CLUSTERING DATA FROM KITTI LIDAR-DATASET WITH MACHINE LEARNING MODELS

Luis Felipe Tobar
luis.tobar@uao.edu.co
Engineering Faculty.
Universidad Autónoma de Occidente
Cali, Colombia

Abstract — The following article documents the work carried out to group data from point clouds captured by a HDL-64E LIDAR, which is a sensor used in the KITTI-dataset. In the first instance, it is exposed how to visualize the data, either in the Jupyter environment or in the ROS environment, obtaining as a result between this comparison that using Rviz (in ROS) the visualization is much better because of the ease that is has to incorporate new data to the display and that the viewing angle can be changed while the code is running. Then it goes to a preprocessing stage where it is mainly sought to segment the data that are part of the ground of the point cloud, in order for the clustering models to work better. In the same way as before, tests are carried out in the Jupyter and ROS environment. The first environment is used to do rapid visualization tests using the RANSAC and Open3D algorithm for visualization, and in ROS PCL is used to test various segmentation algorithms, obtaining the best segmentation results and computational performance with the ROI filter, Ground Plane Fitting Segmenter and Euclidean Segmenter. For the last stage of clustering with machine learning, three different algorithms are tested in the Jupyter environment: K-means, DBSCAN and HDBSCAN, obtaining the best results with the last two. Finally, in the ROS environment, the LidarPerception repository is used to show an application example of clustering, using the preprocessing done in the previous stage.

Keywords – Clustering, LIDAR, KITTI-dataset, Jupyter, ROS, Rviz, RANSAC, Open3D, PCL, Segmentation, Machine Learning, K-means, DBSCAN, HDBSCAN.

I. PROBLEM STATEMENT

Throughout the years, autonomous driving has been the focus of attention due to its great advances in recent years and for this reason it has mainly interested sectors such as education, research and even the industrial sector. This branch of engineering has become of great interest due to its promise of safer and more efficient driving, as it seeks to reduce the main source of accidents that occur: human error.

The World Health Organization (WHO) gave in 2020 the following figures that denote the importance of safe driving[1]:

- "Approximately 1.35 million people die each year as a result of road traffic crashes".
- "Road traffic crashes cost most countries 3% of their gross domestic product".
- "93% of the world's fatalities on the roads occur in low- and middle-income countries, even though these countries have approximately 60% of the world's vehicles".

Based on the above, a list of the most common causes of accidents is also presented, which strengthens the premise of the first paragraph:

- "An increase in average speed is directly related both to the likelihood of a crash occurring and to the severity of the consequences of the crash. For example, every 1% increase in mean speed produces a 4% increase in the fatal crash risk and a 3% increase".
- "Drivers using mobile phones are approximately 4 times more likely to be involved in a crash than drivers not using a mobile phone. Using a phone while driving slows reaction times (notably braking reaction time, but also reaction to traffic signals), and makes it difficult to keep in the correct lane, and to keep the correct following distances".

As exemplified above, much of the causes of accidents are due to driver errors, whether due to reckless driving, distractions while driving or errors in making decisions.That is why autonomous driving proposes a very striking technological solution that not only seeks to reduce all these figures, but also solves other issues such as facilitating the transport of specific social groups such as the elderly or people with reduced mobility.

Now, as it was said at the beginning, today autonomous driving can be seen in various sectors, but in which there was always great interest in this topic was in education and research. An example is the large number of projects with multiple applications throughout educational institutions or research companies that exist at a global level, where its use is seen from education, such as the DuckieTown foundation[2] or the Turtlebots by ROS-components[3]. In addition, in a short time research and industry have come together to make great advances in the subject, examples being large companies such as Tesla, BMW, Mercedes-benz, etc. and more recently Lucid motors.

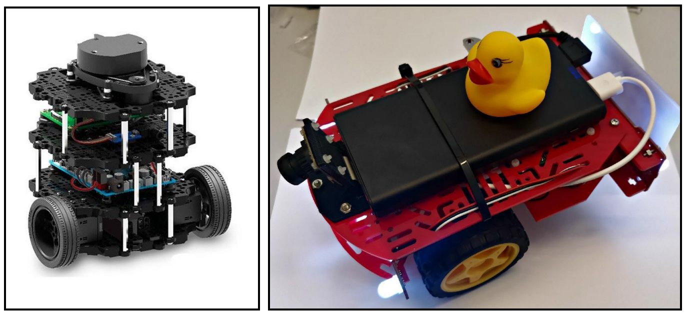


Figure 1: Robotic mobile platforms "Turtlebot3" and "Duckiebot".

[1] Road traffic injuries (2020). World Health Organization. Available online: https://www.who.int/es/news-room/fact-sheets/detail/road-traffic-injuries.

[2] The Duckietown Foundation. Available online: https://www.duckietown.org/about/duckietown-foundation.

[3] Mobile Robots. ROS components. Available online: https://www.roscomponents.com/es/12-robots-moviles.

Figure 2: Ford and Baidu autonomous vehicles.

Now that it has been shown that there is really a great interest in autonomous driving, it must be understood how important it is that these cars can recognize the obstacles that are around them.Chen, Ma, and Wan explain it very well, saying that "3D object detection plays an important role in the visual perception system of Autonomous driving cars. Modern self-driving cars are commonly equipped with multiple sensors, such as LIDAR and cameras. Laser scanners have the advantage of accurate depth information while cameras preserve much more detailed semantic information. The fusion of LIDAR point cloud and RGB images should be able to achieve higher performance and safety to self-driving cars."[4]

Based on the above information, it wanted to carry out an approach with machine learning that allows a system to be able to recognize the objects around it. This system is specifically a mobile robot called "Chimuelo" that is part of the robots that the Robotics and Autonomous Systems seedbed of the Universidad Autónoma de Occidente has. The objective with said robot is to repower it to be an autonomous guide robot. Until now, it has a robotic perception system that allows it to be located in closed environments (with an RGBD camera)[5] but it is desired to project to an environment outside, so it is necessary to migrate to another sensor such as the LIDAR that is named in the text by Chen, Ma and Wan.

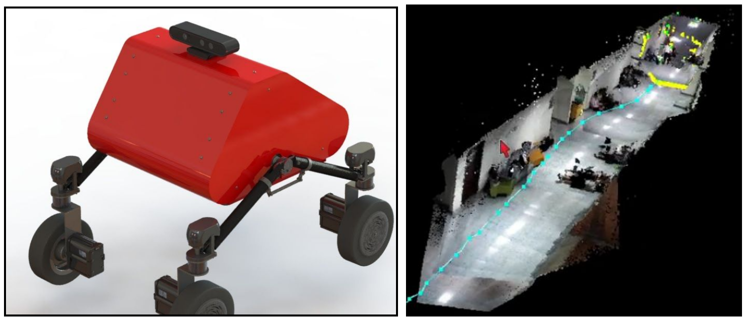


Figure 3: Previous work on the robot "Chimuelo".

It is wanted to migrate to the use of a LIDAR sensor, this work seeks to be a first test to start processing point clouds, seeking to be able to group several of these points to determine whether or not they belong to the same object and thus begin to build a new perception system for the robot.

Due to the fact that there is not yet an own dataset with which these first tests can be carried out, a dataset that has LIDAR information must be used and thus be able to apply Machine Learning algorithms to start with the tests.

II. APPROACH TO THE SOLUTION

Dataset explanation:

As previously mentioned, a dataset was going to be sought that would allow testing with information obtained from a LIDAR sensor (point cloud data). For this, the use of the KITTI dataset was proposed, which is a database provided by the Karlsruhe Institute of Technology and Toyota Technological Institute at Chicago. This dataset has information from various sensors, including monocular cameras, IMU, GPS, stereo camera, and LIDAR.

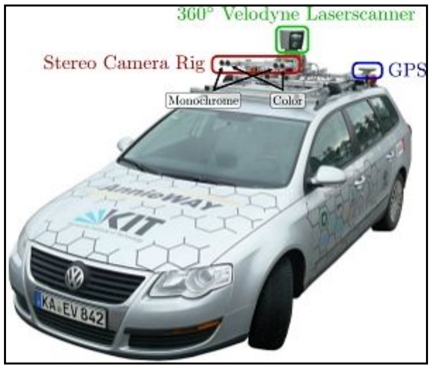


Figure 4: Mobile platform used to make the KITTI dataset.

For the case of this work, the information provided by the Velodyne HDL-64E LIDAR sensor is used which "is designed for obstacle detection and navigation of autonomous ground vehicles and marine vessels"[6], has "64 channels, 120m of range, 360° Horizontal FOV and 26.9° Vertical FOV".[6]

"For efficiency, the Velodyne scans are stored as floating point binaries… Each point is stored with its (x = forward, y = left, z = up) coordinate and an additional reflectance value (r). While the number of points per scan is not constant, on average each file/frame has a size of ~ 1.9 MB which corresponds to ~ 120, 000 3D points and reflectance values. Note that the Velodyne laser scanner rotates continuously around its vertical axis (counter-clockwise), which can be taken into account using the timestamp files".[7]

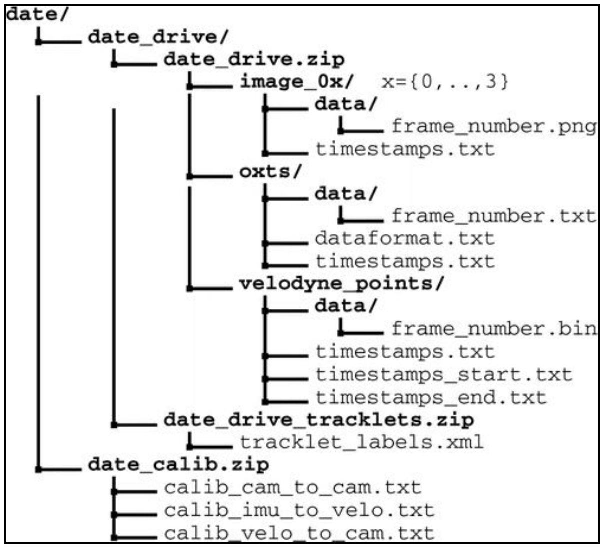
The dataset can be downloaded directly from the KITTI official page, where you can download the information prepared to be able to work with it, which is arranged in the following way:

[4] Chen, X., Ma, H., Wan, J., Li, B., & Xia, T. (2017). Multi-view 3d object detection network for autonomous driving. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (pp. 1907-1915).

[5] Rangel, S., Tobar, L. F., & Escobar, D. A. (2020). Perception system for the vehicle land exploration "Chimuelo" focused on orientation works in the basement laboratories 2 of the Universidad Autónoma de Occidente. Available online on Researchgate.

[6] HDL-64E. Velodyne Lidar. Available online: https://velodynelidar.com/products/hdl-64e/.

[7] Geiger, A., Lenz, P., Stiller, C., & Urtasun, R. (2013). Vision meets robotics: The kitti dataset. The International Journal of Robotics Research, 32(11), 1231-1237.



```
date/
└── date_drive/
    └── date_drive.zip
    │   └── image_0x/   x={0,..,3}
    │   │   └── data/
    │   │   │   └── frame_number.png
    │   │   └── timestamps.txt
    │   └── oxts/
    │   │   └── data/
    │   │   │   └── frame_number.txt
    │   │   └── dataformat.txt
    │   │   └── timestamps.txt
    │   └── velodyne_points/
    │       └── data/
    │       │   └── frame_number.bin
    │       └── timestamps.txt
    │       └── timestamps_start.txt
    │       └── timestamps_end.txt
    └── date_drive_tracklets.zip
        └── tracklet_labels.xml
└── date_calib.zip
    └── calib_cam_to_cam.txt
    └── calib_imu_to_velo.txt
    └── calib_velo_to_cam.txt
```

Figure 5: "Structure of the provided zip files and their location within a global file structure that stores all KITTI sequences" . [7]

Development process:

When you download the data from the velodyne and unzip it, you get two folders: training and testing.

Figure 6: Folders obtained by downloading and unzipping the Velodyne point clouds file from the 3D Object Detection section of the KITTI website.

The way they are organized lends itself to implementing a development process like the following:

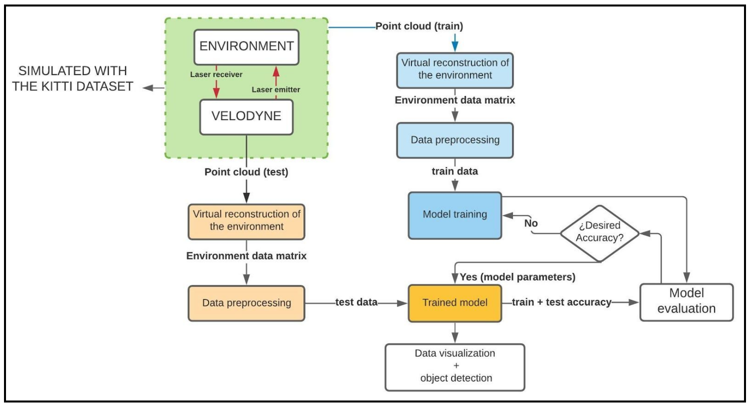


Figure 7: Schematic diagram of the solution process for a supervised learning object detector (See annexes).

In the case of doing supervised learning, the previous diagram would be the sequence by which this project would be done, where the dataset is separated into a test set and another for train, but both are used by the algorithm to determine an accuracy, giving results of how is the performance when generalizing for new data. This way of posing the problem would be fine if what you wanted to do was detection of objects, where you already know the objects of interest to be identified, but when clustering you want the same algorithm to identify the different objects in the dataset, without having specific objects that you want to obtain. Due to the above, the following diagram is proposed that illustrates the process to follow to clustering with unsupervised algorithms:

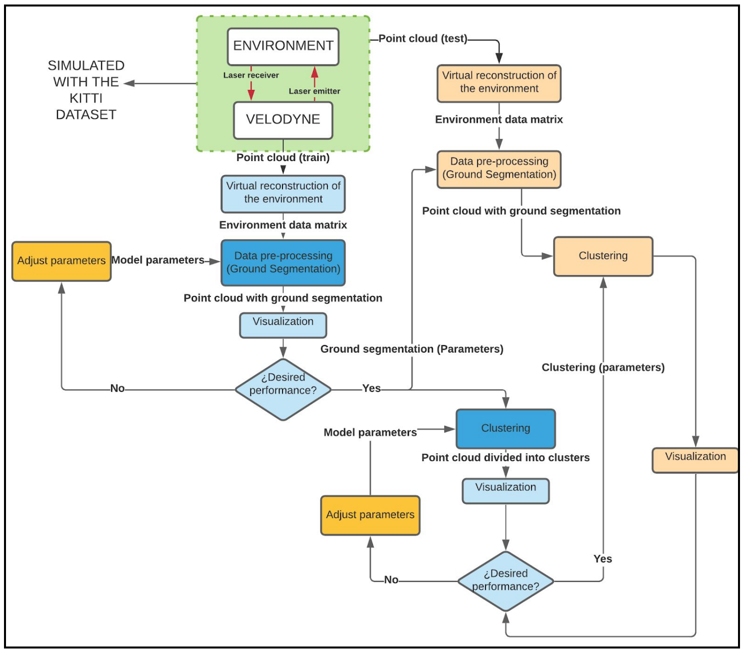

SIMULATED WITH THE KITTI DATASET
ENVIRONMENT
Laser receiver
Laser emitter
VELODYNE
Point cloud (test)
Virtual reconstruction of the environment
Environment data matrix
Data pre-processing (Ground Segmentation)
Point cloud with ground segmentation
Clustering
Point cloud (train)
Virtual reconstruction of the environment
Environment data matrix
Adjust parameters
Model parameters
Data pre-processing (Ground Segmentation)
Point cloud with ground segmentation
Visualization
Ground segmentation (Parameters)
No
¿Desired performance?
Yes
Clustering (parameters)
Model parameters
Clustering
Point cloud divided into clusters
Visualization
Adjust parameters
Visualization
No
¿Desired performance?
Yes

Figure 8: Schematic diagram of the solution process for using unsupervised models for clustering (See annexes).

The process seen in the previous figure consists of using a data set which will be used to do the preliminary tests. As already explained, the obtaining of the data from the Kitti dataset will be simulated. These data come in binary format, so they will initially be displayed, obtaining a data matrix. Once we have a method to visualize the data, we proceed to implement segmentation methods to be able to eliminate the ground from the point cloud (this because it makes the clustering process difficult with the models to be implemented). For each algorithm that will be tested to segment the soil, several visualizations will be made to verify its performance, and from the observed results the parameters will be changed until the best ones are obtained.

Once the pre-processing phase is in place, the point cloud with the segmented soil will be used as the input data for the clustering models. The process to follow with these unsupervised models will be similar to the one that was done with the pre-processing phase, since the parameters of each one will be modified until the best results are obtained.

Once you have the algorithms to segment the soil and to do clustering, you must proceed to use new data from the dataset to validate your results with new data with which it has not been tested before, and from which this can be modified or adjusted parameters again.

III.EXPERIMENTS

[1] DATA VISUALIZATION:

In order to make a virtual reconstruction of the data that was captured by the LIDAR, it is necessary to obtain the data matrix of the environment. This can be obtained relatively easily by processing the binary data downloaded from the Kitti page with C ++ or Matlab. The drawback with this is that you want to work in the future with this data with models with free software architecture, so it was proposed to perform the visualization in two different ways: Performing a python script with Jupyter Lab and another visualization in the ROS (Robot Operating System) middleware environment.

Visualization with Python in Jupyter Lab:

This visualization was based on Alex Staravoitau's repository: "KITTI-Dataset".

For this test, a virtual environment was made with the installation of all the libraries that the repository needs, like this:

```
lenovo@lenovo-felipet:~$ virtualenv --system-site-packages -p python3
./kittiv_
```

Once the virtual environment is activated, you need to install pykitti, and it is recommended that it be through pip, with "pip install pykitti".

An example with the file 2011_09_26_drive_000 is shown in the repository, but this can be any other of the files that can be downloaded from Kitti's page. The easiest way to obtain the code is with the command "git clone" and then the address of the repository.

To order the folder with the project, a folder called **data** and another called **video** must be created, where the data obtained by the LIDAR and the frames for the gif that will be made to display the data respectively will be saved. (In the **data** folder you must unzip the files "synced + rectified_data", "calibration" and "tracklets" of the extract of the desired kitti dataset).

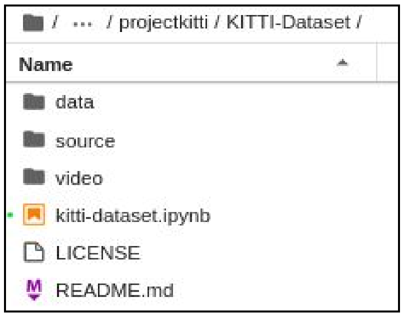

```
/ … / projectkitti / KITTI-Dataset /
Name
data
source
video
kitti-dataset.ipynb
LICENSE
README.md
```

Figure 9: Organization of the folder with the files to visualize with python in Jupyter Lab.

In the first code cell, the address of the folder containing the data is specified in the basedir variable, which in this case is called **data** and the pykitti .raw function is used to read the data.

The next cell defines the function that allows loading the tracklets for each frame. In the following cells, variables are specified with the name of the kitti file that was downloaded and a graph is made with a trimetric, front, top and side view:

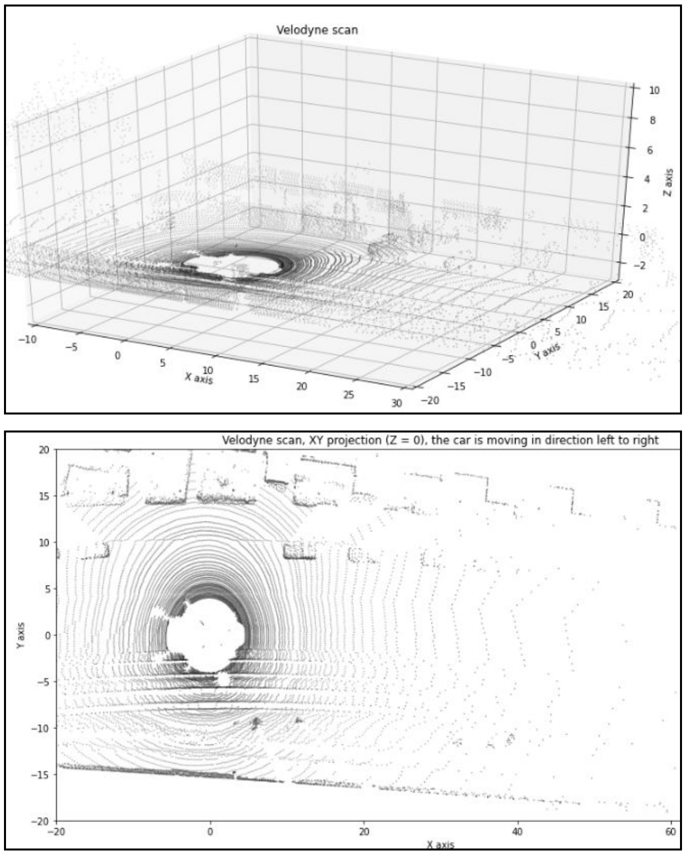


Figure 10: Example of plots with trimetric and top view (See annexes).

(One of the changes made in the code was to comment on the lines of code that generate the bounding boxes that the raw kitti dataset already provides, since one of the objectives in the work is to be able to generate them).

Finally, in order to see all the data taken in a range of time as a sequence and not only as a graph in a single instant of time, the **moviepy** library was used (previously installed with "pip install moviepy" in the virtual environment). This code cell allows you to configure the configuration of how the point cloud will look from the points variable, since the other variables such as **points_step**, **points_size**, **velo_range** and **velo_frame** depend on it. In addition, the name of the folder where all the frames will be saved is specified in the filename (**video**) variable and then "frame_ {0: 0> 4}" to name each one consecutively.

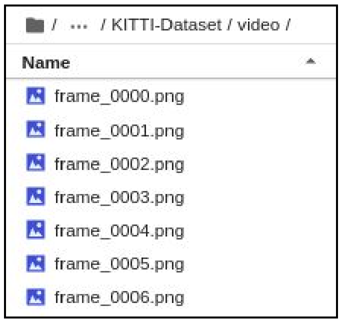

```
/ … / KITTI-Dataset / video /
Name
frame_0000.png
frame_0001.png
frame_0002.png
frame_0003.png
frame_0004.png
frame_0005.png
frame_0006.png
```

Figure 11: Video folder with the frames generated with moviepy.

The code finally concatenates all the frames and returns them a gif in which the sequence can be seen with the following line of code:

```
clip = ImageSequenceClip(frames, fps=5)
% time
clip.write_gif('pcl_data.gif', fps=5)
```

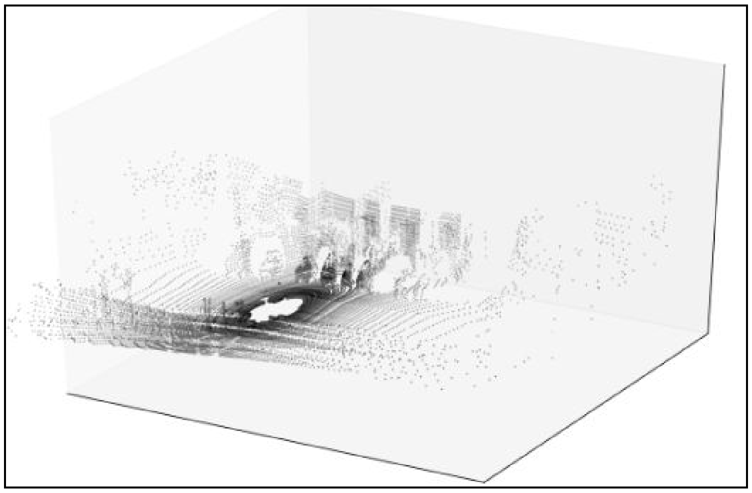


Figure 12: Frame 39 of the gif obtained (See annexes).

Visualization with Rviz and ROS:

Since the previous form of visualization is limited to having the same angle of vision and it is not very efficient when executing, we proceeded to use the visualization program Rviz that is part of the ROS environment.

The first problem to solve is that the encoding as binary files that have the data when downloaded from the KITTI page are not directly interpretable in ROS, for which it is necessary in the first instance to convert these files to a compatible format. Based on the above, the Tomas Krejci repository on github was used, which is called "kitt2bag". This repository is responsible for transforming the binary files to the rosbag file format, which can be played later as long as the Master Uri is running.

In order to convert the files, the same download and unzip process that was done with the downloaded files must be carried out to work in Jupyter Lab and make the git clone of the Krejci repository. In an environment where pykitti and ROS are installed, the "pip install kitti2bag" installation is carried out and the command kitty2bag -r arg1 -t arg2 raw_synced is executed, where in argument 1 goes the name of the folder where the data is and in argument 2 the name of the new document .bag, like this:

```
(kittiv) lenovo@lenovo-felipet:~/kittiv/projectkitti/KITTI-Dataset/data
$ kitti2bag -t 2011_09_26 -r 0002 raw_synced_
```

The previous process was carried out for several files downloaded from KITTI and thus have several samples or data sets with which to apply preprocessing and models:

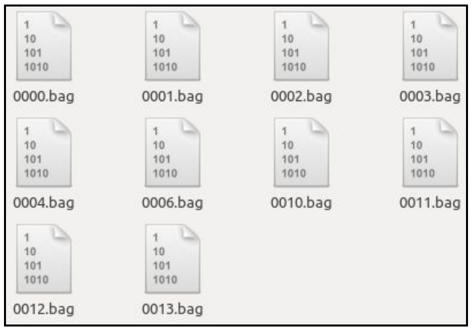

0000.bag 0001.bag 0002.bag 0003.bag
0004.bag 0006.bag 0010.bag 0011.bag
0012.bag 0013.bag

Figure 13: Files processed with kitti2bag.

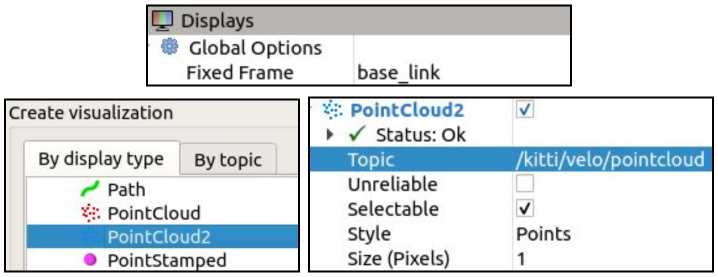
Finally, you must do "catkin_make" in the workspace you are working on and "source devel / setup.bash", then run the Master Uri with "roscore" and another terminal execute "rosbag play -l [path]", like this:

```
lenovo@lenovo-felipet:~/catkin_w$ source devel/setup.bash
lenovo@lenovo-felipet:~/catkin_w$ roscore
```

```
lenovo@lenovo-felipet:~/catkin_w$ rosbag play -l
src/data_lidar/0001.bag_
```

Once this is done, you will see the terminal where the rosbag that starts publishing the data was executed. Because the -l command was used after rosbag play, the sending of the data will be repeated cyclically. Now, to be able to do the visualization, you must open Rviz (in another terminal with the command "rviz") and proceed to make the following settings:

```
lenovo@lenovo-felipet:~/catkin_w$ rviz_
```


Displays
Global Options
Fixed Frame base_link

Create visualization
By display type | By topic
Path
PointCloud
PointCloud2
PointStamped

PointCloud2 ✓
Status: Ok
Topic /kitti/velo/pointcloud
Unreliable ☐
Selectable ✓
Style Points
Size (Pixels) 1

Figure 14: Recommended settings for viewing in Rviz.

Finally, to better understand how this configuration is working, the command "rqt_graph" can be executed in another terminal, obtaining the following:

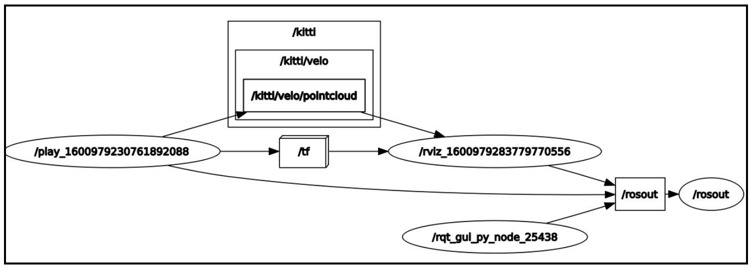


Figure 15: Graph of nodes and active topics generated with rqt_graph for visualization (See annexes).

As can be seen in the previous graph, when running the rosbag, the information to be displayed begins to be published in the topic / kitti / velo / pointcloud (the transform tree is also published). Rviz subscribes to the two topics (which is what is done in the Topic configuration in PointsCloud2 and in Fixed frame).

Finally, with this visualization method, the dataset can be seen in a more comfortable way, since the angle of view can be changed while the visualization is running and allows it to detail better.

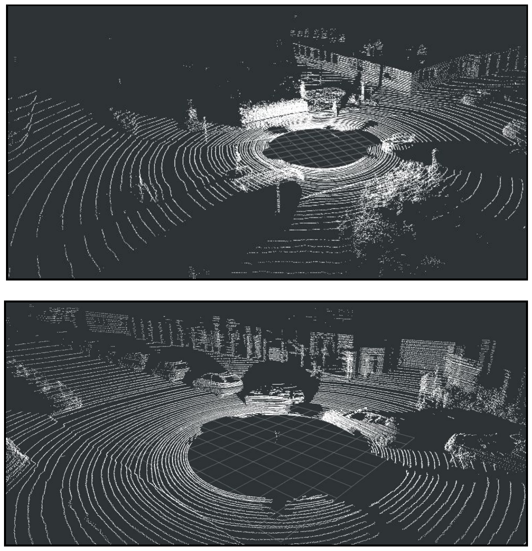


Figure 16: Visualization of the point cloud in Rviz (See annexes).

[2] <u>DATA PRE-PROCESSING:</u>

From the results obtained in the visualization stage, it is proposed to do tests with several models in the Jupyter Lab environment since the way in which you can work in this environment will be more comfortable to be able to evaluate models, but on the other hand wants to show a pre-processing performed in the ROS environment in order to show the great advantage and computational efficiency that this middleware has.

<u>Pre-processing with Python and Jupyter Lab:</u>

Following the process that was carried out with the visualization in the Jupyter environment with Python, it is proposed to carry out a simple preprocessing stage, leaving the tests with various segmentation methods for what it will do in ROS. In this way, the processing that is being carried out is to be able to segment the ground from the other objects, since this favors the accuracy obtained with the models for later clustering.

To perform this segmentation, two functions performed by Ma Nongjiayuan (available in codenong) were used. The first function is to perform Plane Least Square (function required to use the next one), and the next is the implementation of a function to use RANSAC (Random sample consensus) plane segmentation. As Pablo Flores and Juan Braun say, RANSAC "is an iterative algorithm used to estimate the parameters of a mathematical model of a data set that contains (outliers). Other methods to estimate model parameters such as least squares give the same weight to all data, therefore the presence of an outlier can distort the model obtained. This algorithm was used to estimate a homography from corresponding points obtained by algorithms for feature detection."[8]

"From the algorithm there are three parameters to estimate: the distance t to consider whether a point is compatible or not with the model, the number of iterations to search for decision subsets, and the threshold T which is the minimum number of compatible points to consider a good estimation of the model." [8]

In this way the implemented function is added (as an argument) the input data and the parameters named above:

```
def PlaneRANSAC(X:np.ndarray,tao:float,e=0.4,N_regular=100):
```

(The complete code can be seen in the project files).

Now, in order to make a comparison when applying RANSAC, the Open3D library is used, which allows adjusting the viewing angle and zoom to see the results in more detail.

On the official page of the **Open3D** library, various functions are explained and examples are given. In this case the most important are the following:

```
pcd = open3d.geometry.PointCloud()

pcd.points = open3d.utility.Vector3dVector(origindata)
open3d.visualization.draw_geometries([pcd])
```

The first line is used to define that the class to work with is a PointCloud type geometry. Then this variable must be taken to a Vector3dVector type to be able to be read when drawing, and finally the visualization.draw_geometries is used to make the visualization in a pop-up window.

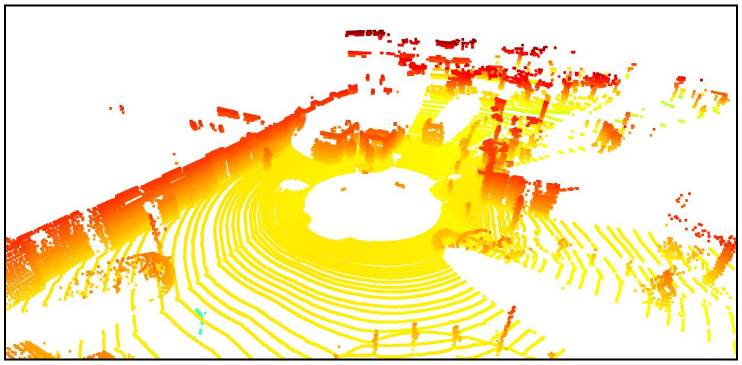


Figure 17: Visualization of the point cloud in Jupyter Lab with Open3D from the original data.

In the RANSAC plane function it is established that all the points belonging to the ground are graphed in blue, so after applying this segmentation this visualization is obtained with Open3D:

[8] Flores, Pablo. Braun Juan. Algoritmo RANSAC: fundamento teórico (2011). Available online: http://iie.fing.edu.uy/investigacion/grupos/gti/timag/trabajos/2011/keypoints/FundamentoRANSAC.pdf

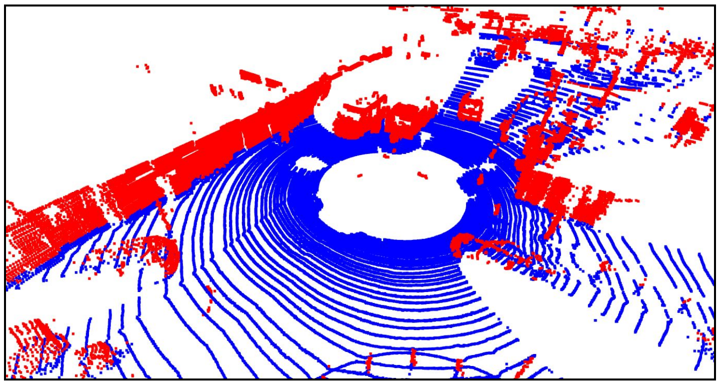


Figure 18: Visualization of the point cloud with the segmentation of the ground in blue with RANSAC.

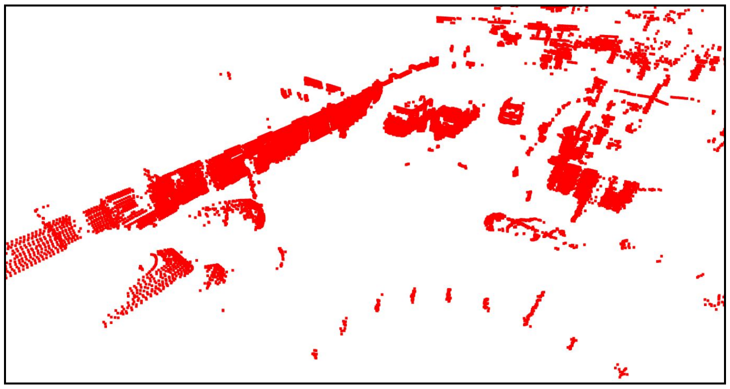


Figure 19: Visualization of the point cloud without the ground.

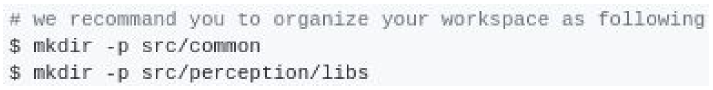
Pre-processing with PCL and ROS enviroment:

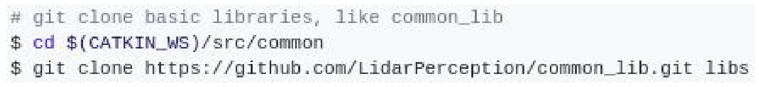
As could be seen in the visualizations both in python with Jupyter Lab and with Rviz with ROS, the information available provides a fairly dense point cloud, which can make the process of future models much slower due to having to process so much information. For this reason, it is necessary to reduce the information that the point cloud has, but not in any way because the information that is valuable can be lost. For this process it is proposed to use the Point Cloud Library (PCL), which as its official website says is “a standalone, large scale, open project for 2D/3D image and point cloud processing. PCL is released under the terms of the BSD license, and thus free for commercial and research use.”[9]

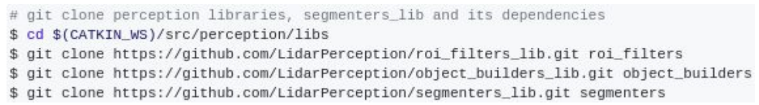
For this task, it is proposed to work on Gary Chan's github repository (LidarPerception), which also already has a model to segment and detect objects, but for this stage it will be used to filter the point cloud. The advantage of this repository is that it has the integration of PCL in the ROS environment, which as seen in the previous section (the display) is a much better option.Another advantage of the repository is that it already brings other libraries necessary to run it, which makes it ideal for the pre-processing task.

Now, it is recommended to follow the steps suggested by the repository to clone it (the following must be in the workspace where the .bag files are):

[9] PCL API Documentation. Point Cloud Library. Available online:https://pointclouds.org/documentation/index.html

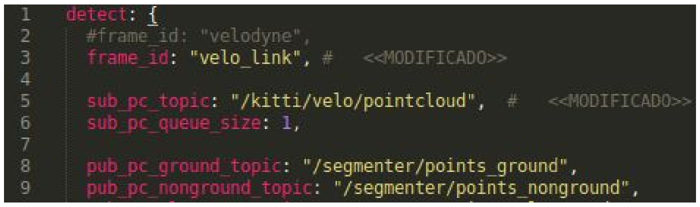

```
# we recommand you to organize your workspace as following
$ mkdir -p src/common
$ mkdir -p src/perception/libs
```


```
# git clone basic libraries, like common_lib
$ cd $(CATKIN_WS)/src/common
$ git clone https://github.com/LidarPerception/common_lib.git libs
```


```
# git clone perception libraries, segmenters_lib and its dependencies
$ cd $(CATKIN_WS)/src/perception/libs
$ git clone https://github.com/LidarPerception/roi_filters_lib.git roi_filters
$ git clone https://github.com/LidarPerception/object_builders_lib.git object_builders
$ git clone https://github.com/LidarPerception/segmenters_lib.git segmenters
```

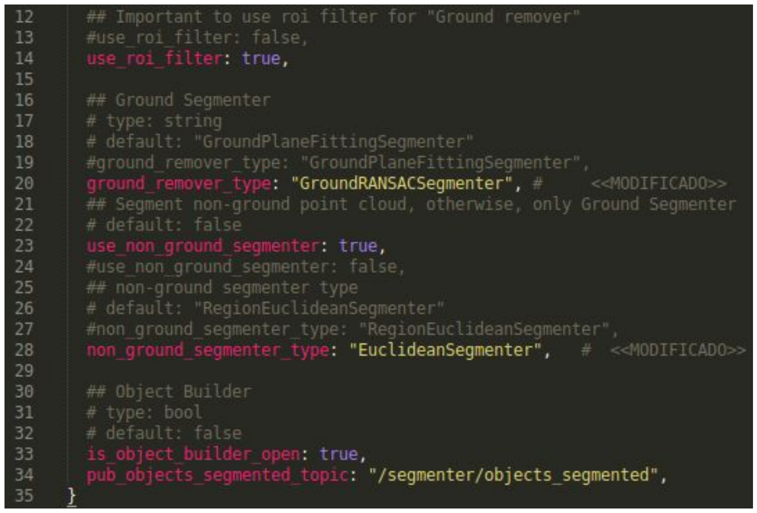
The second item that is suggested in the repository is not necessary to do so, since the point cloud information is being supplied by rosbag.

The first file that is counted to work is the **detection.yaml**, where in the first instance the topics to which we are going to subscribe are configured:


```
detect: {
  #frame_id: "velodyne",
  frame_id: "velo_link", #   <<MODIFICADO>>

  sub_pc_topic: "/kitti/velo/pointcloud",  #   <<MODIFICADO>>
  sub_pc_queue_size: 1,

  pub_pc_ground_topic: "/segmenter/points_ground",
  pub_pc_nonground_topic: "/segmenter/points_nonground",
```

As seen in the previous code extract, the variable frame_id is modified, in which it must be the same as that set in the Fixed frame in the display section with Rviz. In addition, you must indicate the topic to which you are going to subscribe to obtain the point cloud, which is the same that was set in the Topic variable of the PointCloud2 of the visualization. The last values of the topics where the processed point clouds are published are left as they are, but must be remembered for future use.

The next part of the code of the code is the one that we will work with, testing different methods to do the pre-processing of the point cloud, since you can select between different segmentation methods, trying to obtain a good result but that in turn is computationally efficient.


```
  ## Important to use roi filter for "Ground remover"
  #use_roi_filter: false,
  use_roi_filter: true,

  ## Ground Segmenter
  # type: string
  # default: "GroundPlaneFittingSegmenter"
  #ground_remover_type: "GroundPlaneFittingSegmenter",
  ground_remover_type: "GroundRANSACSegmenter", #     <<MODIFICADO>>
  ## Segment non-ground point cloud, otherwise, only Ground Segmenter
  # default: false
  use_non_ground_segmenter: true,
  #use_non_ground_segmenter: false,
  ## non-ground segmenter type
  # default: "RegionEuclideanSegmenter"
  #non_ground_segmenter_type: "RegionEuclideanSegmenter",
  non_ground_segmenter_type: "EuclideanSegmenter",  #  <<MODIFICADO>>

  ## Object Builder
  # type: bool
  # default: false
  is_object_builder_open: true,
  pub_objects_segmented_topic: "/segmenter/objects_segmented",
}
```

In the previous code you can see how you can comment or uncomment some variables to work with different segmentation methods, which are the ones that will be varied to test which ones get better results.

In addition to modifying the previous file, it will also be necessary to modify the parameters of each segmentation method. For this, the **Segmenter.yaml** file will be modified, where the parameters of the "roi_filter", "GroundPlaneFittingSegmenter", "GroundRANSACSegmenter", "RegionEuclideanSegmenter" and "EuclideanSegmenter" are located, which will be modified. (*See sample code for the parameters in the annexes*).

TEST 1: Default parameters (Ground Plane Fitting Segmenter and Region Euclidean Segmenter).

Before starting to change the parameters, it will be tested with the default values suggested by the repository, and it will be seen how well it works on the computer where the tests are developed (the values are directly subject to the processing of the machine with which is being worked on, and they will not be the same for all).

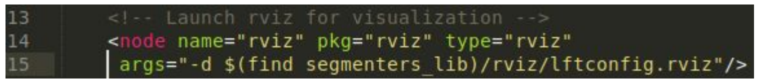
The default values for the detection.yaml file are to activate the **roi_filter** to remove the ground, use the **GroundPlaneFittingSegmenter** and activate the **non_ground_segmenter** with the **RegionEuclideanSegmenter**.

The default values of the Segmenter.yaml file can be seen in the annexes, but it consists of using the roi type as a Cylinder with variables **roi_radius_min_m: 2.** and **roi_radius_max_m: 120.** The ROI filter means "Region of Interest" and it is important because the LIDAR used has a range of 120 meters, which means that we have a lot of point information, which is why we seek to reduce this point range. Therefore, a circular region of interest is established with a minimum and maximum radius.

In the **GroundPlaneFittingSegmenter**, what is sought is to estimate the lowest representative point, the more points are taken into account, the better the result, but it also requires much more computational processing, as well as when choosing a threshold so that the points are considered initial seeds . The above parameters are **gpf_num_lpr: 128,** and **gpf_th_seeds: 0.1 ,**.

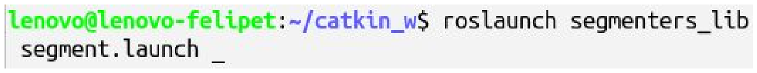
Now, the parameters for the RegionEuclideanSegmenter consist of adjusting the tolerance to the Euclidean distance between adjacent regions. The size of each region can be modified (being 14 by default), also the number of points used for each cluster and you can choose if you want to merge the IoU threshold of overlap of the corresponding ground box. The parameters for the above are **rec_region_size: 14**, **rec_region_initial_tolerance: 0.2**, **rec_region_delta_tolerance: 0.1**, **rec_min_cluster_size: 20**, **rec_max_cluster_size: 30000**, **rec_use_region_merge: true** and **rec_region_merge_tolerance: 4.0**.

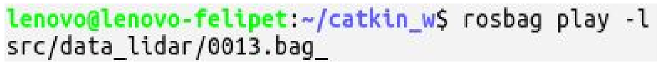
Next, proceed to execute the launch called **segment.launch**, which has a variation with respect to the demo.launch provided by the repository, and it is that a custom rviz configuration is established at the time of launching the launch to avoid be configuring rviz as it was done in the visualization stage.


```
    <!-- Launch rviz for visualization -->
    <node name="rviz" pkg="rviz" type="rviz"
     args="-d $(find segmenters_lib)/rviz/lftconfig.rviz"/>
```

The configuration file **lftconfig.rviz** allows you to view the segmentation that you made of the ground, the point cloud where the ground is filtered and finally the segmentation to a range of the samples closest to the sensor.

Finally, you must execute the segment.launch and the rosbag with one of the .bag that you have:


```
lenovo@lenovo-felipet:~/catkin_w$ roslaunch segmenters_lib
 segment.launch _
```


```
lenovo@lenovo-felipet:~/catkin_w$ rosbag play -l
src/data_lidar/0013.bag_
```

The segmentation obtained with this configuration is shown below:

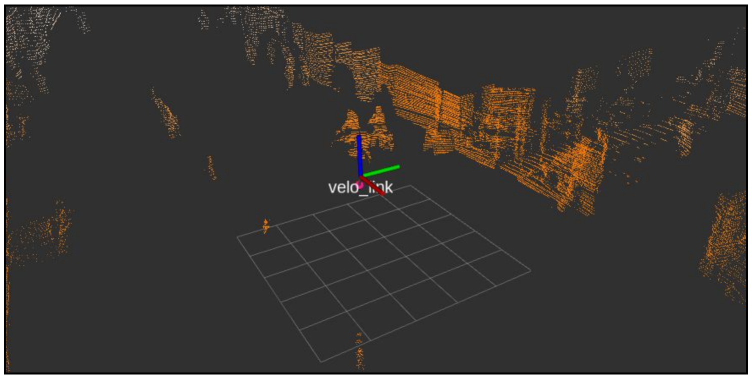


Figure 20: Segmentation obtained with the first test (See annexes).

In the previous figure it can be seen that the ground segmentation is done correctly, although at some points it could be seen that there were points that appear far from the velo_link. Something that cannot be seen in the figure is that the reproduction of the test in Rviz does not exceed 1 fps, which causes the visualization to deteriorate.

The graph obtained from nodes and active topics is the following:

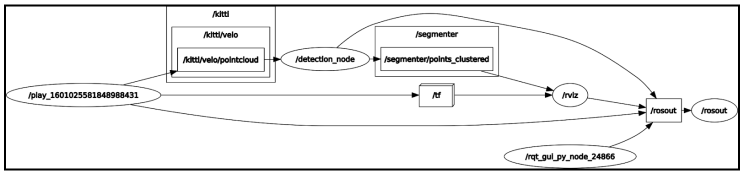


Figure 21: Graph of nodes and active topics generated with rqt_graph for test 1 (See annexes).

As can be seen in the previous figure, the point cloud no longer passes directly to the rviz node, but before the **detection_node** is subscribed, processed and delivered by the **points_clustered** for visualization.

TEST 2: Ground RANSAC Segmenter and Euclidean Cluster Segmenter.

To test these mechanisms, you must first modify the detection.yaml file, where you must remove the variables that call the two previous models and use the variables **ground_remover_type: "GroundRANSACSegmenter",** and **non_ground_segmenter_type: "EuclideanSegmenter" ,**.

For this test, the default parameters that these models bring in the Segmenter.yaml file will be used. To begin with, the roi filter parameters remain unchanged, then for the Ground RANSAC Segmenter the variable **sac_distance_threshold** is set at 0.2, the **sac_max_iteration** at 100 and **sac_probability** at 0.99.

Finally, the parameters of the Euclidean Cluster Segmenter are established, where the Euclidean distance threshold is defined with the variable **ec_tolerance** at a value of 0.2 and the minimum and maximum cluster's point number with two variables: **ec_min_cluster_size** and **ec_max_cluster_size**, with values of 20 and 30000 respectively.

To run the test, the same commands used as in the previous test are used, obtaining the following results:

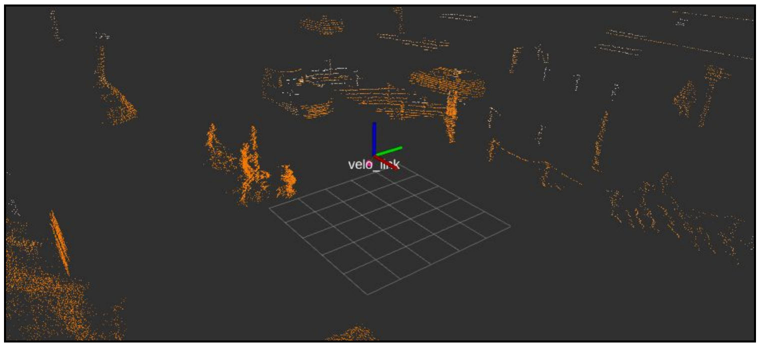


Figure 22: Segmentation obtained with the second test (See annexes).

(The graph obtained with rqt_graph is the same again because the same topics are used for Rviz to subscribe and view its information).

The results obtained with these methods are much better in terms of computational efficiency, since for example the Region Euclidean Segmenter is a much more complete algorithm than the one used in this test (Euclidean Cluster Segmenter).

Although the visualization looks much more fluid, it can be seen in the results obtained that the segmentation of the ground is not done well in some areas, so it must be determined which combination of methods is better and modify the values of the Segmented.yaml parameters to strike a balance between good segmentation and good computational performance.

TEST 3: Combination between models and parameters setting.

To obtain the best combination of the best results with the roi filter, ground segmentation and noise removal, several tests were carried out until the balance between good segmentation and acceptable computing performance was achieved. Finally, it was decided to use the same algorithms as test 2 but modifying the parameters.

For the ROI filter, the area of interest was delimited as Square, with **roi_radius_min_m: 15.** and **roi_radius_max_m: 20** for the distance of the area in XY, and for the vertical range **roi_height_below_m: 1.** and **roi_height_above_m: 20.** For the RANSAC segmenter **sac_distance_threshold** was changed to 0. and for the Euclidean Segmenter it was returned to the default **value ec_tolerance: 0.25** but with **ec_min_cluster_size: 50.**

To make the visualization the same commands used previously are used.

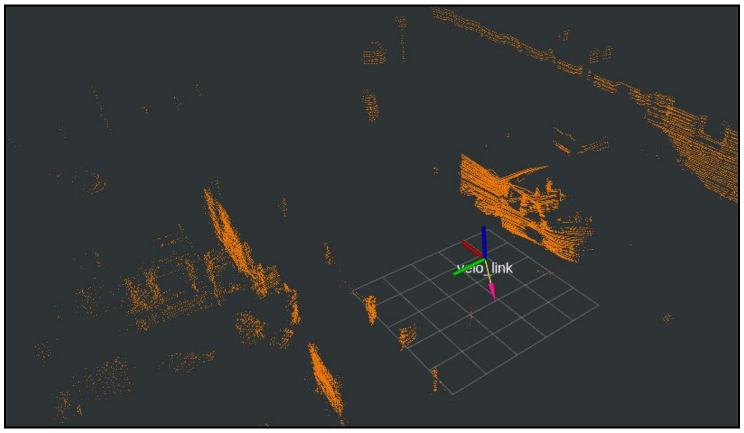


Figure 23: Segmentation obtained with the final test (See annexes).

The reason why these algorithms and those parameters were used is because they not only provide good performance for segmentation of the soil and point clouds from which you do not want to obtain information, but also allow you to see the data sequence of a more fluid way (It is wanted to clarify that the machine that is working does not have a dedicated graphics card and that the processor is an AMD A12 7th Generation, so with computers with better characteristics, better results can be obtained).

[3] MODELS AND EVALUATION:

As mentioned in the previous stages, two pre-processes were carried out in different environments (Jupyter lab and ROS). The environment where the clustering models were tested was that of Jupyter Lab, since it is much simpler and allows tests to be carried out more quickly and efficiently, while ROS also works with C ++ and must be done publications and subscriptions to topics from different nodes that can each be programmed with a different programming language.

Clustering with Python in Jupyterlab environment:

The way in which various algorithms for clustering were tested (among which are the unsupervised learning algorithms seen in the Machine Learning course) was from the .cluster module that the sklearn library has. In addition to this library, Open3D will be used again to make the visualization.

K-Means clustering:

As previously said, the clustering algorithms will be applied with sklearn, so the import must be carried out with "**import sklearn.cluster**". It should be taken into account that the parameter that must be taken into account to do segmentation with K-Means is the number of classes that are known to be in the data set, and based on this the algorithm separates the cloud from points. Then we proceed to make the fit but with the data obtained in the preprocessing stage (**otherdata**):

```
othersids=[]
for i in range(origindata.shape[0]):
    if i not in planeids:
        othersids.append(i)
otherdata=origindata[othersids]
```

Applying the K-Means on this data would look like this:

```
Css=sklearn.cluster.KMeans(n_clusters=10).fit(otherdata)
```

Once you have the label for each point, a new color is also assigned to each class (the colors are established in the **colorset** array). Finally with Open3D, the new point cloud is passed to Vector3dVector format and it is graphed in 3d in the pop-up window.

To try to obtain the best possible performance, the **n_clusters** parameter was varied several times, since it indicates the number of classes expected to be obtained in the dataset. Below are the results obtained with example values of 10, 50 and 100 for this parameter:

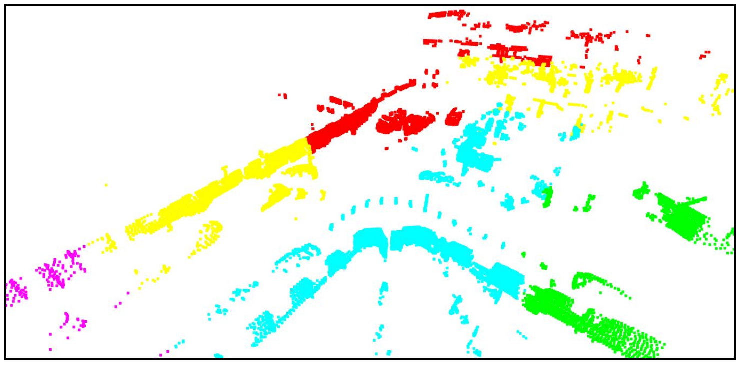


Figure 24: Results with K-means clustering, with n_clusters = 10.

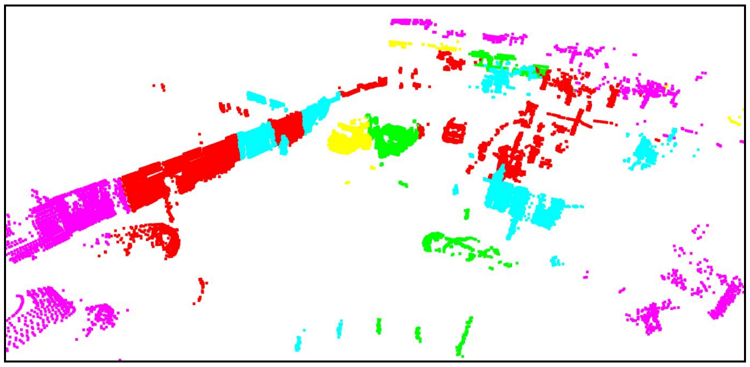


Figure 25: Results with K-means clustering, with n_clusters = 50.

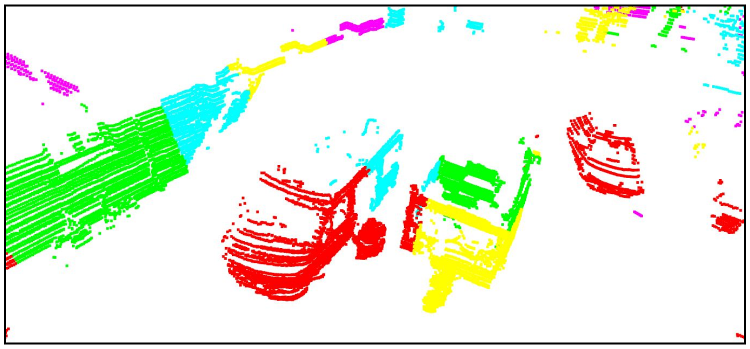


Figure 26: Results with K-means clustering, with n_clusters = 100.

From the results obtained by varying the n_clusters parameter, it can be seen that clustering with this unsupervised algorithm is not ideal. Since not being able to know the exact discrete value of the number of classes that are in a certain scene, estimates must be started. For a very small number compared to the number of objects, the algorithm works poorly, and the more the number of classes increases, the cluster differentiation "improves", but in the same way, as seen in Figure 25, Difficulty with objects that are very close or with objects that appear to be two objects. Finally, if the number of clusters that is estimated is very high, it can be seen that the algorithm only begins to divide the point clouds in order to comply with that number of clusters, and it even assigns up to three different classes to a point cloud. it should be from just one object.

DBSCAN (Density-Based Spatial Clustering):

The following segmentation method is the one frequently used to segment point clouds. As you can read in their documentation on sklearn: "The DBSCAN algorithm views clusters as areas of high density separated by areas of low density. Due to this rather generic view, clusters found by DBSCAN can be any shape, as opposed to k-means which assumes that clusters are convex shaped. The central component to the DBSCAN is the concept of core samples, which are samples that are in areas of high density. A cluster is therefore a set of core samples, each close to each other (measured by some distance measure) and a set of non-core samples that are close to a core sample (but are not themselves core samples). There are two parameters to the algorithm, min_samples and eps, which define formally what we mean when we say dense. Higher min_samples or lower eps indicate higher density necessary to form a cluster."[10]

The parameters to be taken into account for DBSCAN are eps and min_samples:

```
Css=sklearn.cluster.DBSCAN(eps=0.50, min_samples=4).fit(otherdata)
```

The variable eps refers to the closeness distance to be considered between points to group them and n_samples refers to the number of points that must surround another point so that the latter can be considered a center.

Below are the results obtained by doing several tests varying these two parameters:

[10] Custering. Skit learn. Available online: https://scikit-learn.org/stable/modules/clustering.html#

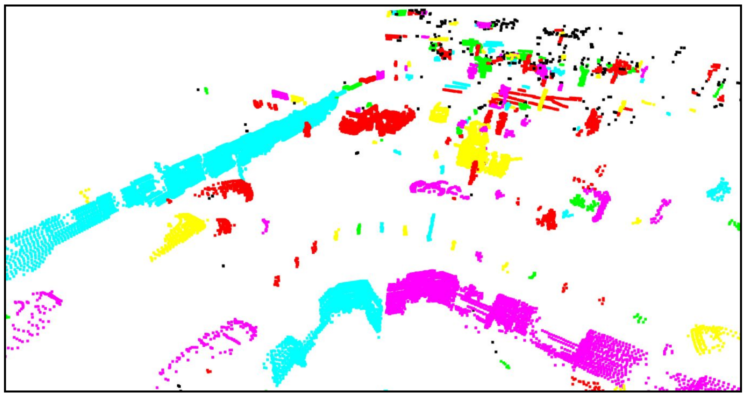


Figure 27: Results with DBSCAN with eps=0.50 and min_samples=4.

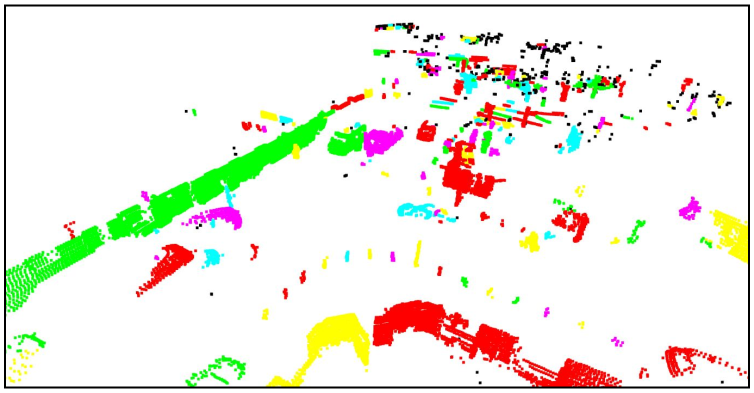


Figure 28: Results with DBSCAN with eps=0.45 and min_samples=4.

As can be seen in Figure 27 and 28, by decreasing the eps value, the division between clusters in point clouds that are very close to each other improves, and you can also see the advantage of using DBSCAN and that is that if there are point clouds that does not comply with the min_samples parameter, it is assigned the noise label, and it is graphed in black. This helps to eliminate measurements made by the dataset where the information obtained from a certain area is not significant.

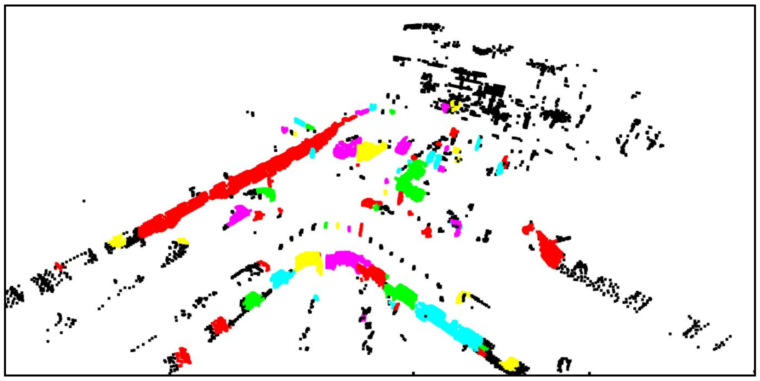


Figure 29: Results with DBSCAN with eps=0.45 and min_samples=22.

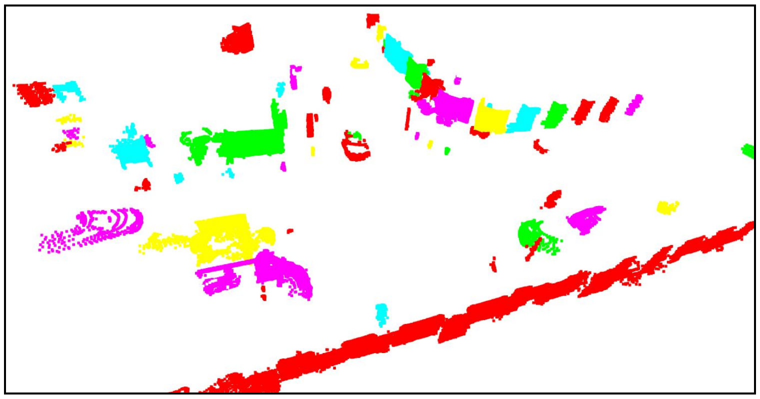


Figure 30: DBSCAN: eps=0.45 and min_samples=22, without noise.

Finally, the min_samples variable was modified to discard even more point clouds that do not provide a sufficiently dense point cloud. In Figure 29 it can be seen how a greater number of points are selected as noise and in Figure 30 only the visualization of the clusters that were assigned a class and that were not discarded as noise.

HDBSCAN (Hierarchical Density-Based Spatial Clustering):

Going to the Skit learn documentation, the following was found: "OPTICS (Ordering Points To Identify the Clustering Structure), closely related to DBSCAN, finds core samples of high density and expands clusters from them [1]. Unlike DBSCAN, it keeps cluster hierarchy for a variable neighborhood radius. Better suited for usage on large datasets than the current sklearn implementation of DBSCAN."[11]

When testing with this method, very poor results were obtained, but it should be clarified that its parameters were not modified many times, since the processing of this algorithm was very slow and it was very inefficient to do them. Regarding the above, the same Scikit learn documentation provides the following information: "Spatial indexing trees are used to avoid calculating the full distance matrix, and allow for efficient memory usage on large sets of samples. Different distance metrics can be supplied via the metric keyword. For large datasets, similar (but not identical) results can be obtained via HDBSCAN. The HDBSCAN implementation is multithreaded, and has better algorithmic runtime complexity than OPTICS, at the cost of worse memory scaling. For extremely large datasets that exhaust system memory using HDBSCAN, OPTICS will maintain n (as opposed to n^2) memory scaling; however, tuning of the max_eps parameter will likely need to be used to give a solution in a reasonable amount of wall time." [11]

Now, it is proceeding to install the hdbscan library in the virtual environment with "**pip install hdbscan**". In order to use the library, the following import must be done:

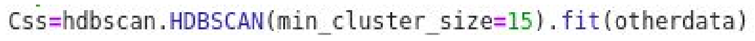
```
import hdbscan
```

In order to start the tests, the documentation provided by the library itself was used[12], where it is indicated that to use this algorithm 4

[11] Clustering-OPTICS. Scikit Learn. Available online: https://scikit-learn.org/stable/modules/clustering.html#optics

[12] Parameter Selection for HDBSCAN. HDBSCAN. Available online: https://hdbscan.readthedocs.io/en/latest/parameter_selection.html

parameters are used. The first is **min_cluster_size**, which refers to the minimum number of points that can form a cluster. Then there are the same two parameters that are used in basic DBSCAN, which are defined as **cluster_selection_epsilon** and **min_samples**. Finally, there is the value of alpha, which the same library recommends not modifying its default value of one, since they indicate that it is better to modify the two previous parameters and as a last option the alpha, because "n practice it is best not to mess with this parameter - ultimately it is part of the Robust Single Linkage code, but flows naturally into HDBSCAN". [12]



```
Css=hdbscan.HDBSCAN(min_cluster_size=15).fit(otherdata)
```

The first test was carried out with the default parameters, except for min_cluster_size, since we wanted to try to eliminate sparse point clouds that could be considered noise or that are from objects with very little information due to a bad reading of data or because they are far away.

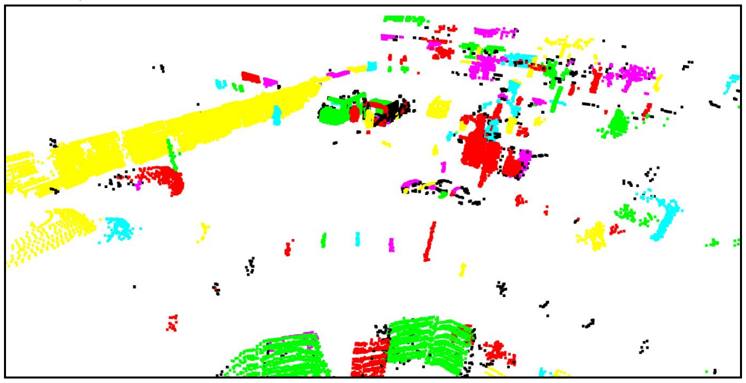


Figure 31: Results with HDBSCAN: min_cluster_size=15.

It can be seen that with only this parameter certain objects are identified as new clusters, which with previous models did not happen, but it is also seen that the noise is not reduced as much as when using DBSCAN.

To check the behavior of the algorithm, another test was carried out using the same parameters that gave better performance with DBSCAN:

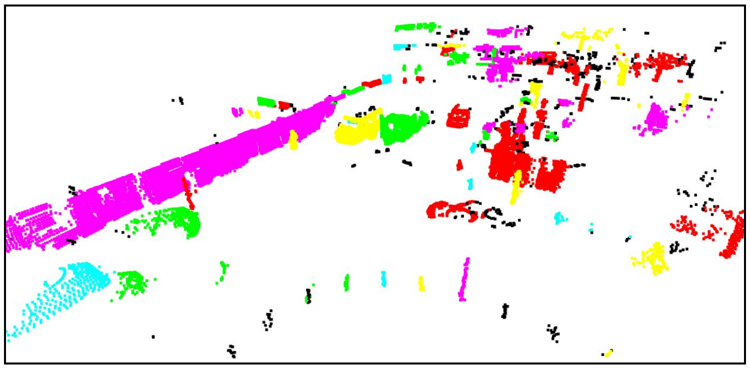


Figure 32: HDBSCAN clustering: cluster_selection_epsilon=0.45 and min_samples=22.

Once the results of Figure 32 were obtained, it was found that the algorithm does not determine as many points as noise as DBSCAN does, but by not implementing the min_cluster parameter.

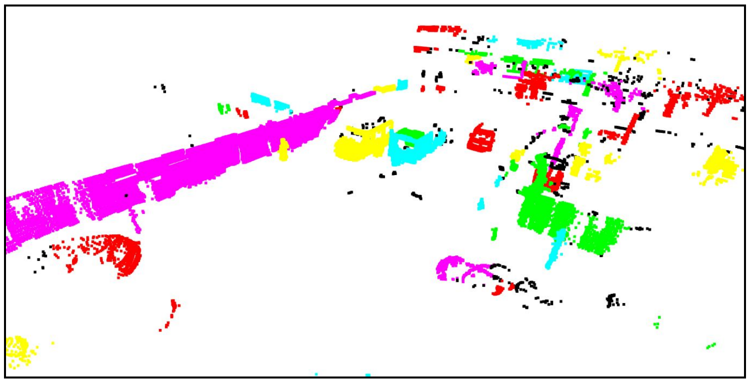


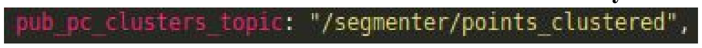
Figure 33: HDBSCAN clustering: min_cluster_size=30 and cluster_selection_epsilon=0.5.

After trying different values and combinations of the parameters, better results were not obtained, so it is understood that this algorithm could serve for a preprocessing stage where you want to filter the point cloud after making the soil segmentation, and then if apply DBSCAN.

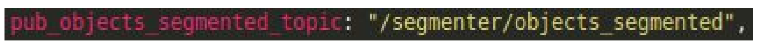
Applicative example of unsupervised clustering algorithms:

In previous tests, segmentation methods were implemented to preprocess point clouds to be able to filter the soil information and eliminate points that were outside the range of interest and tests were also carried out with unsupervised machine learning models such as K-means, DBSCAN AND HDBSCAN, but these last tests were subject to the processing that Open3D provides at the time of graphing, which was not optimal, so now a way to be able to clustering in the ROS environment is presented, seeing both preprocessing and clustering of each binary file in the dataset sequentially (in the order each .bin file was obtained). For the above, the LidarPerception library is used again, which, as mentioned before, already incorporates a clustering method, in addition to creating a box for each cluster that delimits each one.

For these two previous tasks, the topic where the separate point cloud will be published in clusters and the topic where the boxes will be published for each cluster must be defined in the **detection.yam**l.

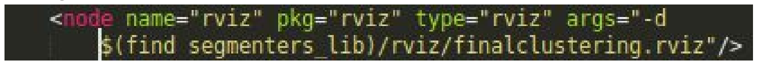


```
pub_pc_clusters_topic: "/segmenter/points_clustered",
```



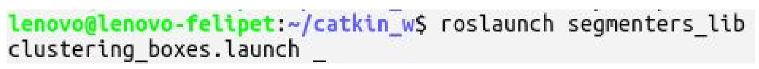
```
pub_objects_segmented_topic: "/segmenter/objects_segmented",
```

In order to visualize each stage that was made, a new .launch file is created which will open a new configuration in rviz that was made for this purpose. The only thing that changes in the new .launch called **clustering_boxes.launch** is where you put the name of the rviz config file:



```
  <node name="rviz" pkg="rviz" type="rviz" args="-d
      $(find segmenters_lib)/rviz/finalclustering.rviz"/>
```

Now, in order to run the test, the following command is used within the work environment:

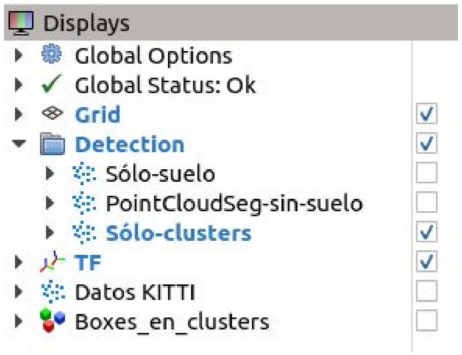


```
lenovo@lenovo-felipet:~/catkin_w$ roslaunch segmenters_lib
clustering_boxes.launch _
```

Also in another terminal (and also in the work environment) the publication of the point cloud with rosbag starts:

```
lenovo@lenovo-felipet:~/catkin_w$ rosbag play -l src/
data_lidar/0000.bag _
```

Una vez ejecutado se podrán ver las siguientes opciones en Rviz:



Displays
- Global Options
- ✓ Global Status: Ok
- Grid ✓
- Detection ✓
  - Sólo-suelo ☐
  - PointCloudSeg-sin-suelo ☐
  - Sólo-clusters ✓
- TF ✓
- Datos KITTI ☐
- Boxes_en_clusters ☐

Figure 34: Visualization options with **finalclustering.rviz**.

In the detection folder you can see (from top to bottom) the point cloud of only the segmented soil, the point cloud without soil, and finally the point cloud divided into groups. You can view the Velodyne LIDAR coordinate system and finally the option to view the boxes that each group contains.

The results obtained from grouping are as follows:

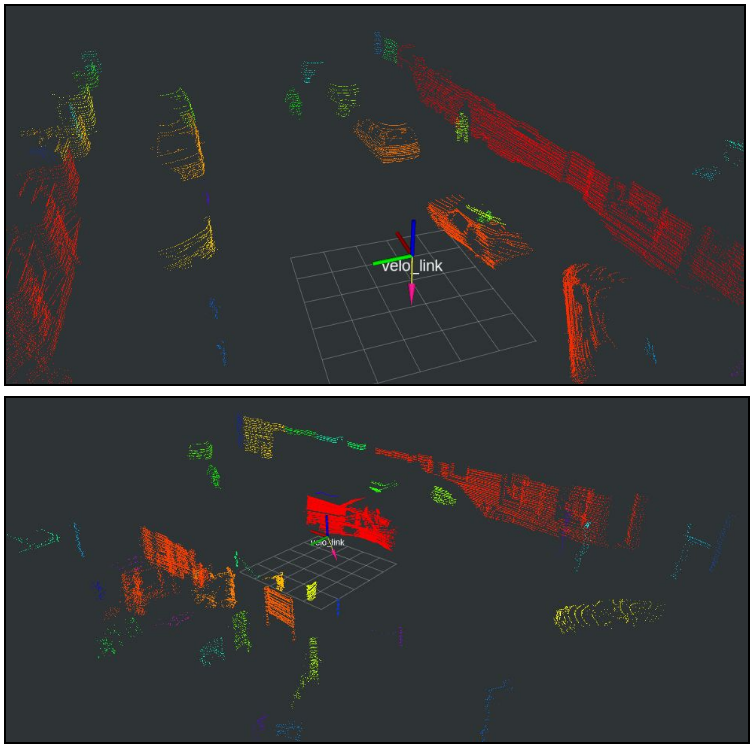


Figure 35: Results obtained from clustering in the ROS environment (See annexes).

As can be seen in the previous images, the segmentation that is obtained is good, however, in certain frames, two clusters are assigned to the same object by the separation of the sets of point clouds that make up said object. Also due to the random way in which the colors are assigned, it is difficult to differentiate the clusters when they are very close together, so activating the display of the boxes facilitates this task.

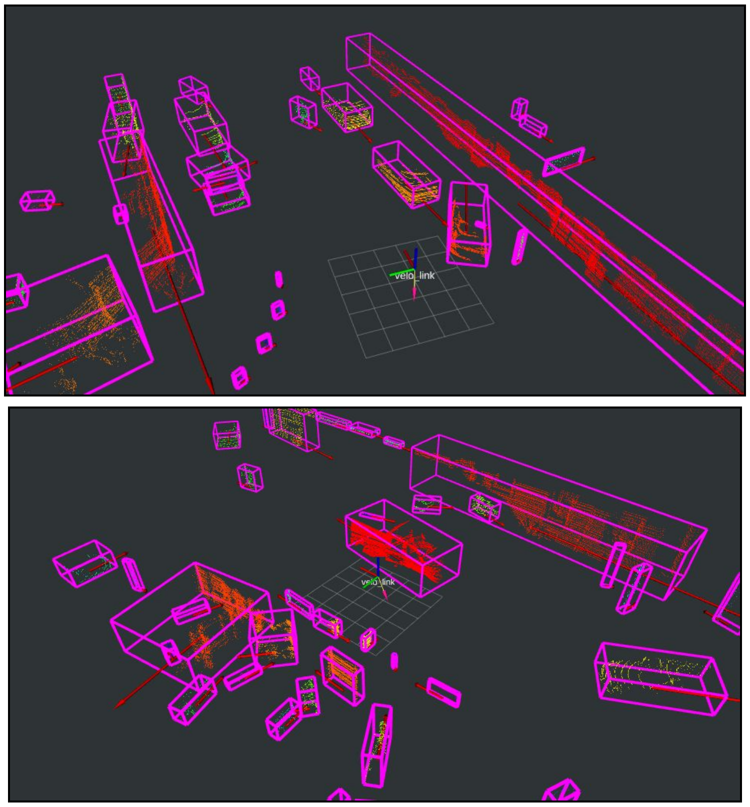


Figure 36: Visualization of clusters and container boxes (See annexes).

Finally, the rqt_graph obtained when visualizing in Rviz the point clouds in the topics of interest that were explained previously is shown:

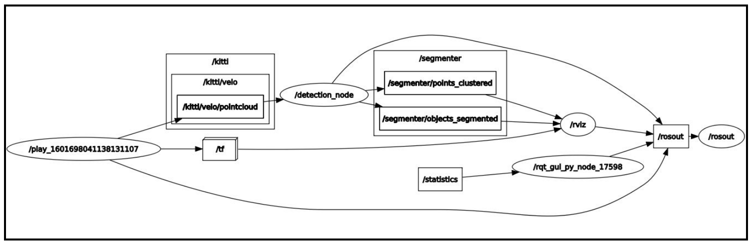


Figure 37: Graph obtained with rqt_graph of nodes and active topics when executing clustering and container boxes (See annexes).

IV. CONCLUSIONS

- The ROS environment provides many more tools to be able to work with point cloud type data. Among their advantages is that they have different shapes, sizes, colors, etc. that can be adjusted while the visualization is running, in the same way that the viewing angle and zoom can also be modified while this happens. In Open3d the latter can also be done, but because the Vector3dVector command is poorly optimized it would not be possible to see the graph of each binary file that represents only one frame in a fluent way.
- Regarding the pre-processing of the data, although the Ground RANSAC Segmenter was chosen with the Euclidean Segmenter, it must be said that the best results were obtained with the Ground Plane Fitting Segmenter and the Region Euclidean Segmenter, especially for the latter, since it is a very more complex than the Euclidean Segmenter, which

when applied to areas a better segmentation is achieved, but its computational cost is much higher, that being the only reason why it did not choose.

- When applying three different clustering models with unsupervised machine learning algorithms, it can be said that it is recommended to use K-means only for simple forms in two dimensions, since having a parameter as restrictive as knowing the number of clusters in a Scene becomes inefficient for an application like the one you propose, since a different number of clusters is expected in each frame. Regarding DBSCAN and HDBSCAN, it was observed that the segmentation that DBSCAN does on the groups of sparse points can become aggressive, eliminating almost as many points as with the ROI filter in ROS. On the other hand, with HDBSCAN the elimination of these points is not so strong, and it was seen that with certain parameter values it was capable of obtaining complicated clusters such as the people near the trucks in Figure 31. Based on the above, I would suggest using the HDBSCAN method if the amount of data you have is very large and you want to clean the data in some way, but for the application proposed in this work it is considered that it is better to use segmentation methods to filter very well the data and later implement DBSCAN with values in its parameters not so high.

REFERENCES

[1] Chan, Gary. LidarPercepction/segmenters_lib. Available online: https://github.com/LidarPerception/segmenters_lib

[2] Chen, X., Ma, H., Wan, J., Li, B., & Xia, T. (2017). Multi-view 3d object detection network for autonomous driving. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (pp. 1907-1915).

[3] Clustering. Skit learn. Available online: https://scikit-learn.org/stable/modules/clustering.html#.

[4] Clustering-OPTICS. Scikit Learn. Available online: https://scikit-learn.org/stable/modules/clustering.html#optics.

[5] Flores, Pablo. Braun Juan. Algoritmo RANSAC: fundamento teórico (2011). Available online: http://iie.fing.edu.uy/investigacion/grupos/gti/timag/trabajos/2011/keypoints/FundamentoRANSAC.pd

[6] Geiger, A., Lenz, P., Stiller, C., & Urtasun, R. (2013). Vision meets robotics: The kitti dataset. The International Journal of Robotics Research, 32(11), 1231-1237.

[7] HDL-64E. Velodyne Lidar. Available online: https://velodynelidar.com/products/hdl-64e/.

[8] Karlsruhe Institute of Technology and Toyota Technological Institute at Chicago. KITTI dataset. Available online: http://www.cvlibs.net/datasets/kitti/.

[9] Krejci, Tomas. Github Repository: kitti2bag. Disponible en https://github.com/tomas789/kitti2bag.

[10] Ku, J., Pon, A. D., & Waslander, S. L. (2019). Monocular 3d object detection leveraging accurate proposals and shape reconstruction. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (pp. 11867-11876).7

[11] Li, B. (2017, September). 3d fully convolutional network for vehicle detection in point cloud. In 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (pp. 1513-1518). IEEE.

[12] Li, B., Zhang, T., & Xia, T. (2016). Vehicle detection from 3d lidar using fully convolutional network. arXiv preprint arXiv:1608.07916.

[13] Ma, Nongjiayuan. Kitti-clustering-homework. Available online: https://www.codenong.com/cs106008457/.

[14] Mobile Robots. ROS components. Available online: https://www.roscomponents.com/es/12-robots-moviles.

[15] Parameter Selection for HDBSCAN. HDBSCAN. Available online: https://hdbscan.readthedocs.io/en/latest/parameter_selection.html

[16] PCL API Documentation. Point Cloud Library. Available online:https://pointclouds.org/documentation/index.html.

[17] Rangel, S., Tobar, L. F., & Escobar, D. A. (2020). Perception system for the vehicle land exploration “Chimuelo” focused on orientation works in the basement laboratories 2 of the Universidad Autónoma de Occidente. Available online on Researchgate.

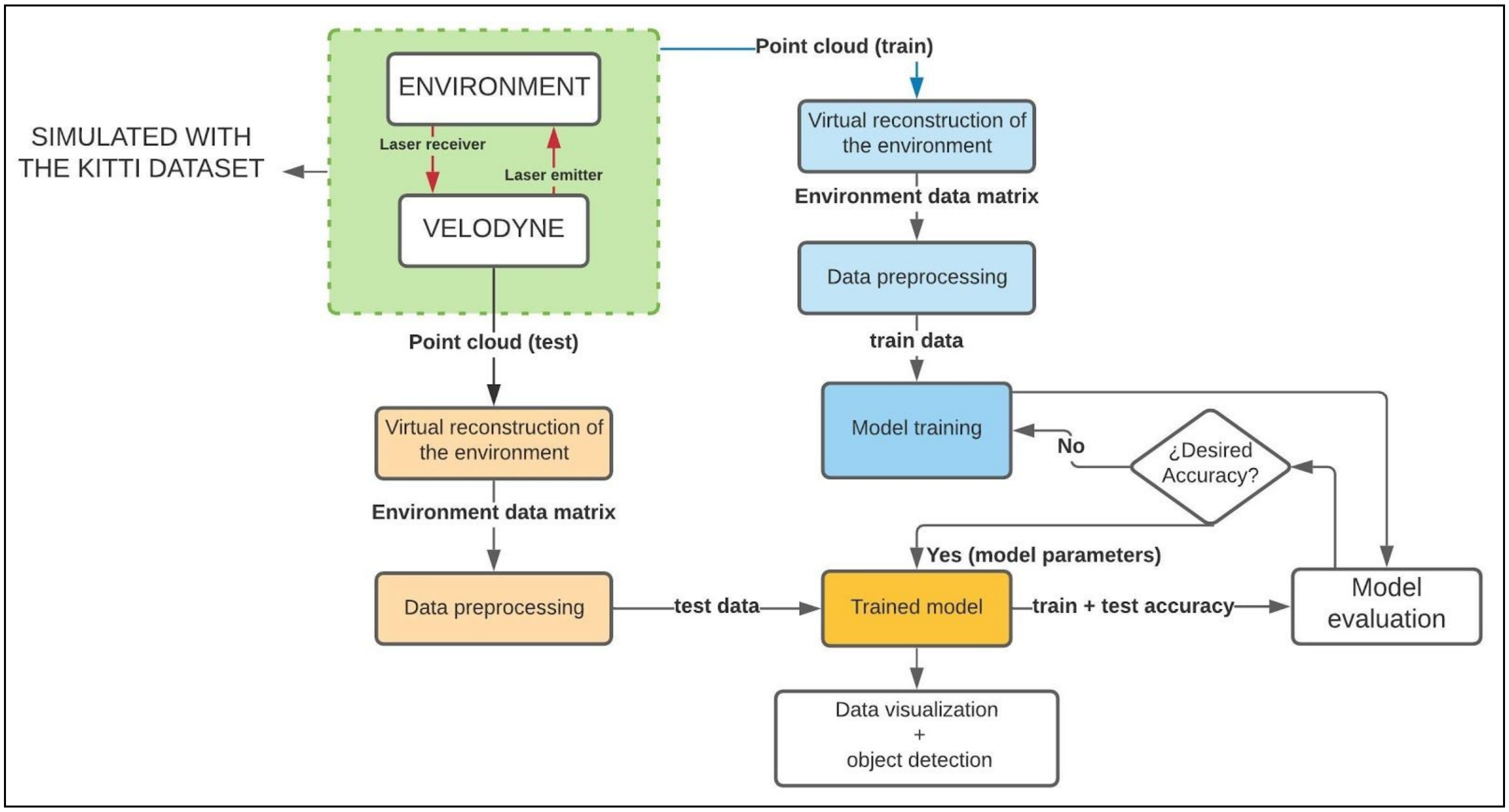
[18] Road traffic injuries (2020). World Health Organization. Available online: https://www.who.int/es/news-room/fact-sheets/detail/road-traffic-injuries.

[19] Space and Terrestrial Autonomous Robotic Systems Laboratory at the University of Toronto. Github repository: https://github.com/utiasSTARS/pykitti.

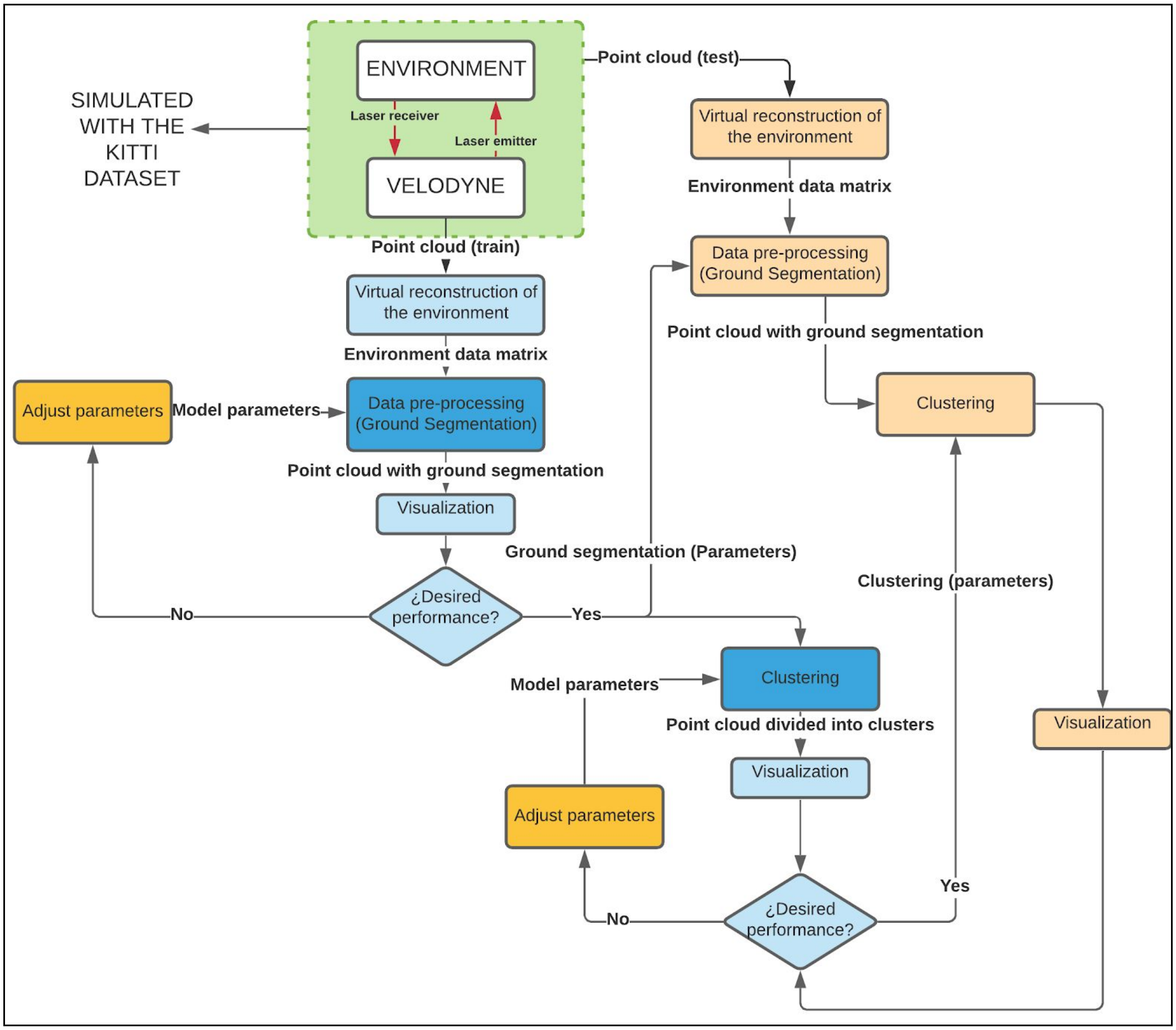
[20] Staravoitau, Alex. Github Repository: KITTI-Dataset. Disponible en https://github.com/navoshta/KITTI-Dataset.

[21] The Duckietown Foundation. Available online: https://www.duckietown.org/about/duckietown-foundation.

V. ANNEXES

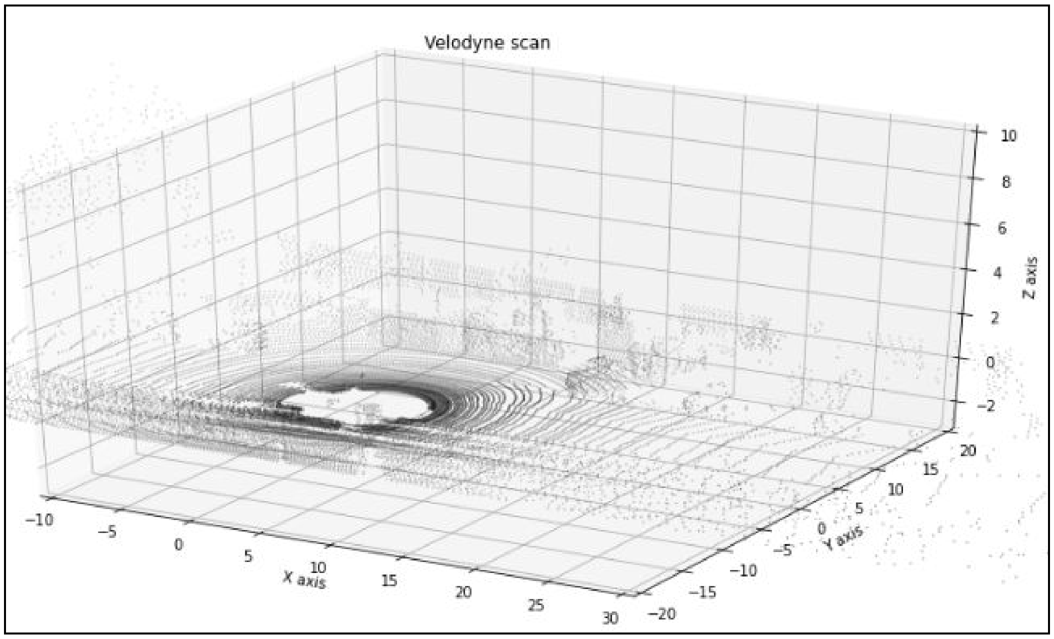


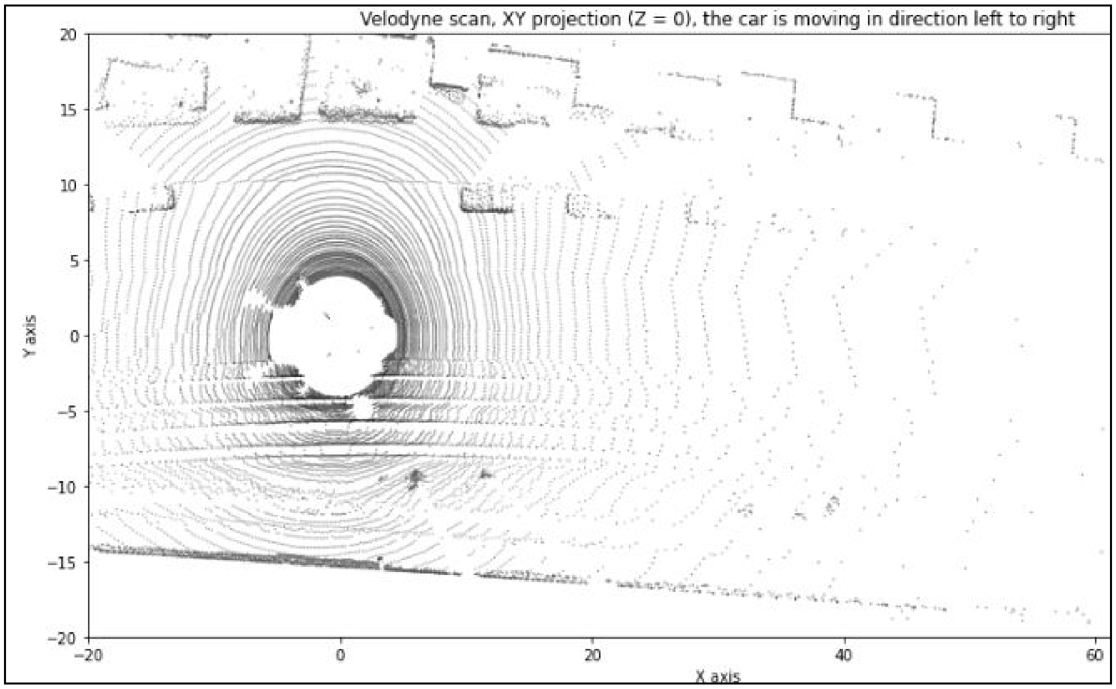
Annex 1: Schematic diagram of the solution process for a supervised learning object detector.



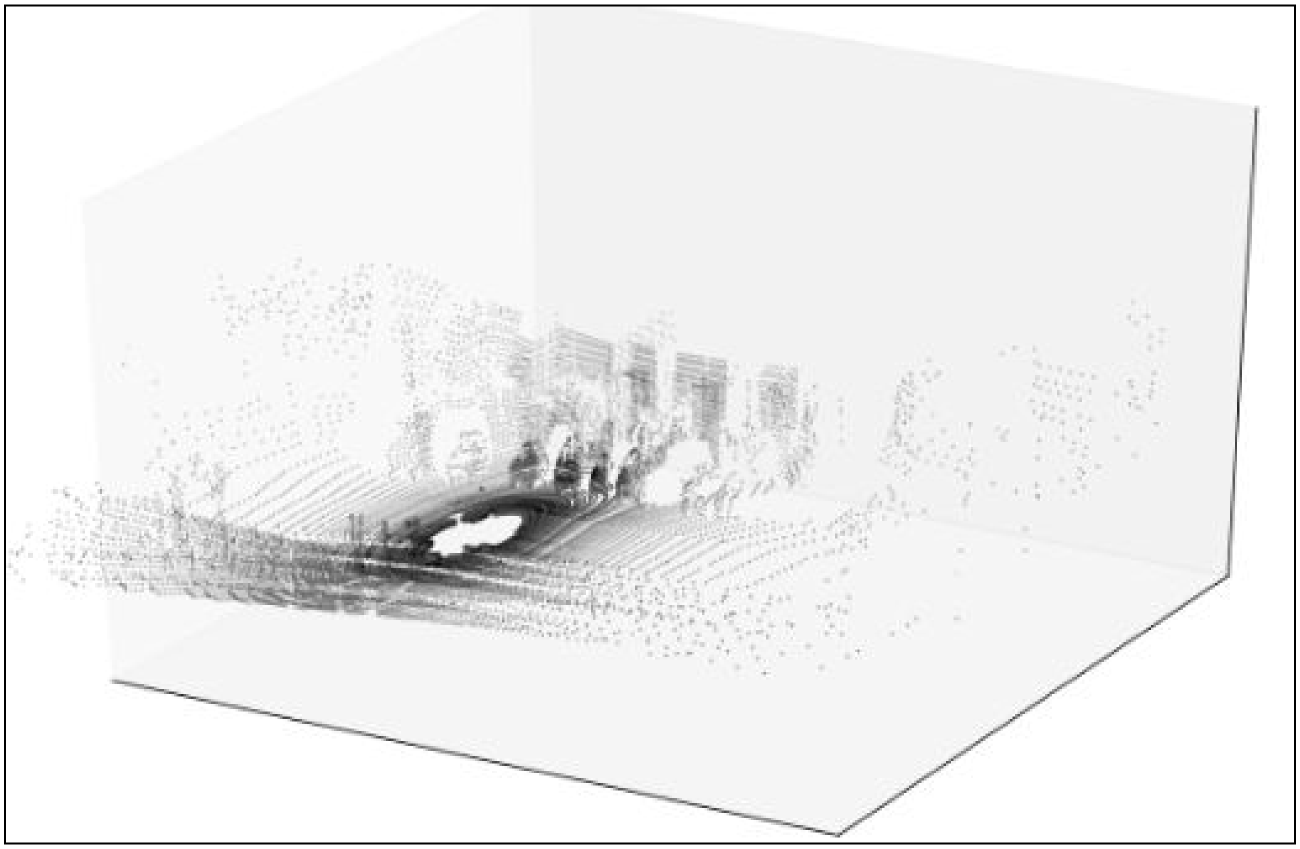
ENVIRONMENT
Laser receiver
Laser emitter
VELODYNE
SIMULATED WITH THE KITTI DATASET
Point cloud (test)
Point cloud (train)
Virtual reconstruction of the environment
Environment data matrix
Data pre-processing (Ground Segmentation)
Adjust parameters
Model parameters
Point cloud with ground segmentation
Visualization
Ground segmentation (Parameters)
¿Desired performance?
No
Yes
Clustering
Clustering (parameters)
Model parameters
Point cloud divided into clusters
Visualization
Adjust parameters
¿Desired performance?
No
Yes
Virtual reconstruction of the environment
Environment data matrix
Data pre-processing (Ground Segmentation)
Point cloud with ground segmentation
Clustering
Visualization

Annex 2: Schematic diagram of the solution process for using unsupervised models for clustering.

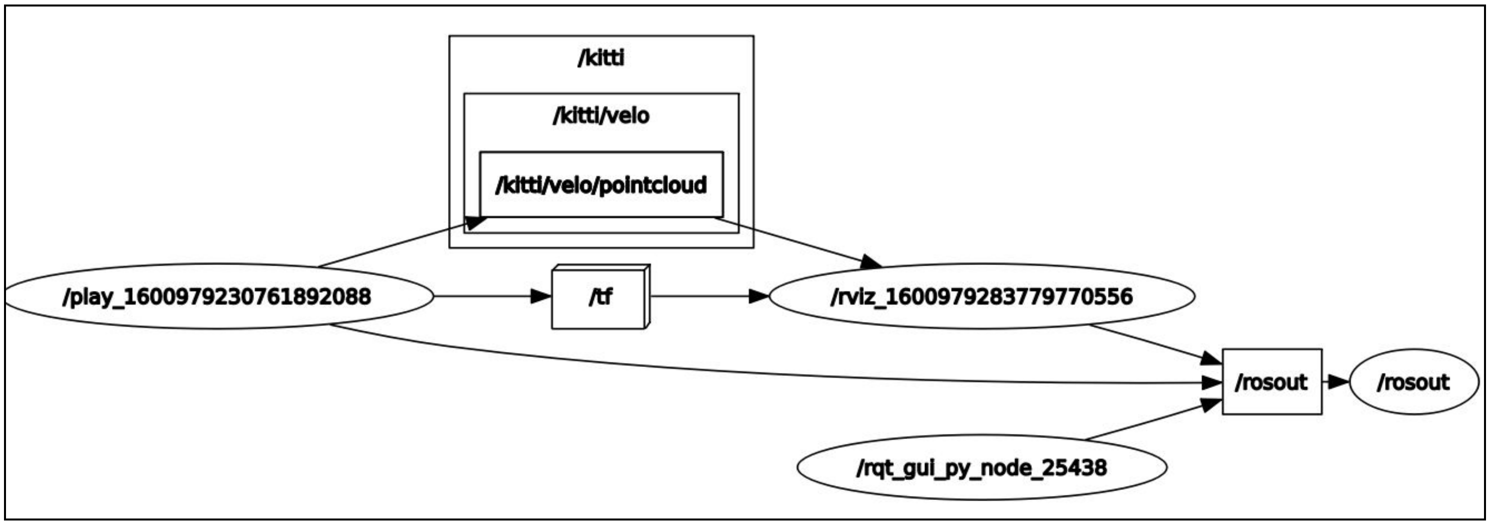




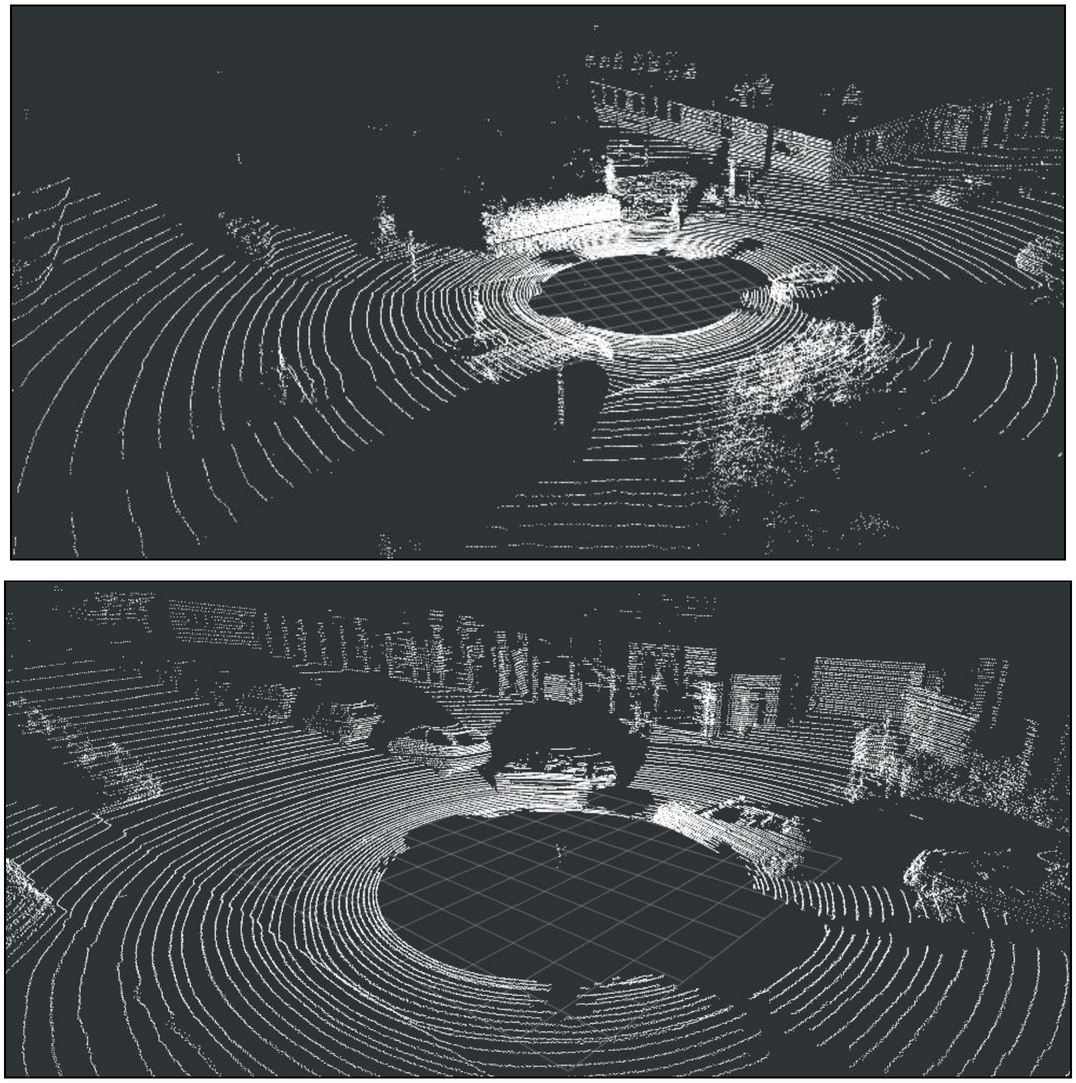
Annex 3: Example of plots with trimetric and top view.

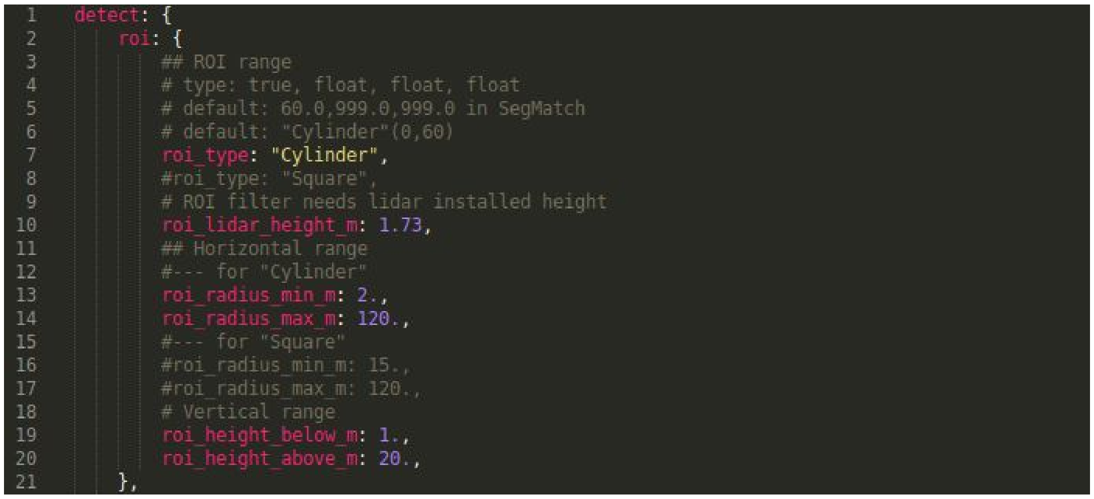


Annex 4: Frame 39 of the gif obtained.

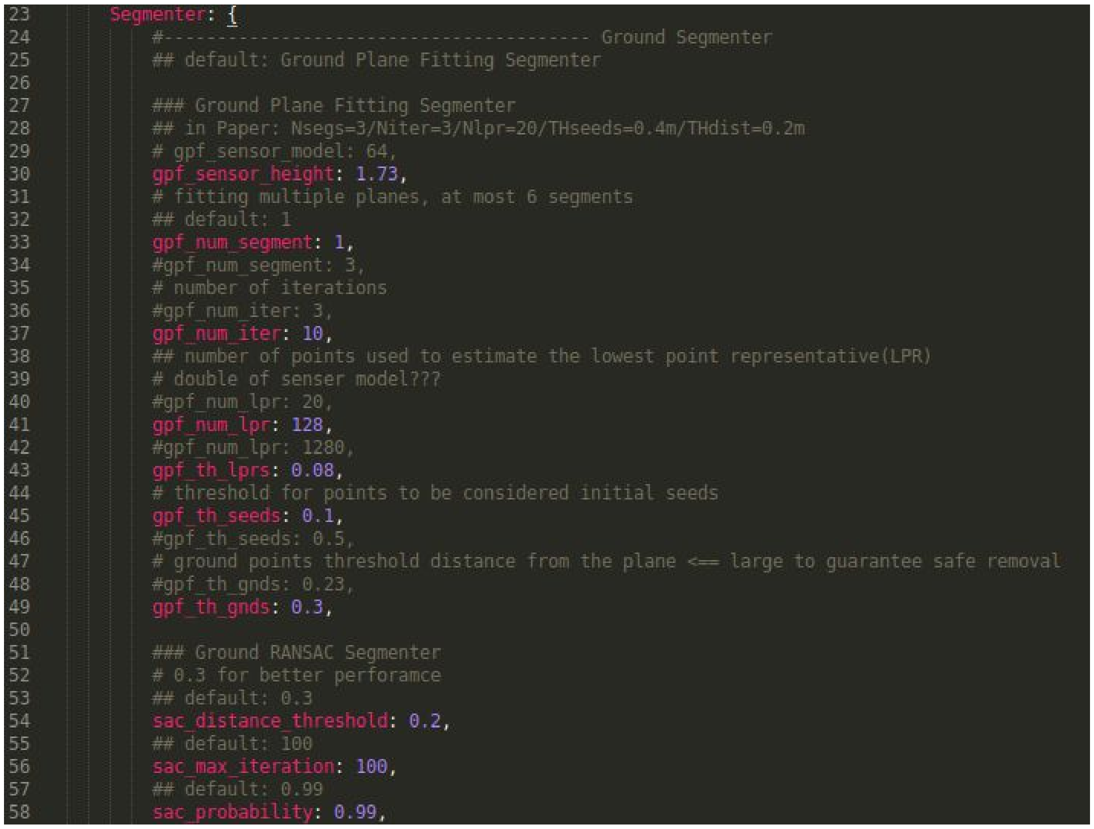


Annex 5: Graph of nodes and active topics generated with rqt_graph for visualization.

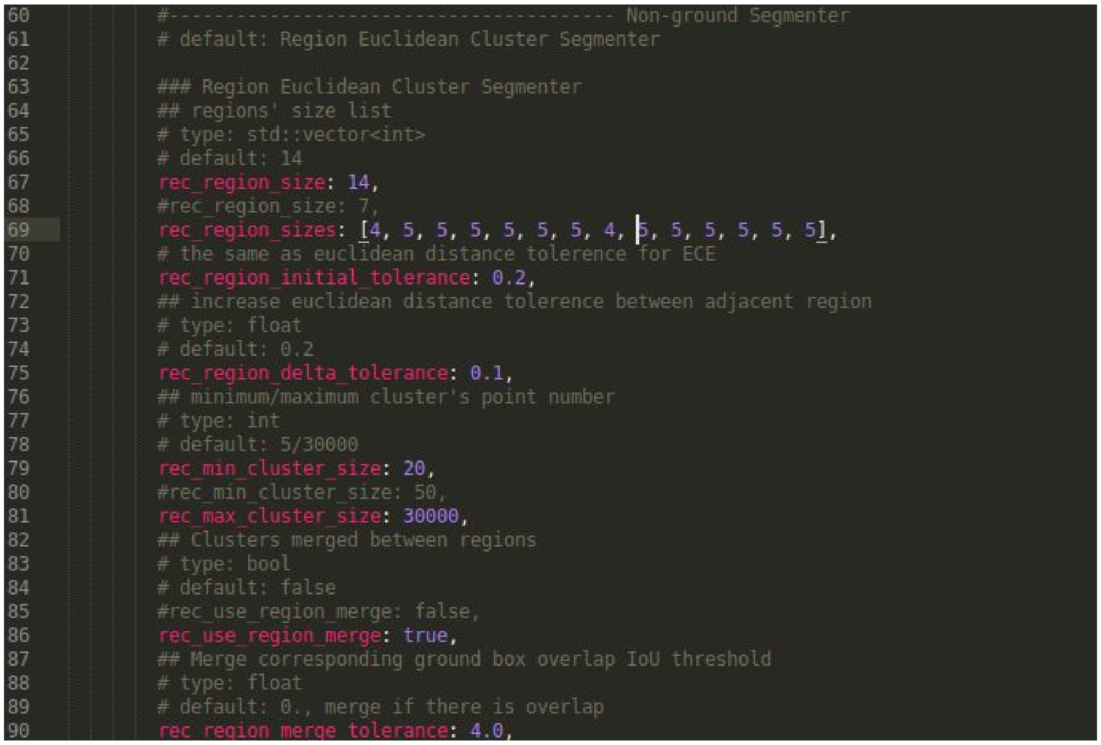

Annex 6: Visualization of the point cloud in Rviz.



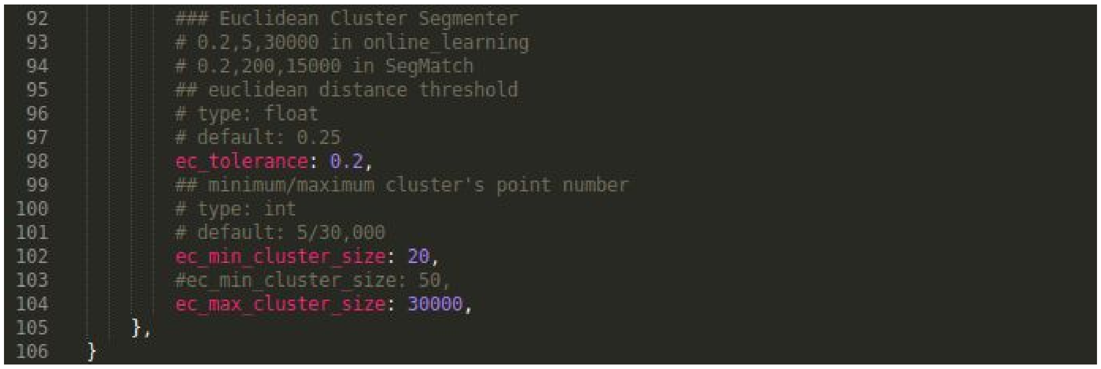
```
detect: {
    roi: {
        ## ROI range
        # type: true, float, float, float
        # default: 60.0,999.0,999.0 in SegMatch
        # default: "Cylinder"(0,60)
        roi_type: "Cylinder",
        #roi_type: "Square",
        # ROI filter needs lidar installed height
        roi_lidar_height_m: 1.73,
        ## Horizontal range
        #--- for "Cylinder"
        roi_radius_min_m: 2.,
        roi_radius_max_m: 120.,
        #--- for "Square"
        #roi_radius_min_m: 15.,
        #roi_radius_max_m: 120.,
        # Vertical range
        roi_height_below_m: 1.,
        roi_height_above_m: 20.,
    },
```



```
    Segmenter: {
        #---------------------------------------- Ground Segmenter
        ## default: Ground Plane Fitting Segmenter

        ### Ground Plane Fitting Segmenter
        ## in Paper: Nsegs=3/Niter=3/Nlpr=20/THseeds=0.4m/THdist=0.2m
        # gpf_sensor_model: 64,
        gpf_sensor_height: 1.73,
        # fitting multiple planes, at most 6 segments
        ## default: 1
        gpf_num_segment: 1,
        #gpf_num_segment: 3,
        # number of iterations
        #gpf_num_iter: 3,
        gpf_num_iter: 10,
        ## number of points used to estimate the lowest point representative(LPR)
        # double of senser model???
        #gpf_num_lpr: 20,
        gpf_num_lpr: 128,
        #gpf_num_lpr: 1280,
        gpf_th_lprs: 0.08,
        # threshold for points to be considered initial seeds
        gpf_th_seeds: 0.1,
        #gpf_th_seeds: 0.5,
        # ground points threshold distance from the plane <== large to guarantee safe removal
        #gpf_th_gnds: 0.23,
        gpf_th_gnds: 0.3,

        ### Ground RANSAC Segmenter
        # 0.3 for better perforamce
        ## default: 0.3
        sac_distance_threshold: 0.2,
        ## default: 100
        sac_max_iteration: 100,
        ## default: 0.99
        sac_probability: 0.99,
```

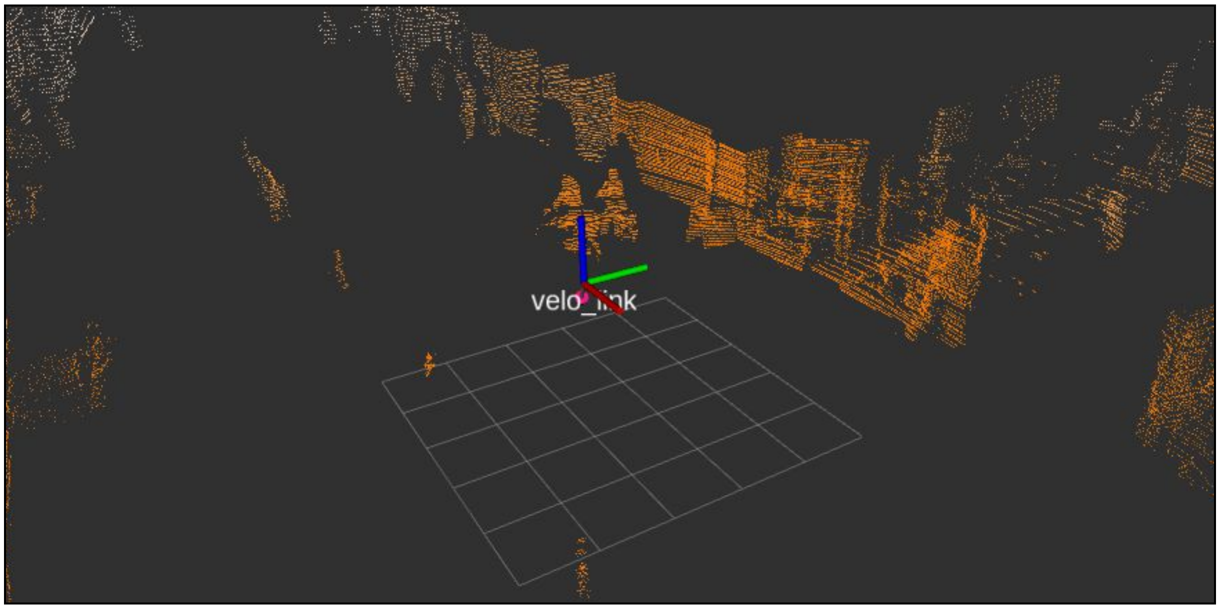


```
        #---------------------------------------- Non-ground Segmenter
        # default: Region Euclidean Cluster Segmenter

        ### Region Euclidean Cluster Segmenter
        ## regions' size list
        # type: std::vector<int>
        # default: 14
        rec_region_size: 14,
        #rec_region_size: 7,
        rec_region_sizes: [4, 5, 5, 5, 5, 5, 5, 4, 5, 5, 5, 5, 5, 5],
        # the same as euclidean distance tolerence for ECE
        rec_region_initial_tolerance: 0.2,
        ## increase euclidean distance tolerence between adjacent region
        # type: float
        # default: 0.2
        rec_region_delta_tolerance: 0.1,
        ## minimum/maximum cluster's point number
        # type: int
        # default: 5/30000
        rec_min_cluster_size: 20,
        #rec_min_cluster_size: 50,
        rec_max_cluster_size: 30000,
        ## Clusters merged between regions
        # type: bool
        # default: false
        #rec_use_region_merge: false,
        rec_use_region_merge: true,
        ## Merge corresponding ground box overlap IoU threshold
        # type: float
        # default: 0., merge if there is overlap
        rec_region_merge_tolerance: 4.0,
```



```
            ### Euclidean Cluster Segmenter
            # 0.2,5,30000 in online_learning
            # 0.2,200,15000 in SegMatch
            ## euclidean distance threshold
            # type: float
            # default: 0.25
            ec_tolerance: 0.2,
            ## minimum/maximum cluster's point number
            # type: int
            # default: 5/30,000
            ec_min_cluster_size: 20,
            #ec_min_cluster_size: 50,
            ec_max_cluster_size: 30000,
        },
    }
```

Annex 8: Demonstration code of the parameters to be modified in the **Segmenter.yaml**.



Annex 9: Segmentation obtained with the first test.

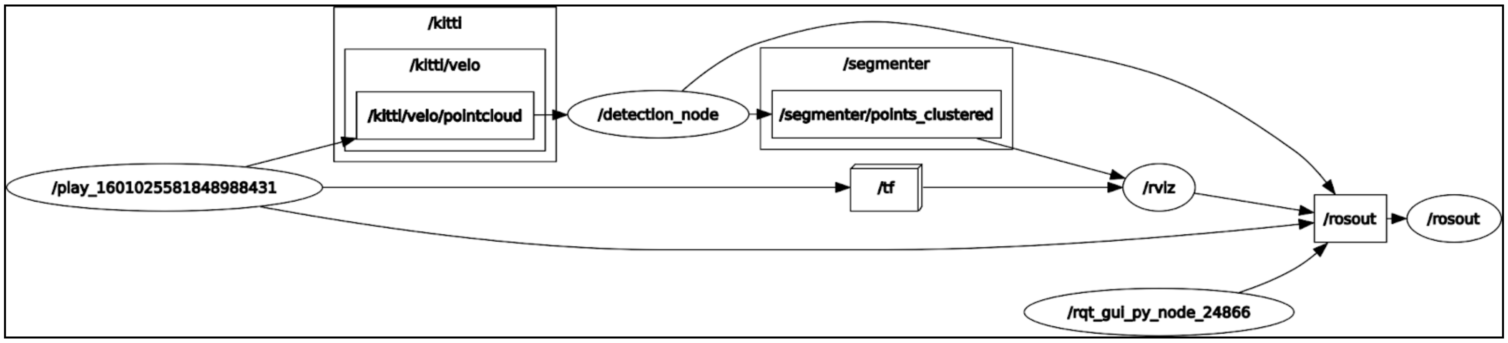
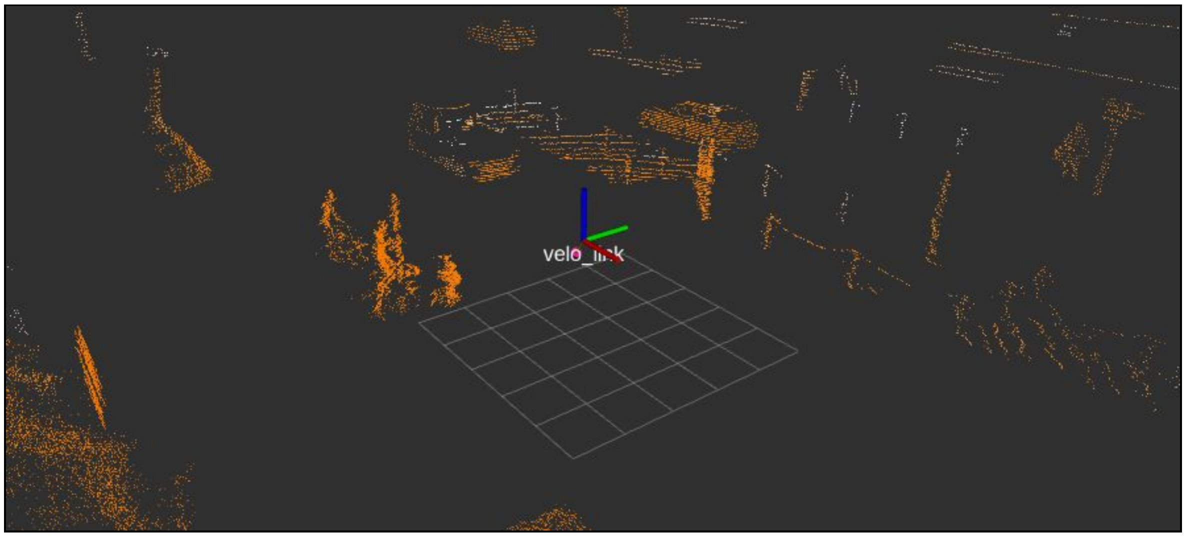
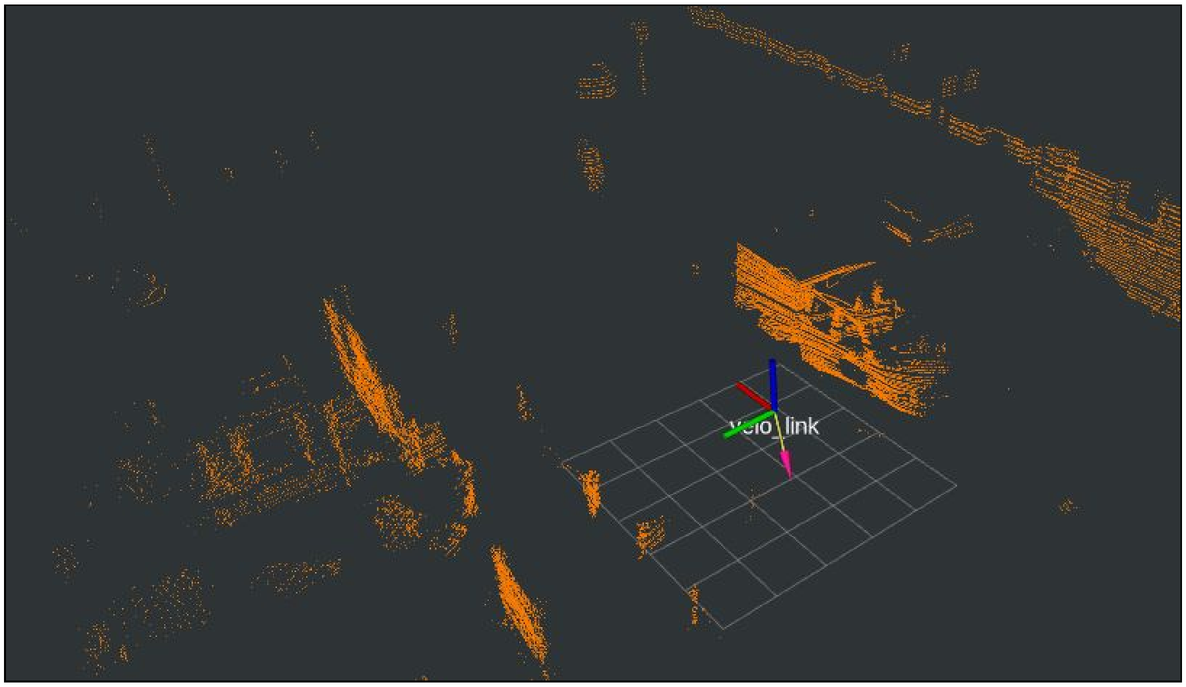


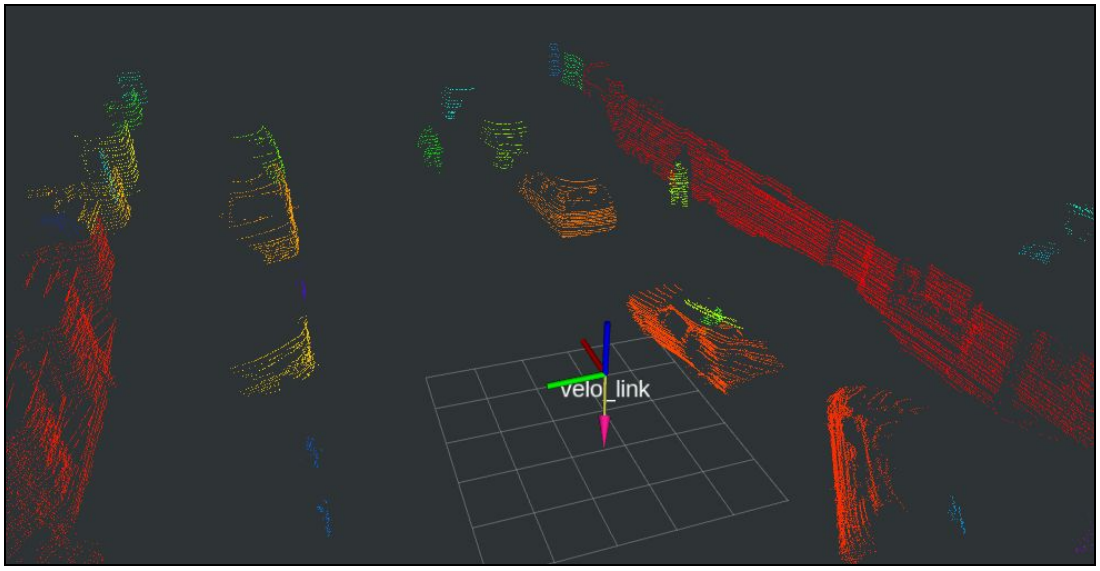
Figure 10: Graph of nodes and active topics generated with rqt_graph for test 1 (See annexes).

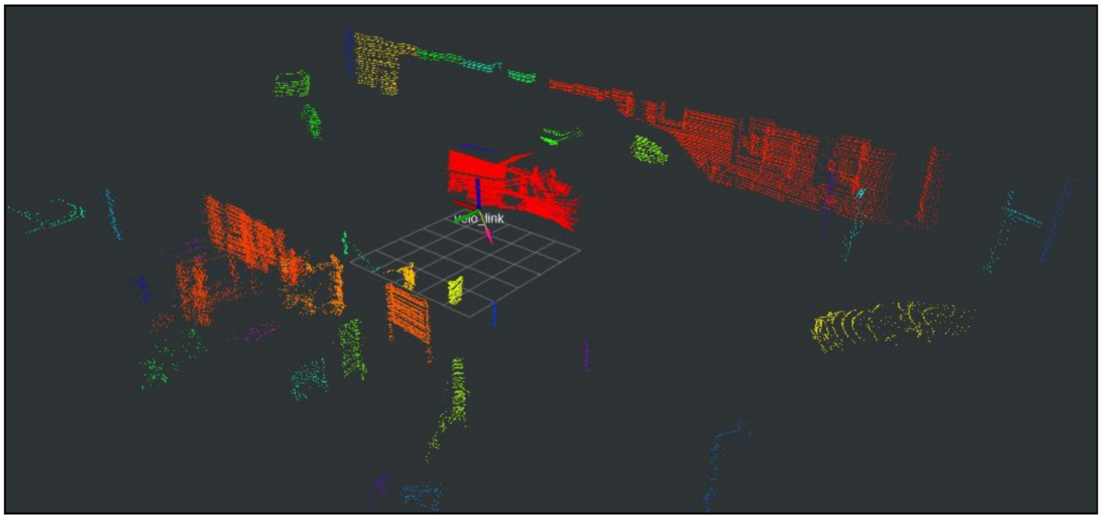


Annex 11: Segmentation obtained with the second test.

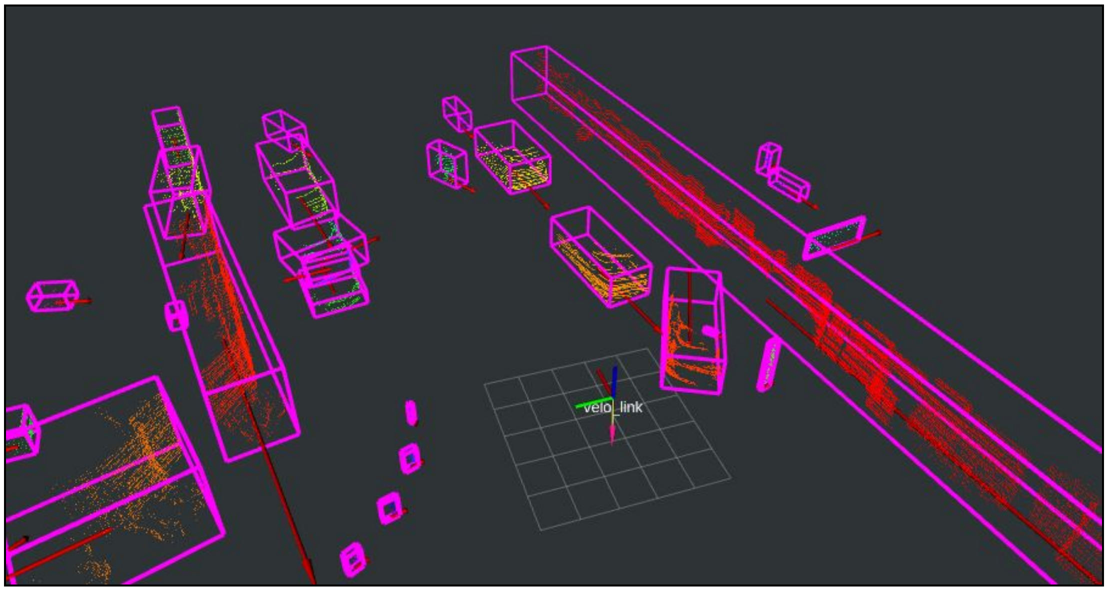


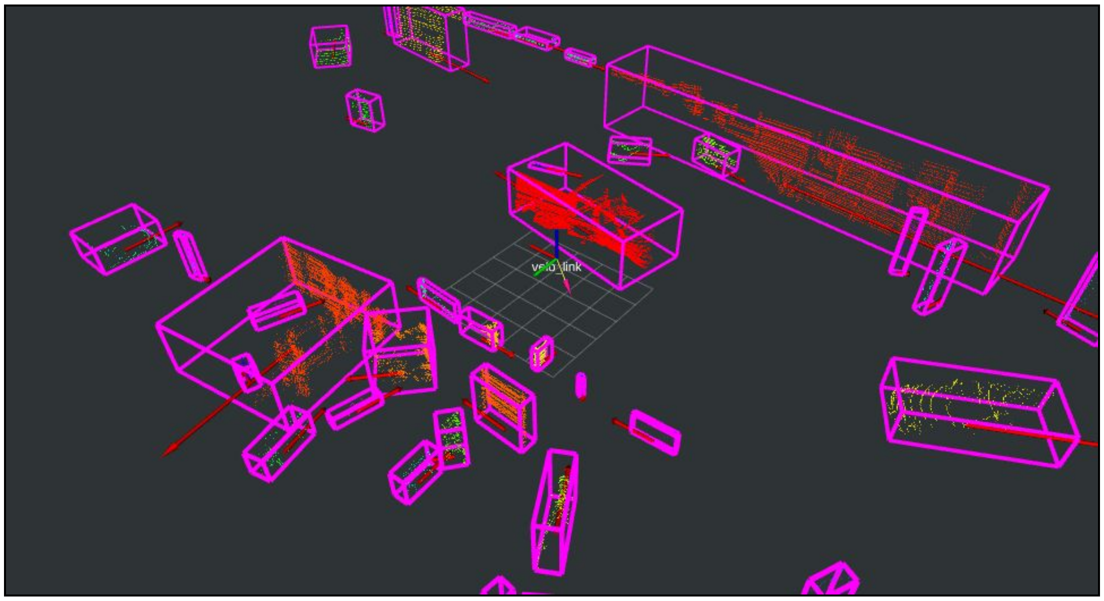
Annex 12: Segmentation obtained with the final test



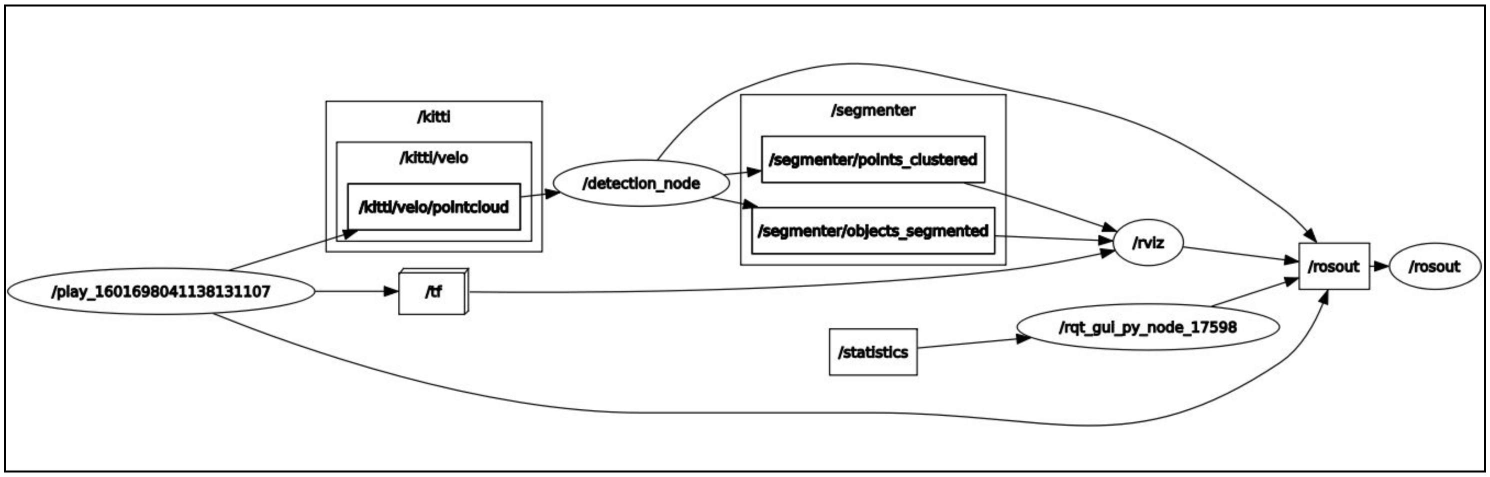


Annex 13: Results obtained from clustering in the ROS environment.





Annex 14: Visualization of clusters and container boxes.



Annex 15: Graph obtained with rqt_graph of nodes and active topics when executing clustering and container boxes.