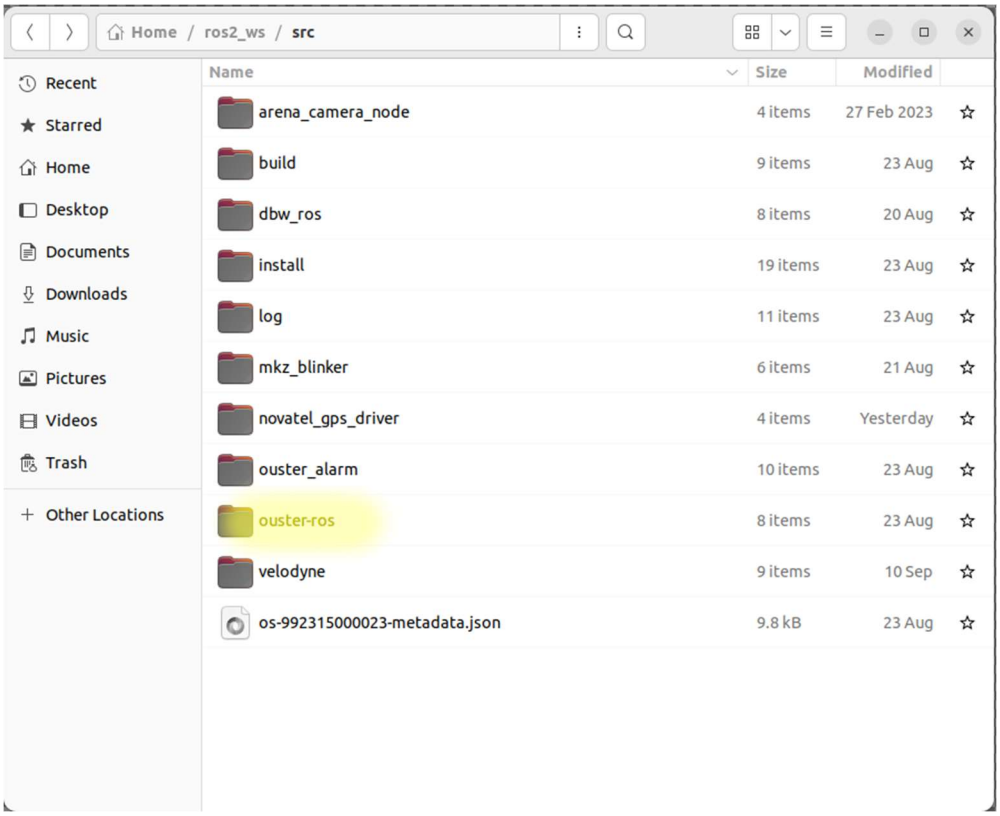


Figure 1: Cloned ouster-ros in our src folder

2. Build our workspace and sanity check that we can visualize the OS2 in ROS2's Rviz gui.
    a. In our terminal, go back one directory to ros2_ws and build our workspace as we've done before.

```
cd ..
colcon build --symlink-install --cmake-args -DCMAKE_BUILD_TYPE=Release
```

    b. Note that the build command is different from when we built our workspaces before. The ouster-ros driver depends on CMake to build the package along with libcurl. If there are issues with building the workspace there are likely missing dependencies and packages. Refer to the official Ouster ROS2 driver webpage the requirements section.
    https://github.com/ouster-lidar/ouster-ros/tree/ros2

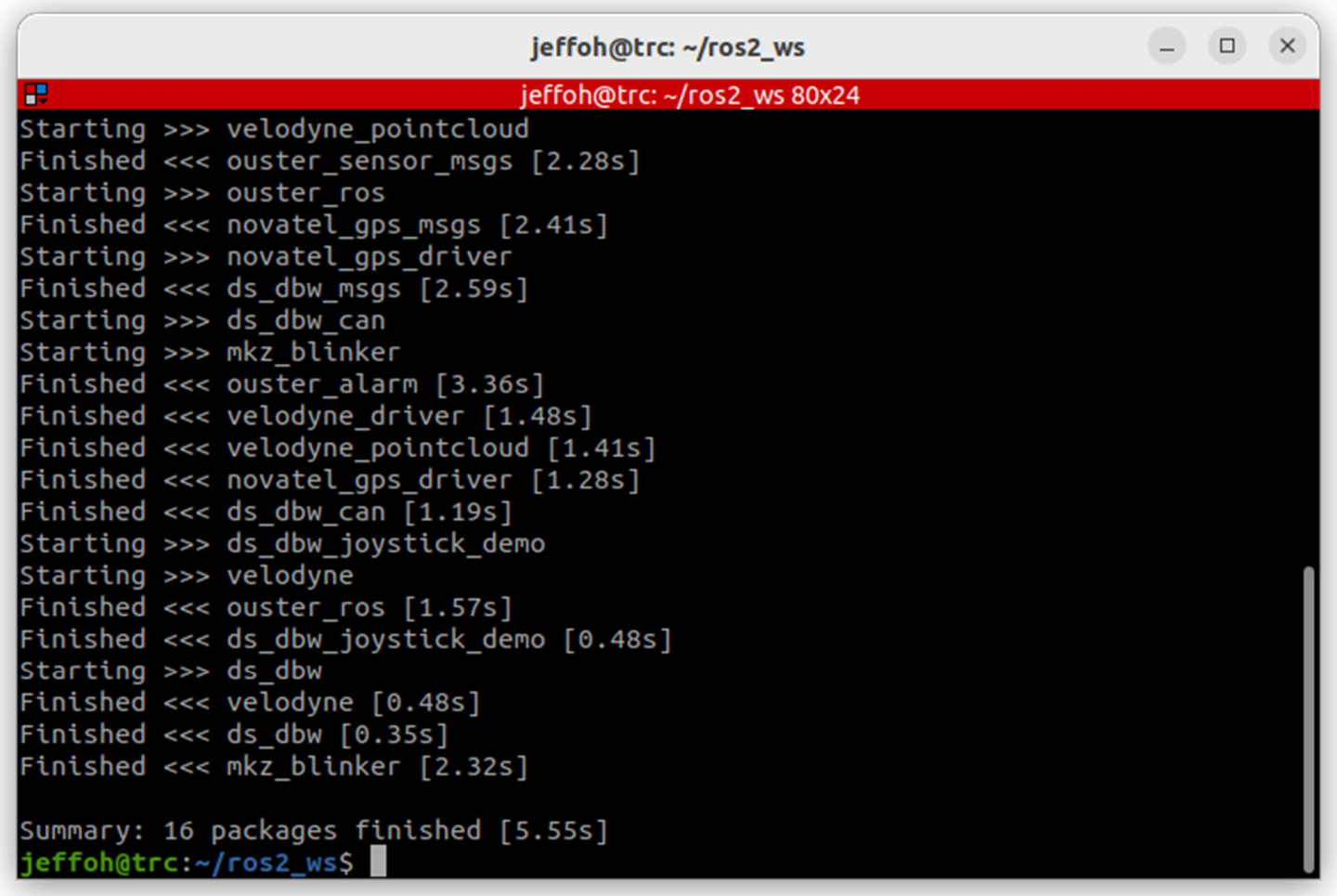
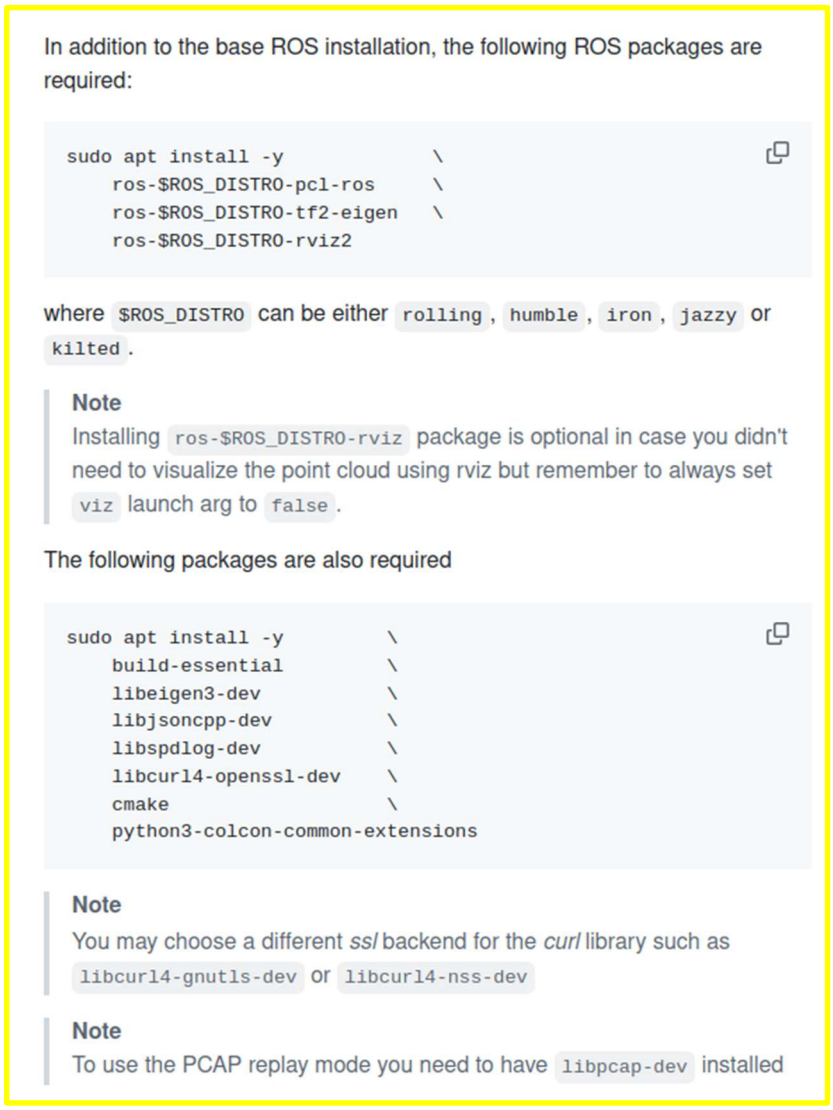


Figure 2: Note ouster_ros and ouster_sensor_msgs have been built



In addition to the base ROS installation, the following ROS packages are required:

```
sudo apt install -y            \
    ros-$ROS_DISTRO-pcl-ros    \
    ros-$ROS_DISTRO-tf2-eigen  \
    ros-$ROS_DISTRO-rviz2
```

where `$ROS_DISTRO` can be either `rolling`, `humble`, `iron`, `jazzy` or `kilted`.

> **Note**
> Installing `ros-$ROS_DISTRO-rviz` package is optional in case you didn't need to visualize the point cloud using rviz but remember to always set `viz` launch arg to `false`.

The following packages are also required

```
sudo apt install -y        \
    build-essential        \
    libeigen3-dev          \
    libjsoncpp-dev         \
    libspdlog-dev          \
    libcurl4-openssl-dev   \
    cmake                  \
    python3-colcon-common-extensions
```

> **Note**
> You may choose a different *ssl* backend for the *curl* library such as `libcurl4-gnutls-dev` or `libcurl4-nss-dev`

> **Note**
> To use the PCAP replay mode you need to have `libpcap-dev` installed

Figure 3: Requirements section from the official Ouster ROS2 driver github

c. Close the terminal and open a new one. This is to ensure the newly built workspace is sourced. We can check our package list by running:

```
ros2 pkg list
```

The Ouster packages should be in the list.

d. Launch the Ouster OS2 through ROS. Recall that when we did our sanity check using Ouster Studio we wrote down our serial number. The serial number can also be found by running the following command.

```
avahi-browse -art | grep os-
```

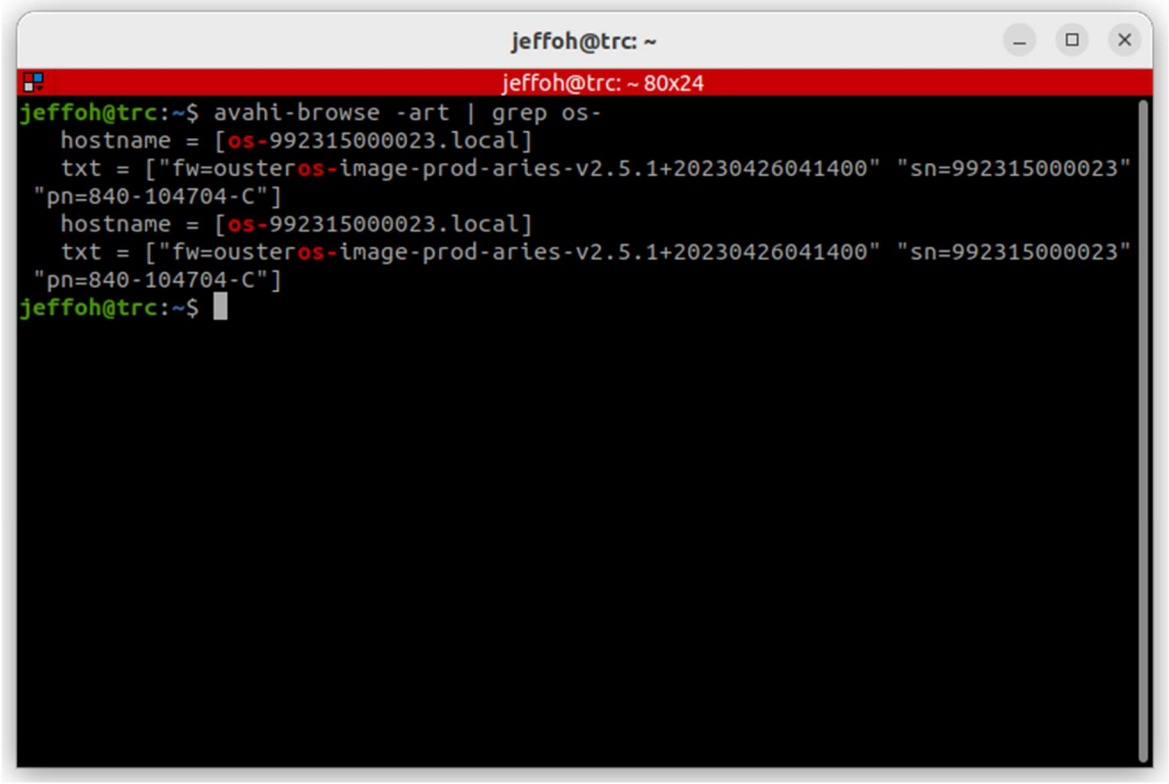


Figure 4: Output of "avahi-browse -art | grep os-" showing the OS2's serial number

Our serial # is 992315000023 so we will launch the OS2 per the official documentation in the repo with the following command.

```
ros2 launch ouster_ros sensor.launch.xml  sensor_hostname:=os-992315000023.local
```

Rviz should launch and you will be able to visualize the OS2. It may take a couple seconds.

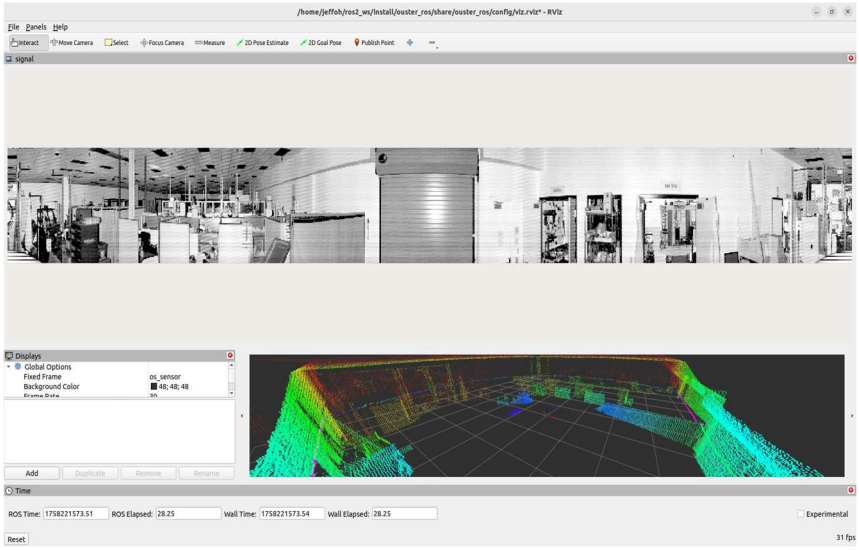


Figure 5: Ouster OS2 visualiztion through ROS2's Rviz

We have completed our OS2 ROS sanity check.

3. Download and build the ouster_alarm package.
    a. Download and place the ouster_alarm package/folder in our workspace's src folder, then colcon build. Follow the same instructions for this as we did for the mkz_blinker package in Hello World I.
    ouster_alarm

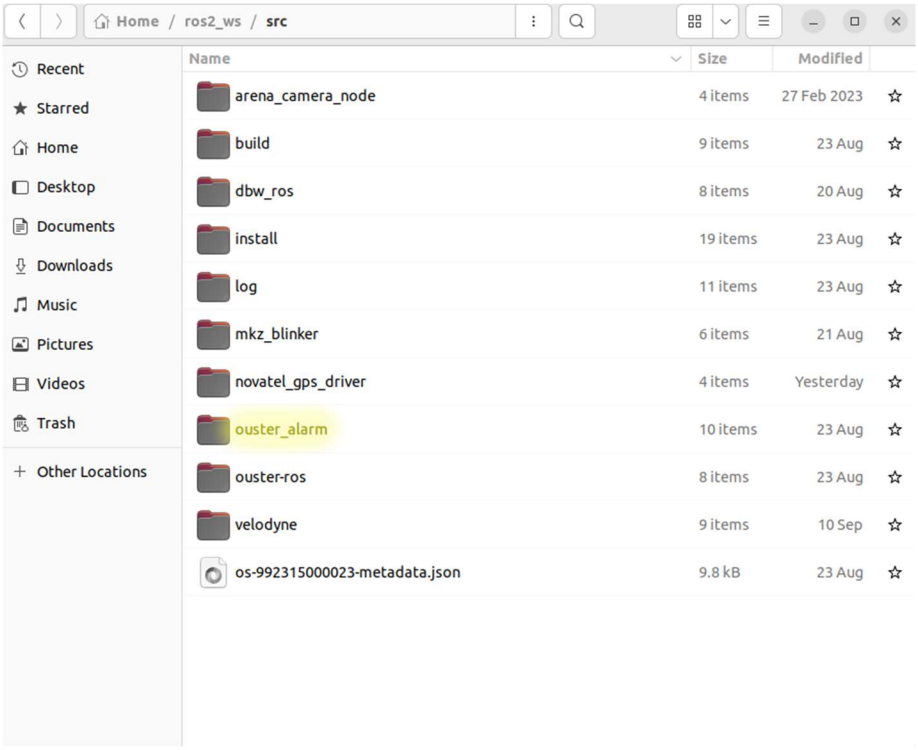


Figure 6: ouster_alarm is in our workspace's src folder

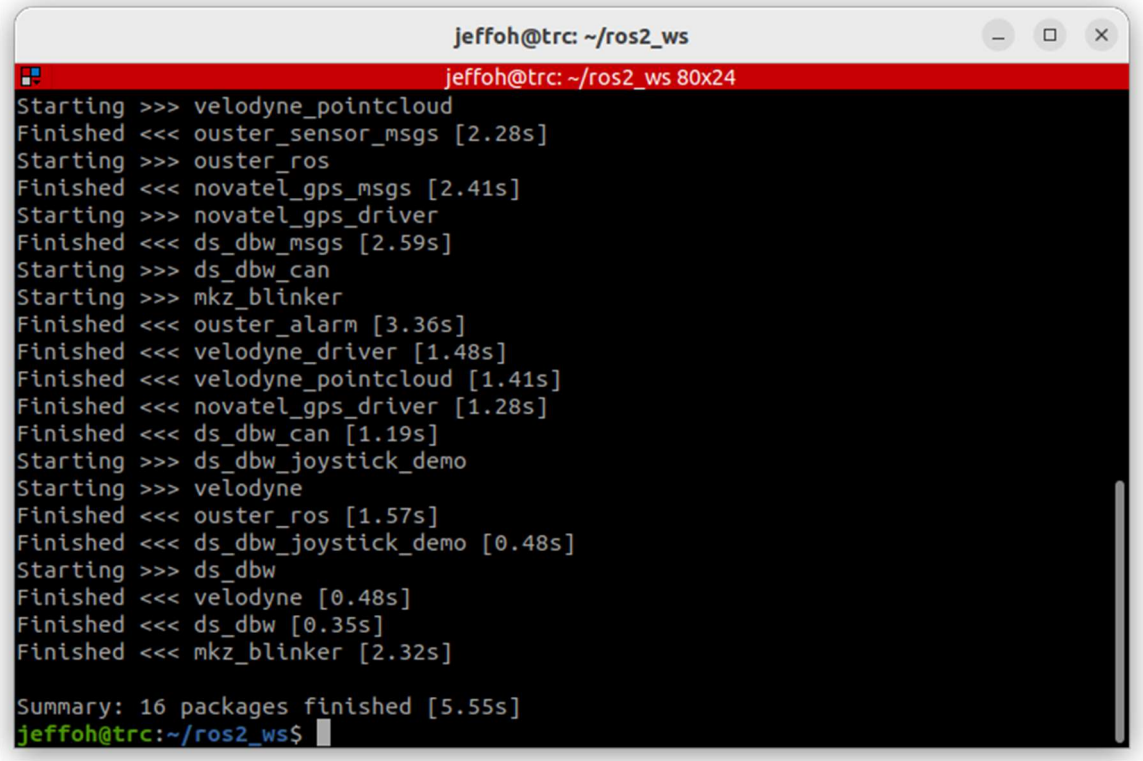


Figure 7: ouster_alarm successfully built

4. Run the Ouster Alarm package.
    a. Place the keys inside the car and press the ENGINE START STOP button without pressing the brake pedal. This will turn the ignition on but not the engine.
    b. (Optional) We will need three terminals for this exercise. Terminator is a terminal emulator that allows you to have multiple terminal sessions open on one window. We will work with Terminator for this and exercises going forwards. Install terminator by opening a new terminal and typing the following command:

```
sudo apt install terminator
```

The new terminals can be opened in the window by right clicking and splitting vertically or horizontally.

    c. In terminal #1 we will launch the Dataspeed DBW package as we did before in Hello World I.

```
ros2 launch dbw_ford_can dbw.launch.xml        # Terminal 1
```

In terminal #2 we will enable DBW and check that it is enabled.

```
ros2 topic pub --once /vehicle/enable std_msgs/msg/Empty "{}"    # Terminal 2 enable DBW
ros2 topic echo /vehicle/dbw_enabled --once    # Terminal 2 check that DBW is enabled.
```

In terminal #3 we launch the Ouster Driver. Rviz will open up and show the point cloud visualization once again.

```
ros2 launch ouster_ros sensor.launch.xml  sensor_hostname:=os-992315000023.local
```

In terminal #2 we will launch our ouster_alarm package with the following commands. All four lines can be copy and pasted. The backslash "\" in a Linux terminal can be used to continue the single command line.

```
ros2 launch ouster_alarm alarm.launch.py \
  points_topic:=/ouster/points \
  misc_topic:=/vehicle/misc_cmd \
  turn_signal_mode:=RIGHT
```

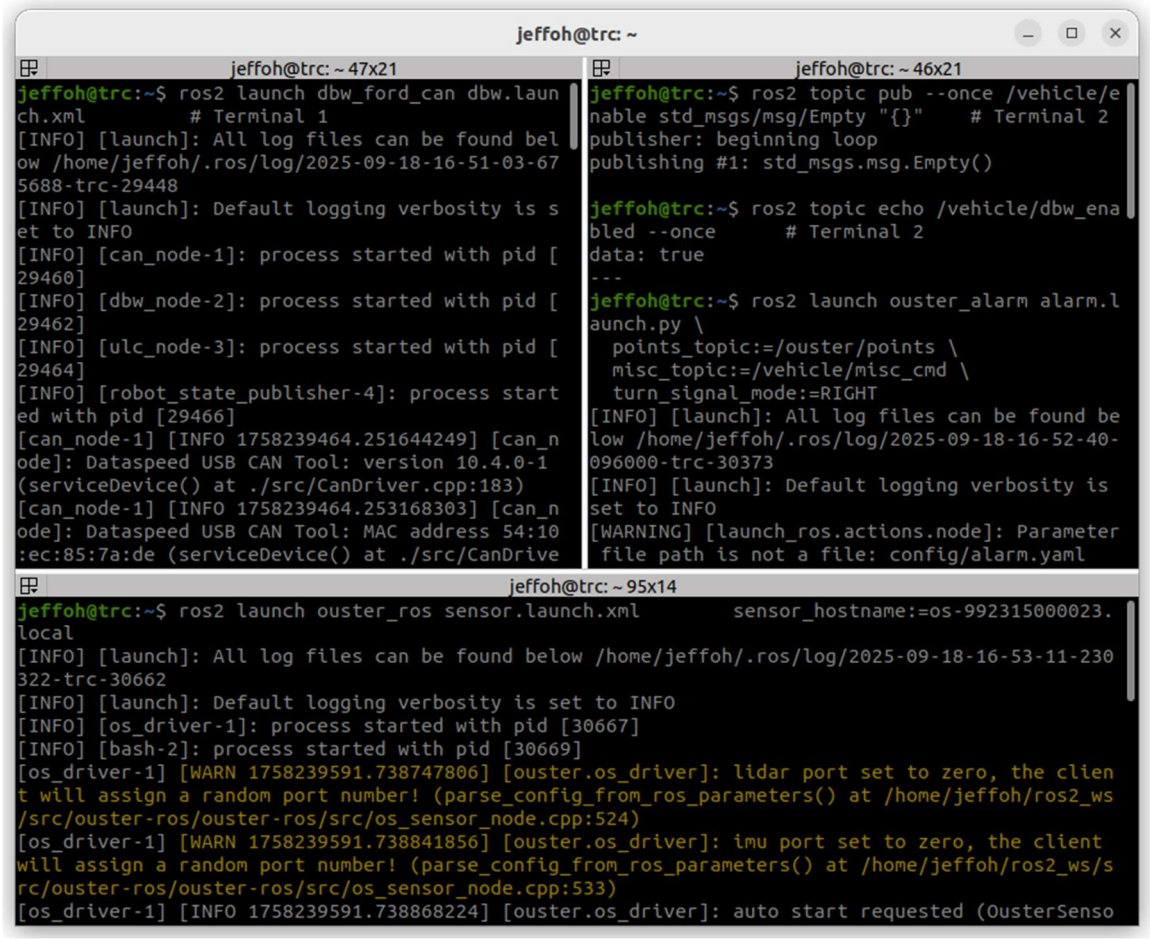


Figure 8: Our launch commands running. Terminal 1-3 starting from the top left going clockwise.

d. Now walk near the vehicle and the Ouster. Once it detects your motion, the right blinker will flash. The Ouster OS2 has a vertical FOV of 22.5 degrees, so you may need to wave your hand up high for it to detect motion. The default radius of detection is set to 5 ft. The result should be similar to this video: ouster_alarm (YouTube)

Summary

We have completed Hello World II and implemented the Ouster OS2 **AND** Dataspeed to create a package that receives an input (motion detected by lidar) and creates an output (blink the right turn signal). The code voxelizes the lidar cloud points and determines changes in voxels between frames, and when the change in frames exceeds a certain threshold, the turn signal activates. We can see from the rqt_graph below that the Ouster topic publishes to our proximity alarm which publishes to our vehicle topic which sends the message to enable the right blinker to the vehicle's computer.

Figure 9: Voxel representation of a pine tree. Our alarm system converts lidar cloud points into voxels.

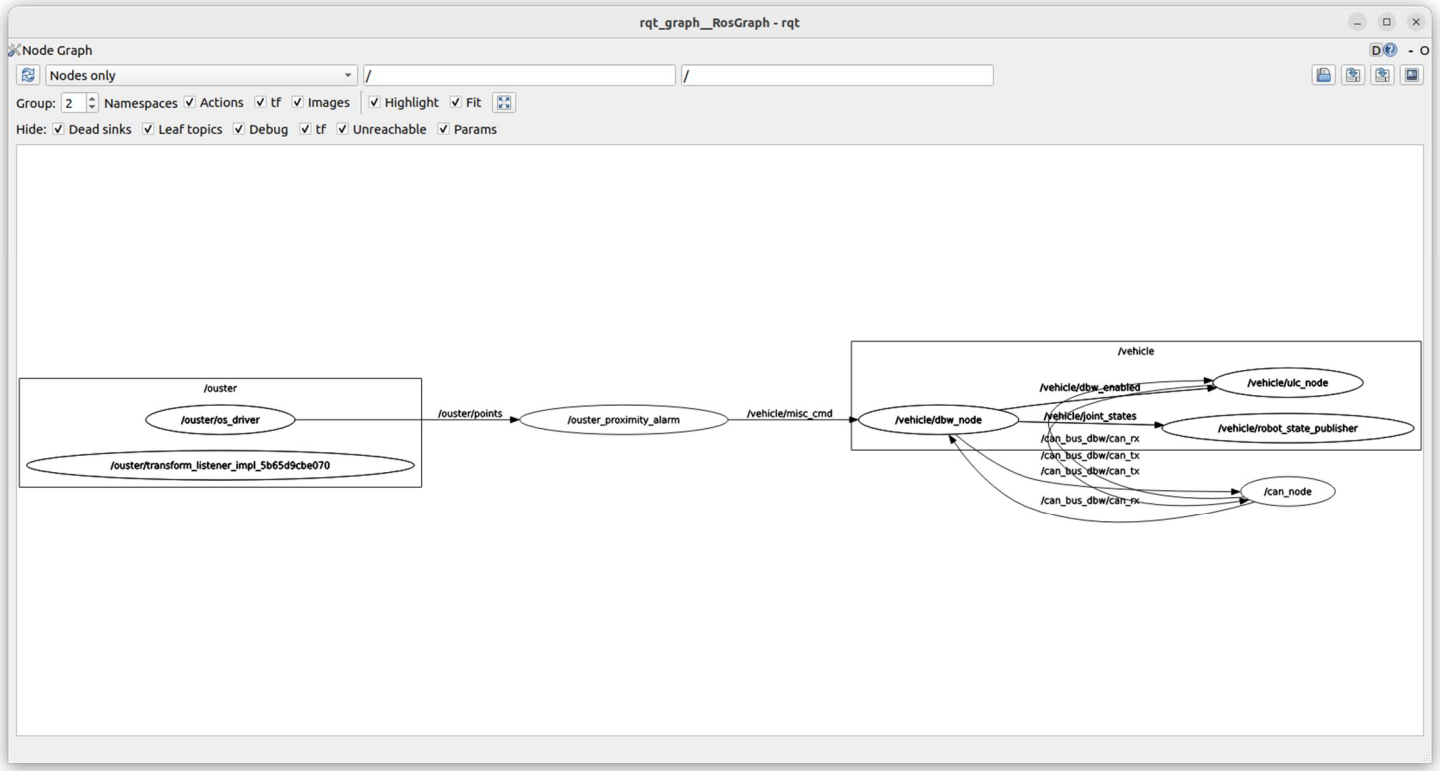


Figure 10: rqt_graph of our Hello World II

```
#!/usr/bin/env python3
# alarm_node.py
import math
import time
from typing import Optional, Set, Tuple

import numpy as np

import rclpy
from rclpy.node import Node
from rclpy.qos import QoSProfile, QoSReliabilityPolicy, QoSHistoryPolicy

from sensor_msgs.msg import PointCloud2
from sensor_msgs_py import point_cloud2 as pc2

# Dataspeed / Ford DBW: use MiscCmd for turn signals on /vehicle/misc_cmd
from dbw_ford_msgs.msg import MiscCmd, TurnSignal, ParkingBrakeCmd


class OusterProximityAlarm(Node):
    """
    Detect motion within a small cylindrical ROI around the LiDAR by comparing
    voxel occupancy between consecutive frames. On motion, publish a turn-signal
    command via dbw_ford_msgs/MiscCmd for a fixed hold duration.

    Assumes the vehicle is stationary. If ego moves, you should compensate using odom/IMU.
    """

    # Logical mapping we use internally; matches common enum values
    ENUM = {'NONE': 0, 'LEFT': 1, 'RIGHT': 2, 'HAZARD': 3}

    def __init__(self) -> None:
        super().__init__('ouster_proximity_alarm')

        # ---------- Parameters ----------
        self.declare_parameter('points_topic', '/ouster/points')
        self.declare_parameter('misc_topic', '/vehicle/misc_cmd')  # where DBW listens
```

Figure 11: alarm_node.py in our ouster_alarm package

alarm_node.py code explanation: Our alarm_node.py code is a node that subscribes to the Ouster OS2's PointCloud2 messages, then the cloud points in a cylinder around the sensor and detects motion by comparing new voxels to old voxels between frames. When the new voxel fraction exceeds our threshold, the DBW turn signal command is published at a fixed rate for a certain duration, then turns the signal off and enforces a short cooldown before retriggering.

```
from launch import LaunchDescription
from launch.actions import DeclareLaunchArgument
from launch.substitutions import LaunchConfiguration
from launch_ros.actions import Node

from ament_index_python.packages import get_package_share_directory
import os

def generate_launch_description():
    pkg_share = get_package_share_directory('ouster_alarm')
    default_params = os.path.join(pkg_share, 'config', 'alarm.yaml')

    return LaunchDescription([
        DeclareLaunchArgument('points_topic', default_value='/ouster/points'),
        DeclareLaunchArgument('misc_topic', default_value='/vehicle/misc_cmd'),
        DeclareLaunchArgument('radius_feet', default_value='5.0'),
        DeclareLaunchArgument('turn_signal_mode', default_value='RIGHT'),

        # optional: allow overriding the params file from CLI
        DeclareLaunchArgument('misc_topic', default_value='/vehicle/misc_cmd'),

        Node(
            package='ouster_alarm',
            executable='alarm_node',
            name='ouster_proximity_alarm',
            parameters=[
                {
                    'points_topic': LaunchConfiguration('points_topic'),
                    'misc_topic': LaunchConfiguration('misc_topic'),
                    'radius_feet': LaunchConfiguration('radius_feet'),
                    'turn_signal_mode': LaunchConfiguration('turn_signal_mode'),
                },
                LaunchConfiguration('params_file'),
            ],
            output='screen',
        ),
    ])
```

Figure 12: Code for our alarm_launch.py file.

alarm_launch.py code explanation: For Hello World II we created a launch file so that we can create launch parameters (the points_topic:=/ouster/points \ misc_topic:=/vehicle/misc_cmd \ turn_signal_mode:=RIGHT in our launch command). The launch file declares command line interface arguments for the lidar points topic, DBW misc topic, detection radius and turn signal mode then starts the ouster_alarm/alarm_node with the declared parameters.