Group :
BRIAND Naël
DOAN Alexis
DU Jack
MUKHANOV Dias
NGUYEN Denis

Project supervisor :
Mr. SIMOND Nicolas

<u>**2nd year project**</u> **:**

Big data for automated driving

SuburVAN

ENSEA
Beyond Engineering

Table of Contents

I - Introduction

In the daily routine of European urban life, 85% of the population relies on individual cars, often commuting alone to work. This preference stems from challenges accessing suburban train stations in peri-urban or rural areas. Addressing this, SuburVAN, a startup born from 30 years of INRIA's research on automated urban transport, is developing a solution – electric autonomous vans – to provide fast commuting to suburban train stations.

Currently testing a prototype in Greece, SuburVAN aims to revolutionize daily mobility. The startup presents two key projects: first, experimental validation of autonomous vehicle results in Greece or Rocquencourt, and second, a collaboration with INRIA on leveraging Big Data for automated driving.

Focusing on the second initiative, SuburVAN aims to collect crucial driving data along the (Le Chesnay-)Rocquencourt to Vaucresson route, ideal for testing their autonomous vans. Two sensor-equipped vehicles, initially operated by professionals, will gather data from LiDARs, cameras, GPS, and internal sensors.

Our tasks include **organizing the database**, establishing secure data access protocols, **discerning object semantics**, and **contributing to data annotation**. This dataset is pivotal for training models, propelling ongoing advancements in this groundbreaking transportation technology.

II - Technical Setup and Infrastructure

1. Hardware setup

To begin with, we had to think about the technical setup we are currently working with.
We had some technical constraints :

- Working with several softwares including ROS Noetic suite and Docker
- Manipulating big files
    - ➔ The data extracted from the LiDAR sensor are ~ 5 GB/min.
    - ➔ We have to visualize the point cloud processed by the LiDAR sensor.

The solution provided after consideration with Mr. Simond and Mr. Papazoglou is a centralized Linux server on which we connect with our own computer with personal user sessions. The server have these hardware specifications :

- **CPU** : Intel(R) Xeon(R) CPU E5-2650

➔ We need to have a fast CPU to have a fluent experience if we're all working on our own sessions at the same time

- **64 GB of memory RAM**

➔ Provide fluent experience if there are 5 simultaneous usage

- **GPU** : Nvidia Quadro RTX 4000

➔ By working on point clouds to visualize the data, we need a relatively strong GPU.

- **Storage** : KINGSTON SKC6002 (256 GB), Seagate Ironwolf ST20000NT001-3LT 20 TB

➔ The data extracted from the LiDAR sensor are ~ 5 GB/min, so we need high amount of storage (Seagate 20TB) and a fluent OS (Kingston 256GB SSD)

2. Software setup

After getting the hardware, we had to set up softwares we use.

2.1. OS installation

First of all, we had to clean up the hard drive and install a fresh new OS. We firstly chose to work on Ubuntu 22.04, the latest stable version of Ubuntu which is a typical OS for working on servers. But due to compatibility issues with ROS Noetic, which only works with Ubuntu 20.04 or below, we reinstalled Ubuntu with the 20.04 version.

- Explanation of the use of the Linux server

Setting up a Linux server for our project which involves LiDAR sensor data and the Robot Operating System (ROS) can offer several advantages.

1. Open Source and Compatibility with ROS
Linux is the preferred operating system for ROS, an open-source middleware framework. ROS is designed to work seamlessly with Linux, providing a robust and flexible platform for developing, managing, and controlling robotic systems. Using Linux ensures compatibility with ROS libraries, tools, and packages.
2. High Storage Capacity and Efficient File Systems
Linux provides support for high-capacity storage solutions. This is essential for managing and storing the large datasets generated by LiDAR sensors.
3. Security and access control
Security is crucial, especially when dealing with sensitive LiDAR data for a company such as SuburVAN.
Linux provides a wide range of security tools, access controls, and community-supported resources to help protect the server and data. We could download and automate the data transfer between our server and SuburVAN's one directly with a unified solution.
4. Command-Line Interface (CLI) for Efficient Management
Linux offers a powerful command-line interface (CLI) that allows us to efficiently manage and automate tasks. For a project involving LiDAR sensor data like ours, we can in the future automate and make a script of repetitive tasks to make it more efficient.

- Explanation of the installation process

The installation process is clear and fast :
- Download the right Ubuntu Image version (ISO image)
- Use Rufus (on Windows) or balenaEtcher to create a bootable USB stick (you'll have to flash the USB with the downloaded ISO image you have downloaded before)
- Insert your fresh new USB stick in the computer and power it on
- Enter the UEFI mode by pressing F12 (may change depending of the manufacturer)
- Change the boot order in the UEFI by putting the USB stick first
- Save and exit
- Follow the instructions to install Ubuntu 20.04 with the information displayed

2.2. User creation

We now have to create for each one of us a user session on which we could work on simultaneously.

First of all, we have to update the package from Linux.

- Update the list :

```
$ sudo apt update
```

- Update every package from the list :

```
$ sudo apt upgrade
```

Now that the packages are all updated, we have to create every user session.

- Create manually every user session
- Add a user in the sudo group :

```
$ usermod -aG sudo username
```

Make sure to replace *username* with the right session username (no uppercase)

- Repeat this step for every session
- Restart the sessions to apply modifications

➔ We did think about security and access rights on the server, especially as it contains sensitive data from SuburVAN and gives access to the school network with more rights than we should, as students, but as our own sessions are protected by passwords and there are logs of what is done.
The sudo (superuser) rights were needed here for more flexibility as we had many packages to install.
We did not give root rights.

2.3. Mounting the 20 TB hard disk

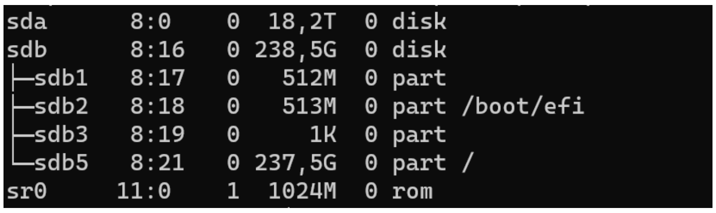
To deal with such a large amount of data, we chose to work with a dual disk system. A 256 GB SSD for the OS (Ubuntu 20.04) to work fluently and a 20 TB hard disk.
We now have to mount the second hard disk by following these steps :

1. Show all the running harddrives

```
$ sudo fdisk -l
```

Or, the following one to show all the hard drives installed

```
$ lsblk
```



```
sda       8:0     0  18,2T  0 disk
sdb       8:16    0 238,5G  0 disk
├─sdb1    8:17    0   512M  0 part
├─sdb2    8:18    0   513M  0 part /boot/efi
├─sdb3    8:19    0     1K  0 part
└─sdb5    8:21    0 237,5G  0 part /
sr0      11:0     1  1024M  0 rom
```

2. Create a directory to mount the drive into.

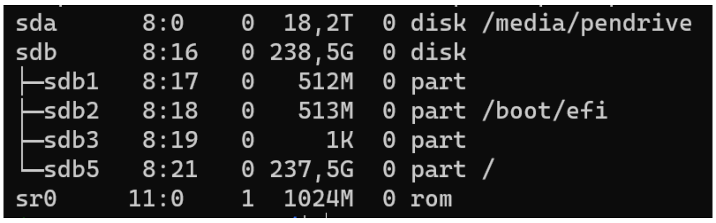
```
$ sudo mkdir /media/pendrive
```

3. Mount the drive to /media/pendrive directory

```
$ sudo mount /dev/sda /media/pendrive
```

4. Check if the drive has been mounted

```
$ lsblk
```



```
sda       8:0     0  18,2T  0 disk /media/pendrive
sdb       8:16    0 238,5G  0 disk
├─sdb1    8:17    0   512M  0 part
├─sdb2    8:18    0   513M  0 part /boot/efi
├─sdb3    8:19    0     1K  0 part
└─sdb5    8:21    0 237,5G  0 part /
sr0      11:0     1  1024M  0 rom
```

Here, we can see that the sda hard drive of 18,2 TB capacity has been mounted to the directory "/media/pendrive".

2.4. SSH Server Configuration

Now that the server is working and has different sessions, we have to set up the network settings in order to access the server from our own computer.

- Explanation of the use of the SSH server

Setting up a SSH Linux server for our project which involves LiDAR sensor data and the Robot Operating System (ROS) can offer several advantages.

1. Remote Access via SSH
Linux servers support SSH, a secure and efficient protocol for remote access. With SSH, up to five individuals can remotely connect to the server from different locations. This allows team members to collaborate on the project, access files, and execute commands securely over an encrypted connection, enhancing teamwork and flexibility.

2. Security and access control
As it was previously mentioned, by using SSH, connections to the server are encrypted, minimizing the risk of unauthorized access. Linux allows fine-grained access control through user permissions and firewall configurations, ensuring that only authorized team members can access specific resources and data on the server.

3. Command-Line Interface (CLI) for Efficient Management
Linux's powerful command-line interface (CLI) enables efficient server management, especially when dealing with complex tasks such as data processing, file management, and system configuration. Team members can use SSH to access the server's CLI remotely, facilitating centralized control and administration of the project.

- Configuration of IP address and SSH

We have been working on a server that has been established in the school network. This provides advantages but also drawbacks.

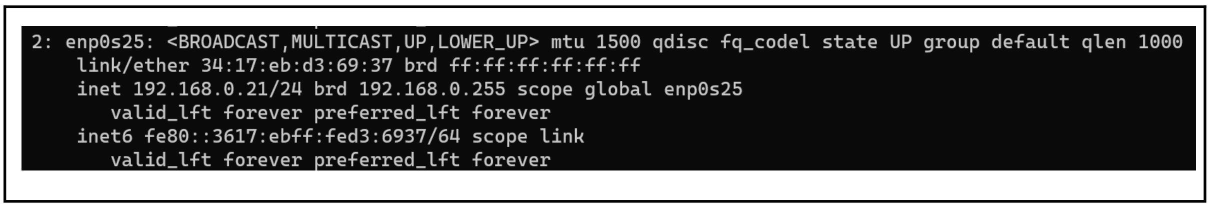
Advantages :
- The whole network system is already implemented. Our Linux server will be one device among the others. This means that a fixed IP address can be provided by the IT department of our school.
- We don't have to set up the whole security system to protect our server as it's already done.

Drawbacks :
- We have to follow strict network rules for safety reasons. All the configurations are set and we can't choose another port or IP address.

❖ Server IP address setup

To configure the IP address of the server, we have to follow these steps :

```
$ ip addr //to view all the network interfaces
```



```
2: enp0s25: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 34:17:eb:d3:69:37 brd ff:ff:ff:ff:ff:ff
    inet 192.168.0.21/24 brd 192.168.0.255 scope global enp0s25
       valid_lft forever preferred_lft forever
    inet6 fe80::3617:ebff:fed3:6937/64 scope link
       valid_lft forever preferred_lft forever
```

<u>**Some explanation :** (cf. Basics of network courses, 2nd year I.T. major at ENSEA)</u>

Interface Name (2: enp0s25) :
- ➔ 2 is the index number of the network interface.
- ➔ enp0s25 is the name of the interface.

Interface Status : <BROADCAST, MULTICAST, UP, LOWER_UP>: This section indicates the current status of the interface.
- ➔ BROADCAST: The interface supports broadcast packet transmission.
- ➔ MULTICAST: The interface supports multicast packet transmission.
- ➔ UP: The interface is enabled.
- ➔ LOWER_UP: The physical link is up.

MTU (Maximum Transmission Unit) :
mtu 1500: The maximum packet size this interface can transmit without fragmentation.

Qdisc (Queueing Discipline) :
- ➔ qdisc fq_codel: This indicates the queueing discipline used. Here, it's "fq_codel," which is a queue management algorithm used for network congestion control.

Interface State :
- ➔ state UP: The state of the interface is active.

Default Group :
- ➔ group default: This refers to the default group to which this interface belongs.

Queue Length (Qlen) :
- ➔ qlen 1000: The length of the queue, indicating the maximum number of packets that can be waiting in the queue for this interface.

MAC Address (link/ether) :
- ➔ link/ether 34:17:eb:d3:69:37: The MAC (Media Access Control) address of the interface.

IPv4 Address (inet) :
- ➔ inet 192.168.0.22/25: The IP address assigned to this interface with a subnet mask of 25 bits.
- ➔ brd 192.168.0.254: The broadcast address for the subnet.

Scope :
- ➔ scope global: The scope of the IP address is global, meaning it is routable on the network.

Address Lifetime (valid_lft and preferred_lft) :
- ➔ valid_lft forever: The validity duration of the address is set as "forever."
- ➔ preferred_lft forever: The preferred duration of the address is also set as "forever."

IPv6 Address (inet6) :

➔ fe80::3617:ebff:fed3:6937/64: The IPv6 address of the interface.

IPv6 Address Scope:

➔ scope link: The scope of the IPv6 address is limited to the link-local network.

We want to configure the “enp0s25” network interface. Its IP address is 192.168.XX.XX and the port we use is XX.

- **Edit the netplan configuration file :**

```
$ sudo nano /etc/netplan/01-network-manager-all.yaml
```

- **Add IP configuration :**

```
network:
  version: 2
  renderer: networkd
  ethernets:
    enp0s25:
      addresses:
        - 10.10.XX.XXX/XX
      gateway4: 192.168.XX.XX
      nameservers:
        addresses: [10.10.2.250, 10.10.17.220]
```

➔ save and exit

- **apply configuration changes**

```
$ sudo netplan apply
```

- **verify configuration**

```
$ ip addr show
```

- **check connectivity**

```
$ ping www.ensea.fr
```

❖ Network configuration explained

1. IP Address :

An IP address is a **numerical label assigned to each device connected** to a computer network that uses the Internet Protocol (IP) for communication.
IP addresses serve to uniquely identify and locate devices on a network. There are two main versions of IP addresses:
IPv4 (192.168.XX.XX) and IPv6 (2001:0db8:85a3:0000:0000:XXXX:XXXX:XXXX).

2. Subnet Mask :

The subnet mask accompanies the IP address to divide the IP address space into smaller subnets. The subnet mask indicates which bits of the IP address are used to identify the network and which bits are reserved for hosts.
For example, in an IP address like 192.168.XX.XX with a subnet mask of 255.255.254.0 (or simply /24 in CIDR notation), the first three octets (24 bits) are reserved to identify the network, and the last octet (8 bits) is reserved for hosts. This means there can be **256 IP** addresses (2^8) in this network.

3. Gateway :

The gateway, also known as a router, is a device that connects different computer networks. It acts as a bridge between the **local** network and **other** networks, such as the Internet. When a device in a local network wants to communicate with a device outside that network, it sends the **traffic to the gateway**, which is responsible for routing the traffic to the appropriate destination.
In a network configuration, the gateway is typically the IP address of the router connecting the local network to the wide-area network (WAN).

4. DNS Server :

The DNS server is the server that translates human-readable domain names (such as google.com) to IP addresses that are understandable on a network (8.8.8.8 for www.google.com).
When a user enters a domain name into a web browser or attempts to access any online service, the computer initiates a DNS resolution process.
The DNS server is responsible for looking up the corresponding IP address associated with the provided domain name.

❖ SSH connection setup

- Update the list of available packages

```
sudo apt update
```

- Install the OpenSSH server package

```
sudo apt install openssh-server
```

- Attempt to SSH into the server using the IP address

```
ssh username@10.10.3.151
```

- Edit the local hosts file to associate the server's IP with a domain name

```
sudo nano /etc/hosts
10.10.3.151 routeur.ensea.fr
```

- Attempt to SSH into the server using the domain name

```
ssh username@routeur.ensea.fr
```

- Edit the SSH server configuration file to change the listening port to 2222 (an exemple, depending of the network configuration)

```
sudo nano /etc/ssh/sshd_config
Port 2222
```

- Restart the SSH service to apply the configuration changes

```
sudo service ssh restart
```

- SSH into the server using the modified port and domain name

```
ssh username@routeur.ensea.fr
```

- Enter the password to access

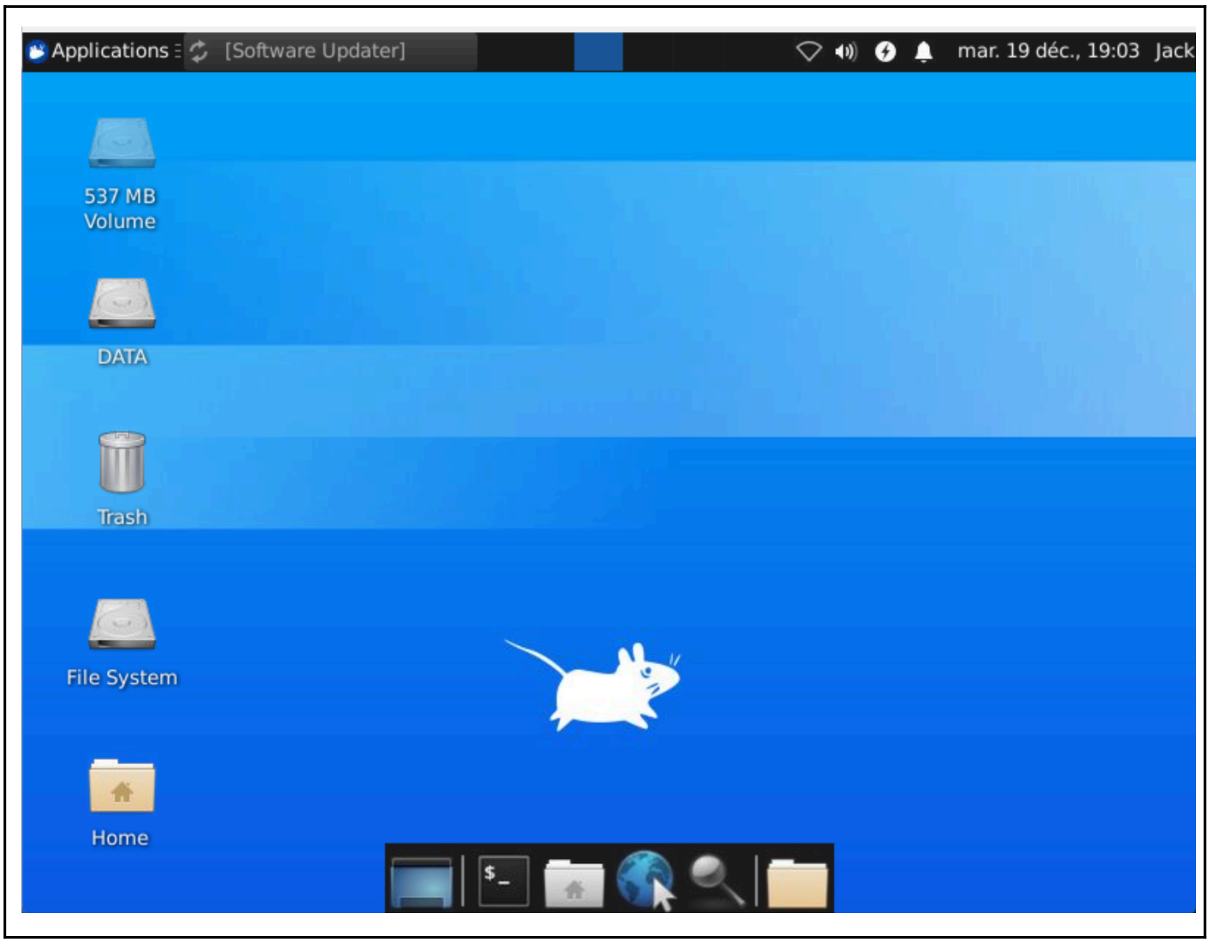
❖ Connection interface we can use

- **MobaXterm**

MobaXterm is a versatile and enhanced terminal for Windows that integrates various network tools, X11 server, tabbed SSH client, and more.
For the cloud points visualization with RVIZ, we may want to have a graphic feedback from the server. MobaXterm is not the right solution.

- **x2Go**

x2Go is an open-source remote desktop solution that enables you to access a graphical desktop of a computer over a low bandwidth connection.
It is designed to provide a seamless remote desktop experience and supports various operating systems, including Linux, Windows, and macOS.



<u>Be careful</u> : X2go has compatibility issues related to the desktop environment and works **only** with **XFCE**.

- **Any terminal on Windows/Ubuntu/MacOS**

We can connect using any terminal under Windows, Ubuntu or MacOS.

```
$ ssh username@servername.ensea.fr
$ type password of your session linked to your username
```

The connection through the terminal is much more convenient and faster but we don't have any graphical feedback.

2.5. ROS/Rviz (Visualisation) Integration

- Presentation of ROS (Robot Operating System) and Rviz

ROS, or Robot Operating System, is an open-source middleware framework designed to develop and control robots. Contrary to its name, ROS is not an operating system in the traditional sense but rather a collection of software libraries and tools that facilitate the development of robotic software. ROS provides services such as hardware abstraction, device drivers, communication between processes, and package management, making it easier for developers to create complex robot applications.

Rviz, short for ROS Visualization, is a powerful 3D visualization tool that comes bundled with ROS. It allows users to visualize and interact with various aspects of a robotic system in a three-dimensional space. Rviz is particularly useful for debugging and understanding the behavior of robot systems.

ROS and Rviz work hand-in-hand to provide a comprehensive and flexible platform for developing, simulating, and controlling robotic systems. ROS abstracts the complexity of robot development, while Rviz enhances the understanding and debugging process through intuitive 3D visualization. Together, they empower developers to create sophisticated robotic applications across various domains.

- Role of ROS/Rviz in the SuburVAN project

We use ROS Noetic to visualize point clouds sent by SuburVAN on Rviz. SuburVAN send us files recovered with the LiDAR sensor. These files are .bag file and they suffered a pretreatment by SuburVAN. We have to segment these files then we visualize on Rviz.

- Practical usage and application

Prepare the ROS environment by sourcing the setup file:

```
$ source /opt/ros/noetic/setup.bash
```

Initiate the central ROS component with:

```
$ roscore
```

It serves as the master node, parameter server, and naming service, playing a vital role in coordinating communication, dynamic parameter storage, and network address resolution within the ROS system.

In a new terminal tab/ window, visualize the content of the '.bag' data files using the command:

```
$rosbag play --clock ‘filepath/filename.bag’
```

The '--clock' option ensures synchronization between the ROS time within the bag file and the current time on the machine.

In another tab/window, list the available data formats using :

```
$ rostopic list
```

To select a specific data format, execute :

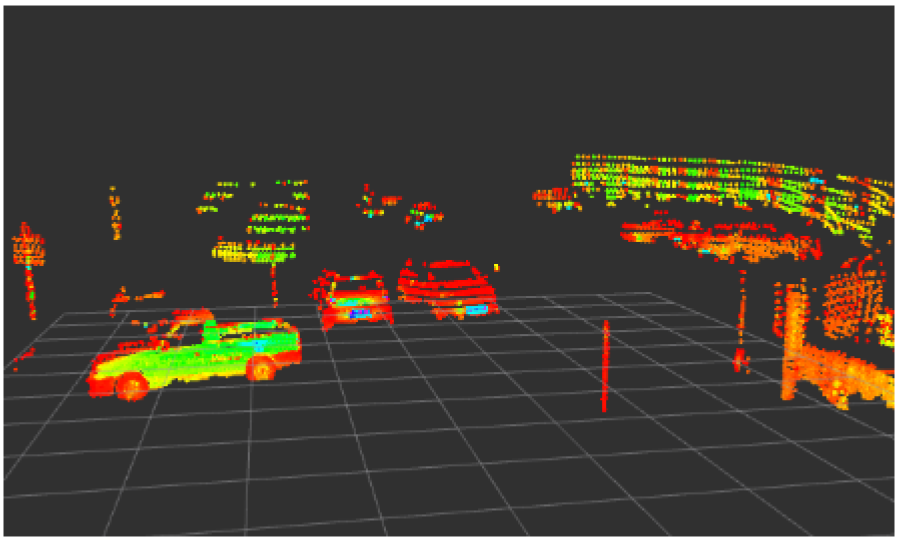
```
$ rostopic echo/data_format
```

Note: This will not work if the bag file is not playing in another tab/window

Open Rviz using :

```
$ rosrun rviz rviz
```

In the interface that appears, choose the desired data format (e.g., /Pointclouds), and the running bag file will be visualized in Rviz.



2.6. Docker Integration

To utilize ROS on our SSH server, SuburVAN suggested employing Docker.

- What is Docker ?

Docker is an open-source platform that automates the deployment, management, and containerization of applications. Containers are lightweight and portable environments that encapsulate an application and its dependencies, ensuring consistent execution across diverse computing environments.

The primary benefits of Docker include :

- **Portability**: allowing consistent execution on various systems.
- **Isolation**: preventing conflicts between applications and ensuring consistency across development, testing, and production environments.
- **Speed**: with containers starting in seconds for rapid scaling and resource-efficient management.
- **Simplified dependency management**: as Docker encapsulates all necessary libraries and components within a container.

- Why use Docker ?

In our context, using Docker allows us to take advantage of the :

- **Resource Efficiency:** Docker's lightweight containers and shared kernel enhance resource efficiency, ideal for working with limited resources on the server.
- **Portability:** Docker's portability enables easy sharing and deployment of your ROS application on various servers, eliminating concerns about underlying infrastructure differences.
- **Isolation:** Docker isolates ROS and its dependencies from the host system, preventing interference with existing software on the server.

Beyond these advantages, incorporating Docker into the project also provides :

- **Security:** Docker's container isolation enhances security, containing potential vulnerabilities within the ROS application and reducing the host system's attack surface.
- **Consistent Development Environment:** Docker ensures a uniform development environment by packaging ROS and project dependencies into a container, reducing compatibility issues across different machines.
- **Easy Deployment:** Docker simplifies deployment by allowing you to start a ROS container on the server without manual installation, particularly useful for remote servers.
- **Versioning and Rollback:** Docker supports image versioning, enabling easy rollback to a previous state for maintaining reproducibility in your ROS application.

- <u>Docker and ROS Noetic integration</u>

❖ Installation

1 - Packages update

```
$ sudo apt update
$ sudo apt upgrade
```

2 - Installation of the required dependencies

```
$ sudo apt install apt-transport-https ca-certificates curl
software-properties-common
```

3 - Add Docker official GNU Privacy Guard (GPG) in system

```
$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo
gpg --dearmor -o /usr/share/keyrings/docker-archive-keyring.gpg
```

<u>Note:</u> if curl is not installed yet :

```
$ sudo apt-get install curl
```

4 - Add stable Docker repository in the APT manager and update packages

```
$ echo "deb [arch=amd64
signed-by=/usr/share/keyrings/docker-archive-keyring.gpg]
https://download.docker.com/linux/ubuntu $(lsb_release -cs)
stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
$ sudo apt update
```

5 - Docker Engine installation

```
$ sudo apt install docker-ce docker-ce-cli containerd.io
```

6 - Start the Docker service

```
$ sudo systemctl start docker
```

Optional :

- Add users in the Docker group to avoid using sudo at each command :

```
$ sudo usermod -aG docker username
```

Replace *username* by the right user username

- Make Docker run automatically on system startup :

```
$ sudo systemctl enable docker
```

❖ Installation and use of ROS Noetic

Prerequisite Docker vocabulary :

- **Dockerfile:** A text file with build instructions for a Docker image, specifying the base image, environment setup, and application configuration.
- **Docker image:** A self-sufficient package with everything to run software, serving as a blueprint for Docker containers.
- **Docker container:** A runnable instance isolated from the host system, created from a Docker image, encapsulating an application and its dependencies.

1 - Get a Dockerfile and build the ROS image

SuburVAN supplied a Dockerfile on Github. We simply cloned the repository, navigated to the relevant workspace with the Dockerfile, and built the ROS image :

```
$ sudo docker build -t image_name .
```

2 - Run a container with a shared folder

We can now initiate a container from this image. To enable ROS access to locally stored data on the server, we must establish a shared folder within the Docker container :

```
$ sudo docker run -it  -v
data_folder_server_path:container_folder_path image_ID
```

Notes: Create the folder in the container if it does not exist.
Replace *image_ID* with the ROS image ID.

3 - Run ROS within the container

- Run the 'roscore' command
- Open a new terminal window/tab
- Open another session in the same container

```
$ sudo docker exec -ti container_ID bash
```

- Load the environment variables to ensure a clean and properly configured ROS environment is set up before executing any subsequent commands.

```
$ set -e
$ source "/opt/ros/noetic/setup.bash" --
$ exec "$@”
```

- ROS is operational
- Repeat these three last steps each time a new window/tab is needed

4 - Do not terminate the container upon exiting.

One common mistake is exiting the Docker container using the 'exit' command, which closes the session without an easy way to restart. Get out of the container using 'Ctrl+P' followed by 'Ctrl+Q' to detach the terminal.

To resume a session in the same container, either run :

```
$ docker attach container_name
```

Or utilize the 'docker exec' command mentioned in the **Run ROS within the container** section.

Keep in mind that detaching the terminal keeps a container running, consuming resources (RAM, CPU, Storage, Network).

For longer breaks, consider stopping the container temporarily with:

```
$ sudo docker stop container_name
```

To restart it later:

```
$ sudo docker start container_name
```

➔ More useful Docker commands in Appendix.

2.7. Data Segmentation using DBSCAN Algorithm in C++

Data segmentation is a critical aspect of the SuburVAN project, particularly in the context of point clouds derived from Lidars, cameras, GPS, and accelerometers. The primary objective is to achieve in-depth analysis and accurate interpretation of information gleaned from these sensors.

By employing the Density-Based Spatial Clustering of Applications with Noise (DBSCAN) algorithm in C++, we aim to enhance the segmentation process. This algorithm is instrumental in identifying clusters or groups of points that share similar characteristics. The segmentation of point clouds allows for the recognition of significant structures or patterns within the data, thereby facilitating a more precise understanding of the surrounding environment.

This strategic approach not only contributes to efficient data processing but also lays the foundation for subsequent phases, including the development of the database architecture and the preparation for machine learning model training.

III - Progress Update

1. Summary of the current project status

The SuburVAN project has reached a noteworthy milestone in its technical implementation. With a **fully operational SSH server** providing round-the-clock access, **ROS and Rviz** are seamlessly **installed and functional** on the server, even though they are not yet encapsulated in a Docker container. The team has developed proficiency in navigating the Rviz interface and effectively **displaying Lidar data Point Clouds**. Currently, the focus is on **data segmentation**, a critical step preceding the construction of the database architecture. This progress positions the project well for the upcoming phases, establishing a robust foundation for further development.

2. Achievements and challenges encountered so far

❖ **Issues with the OS installation process :**

Firstly, we had issues with the computer provided as it wasn't showing any sign of life. It was in a reboot loop mode.

➔ It turns out that a stick of memory RAM **was not correctly connected** to the motherboard which is why the computer failed the boot process.

Secondly, we also had issues with the graphic card after installing Ubuntu 20.04. We had no graphic output from the computer. The graphic card (Nvidia Quadro RTX 4000) needed specific drivers that Ubuntu 20.04 couldn't provide (and that Ubuntu 22.04 had, we didn't have issues on it).

➔ We had to install the driver manually by changing the graphic card with an older one that could work and have graphic output. Then, we had to install the driver by :

1) **Add the NVIDIA Drivers PPA**
```
$ sudo add-apt-repository ppa:graphics-drivers/ppa
```
2) **Update the package list**
```
$ sudo apt update
```
3) **Install the Specific Driver Version compatible with the Quadro RTX 4000**
```
$ sudo apt install nvidia-driver-418=418.88-0ubuntu0~gpu18.04.1
```
4) **Reboot the system**
```
$ sudo reboot
```
5) **Check if the driver has been successfully installed**
```
$ nvidia-smi
```

➔ If the installation is correct, just swap the graphic card and the computer should work properly.

❖ **Issues with Rviz in Docker and in SSH :**

Running Rviz within a Docker container poses significant challenges due to the interaction between the containerized environment and the graphical user interface (GUI) requirements of RViz, this leading to various problems :

- **X Server access issue in the Container :** RViz requires access to the X (X11) server for graphical rendering, but Docker containers run in isolated environments.
- **OpenGL support :** RViz relies on OpenGL for rendering, and the container might lack proper OpenGL support.
- **Graphics Drivers incompatibility :** The graphics drivers on the host machine might not be compatible or might not support hardware acceleration inside the container.

We explored different potential solutions:

1 - Adapt the Dockerfile
The goal was to add the dependencies and libraries necessary to run Rviz when building the Docker image.

➔ However, the container failed to identify certain libraries, and the X server access problem persisted.

2 - Use the NVIDIA runtime image
Using this image could help resolve the hardware acceleration support issue. The NVIDIA Container Runtime enables Docker containers to access NVIDIA GPUs efficiently, providing improved performance for GPU-accelerated applications like RViz.

➔ The documentation we found applies this with previous versions of ROS like ROS Indigo or Melodic, so we could not adapt this method for ROS Noetic yet.

3 - Use a Docker Compose File
A **Docker Compose file** is a YAML configuration file used to define and configure multiple Docker containers, their settings, and the services they provide. It allows you to define an entire multi-container Docker application stack, specifying details such as container images, volumes, network configurations, and environment variables.
We attempted to create separated containers for Rviz and ROS and link them through the Compose File.

➔ We haven't yet acquired proficiency in generating a proper Docker Compose File, and some of the examples available also utilized outdated versions of ROS.

4 - ‘xhost +’ command
Many sources were fixing the X server access issue by using the **‘xhost +’** command. Using this command is highly discouraged due to significant security risks. It opens unrestricted access to the X Window System, allowing any remote host to display graphical applications on the screen without limitations. This exposes the system to potential malicious exploitation, compromising security, privacy, and overall system integrity.

➔ Despite the potential resolution it could have offered, we opted not to take the risk, particularly considering the fundamental importance of security measures on an SSH server.

There were also issues related to the SSH connection.

- **Graphical incompatibility between the local machine and the server :** X11 forwarding needs to be correctly set up to display graphical applications on the local machine.
- **SSH Server Configuration:** The SSH server on the host may not be configured to allow X11 forwarding.

➔ Temporary solution : As there was a need to perform server-related tasks, we, with SuburVAN's approval, opted to install ROS directly on the server. Currently, Rviz is functional on the server and operates seamlessly when connected via SSH from a Linux terminal by adding **'-X'** after **'ssh'** when connecting on the server.

For now, we have not yet identified the method to use it on x2go, we have to include X11 in the client.

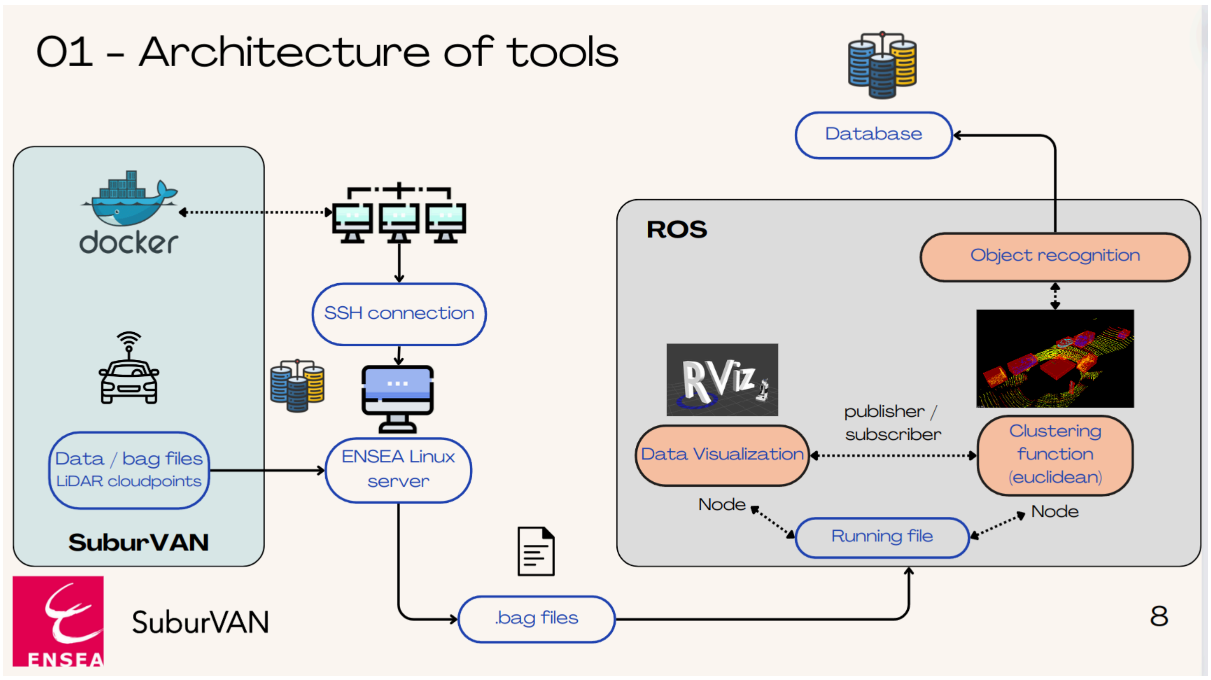
Rviz is essential for visually representing our actions, although its necessity for the database architecture is minimal. Ideally, the data structure work should be conducted within a Docker container. At present, we find ourselves in a situation with a dual installation of ROS, both with and without Docker. While not an optimal approach, we are adopting this method temporarily as we await a resolution for using Rviz within Docker.

❖ **Issues with displaying Point clouds on Rviz**

For a while, we encountered challenges displaying on Rviz the Lidar-generated Point Clouds processed upstream by SuburVAN. Rviz struggled to recognize the format of the Lidar data, despite various attempts to resolve the issue with guidance from the company. We explored solutions such as acquiring the correct packages and implementing code in C.

➔ It was later discovered that the Lidar data were originally in the robot/Point Clouds format, which Rviz does not support. The key to resolving the problem was ensuring the data were in the correct format, and once that adjustment was made, everything worked as intended.

V - Relevant information of the report



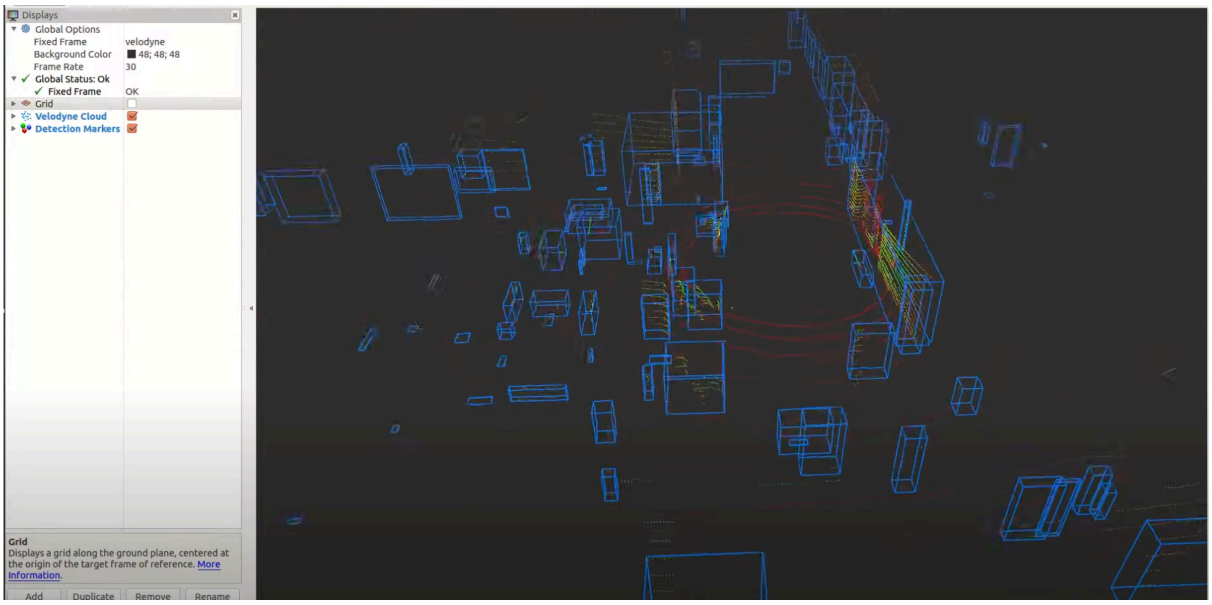
ROS Wiki has been a really useful documentation throughout the project. Most of the information we get are from here. So, pay attention to the whole documentation, especially the ROS Tutorials.
It helps in understanding the basic architecture of a ROS project.
ROS architecture is based on :

- a *Package* (catkin)
- Nodes: A node is an executable that uses ROS to communicate with other nodes.
- Messages: ROS data type used when subscribing or publishing to a topic.
- Topics: Nodes can *publish* messages to a topic as well as *subscribe* to a topic to receive messages.

To work more efficiently, we decided to work on different projects. I have been working on existing projects and bringing out useful information from them. We have looked at different projects on GitHub. A few were relevant and I've been working on this one :
Adaptive clustering from yzrobot

This project aims to develop a clustering algorithm that creates from LiDAR Cloud Points different clusters of different sizes. It's actually what we aim to do.

Adaptive Clustering: A lightweight and accurate point cloud clustering method



Displays
Global Options
Fixed Frame velodyne
Background Color ■ 48; 48; 48
Frame Rate 30
✓ Global Status: Ok
✓ Fixed Frame OK
Grid ☐
Velodyne Cloud ✓
Detection Markers ✓

Grid
Displays a grid along the ground plane, centered at the origin of the target frame of reference. More Information.

To adapt this project to ours, we have to understand its architecture well and especially the CMakeList.txt file that declares dependencies between the different packages, the executables, the libraries and the compilation parameters.
We first have to rebuild the *CMakeList.txt* file and especially :

- The messages that are generated are using the *ClusterArray.msg* file.

```
$ add_message_files(FILES ClusterArray.msg)
```

- ROS generates the necessary files to support the messages that are defined within the package (which are defined such as “std_msgs”, “sensor_msg” and “visualization_msg”)

```
$ generate_messages(DEPENDENCIES std_msgs sensor_msgs visualization_msgs)
```

- The paths of our project, especially with the *adaptive_clustering.cpp* which is an executable from the *adaptive_clustering* package.

```
$ add_executable(adaptive_clustering
adaptive_clustering/src/adaptive_clustering.cpp)
```

However, we faced different issues :

- Building the project

```
$ catkin_make
```

→ Building the project has not been possible. We had issues with compiling the *adaptive_clustering.cpp* file that we modified to adapt it to our project and especially with :

- Undefined references to PCL symbols:

```
$ // PCL
$ #include <pcl_conversions/pcl_conversions.h>
$ #include <pcl/filters/voxel_grid.h>
$ #include <pcl/segmentation/extract_clusters.h>
$ #include <pcl/common/common.h>
$ #include <pcl/common/centroid.h>
```

We need to include different PCL libraries such as *pcl_conversions* that provide functions to convert Cloud points into message data.

These errors indicate that the compiler cannot find the definitions of certain functions or classes in the PCL library while creating the executable file. A PCL symbol refers to any function, class, variable defined in the **Point Cloud Library** (PCL). These symbols represent the functionalities and capabilities provided by the PCL library for processing and analyzing point cloud data. This could be due to configuration issues, incorrect links, or missing dependencies that I wasn't able to find.

- Missing VTK files:

The compiler reports that some VTK files are missing, which may indicate problems installing or configuring VTK.
VTK files directories are indicated by the *CMakeLists.txt* file by the variable

```
$ // PCL
$ PCL_INCLUDE_DIRS
$ include_directories(
        ...
        /usr/include/pcl-1.10
        ...)
```

But still, there's an environment configuration issue.

→ What is relevant in this project ?

adaptive_clustering.cpp is a really interesting file with relevant functions that we used in our project : **pointCloudCallback function**
This function is the callback that gets executed whenever a new point cloud message is received.

It applies several sub-functions :

- It **converts the ROS message to a PCL point cloud**

```
$ /*** Convert ROS message to PCL ***/
$  pcl::PointCloud<pcl::PointXYZI>::Ptr pcl_pc_in(new
pcl::PointCloud<pcl::PointXYZI>);
$  pcl::fromROSMsg(*ros_pc2_in, *pcl_pc_in);
```

- It performs **downsampling** (to take up less space) and filters out points based on the z-axis range (**to remove the ground**). After discussing with Carlos, It has been clear that the data has been preprocessed before being concatenated into a bag file and this is exactly what has been done.

```
$ /*** Downsampling + ground & ceiling removal ***/
$  pcl::IndicesPtr pc_indices(new std::vector<int>);
$  for(int i = 0; i < pcl_pc_in->size(); ++i) {
$    if(i % leaf_ == 0) {
$      if(pcl_pc_in->points[i].z >= z_axis_min_ && pcl_pc_in->points[i].z <=
z_axis_max_) {
$  pc_indices->push_back(i);
$      }
```

- It divides the **point cloud into nested circular regions** to allow adaptive clustering and efficient processing as the algorithm could be adapted to the size and density of each region.
-

```
$ /*** Divide the point cloud into nested circular regions ***/
$  boost::array<std::vector<int>, region_max_> indices_array;
$  for(int i = 0; i < pc_indices->size(); i++) {
$    float range = 0.0;
$    for(int j = 0; j < region_max_; j++) {
$      float d2 = pcl_pc_in->points[(*pc_indices)[i]].x *
pcl_pc_in->points[(*pc_indices)[i]].x +
$      pcl_pc_in->points[(*pc_indices)[i]].y *
pcl_pc_in->points[(*pc_indices)[i]].y +
$      pcl_pc_in->points[(*pc_indices)[i]].z *
pcl_pc_in->points[(*pc_indices)[i]].z;
$      if(d2 > range * range && d2 <= (range+regions_[j]) * (range+regions_[j])) {
$            indices_array[j].push_back((*pc_indices)[i]);
$            break;
$      }
$      range += regions_[j];
$    }
$  }
```

- It performs **Euclidean clustering** on the different regions.

This is the main part of the algorithm we want to perform. Euclidean clustering is a technique that measures the straight-line distance between two points in Cartesian coordinates in order to group nearby points together based on their spatial proximity in 3D space. It is a really simple method for grouping spatially related point clouds.

```
$ *** Euclidean clustering ***/
$  float tolerance = 0.0;
$  std::vector<pcl::PointCloud<pcl::PointXYZI>::Ptr,
Eigen::aligned_allocator<pcl::PointCloud<pcl::PointXYZI>::Ptr > > clusters;

$  for(int i = 0; i < region_max_; i++) {
$    tolerance += 0.1;
$    if(indices_array[i].size() > cluster_size_min_) {
$      boost::shared_ptr<std::vector<int> > indices_array_ptr(new
std::vector<int>(indices_array[i]));
$      pcl::search::KdTree<pcl::PointXYZI>::Ptr tree(new
pcl::search::KdTree<pcl::PointXYZI>);
$      tree->setInputCloud(pcl_pc_in, indices_array_ptr);
$       std::vector<pcl::PointIndices> cluster_indices;
$       pcl::EuclideanClusterExtraction<pcl::PointXYZI> ec;
$       ec.setClusterTolerance(tolerance);
$       ec.setMinClusterSize(cluster_size_min_);
$       ec.setMaxClusterSize(cluster_size_max_);
$       ec.setSearchMethod(tree);
$       ec.setInputCloud(pcl_pc_in);
$       ec.setIndices(indices_array_ptr);
$       ec.extract(cluster_indices);

$       for(std::vector<pcl::PointIndices>::const_iterator it =
cluster_indices.begin(); it != cluster_indices.end(); it++) {
$         pcl::PointCloud<pcl::PointXYZI>::Ptr cluster(new
pcl::PointCloud<pcl::PointXYZI>);
$         for(std::vector<int>::const_iterator pit = it->indices.begin(); pit !=
it->indices.end(); ++pit) {
$           cluster->points.push_back(pcl_pc_in->points[*pit]);
$     }
$         cluster->width = cluster->size();
$         cluster->height = 1;
$         cluster->is_dense = true;
$   clusters.push_back(cluster);
$       }
$     }
$   }
```

Overall Functionality

The script subscribes to a ROS topic that publishes sensor data in the form of a PointCloud2 message. It processes this data to segment out the main plane (typically the ground or other large, flat surfaces) and then finds clusters of points that are not part of the main plane, treating these clusters as potential obstacles.

Detailed Breakdown:

1. Imports and ROS/PCL Setup:
    - Import necessary Python packages and modules for ROS and PCL handling.
    - Import numpy for numerical operations on point data.
2. PointCloud Callback Function:
    - Input: Receives PointCloud2 messages from a ROS topic.
    - PointCloud Conversion: Converts the PointCloud2 message to a PCL-compatible point cloud object. This involves reading the point data (x, y, z coordinates), handling any NaN values, and transforming this data into a NumPy array that PCL can use.
    - Plane Segmentation: Uses PCL's segmentation capabilities to identify the largest plane in the point cloud, like the ground or a wall, based on the RANSAC algorithm. Points that are part of this plane are found and can be excluded in subsequent steps.

- Obstacle Extraction: Extracts points not belonging to the identified plane to focus on potential obstacles.
- Obstacle Clustering: Performs Euclidean clustering on the remaining points to group nearby obstacles into distinct clusters based on proximity.
- Cluster Handling: Each cluster of points identified as an obstacle is converted back into a PointCloud2 message format and published on a different ROS topic. This allows other parts of the robotic system to use the identified obstacles for navigation and decision-making.

3. Helper Function pcl_to_ros:
    - Converts PCL point cloud data back into a ROS PointCloud2 message format, making it suitable for publication on a ROS topic.
4. Main Function:
    - Initializes the ROS node and sets up subscriptions to receive point cloud data.
    - Publishes processed point cloud data (segmented and clustered) to a new ROS topic.
    - Keeps the node running and handling incoming data until the program is terminated.

Execution Flow

- When the script runs, it starts the ROS node, subscribes to the point cloud data from specified ROS topics, and continuously receives and processes incoming point cloud messages.
- For each point cloud message received, it segments out the main plane, isolates potential obstacles, clusters these obstacles, and publishes the results for further use.

This script is typical for applications in autonomous navigation, where detecting and navigating around obstacles is crucial. The use of PCL for point cloud processing and ROS for message handling is standard in robotics applications, especially those involving environmental perception and interaction.

This is the algorithm we have been trying to implement. We first implemented it in Python which package was much easier to build. (Look at the 2nd part of the report)

- In the end, it **publishes** the resulting clusters.

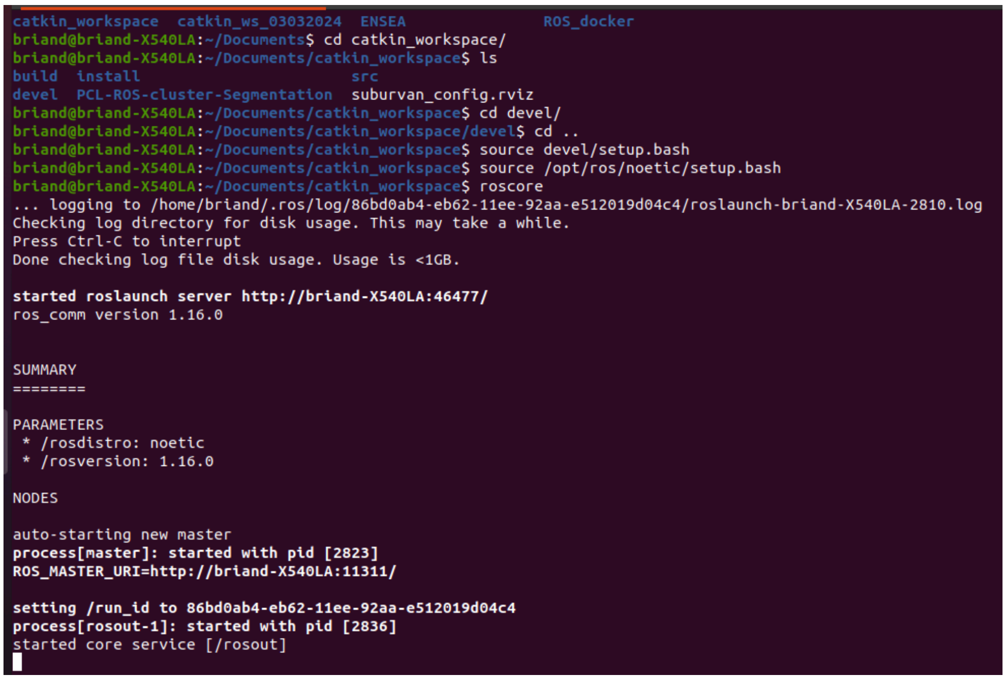
```
$ adaptive_clustering::ClusterArray cluster_array;
$  geometry_msgs::PoseArray pose_array;
$  visualization_msgs::MarkerArray marker_array;
```

→ The current working project works with euclidean segmentation and is part of a package named *robot*.

First, we download the *catkin_ws* project.

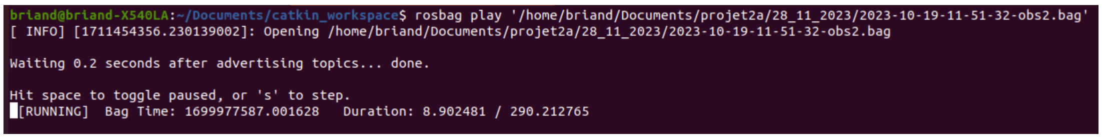
Get to the root of the project and source it for ROS :

```
$ cd ~/catkin_ws/
$ source /opt/ros/noetic/setup.bash
$ source devel/setup.bash
$ catkin_make //to build the project
$ roscore
```



```
catkin_workspace  catkin_ws_03032024  ENSEA               ROS_docker
briand@briand-X540LA:~/Documents$ cd catkin_workspace/
briand@briand-X540LA:~/Documents/catkin_workspace$ ls
build  install                    src
devel  PCL-ROS-cluster-Segmentation  suburvan_config.rviz
briand@briand-X540LA:~/Documents/catkin_workspace$ cd devel/
briand@briand-X540LA:~/Documents/catkin_workspace/devel$ cd ..
briand@briand-X540LA:~/Documents/catkin_workspace$ source devel/setup.bash
briand@briand-X540LA:~/Documents/catkin_workspace$ source /opt/ros/noetic/setup.bash
briand@briand-X540LA:~/Documents/catkin_workspace$ roscore
... logging to /home/briand/.ros/log/86bd0ab4-eb62-11ee-92aa-e512019d04c4/roslaunch-briand-X540LA-2810.log
Checking log directory for disk usage. This may take a while.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://briand-X540LA:46477/
ros_comm version 1.16.0


SUMMARY
========

PARAMETERS
 * /rosdistro: noetic
 * /rosversion: 1.16.0

NODES

auto-starting new master
process[master]: started with pid [2823]
ROS_MASTER_URI=http://briand-X540LA:11311/

setting /run_id to 86bd0ab4-eb62-11ee-92aa-e512019d04c4
process[rosout-1]: started with pid [2836]
started core service [/rosout]
```

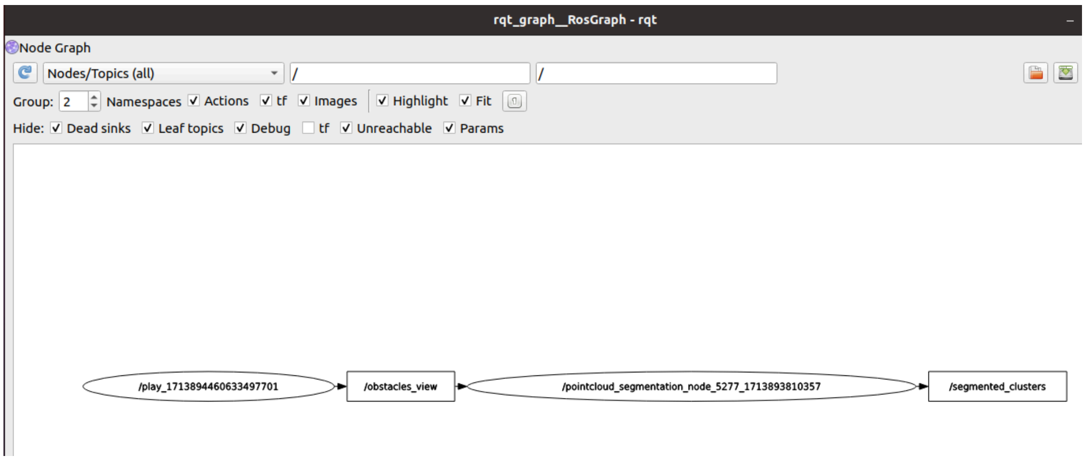
```
$ rosbag play 'file'
```



```
briand@briand-X540LA:~/Documents/catkin_workspace$ rosbag play '/home/briand/Documents/projet2a/28_11_2023/2023-10-19-11-51-32-obs2.bag'
[ INFO] [1711454356.230139002]: Opening /home/briand/Documents/projet2a/28_11_2023/2023-10-19-11-51-32-obs2.bag

Waiting 0.2 seconds after advertising topics... done.

Hit space to toggle paused, or 's' to step.
 [RUNNING]  Bag Time: 1699977587.001628   Duration: 8.902481 / 290.212765
```

And to show the nodes, their topics and understand better their dependencies, we can do :

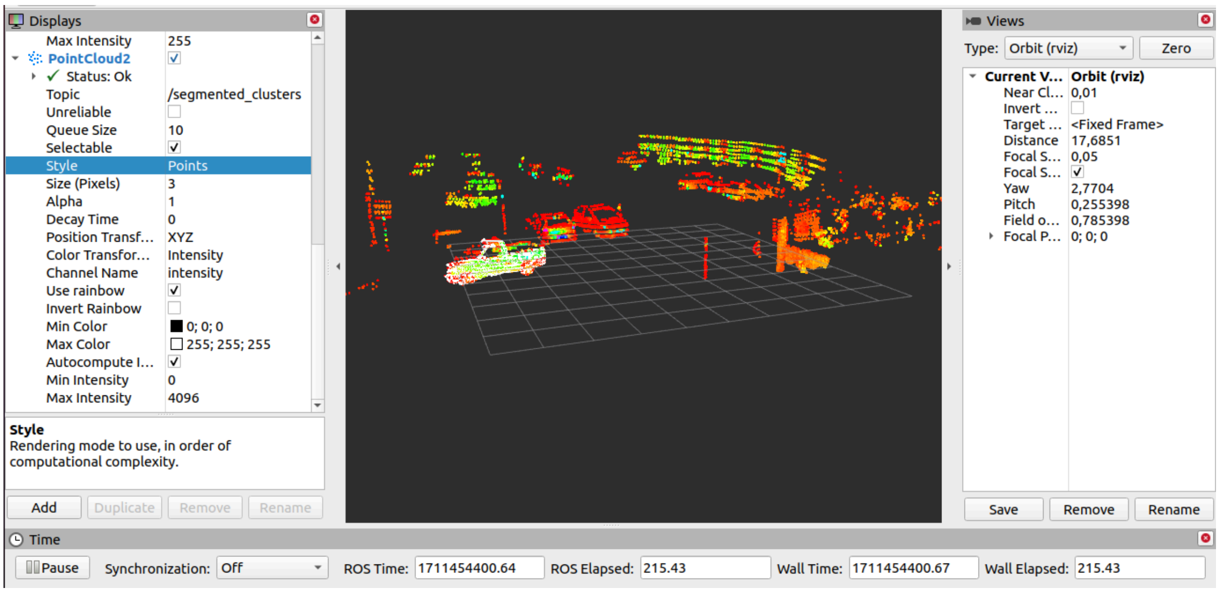
```
$ rosrun rqt_graph rqt_graph
```



The "rqt_graph" above shows the links between nodes and topics. A node can take two actions with respect to a topic:

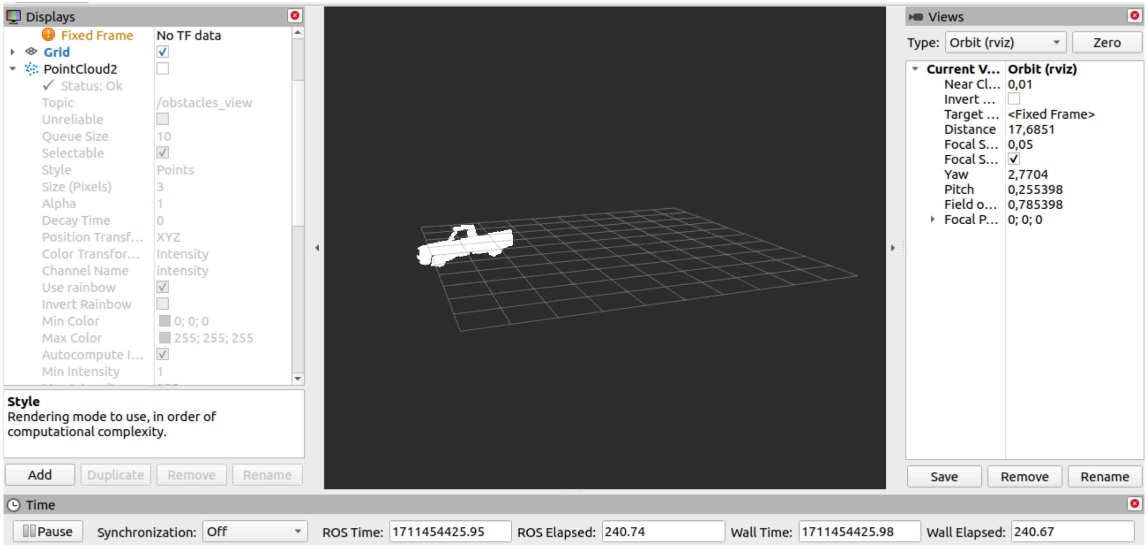
- **Subscribe** to a topic
- **Publish** on a topic

In the case opposite, the "/play_..." node allows us to read and then view the "/obstacle_view" topic, so the node publishes on the latter. Our next objective is to segment the "/obstacle_view" topic. To do this, we create a node that subscribes to "/obstacle_view" but also publishes to the "/segmented_clusters" topic. We've created the latter topic to visualize the segmentation performed by our node. The segmentation node contains the Euclidean clusthering algorithm.



```
Displays
  Max Intensity        255
▾ PointCloud2          ✓
  ▸ ✓ Status: Ok
  Topic                /segmented_clusters
  Unreliable           ☐
  Queue Size           10
  Selectable           ✓
  Style                Points
  Size (Pixels)        3
  Alpha                1
  Decay Time           0
  Position Transf...   XYZ
  Color Transfor...    Intensity
  Channel Name         intensity
  Use rainbow          ✓
  Invert Rainbow       ☐
  Min Color            ■ 0; 0; 0
  Max Color            ☐ 255; 255; 255
  Autocompute I...     ✓
  Min Intensity        0
  Max Intensity        4096

Style
Rendering mode to use, in order of computational complexity.

Views
Type: Orbit (rviz)
▾ Current V...   Orbit (rviz)
  Near Cl...     0,01
  Invert ...     ☐
  Target ...     <Fixed Frame>
  Distance       17,6851
  Focal S...     0,05
  Focal S...     ✓
  Yaw            2,7704
  Pitch          0,255398
  Field o...     0,785398
▸ Focal P...     0; 0; 0

Time
Synchronization: Off   ROS Time: 1711454400.64   ROS Elapsed: 215.43   Wall Time: 1711454400.67   Wall Elapsed: 215.43
```

Here we can see the overlay of the topic "/obstacle_view" and its segmented version, "/segmented_clusters". Segmentation is visible in white. While the non-segmented point cloud is in color.



```
Displays
  Fixed Frame          No TF data
▸ Grid                 ✓
▾ PointCloud2          ☐
  ✓ Status: Ok
  Topic                /obstacles_view
  Unreliable           ☐
  Queue Size           10
  Selectable           ✓
  Style                Points
  Size (Pixels)        3
  Alpha                1
  Decay Time           0
  Position Transf...   XYZ
  Color Transfor...    Intensity
  Channel Name         intensity
  Use rainbow          ✓
  Invert Rainbow       ☐
  Min Color            0; 0; 0
  Max Color            255; 255; 255
  Autocompute I...     ✓
  Min Intensity        1

Style
Rendering mode to use, in order of computational complexity.

Views
Type: Orbit (rviz)
▾ Current V...   Orbit (rviz)
  Near Cl...     0,01
  Invert ...     ☐
  Target ...     <Fixed Frame>
  Distance       17,6851
  Focal S...     0,05
  Focal S...     ✓
  Yaw            2,7704
  Pitch          0,255398
  Field o...     0,785398
▸ Focal P...     0; 0; 0

Time
Synchronization: Off   ROS Time: 1711454425.95   ROS Elapsed: 240.74   Wall Time: 1711454425.98   Wall Elapsed: 240.67
```

https://enseafr-my.sharepoint.com/:v:/g/personal/nael_briand_ensea_fr/EZP_z3yotmZFvkCXwRHfM7ABzFa5wtR8S1bLq7nrrpPkpw?e=zUjrnC

Issue faced :
The clustering function does not keep up in real-time and can’t be applied to the whole cloud points of the file. We can see that It works partially. Only one cluster seemed to be published.

Possible improvement:

Publishing each cluster as soon as it is extracted can induce additional slow. It may be more efficient to aggregate multiple clusters before publishing them or to use more efficient publication methods. (that’s for example the case in AI with different batch sizes).

VI - Conclusion

This project was really interesting from its debut, especially thanks to the context of the project given by Carlos Holguin and Michel Parent.
However, even though we faced a lot of issues from the beginning and the fact that It was quite exhausting to implement the whole architecture of tools we needed to get started (the server, ROS, Docker), we actually really learnt a lot of things throughout the whole project.

ROS (Robot Operating System) is a really flexible framework that **allows parallel processings of real-time operations**. Its architecture based on topics that publish and subscribe to another through messages allows it to develop really flexible projects such as the Big Data SuburVAN project we have been working on.

Currently, we don't have a database of static and dynamic elements but the clustering algorithm is working, even though it's not perfect. This project has a bright future.

Acknowledgement

We actually want to express our extreme gratitude to every person mentioned below for their contribution to the project and especially our Project Supervisor who really supported us throughout the difficulties we faced.

- Project supervisor :
  - **Nicolas SIMOND**

- SuburVAN :
  - **Carlos HOLGUIN**
  - **Michel PARENT**

- ENSEA teachers :
  - **Nicolas PAPAZOGLOU**
  - **Alexis MARTIN**

- IT departement of ENSEA (SRI) :
  - **Julien NIVET**
  - **Fabrice DUVAL**