POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

An outdoor radar-based 3D mapping system for autonomous navigation in the agricultural framework

TESI DI LAUREA MAGISTRALE IN
MECHANICAL ENGINEERING-INGEGNERIA MECCANICA

Author: **Luca Ferrari**

Student ID:	945650
Advisor Politecnico:	Emanuele Zappa
Advisor ATB Institute:	Dirk-Niklas Müller
Co-advisor:	Tjark Schütte
Academic Year:	2021-22

Abstract

In the agricultural framework, autonomous vehicles represent a disruptive solution to face the exponentially increasing demand of resources from the field while the working power is steadily decreasing. In this context, where innovation meets one of the hardest working environments for a robot, this work proposes an innovative solution to perform 3D outdoor mapping deploying uniquely a radar sensor. Radars' characteristics, with respect to other similar sensors', typically lack in performances as they suffer of multi-path reflection noise, but they have few not negligible advantages: being invariant to weather condition, great operating range of measures and, ultimately, good capacity in detecting moving objects. The developed algorithm aims at dramatically rising the measuring quality of the original radar, in terms of reliability, noisiness and density, while retaining all the good properties mentioned. The improvement attained through this methodology is reached thanks to two subsequent phases: an analytical one of denoising and clustering, and an upsampling one. The latter is obtained through a machine learning algorithm trained with coherent lidar's ground truth data. The dataset for the training and validation has been sampled using the Irus autonomous vehicle at the ATB institute in Potsdam (Germany).

Key-words: Radar, Mapping, Agriculture, Autonomous vehicles, Neural Network

Abstract in lingua italiana

I veicoli autonomi rappresentano, nel quadro dell'agricoltura, una soluzione rivoluzionaria per fronteggiare la richiesta sempre più alta di risorse provenienti dai campi, mentre la forza lavoro continua a diminuire. In questo contesto, dove l'innovazione incontra uno degli ambienti di lavoro più duri per un robot, questo lavoro propone una soluzione innovativa per fare mappatura 3D all'aperto con il solo utilizzo di un sensore radar. Le caratteristiche dei radar, rispetto a quelle di altri sensori simili, tipicamente hanno prestazioni scarse poiché soffrono di rumorosità dovuta a riflessioni multiple, ma possiedono anche alcuni vantaggi non trascurabili: sono invarianti alle condizioni metereologiche, hanno una ottima portata di misurazione e buone capacità di rilevare oggetti in movimento. L'algoritmo sviluppato ha come obiettivo quello di migliorare drasticamente la qualità della misurazione del radar originale, in termini di affidabilità, rumorosità e densità, mentre tutte le buone proprietà menzionate vengono mantenute. Il miglioramento ottenuto dall'algoritmo è raggiunto grazie a due fasi principali: una fase analitica di rimozione del rumore e raggruppamento, e una fase di addensamento ottenuta grazie ad un algoritmo di apprendimento automatico allenato utilizzando dati coerenti di un sensore lidar, considerati come verità assoluta. L'insieme dei campioni per l'allenamento e la validazione è stato raccolto utilizzando il veicolo autonomo Irus all'istituto di ricerca ATB di Potsdam (Germania).

Parole chiave: Radar, Mappatura, Agricoltura, Veicoli Autonomi, Rete Neurale

Contents

Introduction

In the context of agriculture, autonomous vehicles (Figure 0.1) represent a disruptive solution to face the exponential increasing demand of resources from the field while the working power steadily decreases, mainly due to the hardness of the farming tasks on the human body.
Therefore, during the past it has been seen an evolution in the methods and tools used for cultivating, leaving the most difficult and heavy tasks more and more to the machines. The complexity of such a hostile environment as the crop fields makes the designers' job very hard in terms of finding feasible and cheap solution while guaranteeing proper functioning in presence of soil, mud, humidity, leaves etc.

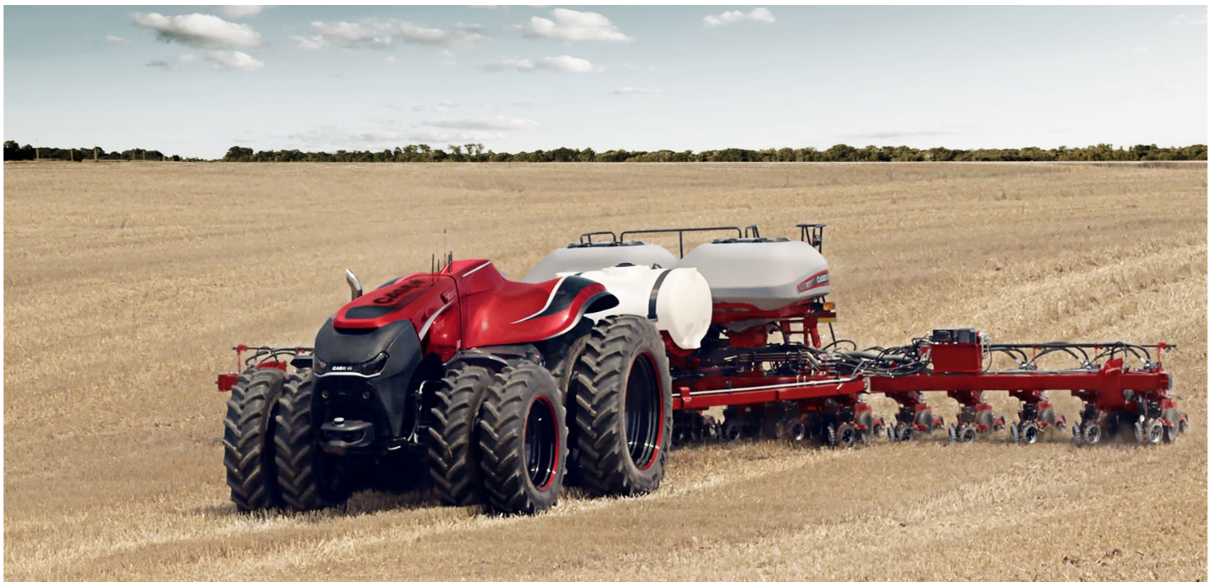


Figure 0.1: Example of autonomous vehicle in the agricultural framework

Moreover, intensive agriculture is not an available option anymore. The latest research showed how this method is neither the most efficient nor preserves biodiversity, instead it introduces in the environment imbalances that cannot be naturally corrected. This fuels the use of other substances that in turns increase the final costs of production propagating to the final consumer while making the final products less natural.

The answer might be in the concept of agroforestry (Figure 2): before moving to the intensive fields, this idea was implicitly applied, and it basically consists of having more complex areas within forests used for farming. The ecosystem is completely capable of sustaining itself by its own, without any addition (thus any added cost), providing not only the cultivated product, but also a vast quantity of biomass such as wood, that can be sold or used for domestic heating by the farmer as well. The natural context of this idea permits the stabilization of the imbalances and the protection of the biodiversity now at risk.
It is obvious then, that the machines performing in these environments have to be highly technologically developed, small enough to operate, with several degrees of freedom in the movement of their parts for meticulous operations and checking, a stable navigation module, equipped with various sensors either for agricultural purposes or for assessing their own proper functioning, positioning etc.

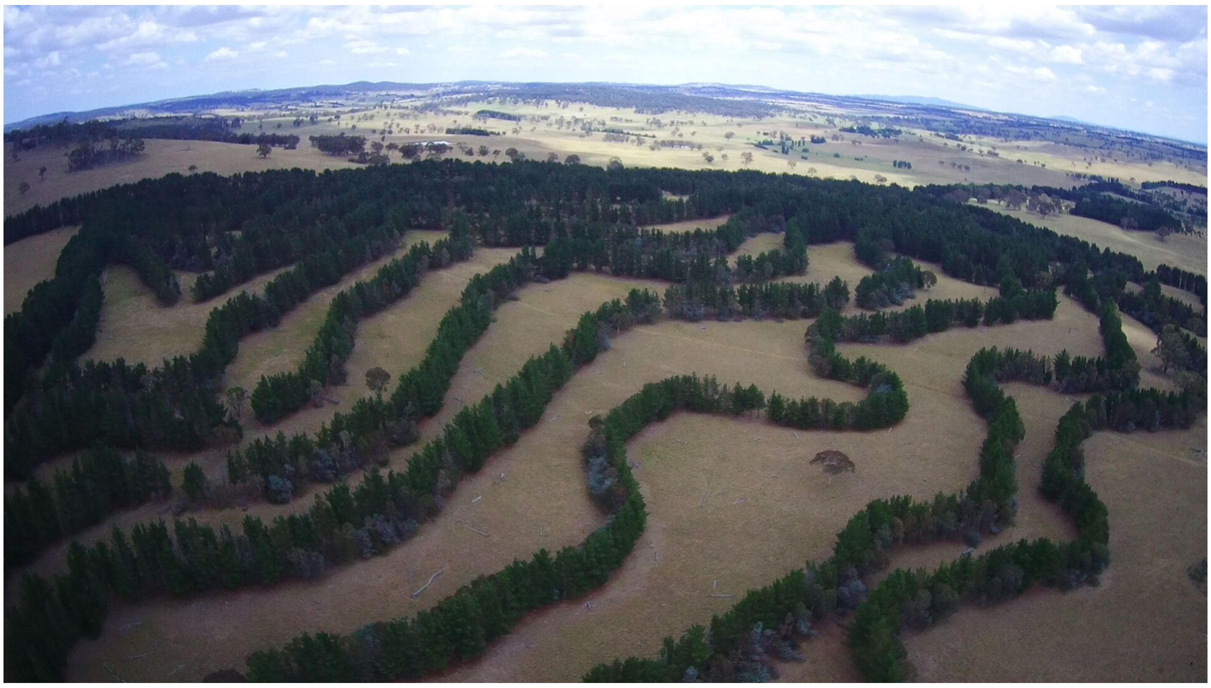


Figure 0.2: Agroforestry

From this perspective, the goal is to maximize the number of performable tasks while keeping at the minimum the resources used. The concept developed by this work stands out inside this idea. The algorithm developed provides an innovative way of performing supervised mapping. On the other hand, the latest works in the literature provide navigation algorithms for global path planning, for obstacle avoidance logics, for object recognition, etc. which are compatible with the a priori knowledge of a map. Moreover, the intrinsic capability of radars to well perform in dynamic environments makes it theoretically ideal to accomplish by itself all these complicated tasks.

Performing environmental mapping means to scan the surroundings in order to collect the necessary data to reproduce its structure and shape digitally. To perform this procedure, it is necessary to use distance measuring sensors that make use of TOF (*Time-Of-Flight*) principle and/or cameras, and then handle the data through a process that makes the signal useful and available to the vehicle. Based on this information, the vehicle can take better and more accurate decisions.
Moreover, one can classify different types of mapping based on the type of information gathered, on the number of spatial dimensions considered (2D or 3D), and on the application which the map is built for. This work aims at generating a 3D map of the surrounding area of an outdoor path in an agricultural context through an off-the-shelf radar sensor coupled with its own evaluation board.

Nowadays, mapping in all its variants is hardly ever performed by radars. Indeed, typically this operation is left to lidars and/or depth-cameras. Considering sensors belonging to the same relative market segment, the latest are more accurate, cleaner in their measurements and the output information is more complete with respect to radars. Radars are noisy, and capable of sensing only specific structure of the environment, but they have many relevant and useful properties, like being invariant to weather conditions, having great capability of sensing moving objects, which is something fundamental when dealing with dynamic ambience, and in most cases, they are also cheaper. Therefore, the task can be looked at from a different perspective: is there a way to join the best characteristics of the two sensors?
This merging capabilities idea could be thought in two distinctive ways: either a radar takes the ability of the lidar, becoming more precise while overcoming data sparsity, or conversely, a lidar can preserve its own great measuring accuracy while working in different weather condition and working in a dynamic environment. This work goes in the direction of the first hypothesis, which is what is considered to be the most feasible approach. In order to increase the radar’s performances, the data coming from a lidar are taken as ground truth information on which the algorithm is trained.

1. Literature Review

1.1 State of the Art

The goal of this chapter is to describe the current state of the art concerning the field of outdoor radar-based mapping. Since the affinity with the topic, some space will be given also to other typical tasks related with autonomous navigation such as pose estimation, object detection and recognition, and also to the most innovative methods applied to carry out those tasks.

First of all, let’s make a distinction between indoor and outdoor mapping.
Indoor mapping considers most of the times small robotic innovative vehicles, and the research are carried out within buildings. Typically, the structure of this environment is vertical and thus it well suits the use of occupancy grids (Elfes, 1989) , which are top view 2D image representations of the environment, where each pixel is associated with the probability of being occupied by an object (Figure 3).

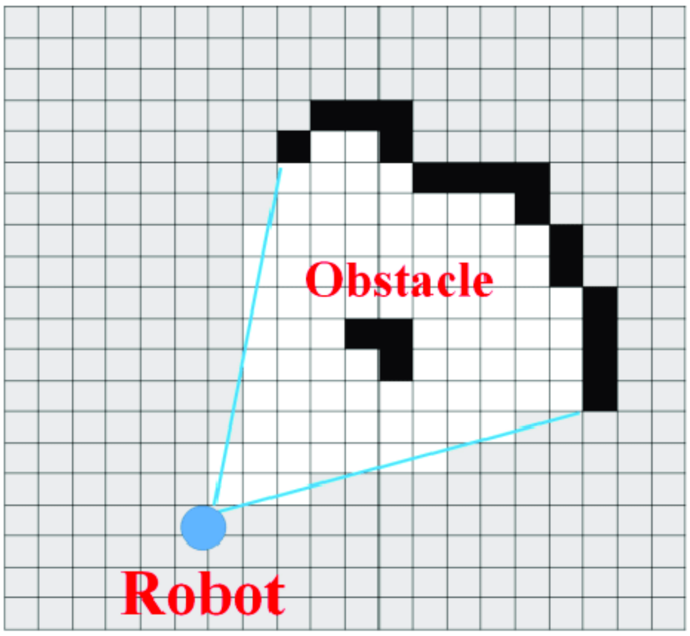


Figure 1.1: 2D Occupancy Grid Mapping

On the other hand, the vast majority of literature reviewed in case of outdoor related analysis revolves around automotive applications. In this field, the use of radars begun in the 90's with the ACC (Adaptive Cruise Control) and collision warning application (M. Schneider, 2005). Thus, it should not be surprising the vastity of research made in this industry.

The Manhattan world assumption (Coughlan et al., 1970) provides structural hypothesis in terms of structure regularities that can be exploited (Figure 3). This work shows that the assumption is made mainly for indoor application, but it finds validity even for certain outdoor environment, including rural scenes. Thus, because of this similarity, to some extent, also the literature providing results of autonomous navigation behavior in indoor environment are considered.

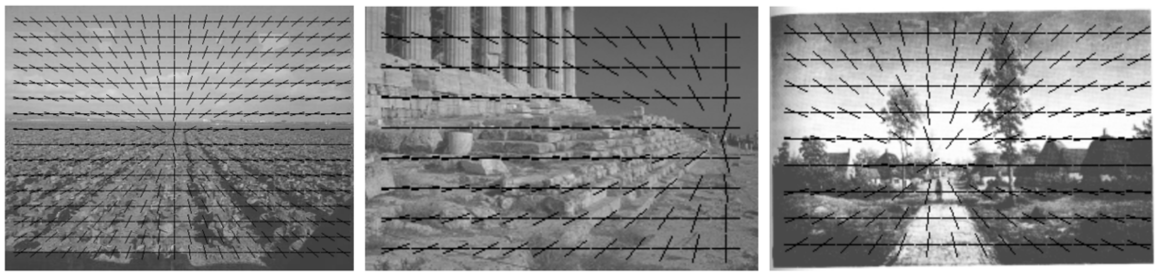


Figure 1.2: Manhattan World Assumption

As already mentioned in the Introduction, radars lack of most of the skills to well perform mapping. Therefore, during the past years, researchers have looked for the possibility to overcome this issue instead of making the hardware extremely complicated and expensive, by moving the complexity towards the algorithm and the processing. From this perspective came the idea of using, in the automotive context, processing-based coherent evaluation, by exploiting only two radars (Gottinger et al., 2021).

In a similar manner, a radar-based occupancy grid mapping technique was developed by exploiting random forest classifiers and in general techniques from the computer vision field, with the intent of mapping parking lots to spot parked vehicles (Dubé et al., 2014). Other radar mapping algorithm developed in the literature uses various algorithm strategies to reduce all kinds of radars side effects (Dogru & Marques, 2019; Lee et al., 2021; Marck et al., 2013).

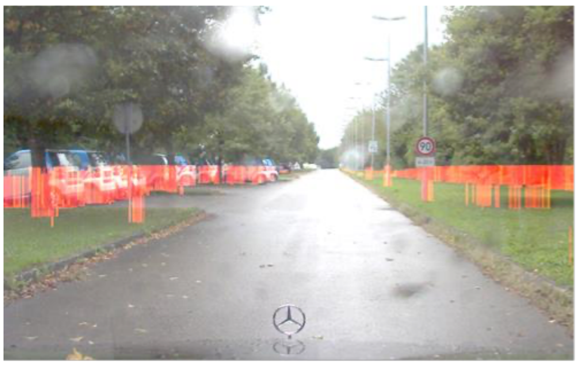


Figure 1.3: R Dubé et al., 2014, Detection of parked vehicles

The innovative theories in the computer vision field had a very strong impact on the mapping framework. Exploiting the use of 2D occupancy grids or 3D voxels, the applicability of those algorithms comes naturally. One clear example of this are cGANs (Mirza & Osindero, 2014) or autoencoders, which are fairly complex Machine Learning algorithm applied in a wide range of related tasks like high resolution imaging through mmWave radars (Guan et al., 2020), object enhancement through cloud upsampling (Yifan et al., 2019), object reconstruction (Lin et al., 2018), or transformation from cloud data to pixel images (Milz et al., 2019).

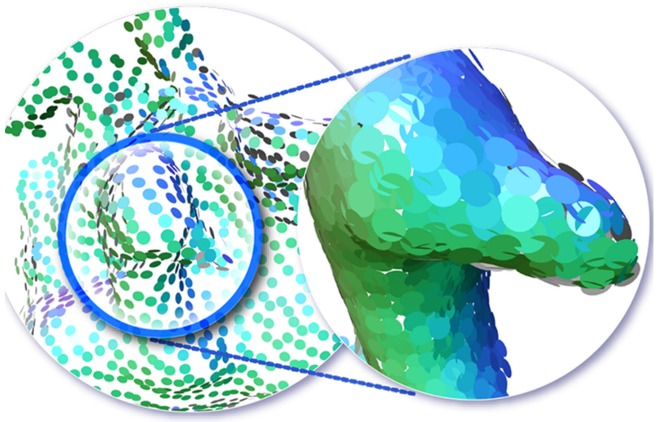


Figure 1.4: Yifan et al., 2019. Patch based upsampling

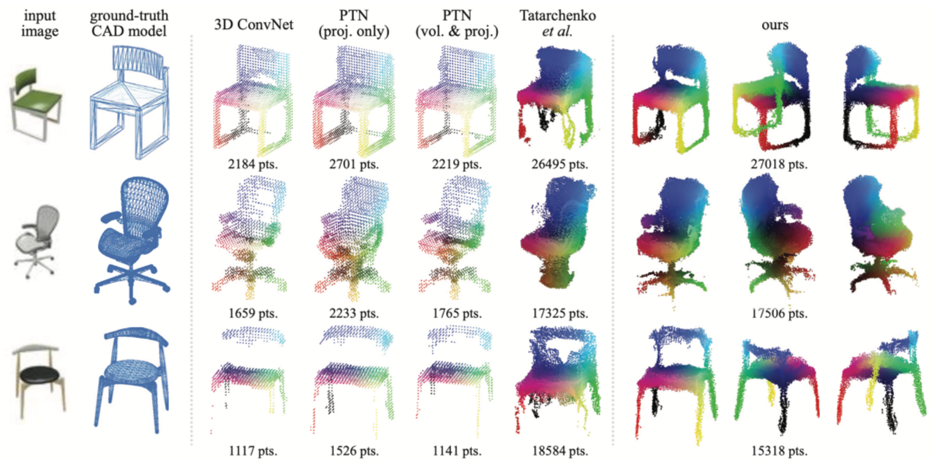


Figure 1.5: Lin et al., 2018. 3D Object Reconstruction

In more recent years, radar applications concerning the autonomous navigation have considered the fusion with different sensors in order to join their best characteristics and provide superior results. In particular, radars have been matched with IMUs (Lu et al., 2020), Cameras (Nobis et al., 2019), Laser scanners (Bouzouraa & Hofmann, 2010), other radars (Diehl et al.., 2020) and Lidars.

In the fusion perspective just introduced, considering radar and lidar characteristics, the goal is to improve the radar sampling performances with the help of a coherent information provided by the better performing sensor. This data can be provided in many ways: for instance, as prior knowledge to perform SLAM in indoor disaster environment (Park et al., 2019). Further, many researchers have experimented linking the information for online applications through neural networks, by producing occupancy grid (Weston et al., 2019; Lu et al., 2020). All these works are characterized by 2D occupancy grid indoor mapping, aiming at autonomous motion of safety robots into buildings where the presence of smoke or other factors is impeding normal visibility (Figure 8).

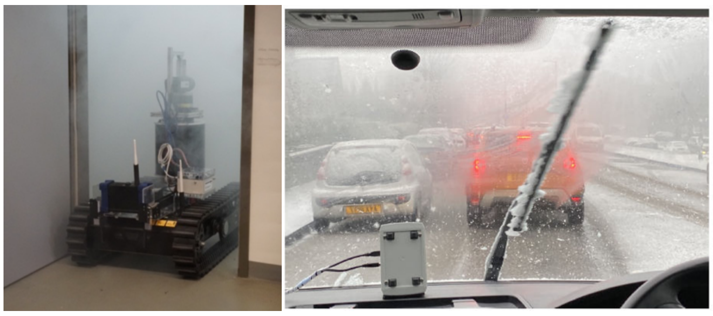


Figure 1.6: Visibility hindered by smoke or snow

Differently from what can be found in the literature, this work yearns at providing a unique interpretation of the mapping problem, avoiding complex and computationally expensive Machine Learning algorithm and by using a different kind of map representing data, lighter and easier to understand, which are marker coordinates. Moreover, neither segmentation algorithms have been used nor object recognition of any kind, but rather it has been proposed another way of shaping different object through union of multiple clusters.
This thesis aims at finding a solution in a structured outdoor environment, where either strongly geometrical obstacles (like walls of buildings) and less structured objects (like trees, loaders of various shape and dimensions etc.) can be found on the walkthrough.

1.2 Research Questions

In this paragraph, there will be defined in detail the research questions, in terms of what the goals of this project are and how they are meant to be achieved.

This Master Thesis aims at building an algorithm to perform the complex task of 3D outdoor mapping. The hardware framework on which this work relies on, is an autonomous vehicle for agricultural purposes called *Irus*. The vehicle is equipped with a radar sensor for frontal distance measurements. Starting from the original signal of this sensor, the complete algorithm developed has the purpose of transforming this data into something more complex and useful.
First of all, the task that this newly proposed system aims at solving is supervised mapping, i.e., the procedure of mapping is performed while a user is driving the vehicle into the environment. In the specific case of this thesis, the driving directions are sent to the robot through a radio waves remote control.

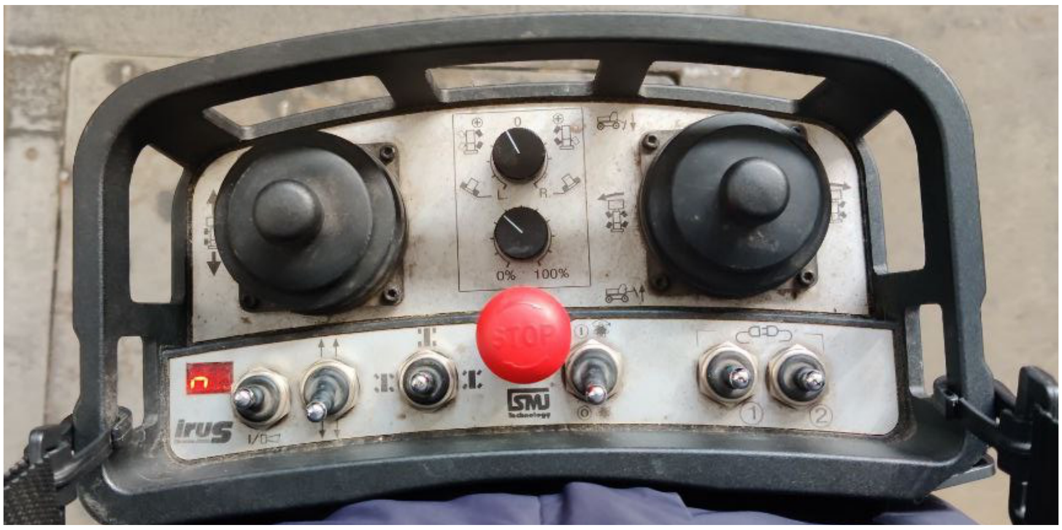


Figure 1.7: Vehicle remote controller

Then, the output has to be visually appealing: by looking at the final map, a human should be able to recognize the presence of objects, obstacles but also the path that the vehicle has followed. Furthermore, in a perspective of full automation, the resulting map generated should be used as source of a priori knowledge of the environment structure. The system will use this information in logics like path planning or self-localization.
As it will be described later in this document, the radar sensor presents some issues when it comes to mapping, because of its intrinsic performances. However, many other aspects of radars are extremely appreciable. Therefore, the goal is to improve the radar mapping characteristics in terms of reliability, noise and cloud sparsity, without compromising the other good capabilities.
Overall, the final objective is to provide an algorithm capable of handling the signal coming from the radar, passing the original information through specific transformations (which will be developed into details in future chapters), to finally obtain a map which is better than the original.
Since the algorithm is a pure data manipulation, it is expected to provide the results without any extra element, thus without any further cost.

2. Fundamentals

In this chapter, the reader will get an understanding of what are the most relevant topics on which this work is based on. Those concepts, when developing the core part of the Master Thesis will be taken for granted.

2.1 TOF Sensor

State of the art regarding the techniques performing mapping involves, most of the times, the use of cameras and TOF (*Time-of-Flight*) sensors, like lidars, sonars, radars etc.
Time of Flight sensors working principle (Figure 10) relies on the phase shift of the illuminating radiation that has been emitted with respect to the reflected one measured by a receiver. The phase shift translates into distance knowing the frequency of the radiation. An imaging sensor designed to respond to the same spectrum receives the light and converts the photonic energy to electrical current. Note that the light entering the sensor has an ambient component and the reflected component. Distance (depth) information is only embedded in the second. Therefore, high ambient component reduces the signal to noise ratio (SNR). (Li, 2014)

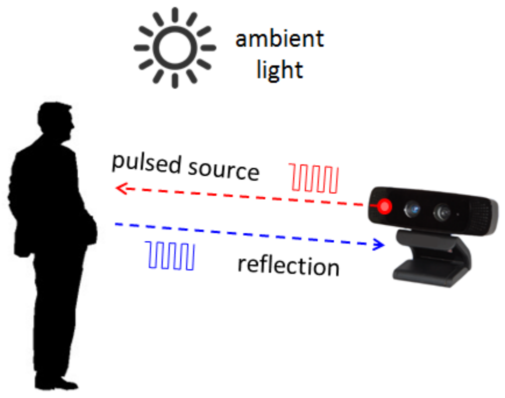


Figure 2.1: TOF working principle

In order to select the proper sensors for a certain application, there are many drivers to take into consideration: the environment in which it will act, the performances the designer wants to achieve, the costs, the compactness are just few examples.
Each sensor typology has pros and cons that may limits its usage and its applicability. However, the huge variety of these sensors in many cases gives the possibility to select a tool that properly match the requirements.

2.1.1 Lidars

The lidar is a *Time-of-Flight* sensor typically used to perform indoor mapping. The emitted radiation belongs to the infrared spectrum, with light wavelength in the range between 850nm and 1050nm.
There are several kinds of lidars (Figure 11), depending for instance on the technology of light emitters like VCSEL (Vertical Cavity Surface Emitting Laser) or LEDs. The first case is the most common, the laser technology permits a more focused impulse that well clarify the reading of the sensor making it the most precise. The LEDs on the other hand outputs an overall power which is greater than the VCSEL for lower current intensity values, but the wave is spatially spread over a bigger angle: this makes it suffer against misreading due to wave reflections and lowers its range since the higher power level is spread over a bigger angle.
Sticking then with VCSEL technology, the aim would be to obtain several measurements for slightly different positions in order to generate a more complex three-dimensional image or point cloud representing the world in front of the sensor. One way to do this, is to have a moving mirror and only one emitter whose light is selectively reflected depending on the position of the mirror. This way the light is always directed towards a new direction and the high speed of this process makes it possible to be used for online application.

Figure 2.2: Examples of lidars

The matrix lidar, on the other hand, is a multi-VCSEL lidar. Each laser is a pixel of the image/point cloud generated. Those kind of lidars are very popular, and one example of it is the Sick Lidar.

Finally, the Velodyne Lidar is a very commonly used lidar, especially in the automotive industry as it grants a 360° cover of the surroundings thanks to its column of VCSEL which are then put in rotation. This provides sampling from all around the vehicle on which it is installed. (C. Rablau, 2019)

2.1.2 Radars

Radars have the same working principle of lidars. The difference stands in the frequency of the emitted light, which is in the range of radio waves, around 80MHz of frequency.
Radars are not usually employed to perform mapping. Their usual application is rather related to object detection and environmental monitoring. The reasons are many: standard commercial radars are not very accurate: they suffer of multi-path noise related to their huge wavelength generating a lot of outliers; they typically suffer of point sparsity as they are not capable to sample enough points and also shortcomings such as a large footprint, sidelobes, specularity effects and limited range resolution, all of which result in poor maps (Bunting et al., 2000). The higher frequency of the radiation drastically increases their range up to 200m but the field of view is normally lower compared to lidars (Rablau, 2019).
One of the main advantages of radars is their behavior in dynamically changing environments. Furthermore, the installation of multiple receiver antennas gives the capability of computing the radial velocity of the object visualized, their direction and thus their overall motion. Therefore, radars are widely use in the automotive industry as a base sensor for many applications.

Figure 2.3: Examples of radars

2.2 ROS framework

ROS is an open-source robot operating system. It is not an operating system in the traditional sense of process management and scheduling; rather, it provides a structured communications layer above the host operating systems of a heterogenous compute cluster.

Writing software for robots is difficult, particularly as the scale and scope of robotics continues to grow. Different types of robots can have wildly varying hardware, making code reuse nontrivial. ROS has been developed in this perspective, focusing on a specific set of challenges to produce an architecture which is more generally exploitable in a huge variety of application. Among those challenges, two has to be mentioned for their relevance for this work: first of all the multi-lingual flexibility, as guarantees supports whether the coding language is Python, C++, Octave or LISP; second of all, it is completely free and open source, which in turns means huge support, either through a wide variety of downloadable content but also through help from experts in forums and other websites and finally many free tutorials or online lectures for fast learning.

The fundamental concepts and building blocks of this framework are nodes, messages, topics and services. Nodes are processes that perform computation. ROS is designed to be modular at a fine-grained scale: a system is typically comprised of many nodes. In this context, the term "node" is interchangeable with "software module".
Nodes communicate with each other by passing messages. A message is a strictly typed data structure. Standard primitive types (integer, floating point, boolean, etc.) are supported, as are arrays of primitive types and constants. Messages can be composed of other messages, and arrays of other messages, nested arbitrarily deep.
A node sends a message by publishing it to a given topic, which is simply a string such as "/odometry" or "/map." A node that is interested in a certain kind of data will subscribe to the appropriate topic. There may be multiple concurrent publishers and subscribers for a single topic, and a single node may publish and/or subscribe to multiple topics. In general, publishers and subscribers are not aware of each other's existence.

Although the topic-based publish-subscribe model is a flexible communications paradigm, its "broadcast" routing scheme is not appropriate for synchronous transactions, which can simplify the design of some nodes. In ROS, we call this a service, defined by a string name and a pair of strictly typed messages: one for the request and one for the response. This is analogous to web services, which are defined by Urls and have request and response documents of well-defined types. (Quigley et al., 2009)

2.2.1 Relevant ROS Packages and Library

Among the thousands packages available for ROS, three have extraordinary importance for this work. The first one is the TF (TransFer) library, designed to provide a standard way to keep track of coordinate frames and transform data within an entire system such that individual component users can be confident that the data is in the coordinate frame that they want, without requiring knowledge of all the coordinate frame in the system (Figure 13) (Foote, 2013). The way the frames can be depicted is in form of trees (Figure 14), as each frame either is the fixed frame, or has a relationship with another defined frame. The handling of different movement given by this library comes with easiness in terms of concept, but also in the practical coding side.

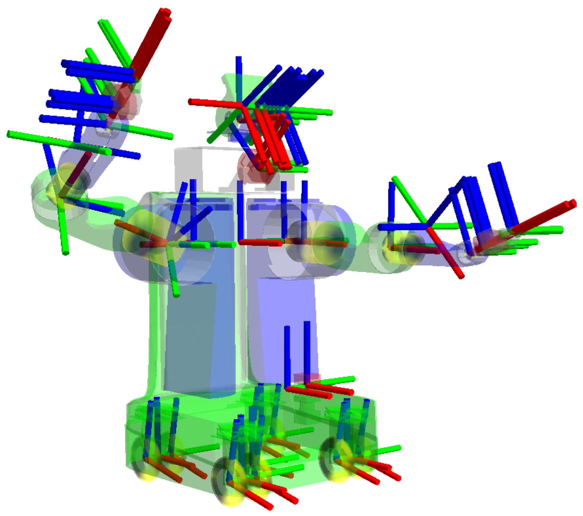


Figure 2.4: Frames from TF library

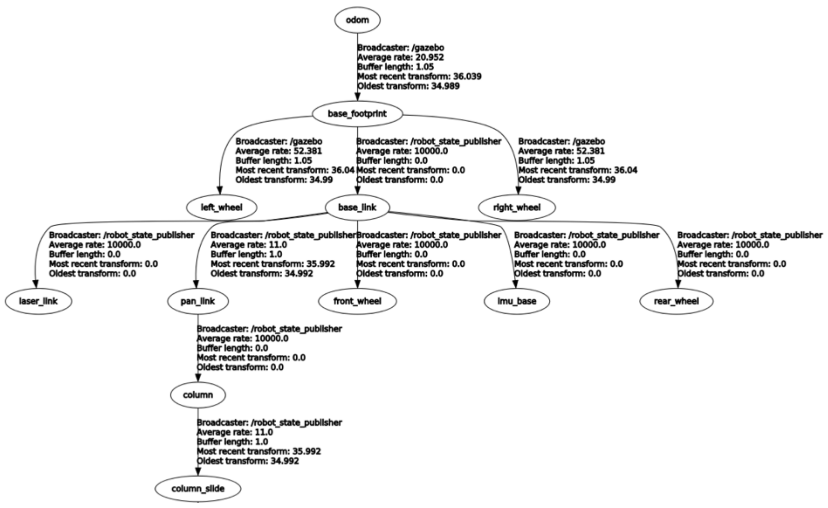


Figure 2.5: TF tree

OctoMap is an integrated framework based on octrees for the representation of three-dimensional environments. Most robotics applications require a probabilistic representation, modeling of free, occupied, and unmapped areas, and additionally efficiency with respect to runtime and memory usage. To this extent, this package deals with three fundamental requirements: probabilistic representation, thus taking into account the uncertainties related with the sensor measurements; modeling of unmapped areas, as a robot is able to plan collision-free paths only for those areas that have been covered by sensor measurements and detected to be free; efficiency, in terms of memory usage and access time, as those two are considered to be the majors bottle-necks of 3D mapping systems. (Hornung et al., 2013)
The working principle of Octomap is based on Octrees, which are a simple and efficient way for space representation. Given a certain value of depth of the tree, the space is divided in cubes of cubes. This encapsulation reaches the resolution limit when the number of layers of the octree is equal to the depth parameter defined (Figure 15).
In any case, this work makes use of this package as a black box: the input to the octomap node lauched at start are the online pointcloud information coming from the sensor as PointCloud2 message and the pose of the sensor itself with respect to a fixed frame, estimated by the GPS module, and published on ROS as Pose message. The output of the node is the generated map as arrays of coordinates. The type of message defined by the ROS framework is MarkerArray, each Marker representing a single point of the map.

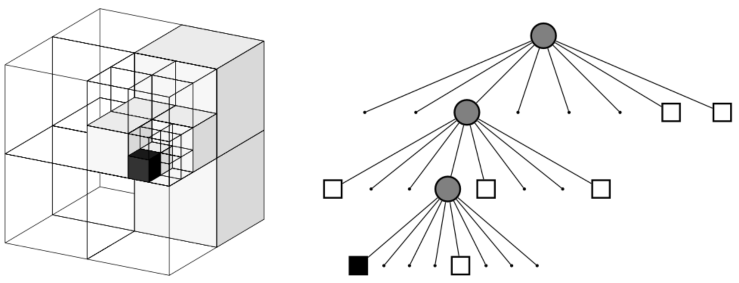
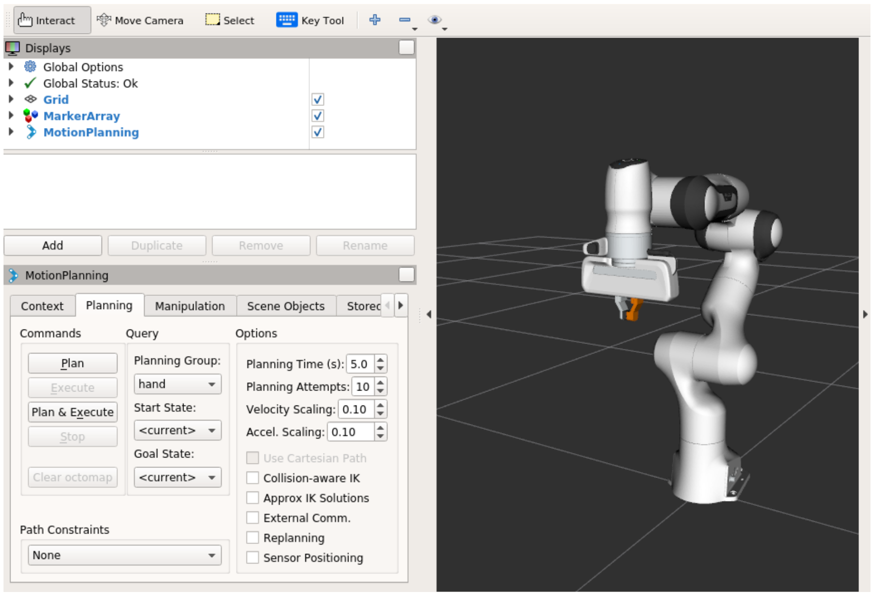


Figure 2.6: Example of an octree

RViz is a 3D visualization environment. It permits to view what the robot is seeing, thinking, and doing. The project was born from the necessity to provide a tool that could model the data coming from the real world. It is very hard to do debugging by looking at pure numbers, conversely visualizing the situation throughout this tool simplify the comprehension of what is happening in the whole system. In other words, it gives the possibility to see the world through the robot's eyes.

This environment is capable of handling sensors and state information, like laser scans, point clouds, cameras and coordinate frames. The tool has many built in parameters to select the best way to view at one's data. RViz is a powerful tool to develop robot's and autonomous vehicle's capabilities and perform research. Finally, just like ROS, it is free and open source. (Figure 2.7)



Global Status: Ok
Planning Group: hand
Start State: <current>
Goal State: <current>
Path Constraints: None
Planning Time (s): 5.0
Planning Attempts: 10
Velocity Scaling: 0.10
Accel. Scaling: 0.10
Use Cartesian Path
Collision-aware IK
Approx IK Solutions
External Comm.
Replanning
Sensor Positioning

Figure 2.7: Standard interface of RViz tool

2.3 Machine Learning/Neural Network concepts.

Machine learning (ML) is the scientific study of algorithms and statistical models that computer systems use to perform a specific task without being explicitly programmed (Mahesh, 2020). In other words, it is the field of study of all those algorithms that make use of databases to learn how to model certain problems that would be too hard to solve deterministically. Machine Learning algorithm can be classified into three categories: Supervised, Unsupervised and Reinforcement learning. This brief introduction to the topic will focus on the first kind.

Supervised models provide mathematical frameworks aiming at finding statistically valid solutions by looking at real inputs and outputs.
The typical example made to clarify one of the many use cases of supervised learning algorithms is to label pictures of animals or objects. Thus, given a matrix of pixels (out of which a human would instantly recognize what is the subject), the aim is to find an algorithm capable of identifying the object as a human would do (Figure 2.8). Clearly, there is no deterministic way to assess this task as all the pictures are unique and the possible patterns are too various. Rather, the approach proposed by this science is to

take a high number of pictures with their correct labels and "train" an algorithm such that at a certain point the algorithm is capable of providing with low uncertainty the probability that a picture contains a certain subject.

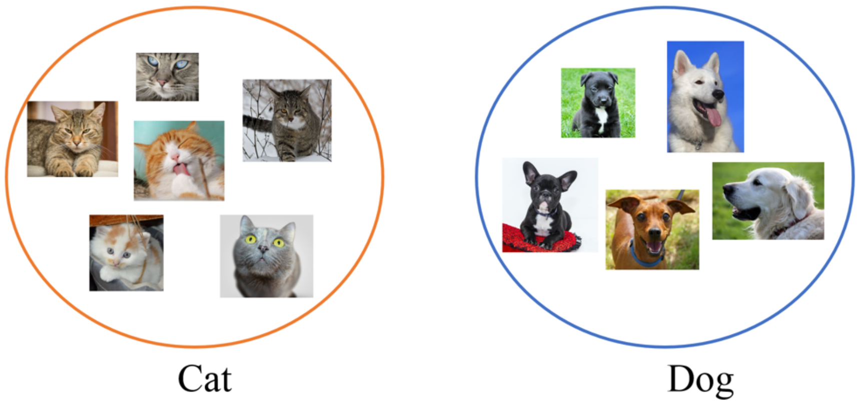


Figure 2.8: Classic didactic example of Machine Learning classification problem

What does it mean to train? Training means updating the value of weights which are part of the algorithm, based on the result of each comparison. In practical terms, those weights are initiated randomly, but the more the algorithm look at new samples, the more their value changes accordingly. Obviously, the way the update is done depends surely on the comparison, but also on a minimization error principle. This latest considers a function of "loss" that has to be minimized in order to converge to the set of weights providing the minimum error.

2.3.1 Neural Networks

Neural Networks are a subcategory of supervised learning algorithms. The mathematical base of those networks is the "Perceptron" that had the goal at that time to be a profusion of brain models which amount simply to logical contrivances for performing particular algorithms (representing "recall", stimulus comparison, transformation, and various kinds of analysis) in response to sequences of stimuli (Rosenblatt, 1958).

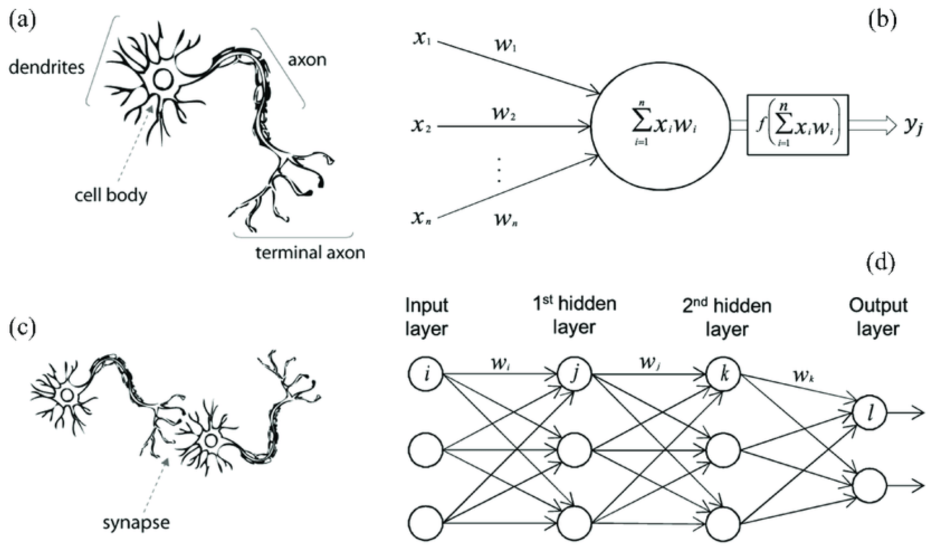


Figure 2.9: Neural Network analogy with cerebral neuron

The perceptron is indeed a biologically inspired programming paradigm, composed by artificial neurons that will be called nodes here. Mathematically, the perceptron node receives several inputs and put them together through a simple weighted sum. Those weights are real numbers expressing the importance of the respective inputs to the output. The neuron's output, 0 or 1, is determined by whether the result is less than or greater than some threshold value. Just like the weights, the threshold is a real number which is a parameter of the neuron.
This simple algorithm by itself is not capable of solving highly complex problems. What it rather shows is the possibility to weigh up different kinds of evidence in order to make decisions. Therefore, increasing the capacity of such algorithm by increasing the amount of nodes and the layers between input and output (Figure 2.10), it should seem plausible that a network of perceptrons is able to make quite subtle decisions.

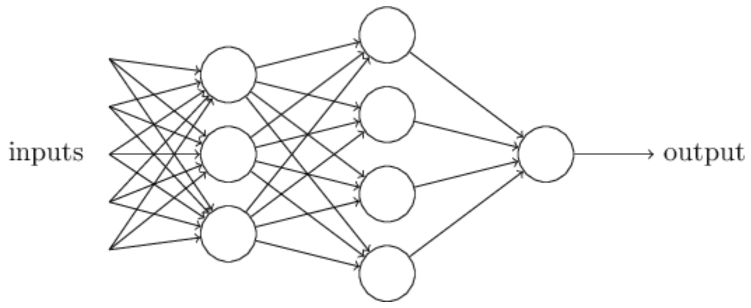


Figure 2.10: Example of Multilayer Perceptron structure

In the picture above, each column of nodes is commonly called layer. The first layer thus represents the inputs (input layer), whereas the last one contains the values of the outputs (output layer). The layers in between are hidden inside the algorithm and are

used to increase the abstraction level of the network and so its capability to model mathematical problems of higher and higher complexity.
The main issue of the perceptron is the output, as it can assume only values 0 and 1. This means that in certain situation small variation of the parameters could drastically modify the output. This makes it difficult to see how to gradually modify the weights and biases so that the network gets closer to the desired behavior. The required development to achieve this flexibility brought to light the sigmoid neuron (Figure 2.11).

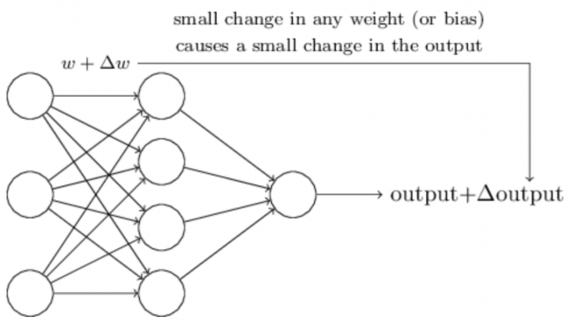

small change in any weight (or bias)
causes a small change in the output
$w + \Delta w$
output+Δoutput

Figure 2.11: Flexibility through sigmoid neuron adoption

Basically, whatever results is provided by the weighted sum, the sigmoid function is capable of mapping it to be between 0 and 1. The sigmoid function is just a smoothed version of the step function which is the one used by the perceptron: the analogy between the two models lays here (Figure 2.12).

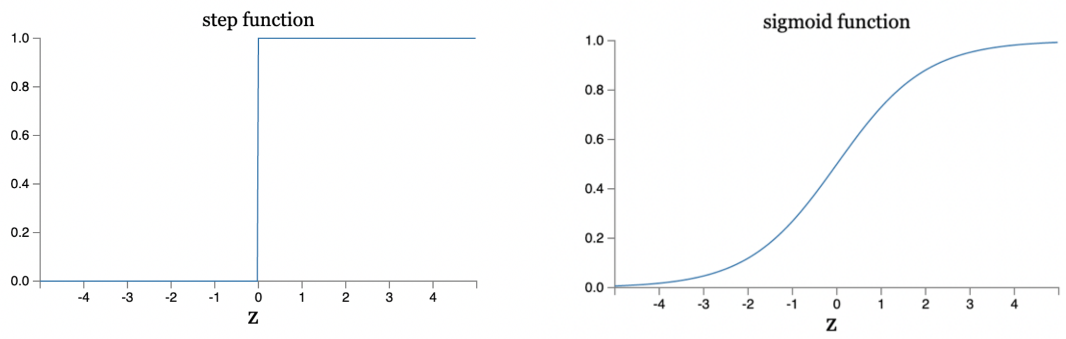


Figure 2.12: Similarity between step and sigmoid activation function

The sigmoid function is an activation function, as it defines how much a neuron is activated by the provided inputs. Later research provided newer, more stable and efficient activation functions like ReLu, Tanh and many others (Figure 2.13).

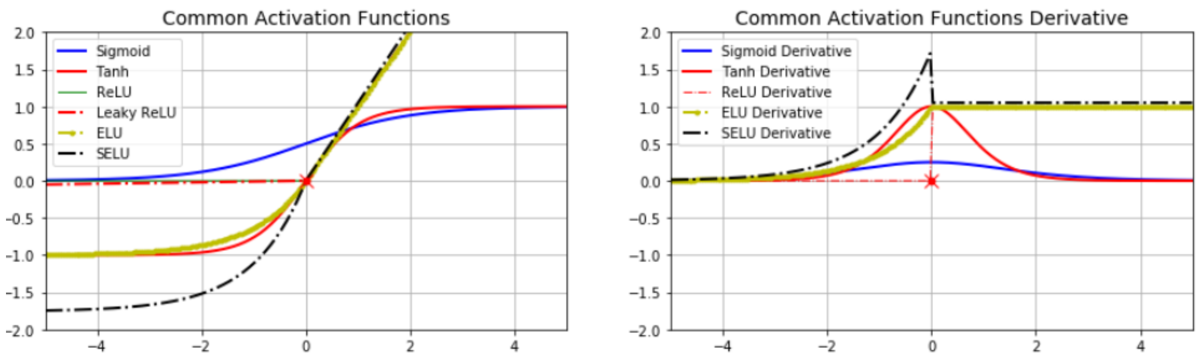


Figure 2.13: Most common activation function and derivatives

Everything described up until now considers only how the network processes the inputs to provide an output. Now, the training procedure will be briefly described below.

Basically, a cost function (also called loss function) has to be defined, measuring how well the algorithm performs in the problem. The objective is to minimize that function that can assume the shape of a mean squared error like the following:

$$C = \frac{1}{2} w(\|y(x) - a\|)^2 \qquad (2.1)$$

such that the model provides the best answers possible.

There are several possible functions nowadays that can be used to be minimized, depending on the application some might be more efficient than others. Some other examples are Poisson class, Crossentropy function, Cosine Similarity function etc.

The complexity due to the great dimensionality of the problem unfortunately makes it impossible to define an analytical solution. Rather, the most convenient possibility to find the minimum is to consider an optimization algorithm. The first algorithms used in this model worked by analyzing the loss function gradient and taking small steps in the opposite direction (Gradient Descend algorithm).

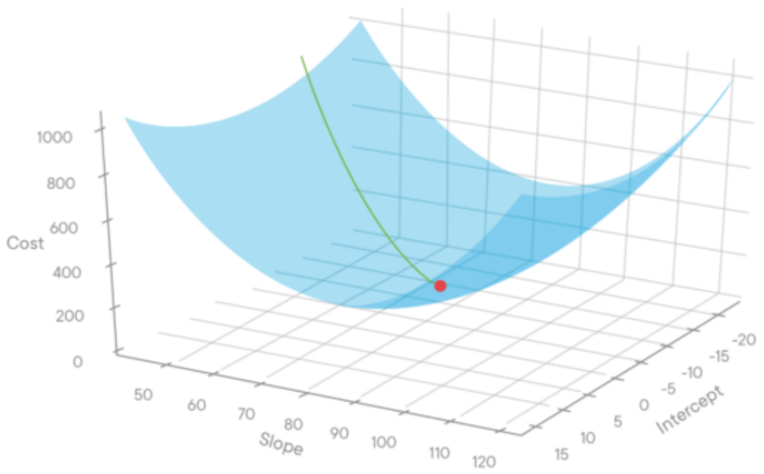


Figure 2.14: Optimization principle

Even in this case there are currently many possible algorithms that can be adopted: Adam is the most popular, but also Adagrad, Adadelta or SDG (stochastic gradient descent) which is a more stable variation of the algorithm just explained.
Operatively then, the optimizer considers the loss function and by taking as input the differences between the output layer values and the ground truth values it computes all the deltas to be applied to the weights of the network such that the new model is one step closer to the minimum of the loss function. The weights are thus updated, and the next sample is considered. (Nielsen, 2015)

3. Tools and Methods

This chapter represents the core part of this work. After the analysis of the tools, the methods section will present into details the developed algorithm by going into details of each step, considering accurately all the decision taken, relevant issues encountered and so on.

3.1 Tools

In this section will be described brefly the tools employed for all the real operations required to carry out this work. Specifically, the description of the hardware (including the vehicle, the radar and the lidar), the software and the description of the environment where the data is sampled.

3.1.1 Hardware

The ATB institute that supported this project made available for research purposes the Irus vehicle (Figure 3.1).
This is an automated vehicle, with the possibility to switch to remote control at will. The motion is provided by an internal combustion engine and the steering dynamics relies on the four wheels steering principle.
The electronic network through which the signal moves on the vehicle is the CAN bus. The CAN bus is a multi-master protocol, and at each communication is associated a level of priority (the lower the level, the higher the priority). During the initial phase of the transmission, the peripheral sending the message with the highest priority gets the control over the whole network, in this way the possible conflicts occurring when two peripherals are trying to send two signals simultaneously are solved. The protocol also defines how the information must be sent between two interfaces, how it has to be structured and what should be contained aside from the actual data to be delivered. This protocol is very common in the automotive industry and also for automated vehicles, it is low cost and does not require long and complex wiring.

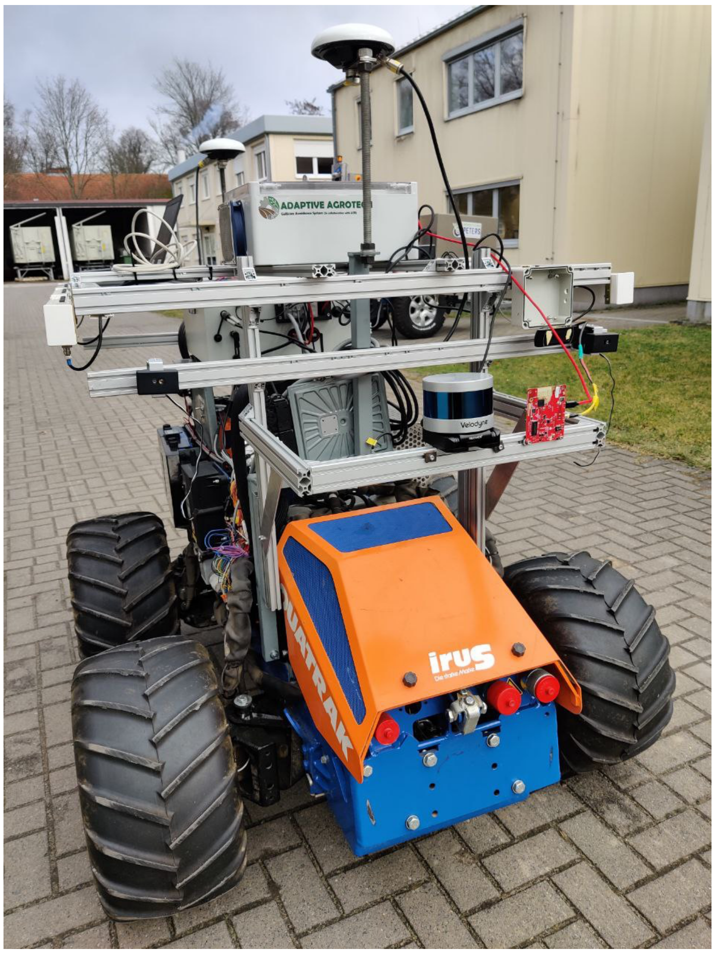


Figure 3.1: Irus vehicle

The vehicle is equipped with two computers, one exclusively dedicated at solving machine learning tasks, comprehensive with gpu modules for fast computation, and the other one considered the main computer on which the linux operative system is installed together with the ROS libraries. As one can see from the picture below (Figure 3.2), the presence of sensors and related wiring is massive all around the vehicle.

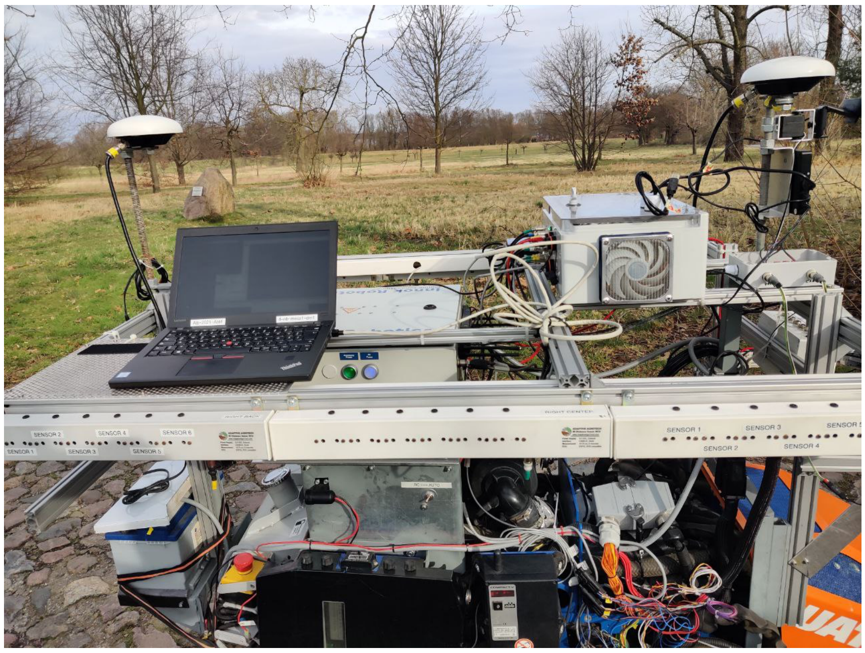


Figure 3.2: Irus hardware and connections

The millimeter-wave sensor device AWR1843BOOST is an off-the-shelf radar produced by Texas Instruments. in the User's Guide one can find the operating the operating frequency range as 76 - 81 GHz. This radar is considered to be a right solution for medium range radar applications, thanks to its small form factor, low power and highly performing range resolution (less than 4 cm). It also has 150 meters of detection range. The drivers necessary to work in the ROS framework are developed by Dr. Leo Zhang from the University of Arizona and are available for free on GitHub.
Finally, the official price for AWR1843BOOST is 299 USD.

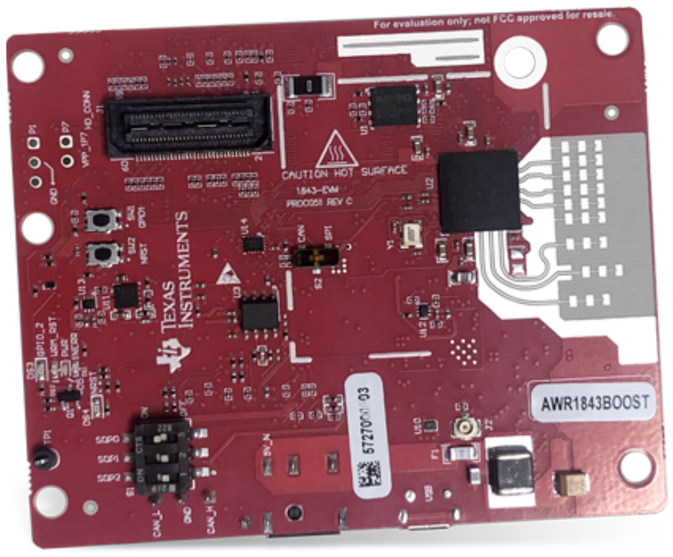


Figure 3.3: AWR1843BOOST Texas Instruments radar

The Velodyne Lidar’s Puck is a small and compact lidar that is performance and power optimized for usage across a variety of applications ranging from automotive, mapping, robotics, security, smart cities and more. The Puck has a range of 100 m and generates up to ~600,000 points/second, across a 360° horizontal field of view and a 30° vertical field of view. It uses proven, Class 1 eye-safe 905 nm technology with substantial autonomous fleet validation, making the Puck a sensor of choice for lower speed autonomous vehicle (AV) applications. The Puck has best-in-class power, which enables operation over a wide temperature range. It's use of off-the-shelf components enables enhanced scalability and attractive volume pricing.
Like for the radar, all the drivers permitting the compatibility of the sensors with the ROS environment are available for free on GitHub.

Figure 3.4: Velodyne Radar Puck

The ROS packages for both sensors provide all the necessary node (written in Python) and nodelets (written in C++) to output the signal data to a higher level corresponding to the PointCloud2 message. The frequency at which the two sensors output their respective messages is slightly different by few milliseconds, but because of the way the algorithm is structure, this will not affect its performances.
It has to be noticed that since the lidar sensor is a rotary sensor, the cloud generated has a horizontal field of view of 360°. Thanks to its ROS package though, it is possible to change through dynamically reconfiguration its field of view to the desired angle. In the specific case, 120° (±60), which is coherent with the radar's FOV angle. On the vertical direction, both systems have ±30° field of view angle.

The two sensors are mounted on the vehicle as shown in the picture down below (Figure 3.5). This positioning information is coded inside the TF package that will be presented later, in order to have complete spatial consistency between the distances computed by the two.

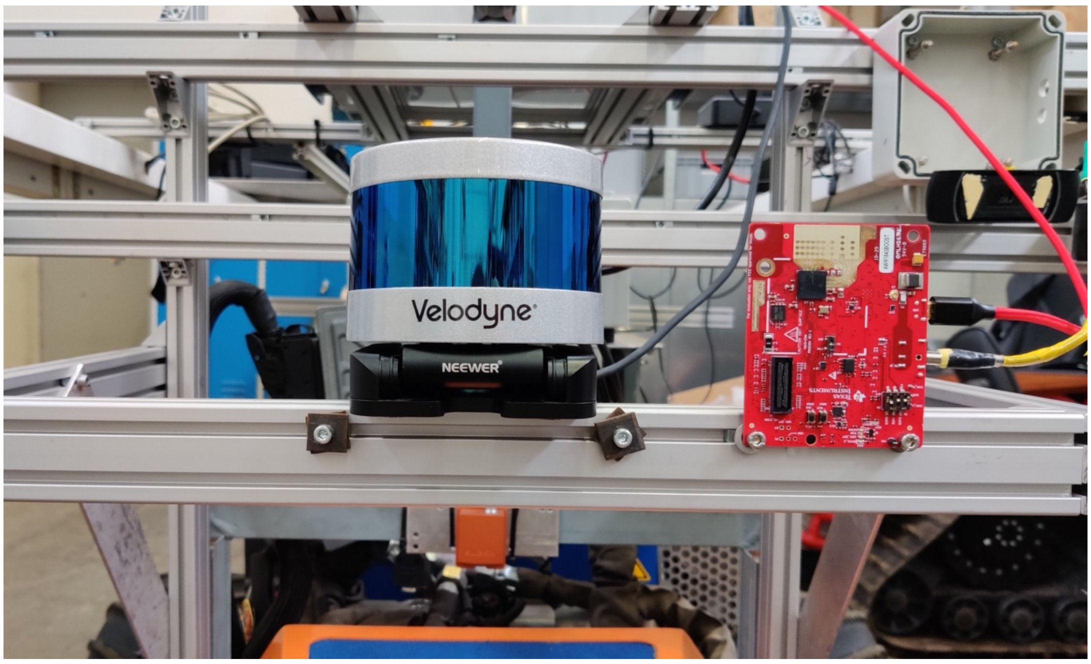


Figure 3.5: Setup arrangement

3.1.2 Software

On the IRUS vehicle the operative system is Linux Ubuntu 18.04, on which it has been installed the version of ROS called Melodic Morenia. This version is currently more developed, supported and stable with respect to the newer Noetic Ninjemys released for Ubuntu 20.04. This is the main reason why an older version was chosen.
The vehicle network is accessible through either Wifi connection or SSH ethernet protocol. For this work, the second one has been used. The code editor used has been Visual Studio Code while the supporting drivers were already present and working on the machine.
The development of the algorithm is divided between Python code and MATLAB mainly depending on the easiness of implementation. The parts related with ROS and the robots were entirely written in Python as it guarantees the full support. Also, all the algorithm related with the machine learning algorithm and the clustering were entirely written in Python. The Keras package of Tensorflow available in Python provides user-friendly tools and functions to set up and run a multitude of machine learning algorithm with easiness.
Then, the relevant data were then moved from there to MATLAB using *bagfiles* which are native ROS files. The principles of the bag file are recording and playback: a bag file is generated by recording the value published by topics; the playback feature given an existing *bagfile* can also be reproduced by time simulation i.e., the file publishes the same value on the same topic as during the recording.

On MATLAB it is performed all the rest, in particular related with data handling, in terms of variables creation and organization, all the phases of normalization, standardization and reconstruction, the lidar side of the dataset generation and finally, the largest part of the plots.

3.1.3 Environment Description

The specific outdoor environment on which this model is trained and developed for, is related with the agricultural context. In particular, this work focuses on the typical paths between the storage, where a loader might be kept at rest, and the field in which it solves its farming tasks when deployed.
The places in which the loader has to move are typically outdoor and full of obstacles (Figures 3.6). Obviously, the type of structure is different either compared with the open crop field, in which the trajectory planning is sometimes supported by GPS or a system for triangulation with fixed antennas because of lacking in visual reference for sensors like radar and lidar and the major characteristics distances of it, and compared with an indoor environment, because of the closeness of the walls, the presence of a ceiling etc. In some ways, the world structure in which this thesis is focusing on can be considered a mid-way between the two, where features and shapes are recurrent, especially from the indoor side: indeed, this kind of ambience, is compliant with the Manhattan assumption (Coughlan et al.., 2000), which has been developed to perform Bayesian interference indoor, but it can be nonetheless applied to a lot of different environments, even outdoor as long as the environment scanned keeps having a structure characterized by objects with geometric rigidity like buildings and walls.

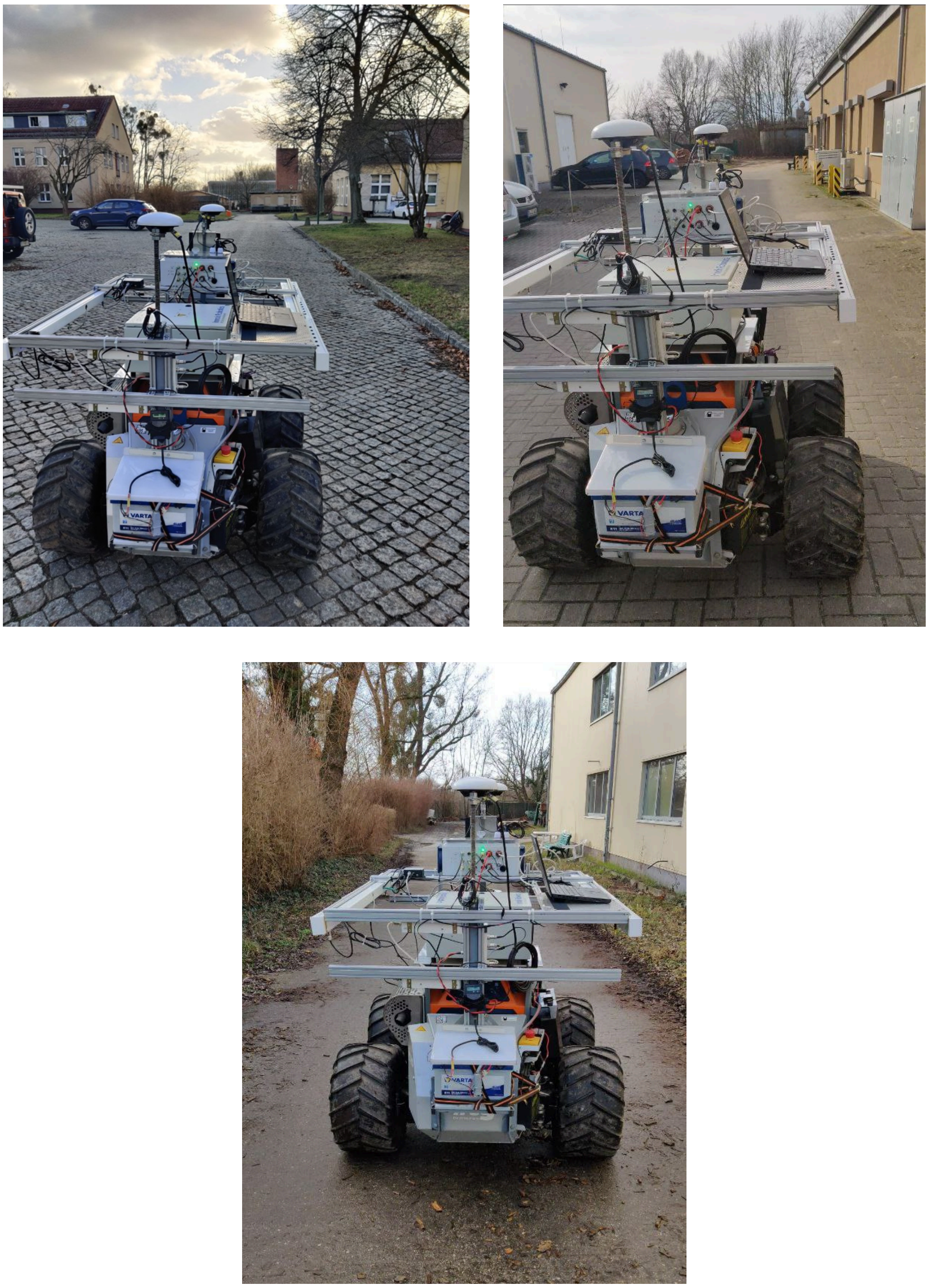


Figure 3.6 (a-b-c): Training and testing environment

Anyway, the solution proposed is not entirely dependent on the environment for which it was developed, but also on the characteristics of the vehicle itself. One main limitation is in the system's performances as they depend on the object enlightening by the sensors: the algorithm is not developed to build information where there is no information, if a certain object is not sampled at all or sufficiently, it will not be present in the final map.

From the algorithm perspective there is effectively no difference between the various environments. Even if this hypothesis was not properly tested, in this way it would be possible to keep the same functionality in many other environments. In support of this theory, the typical obstacles or features sampled within the map, can be considered similar to many other contexts, as explained in the Manhattan assumption, guaranteeing proper functioning even outside of a rural context.

3.2 Methods

In this chapter the developed algorithm will be described into details. The code related to the most relevant phases is available in the Annex section at the end of this document.

3.2.1 Map transformation

The package octomap_server is used here as a black box to generate the initial map. It takes as input the PointCloud2 messages coming from the sensors' ROS packages distributed by the respective vendors, and the pose of the vehicle provided by the GPS already installed on the vehicle. There are many parameters that can be tuned for this tool, but the ones used are:

- the dimension of the voxels composing the map, therefore the map resolution which has been kept as low as computationally possible in order to get enough details of the map.
- the upper and lower bounds for the map generation: in this case it was necessary to cut off all the lidar points composing the ground, as the radar is not able to sample any points out of it. On the other side, the upper bound was set to 2.7 meters considering the maximum height of the vehicle to be 1.5 meters

It has to be noticed that Octomap outputs the map in various possible formats. The one used for this work is through MarkerArrays. In brief, for each point of the map, a Marker containing the three coordinates of the point is generated. This solution is far less memory consuming that using voxel grids. In case of 2D representations, this works very well: the dimension of the map is limited and works nicely with very advanced imaging techniques that can be applied.

Another relevant factor to always keep in mind is that Octomap outputs the entire message of the map at each iteration. In turns, this causes the size of the information to be monotonically increasing, growing at each step.
The whole management of the frame's relationship is handled by the TF library in ROS (Figure 3.7). In this project, the most relevant frames are:

- *map*: fixed absolute frame;
- *base_link*: base frame of the vehicle, positioned at ground level;
- *gps_link*: the frame corresponding with the GPS antenna, the information provided by this system refers to the position of this frame;
- *velodyne*: frame associated with the Velodyne Lidar position;
- *ti_mmwave*: frame associated with the TI Radar position;

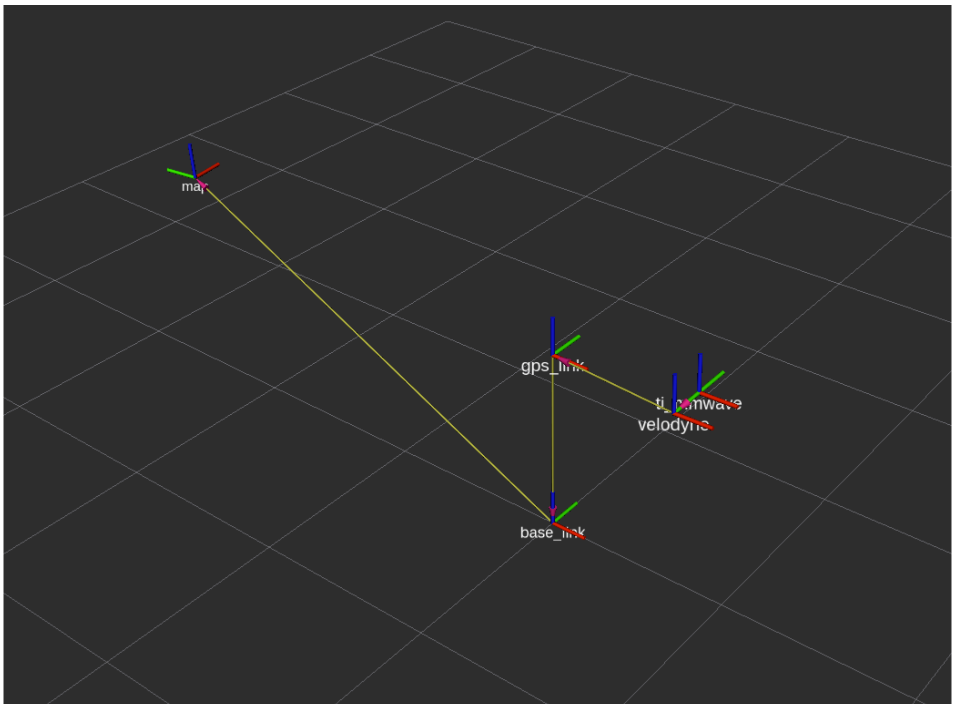


Figure 3.7: TF representation of the vehicle

All the relations intra-robots are completely static. The only one that is time dependent is the one between the *map* frame and the *base_link* frame. This transformation is computed through the GPS data, directly acting on the *gps_link*. The TF library can however perform this frame transformation easily and provide the correct robot motion with no delay.

In order to visualize what is happening and how the map is been built, RViz visualization tool is used. As already explained in the Fundamentals section, this framework is key for proper understanding of all the complex dynamics of 3D mapping (Figure 3.8).

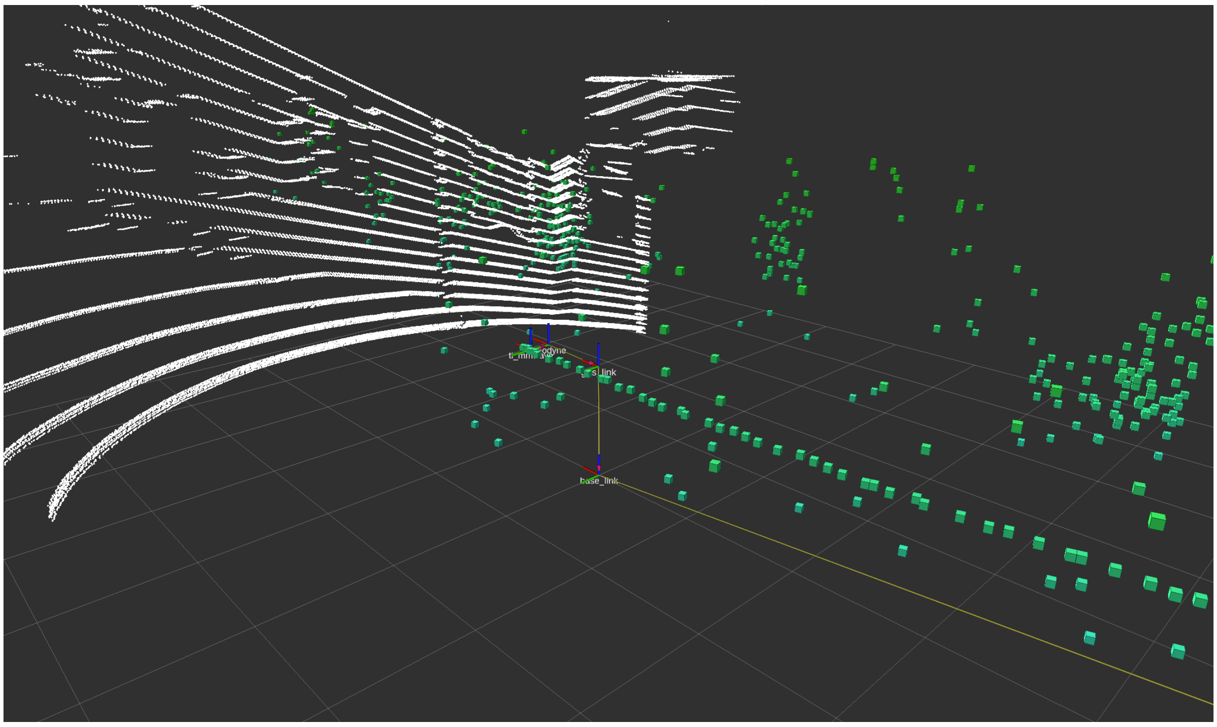


Figure 3.8: RViz visualization for easy understanding and debugging

3.2.2 Denoising

As previously said during the analysis of radar sensors, those are typically very noisy. Moreover, in this particular case, the sensors deployed is an off-the-shelf commercial radar, not intended to have high performances. Therefore, it is crucial to deal as soon as possible with this noise aspect.
When analyzing the raw data from the radar and the lidar, it is immediately noticeable the different level of noise characterizing the two maps (Figures 3.9 and 3.10). As one can see, it is very hard to clearly recognize which is the structure of the environment sampled during the whole experiment. Conversely, the map produced by the lidar is very neat and understandable.

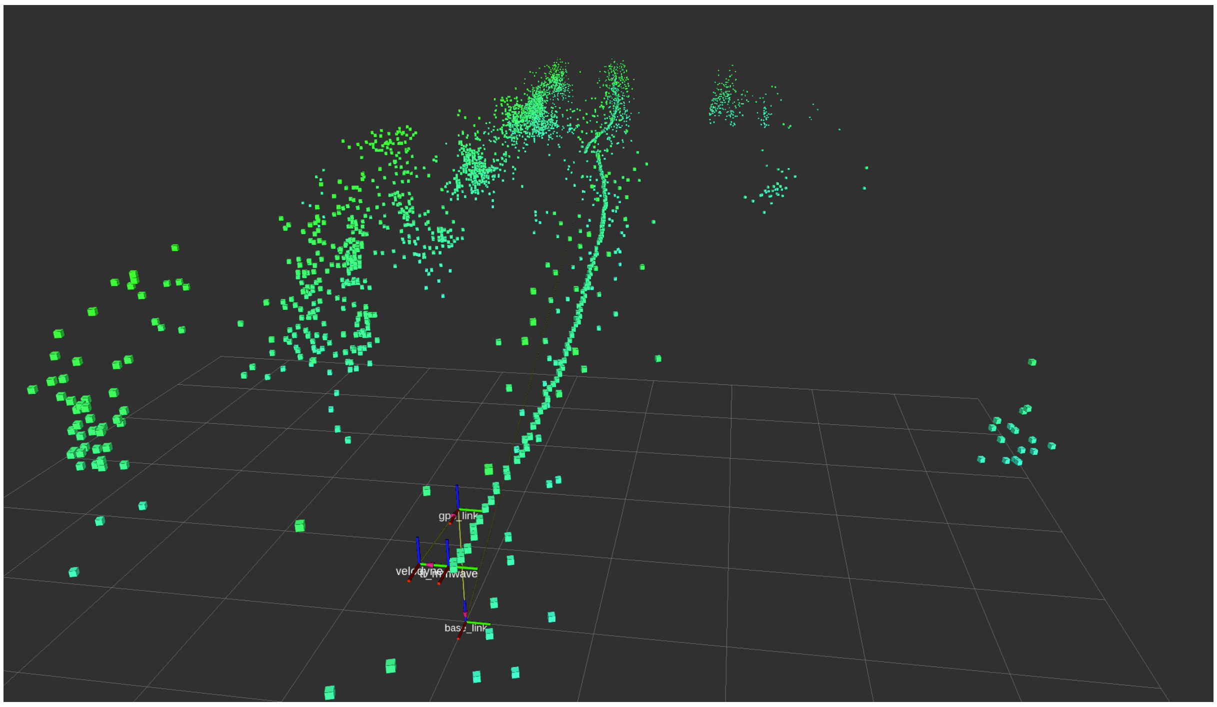


Figure 3.9: Radar original map

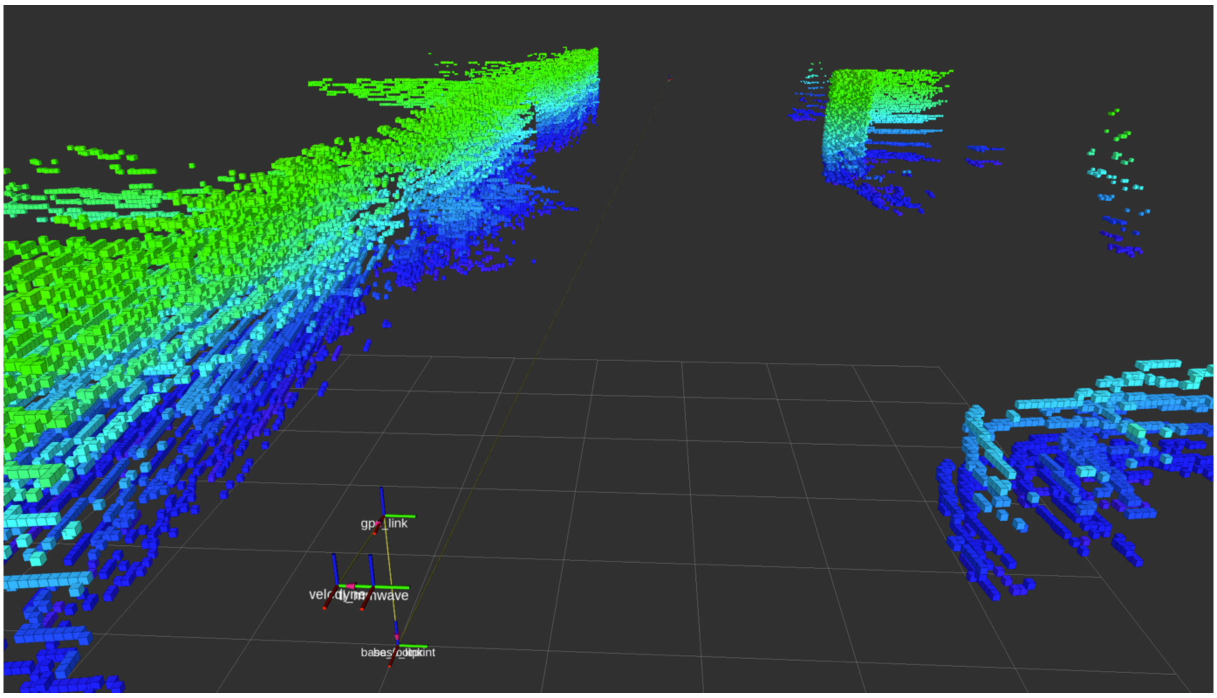


Figure 3.10: Lidar original map (Ground Truth)

First of all, focusing on the radar side, it is immediately noticeable the very dense curved line of points laying on the plane $z = 0$, perfectly following the trajectory of the vehicle. Even though this phenomenon has not been analyzed into details, the signal should be caused by an object reflecting the radio wave at null distance from the

sensor. Initially, once observed this phenomenon, the sensors have been mounted in another position in order to try to avoid any kind of close reflection as such, but even with this setup the issue remained still. The most probable cause of this issue is auto-detection: in brief, the sensor is always sampling itself and this marker is unavoidable using only the *octomap_server* settings. As showed in the section describing this package, no minimum radius for the cloud transformation can be set.
Moreover, comparing the two maps (radar and lidar), one can notice another region characterized by strong noise which is surrounding the trajectory of the vehicle. With respect to the previous case, those points are sparse and randomly positioned. Apart from those, it can be seen that the radar seems to model the shape of the environment overall, capturing the main element of the map even if still characterized by various outliers.

This algorithm focuses on the first two main sources of noise. Luckily, removing those points from the raw map as first data processing operation, is simple: as aforementioned, all this noise sources are focused close to the radar trajectory.
During this work, no algorithm for full automation was coded, but rather a concept for further development and actual implementation. For this phase therefore, in this work the noise has been manually removed using MATLAB (Figure 3.11).

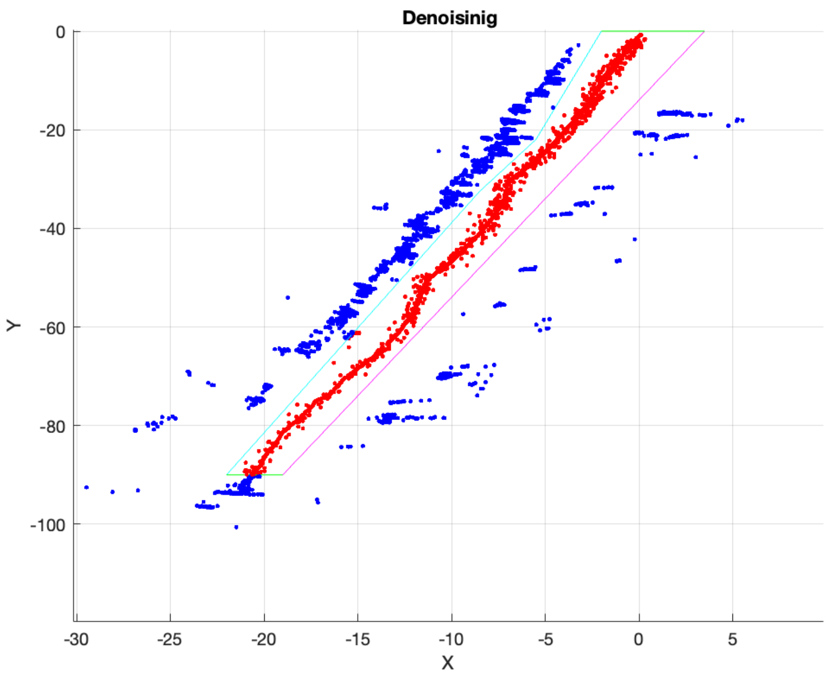


Figure 3.11: Denoising segmentation algorithm

Furthermore, a final note on this outliers' removal. Of course, all the points that has been removed are certainly noise. As we will later see in the results sections, almost the 40% of the radar original data are thus classified as noise, leaving after this processing a cloud a sparser cloud.

3.2.3 Single-linkage clustering, *fclusterdata* and Silhouette Score

Before entering the main cluster phase, it is necessary to introduce two algorithms which of fundamental relevance to understand the following paragraph: the *Single-linkage* algorithm and the *Silhouette Score*.

The *Single Linkage* algorithm, which is a type of agglomerative hierarchical clustering. Basically, the aim is to start considering each point (also called singleton) of the original set as a separate cluster and at each iteration merge the two closest clusters existing at that moment. The distance between two clusters is defined by the minimum Euclidian distance between the points belonging to different sets.
Once all the singletons have been merged into one cluster, the algorithm creates a hierarchical tree based on the joining sequence. This tree graph is more commonly called dendrogram (Figures 3.12 and 3.13) and it plots the cluster singletons (on the x-axis) versus the cluster distance (on the y-axis).

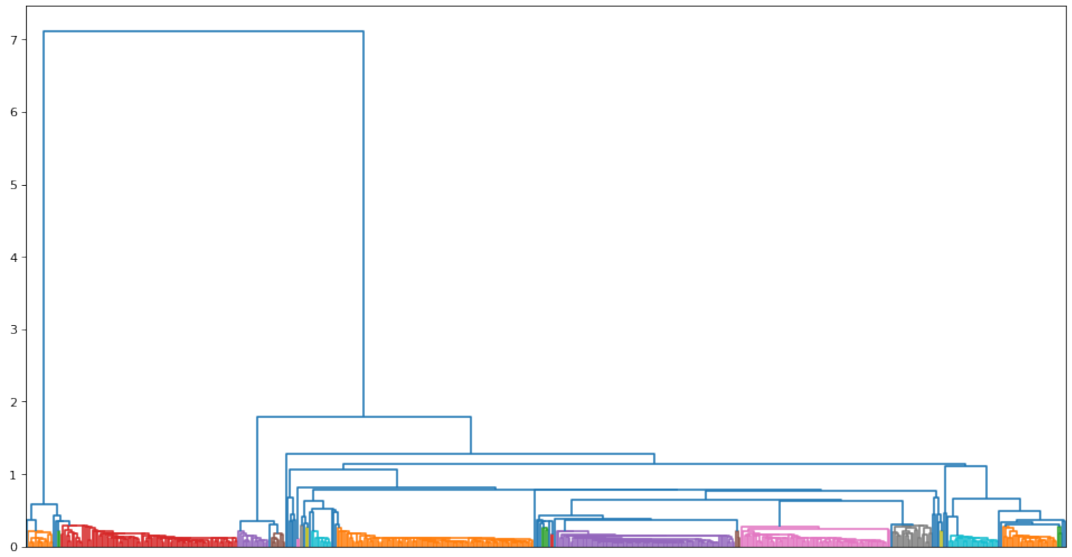


Figure 3.12: Full dendrogram

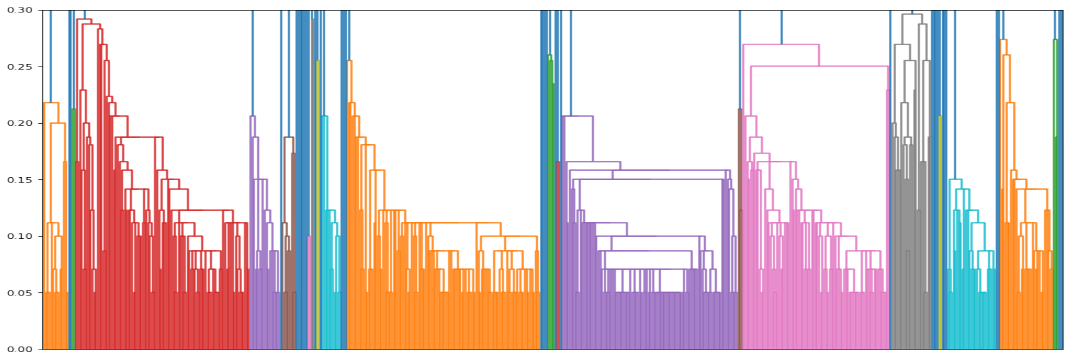


Figure 3.13: Detail of found clusters of Figure 3.12

The chart is ideally cut with a straight line on the value of the distance parameter, which is 0.3 in this work. The clusters beneath that cut are the final clusters.
However, considering what has been said up until now, it is clear to see, as the iterations continues, the minimum distance between clusters computed is monotonically increasing with the iterations. Therefore, it is more computationally efficient if, when the algorithm finds that the distance between the two closest clusters is higher than the distance parameter, the algorithm just stops and output the final clusters as it does exist no other possibility of another merge between two sets sufficiently close to each other. In practical terms, the check is performed at each iteration, therefore the algorithm never computes the full dendrogram. The name of the function performing

The *Silhouette Score* is calculated using the mean intra-cluster distance $a(i)$ and the mean nearest extra-cluster distance $b(i)$ for each sample. The *Silhouette Coefficient* for a single sample i, is given by:

$$S(i) = \frac{(b(i) - a(i))}{\max\,(a(i), b(i))} \qquad (3.1)$$

To clarify, $a(i)$ is the intra-cluster average distance. On the other hand, $b(i)$ is the average distance between a sample and the elements of the nearest cluster that the sample is not a part of. Note that *Silhouette Coefficient* is only defined if the number of labels is:

$$2 \leq n_{labels} \leq n_{samples} - 1 \qquad (3.2)$$

The best value of the score is 1, while the worst is -1. Values near 0 indicate overlapping clusters. Negative values generally indicate that a sample has been assigned to the

wrong cluster, as a different cluster is more similar. In our case, thanks to the *Single Linkage* clustering, all the points will have positive score.
When the silhouette score is at its largest (that is, close to 1) this implies that the 'within' dissimilarity (which is $a(i)$) is much smaller than the smallest 'between' dissimilarity (which is $b(i)$). Therefore, we can say that the single point assessed is 'well-clustered', as there appears to be little doubt that it has been assigned to a very appropriate cluster: the second-best choice (the not-currently-belonging cluster B) is not nearly as close as the actual choice (A) (Figure 3.14). (Rousseeuw, 1987)

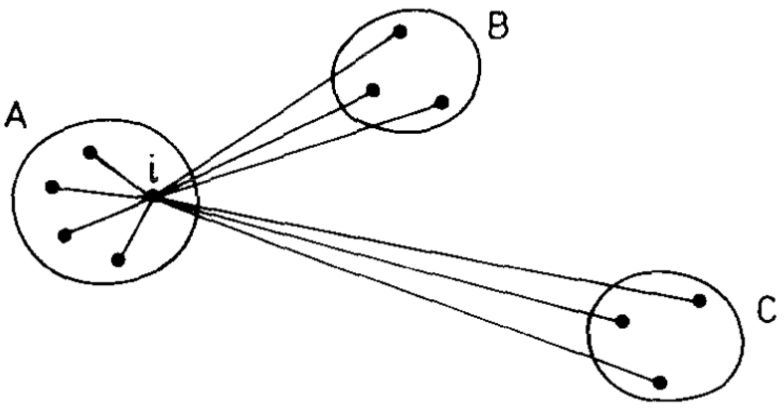


Figure 3.14: Silhouette Score distance computations

3.2.4 Clustering

The clustering algorithm aims at gathering points which are sufficiently close to each other, dividing the full point cloud into smaller sets. It has to be recalled here that in the context of this work, differently with respect to what has been done in the past research (see Chapter (1): Literature Review), no object recognition method will be applied. The first hypothesis made, regards the dimensionality of the problem. Even though the goal of this work is to perform mapping in three dimensions, it makes sense to consider only the top view, thus projecting the points against the x-y plane and performing clustering consequently (Figures 3.15).

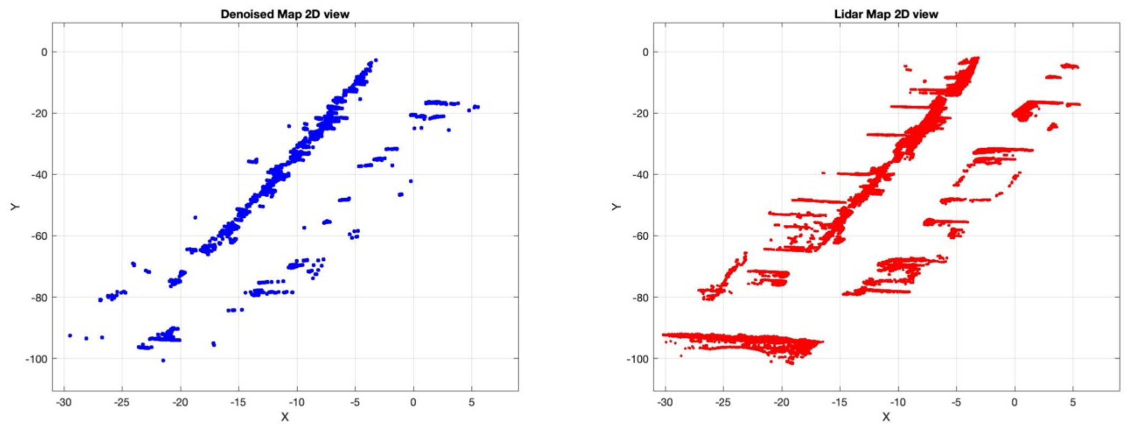


Figure 3.15: Radar and lidar 2D map from top view

Firstly, this simplification reduces the computational weight of the processes overall, as each time a single point is considered, the values used in the calculations are only two instead of three; the second and most relevant aspect, relates considerations on the average shape of a cluster with respect to the one of the objects most likely present within the map. As described previously, the environment on which this algorithm acts, is considerable compliant with the Manhattan assumption, but it is also filled with more natural objects. Typically, those objects have a softer geometry, like trees, shrubs, branches etc.

Another very relevant thing is that the algorithm considers only those points behind the vehicle. Having the robot's pose message from the GPS that also contains the orientation quaternion, it is possible to compute the plane orthogonal to the trajectory at every instant. All the points lying in the half space behind the robot are taken into consideration by the algorithm. The reason for this particular choice stands in the fact that in this way, the objects sampled cannot be shaped any better, as the vehicle has already overcome the object position. On the contrary, all the points still in front of the vehicle (and thus in front of the sensors) are not included, because there may be the chance that few other points, belonging to the same objects, will be sampled in future step iterations. In order not to miss any relevant detail before performing clustering, those frontal points are completely avoided. They will be considered once included into the rear half-space defined as a consequence of the robot's motion as no other information can be added. The only way to add further information would be to scan the same region of the map twice: this case though, will not be considered in this analysis.

The only parameter to be tuned for this algorithm is the maximum distance allowed for two points to be part of the same cluster. This distance was set to be 0.3m, much less than the average dimension of an object. The main reason for this choice is that the final number of clusters found, that also respects the threshold of at least 30 points, is the highest. As one can imagine, the more clusters are collected, the more points the algorithm is modelling, the better the map is capable of properly follow the shape of the original environment (Figure 3.16).

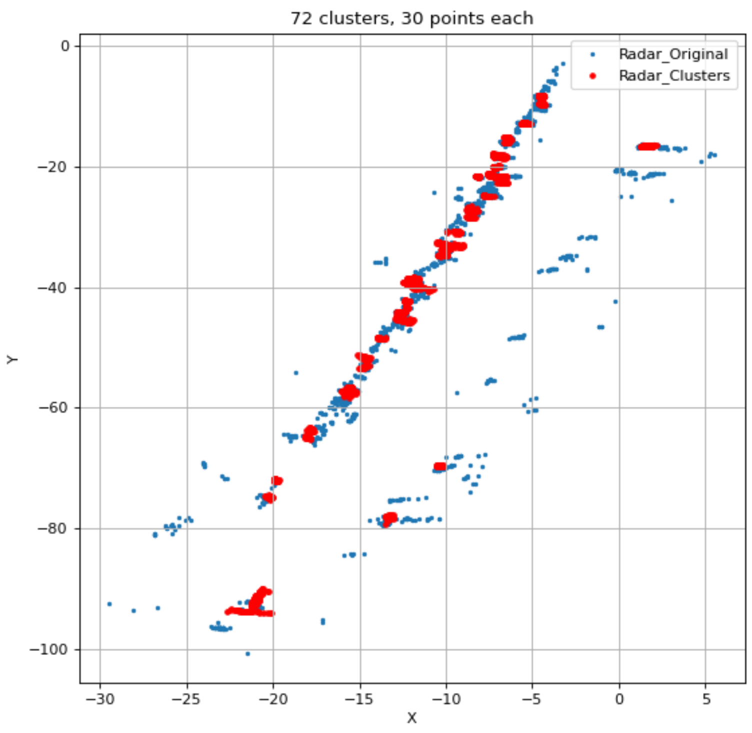
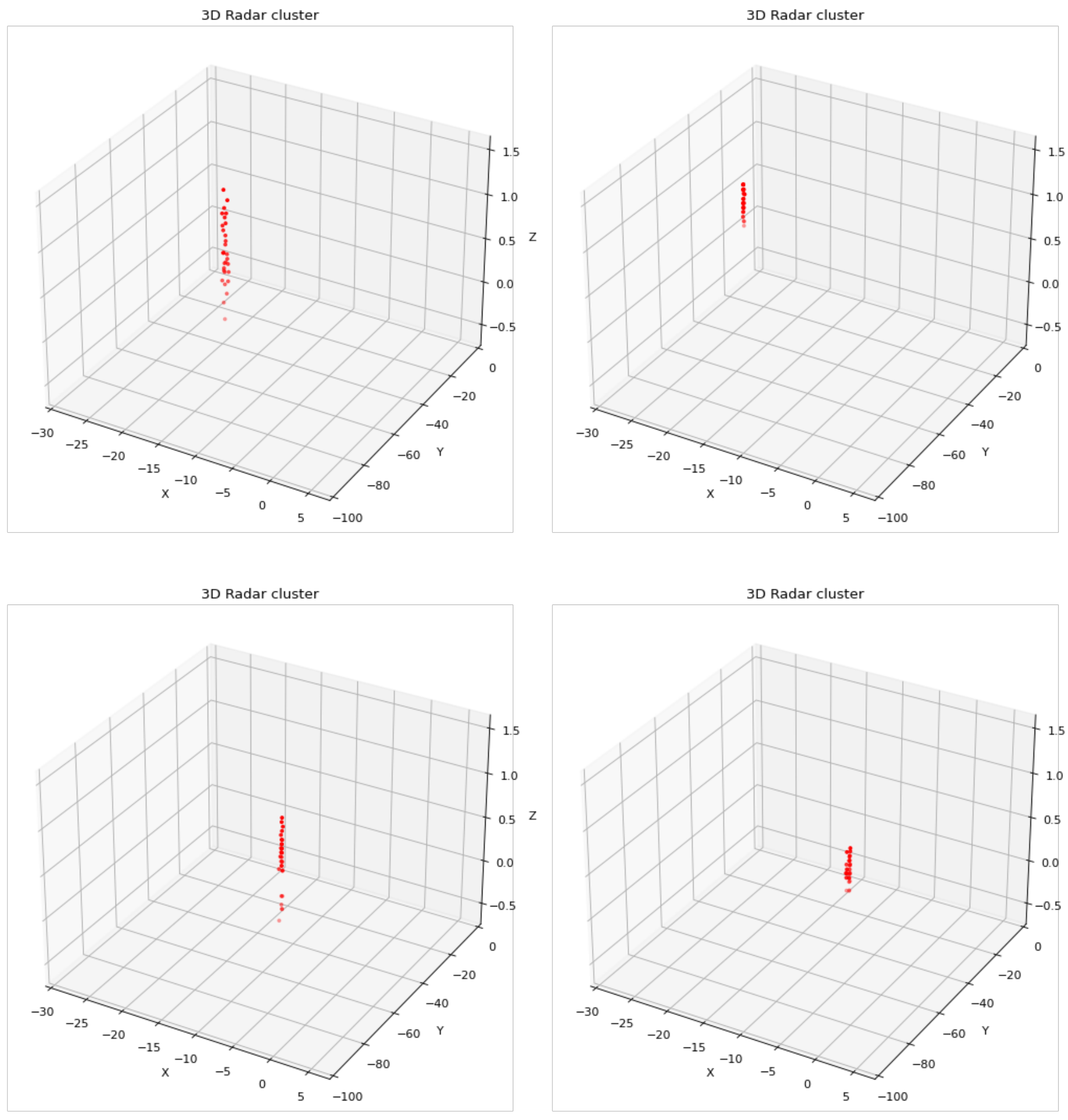


Figure 3.16: Clusters found against the original map

This parameter distance is indeed that one value maximizing the level of clusters found. In this way, the cluster is made on one piece of an object from which one has already gathered the maximum possible number of points that the radar was capable of sampling.
Once the correct points are passed to the clustering algorithm, it executes the *fclusterdata* function building the clusters up until the distance parameter is exceeded. Then the algorithm goes through the different clusters counting how many points they have. Only those with 30 or more points will be taken into account. Out of all the points belonging to each selected cluster, exactly 30 are extracted, due to the constraint imposed by the later processes. In order to remove the excess of points inside a cluster, the *Silhouette Score* is computed for each point and based on it the points are sorted from the highest to the lowest. The algorithm then retains only the first points, which are those with the greatest coherency within the set itself. Considering how the

clustering algorithm has been proposed, the final resulting set forming a single cluster are, very compact and dense. (Figures 3.17)



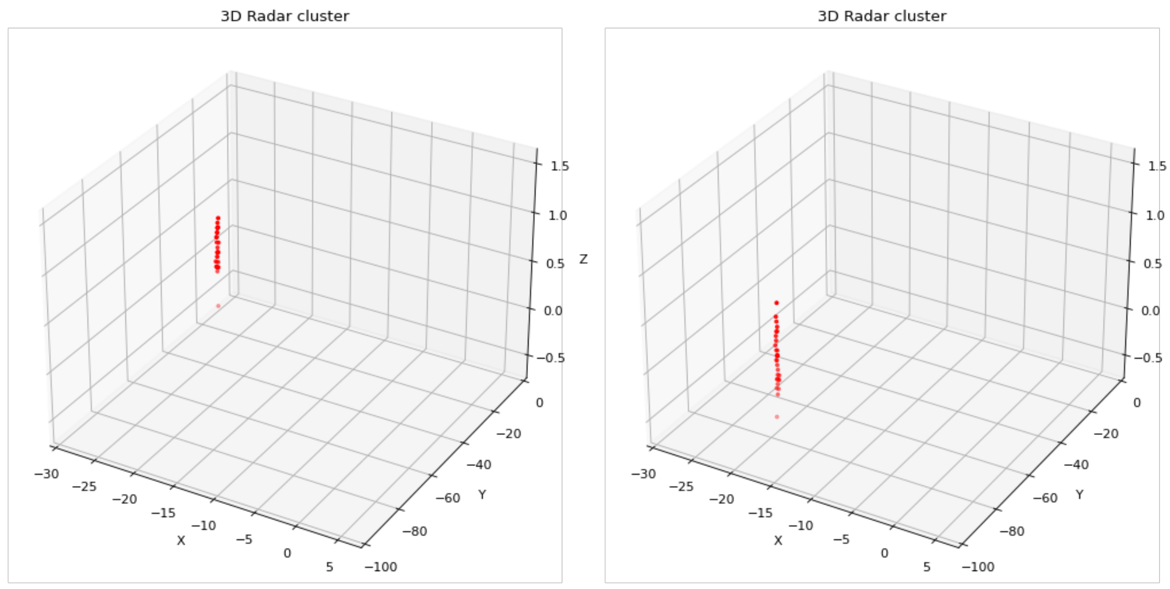


Figure 3.17 (a-b-c-d-e-f): Various radar clusters generated

Moreover, as we have already said, in order to best approximate the shape of the map without performing any object recognition (as proposed in other works present in the literature), it was necessary to build clusters as narrow as possible. In this way, another consideration can be made in contrast with the high level of noise typical of radar sensors. The presence of a high number of points enclosed into a small region of space, suggest that the modeled object is product of the observation of a real object. Normally, noisy observations are sparse, and, in many cases, they appear as outliers with respect to real samples. The algorithm implicitly does not consider these points, simply because of their position and sparsity. Hence, in addiction with the central noise removal operation made beforehand, it can be stated that all the cluster generated by this algorithm are parts of real object with very high probability. To better show this, we plot here the cluster radar map together with the lidar original map (Figure 3.18):

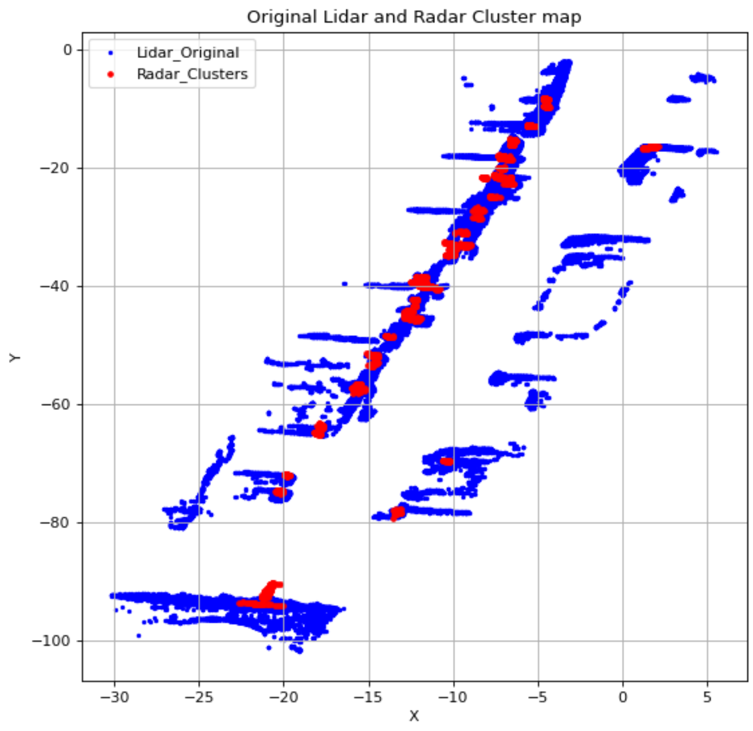


Figure 3.18: Clusters found compared with the lidar map

Unfortunately, as one can clearly see from the picture, the clustering algorithm may be severe enough to avoid generating sets where the radar is not able to sample sufficiently certain objects. Therefore, those are not contained inside the final set of clusters, and likewise before, treated as pure noise.
In order to solve this issue, it might be a solution considering slowing down the speed of the radar, therefore the level of points collected increases and permits an even more accurate shaping of the original map.

As detailed just above explaining the *Silhouette Score*, when this method is applied, not as a metric, but as cutting off threshold, it can be stated that it gives higher importance at those points which are closer to the center of the cluster, such that it produces circular 2D clusters with higher probability.
In three dimensions, this circular shape evolves into a cylinder, which can be considered a representative shape to approximate the objects contained in the context depicted above. The overall shape of the surroundings (from a top view perspective) is approximated using circular and very dense clusters, while on the z axis the different sets follow the vertical mold of the map (Figure 3.19).

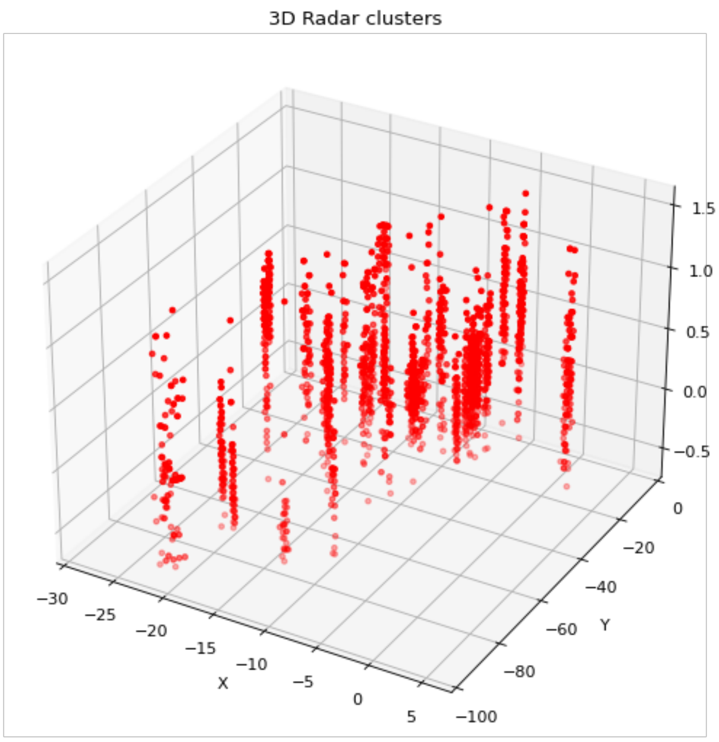


Figure 3.19: 3D cluster map

Finally, the presence of the neural network as subsequent phase imposes a not negligible aspect that is one of the main issues with which this work struggled. Indeed, the choices to retain only a very specific number of points for each cluster (30 in this case), has been made as a response to this problem. Basically, as will be later better explained, the perceptron does not have the flexibility to take as input a different number of values than exactly the dimension of the first layer of the perceptron. Hence, once decided this design parameter, it affects the whole algorithm chain.

3.2.5 Normalization

Given the trajectory and the points belonging to the cluster to be transformed, firstly the position of the centroid cluster point is computed. The radius of the frame rotation is obtained as difference between the centroid and its closest point to the trajectory of the vehicle. Now all the points belonging to the cluster are rotated in order to bring that radius line horizontal. The rotation center is the point belonging to the trajectory and the rotation angle is computed from the derivative of the trajectory curve (Figures 3.20).

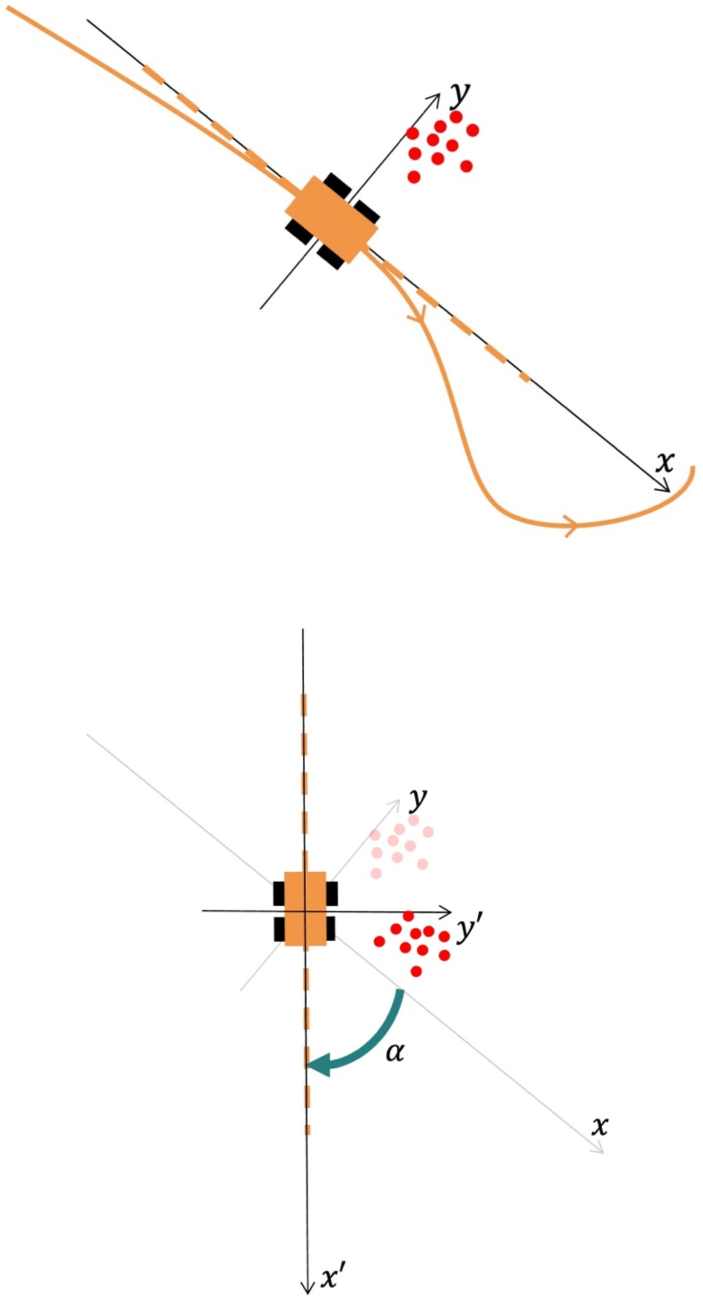


Figure 3.20 (a-b): Principle of frame rotation

Now that all the clusters are oriented the same way, a patch is generated keeping the trajectory point considered even before as center point. The patch has the following dimensions: 27 meters orthogonally with respect to the trajectory direction, 20m longitudinally and 2.7 meters vertically along z. This box is normalized into a cube of dimensions 1x1x1 having the trajectory point positioned as (0.5,0.5,0) (Figure 3.21).
This normalization overall permits to keep both the coherency of the shape and the position with respect to the robot perspective. This information, even if implicitly

stored inside the values of the coordinates of the clusters, provides more capability for a best modeling by the Neural Network Perceptron.

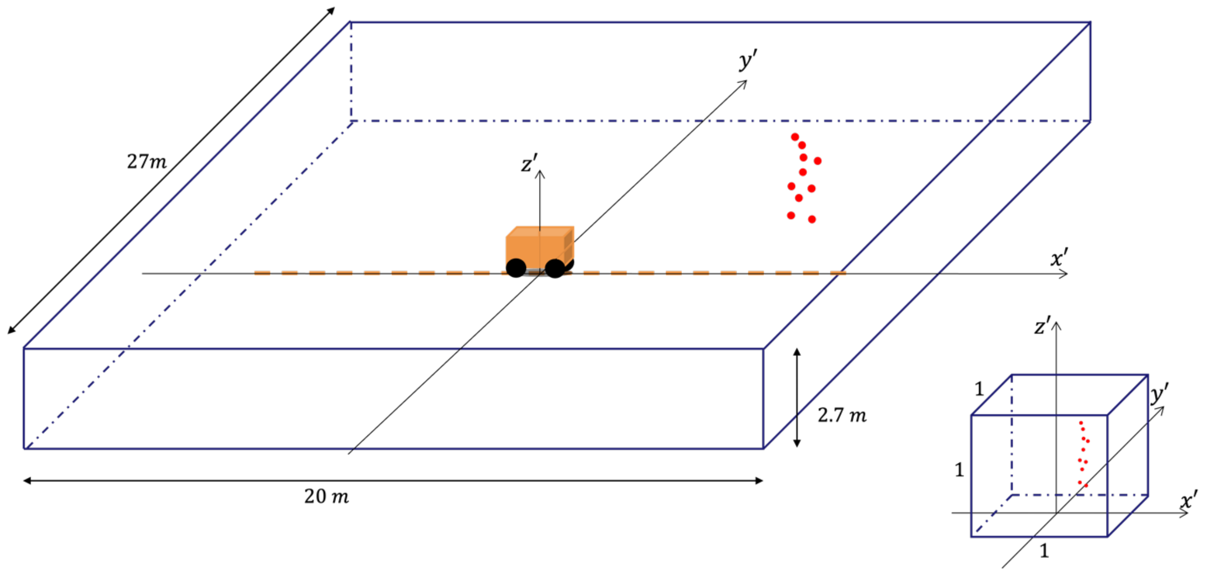


Figure 3.21: Principle of Normalization

To keep the data to rebuild the map afterwards, an array preserving the centroid positions and the trajectory derivative is generated and stored into a .mat file.

3.2.6 Dataset Generation

Unfortunately, the literature does not provide any pre-trained perceptron for coordinates transformation or upscaling. Thus, in order to train the Neural Network, it was necessary to build beforehand a database of values. Various testing campaigns have been carried out with the setup described in the Tools section. To have the maximum consistency between the learning and the actual implementation, the procedure for the data collection and handling of the radar are the same. Also, the samples collected by the radar and the lidar are taken simultaneously and thanks to the TF package in ROS they are also coherent in space.
On the other hand, the lidar signal is treated in the same way with some slight variation: its raw PointCloud2 message passes through the octomap_server node and it is transformed into map.
Now, this map does not present any noise to be removed as we consider it to be the ground truth map, but also the accuracy of the lidar is such that there is no need to apply any denoising algorithm. Nonetheless, a couple of notes on the final map obtained: first, the ground has been completely filtered out because the radar is not capable of sampling any point of it; second, at the beginning of each session, the parameters of the lidar has been reconfigured such that the horizontal angle of field of

view was reduced to a value more similar to the one of the radar sensors. This is better explained in the Fundamentals section.

Now, it has to be said that there is no easy way at this point to proceed: given the radar clusters, producing other point sets from the lidar data which are representative of the radar cluster is not straightforward. Each set does not represent a finite object, but a part of it. Thus, the most reasonable way to produce those lidar samples is manually, by comparing the whole lidar cloud of a map and a single radar cluster.
Using the brush tool present in the MATLAB plot, only the lidar points representative of the clusters are kept and stored (Figures 3.22).

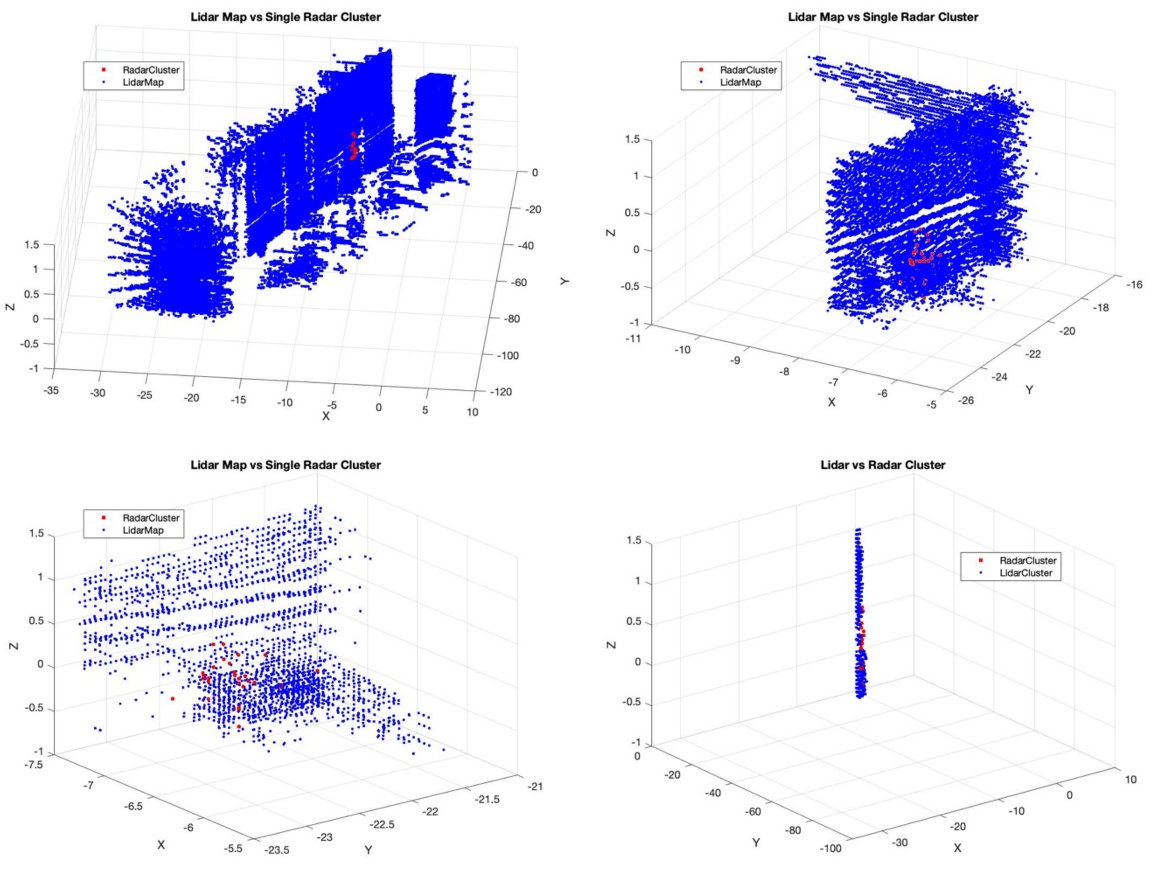


Figure 3.22 (a-b-c-d): Steps of lidar cluster generation with brush tool

Notice here that the two pictures above represent the same two clusters scaled in different ways.
In this particular case, the lidar cluster produce contains 2172 points. The algorithm from this point set produces a final cluster of 400 points (Figures 3.23):

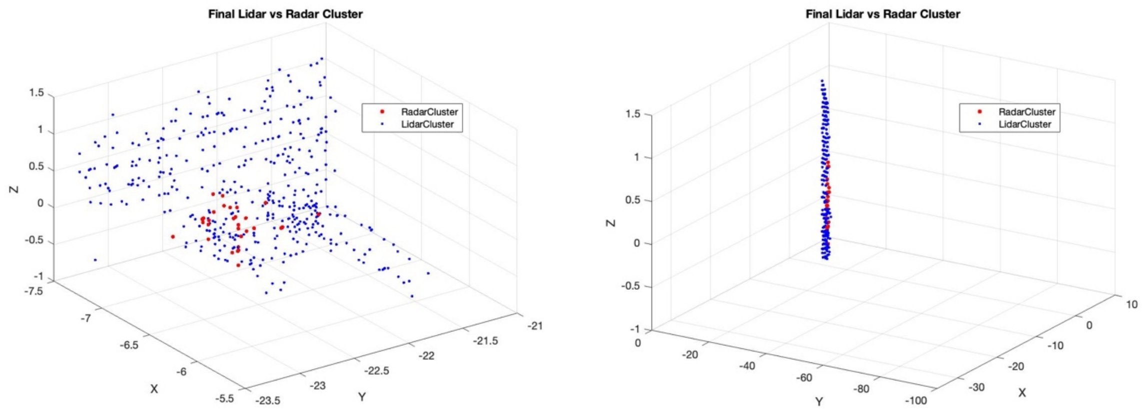
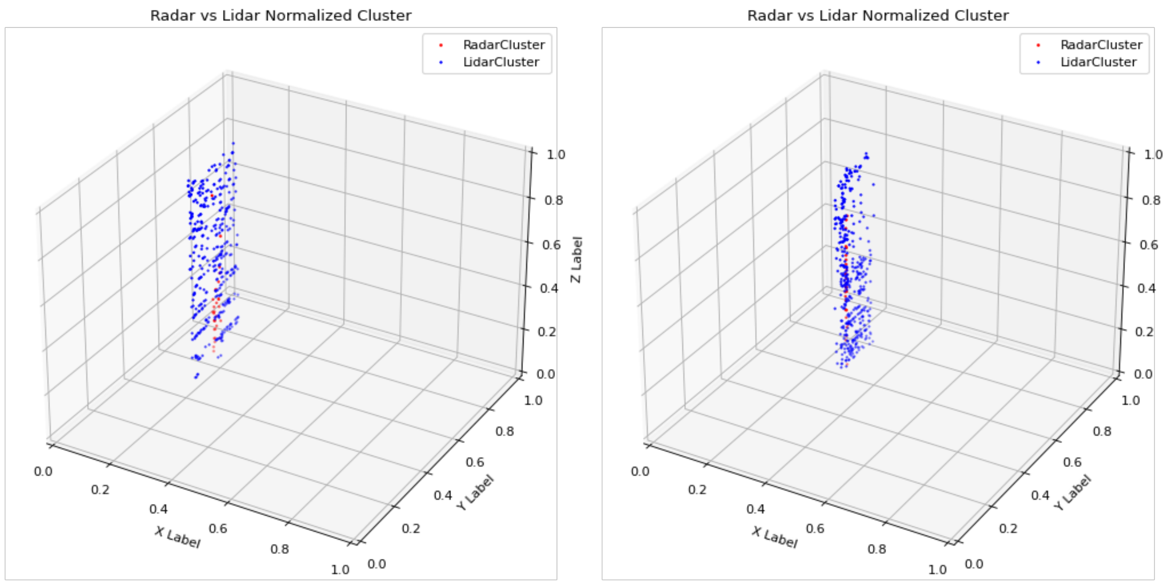
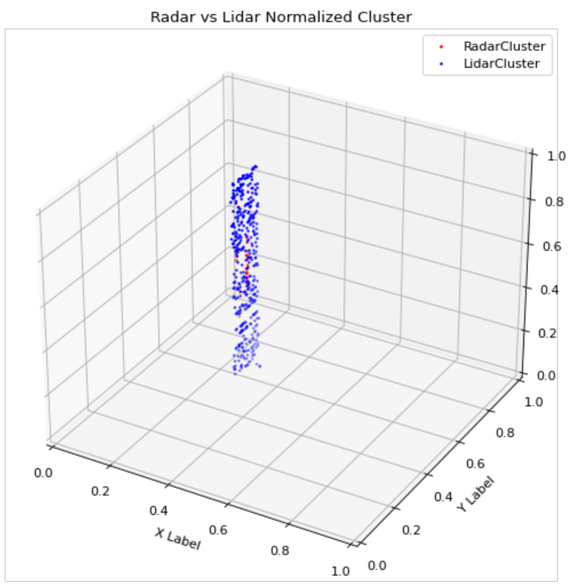
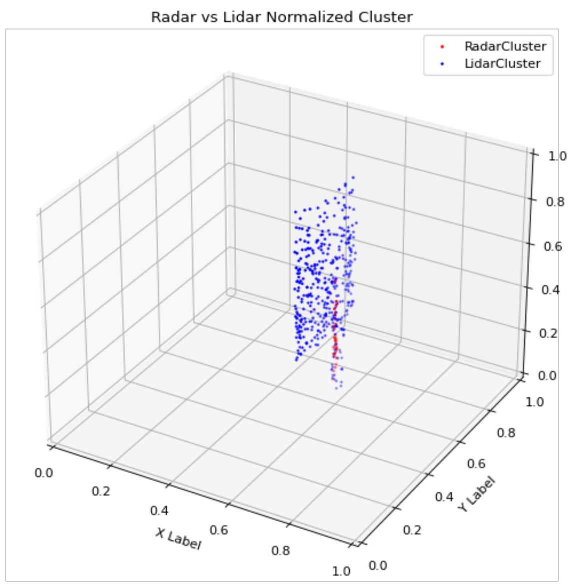


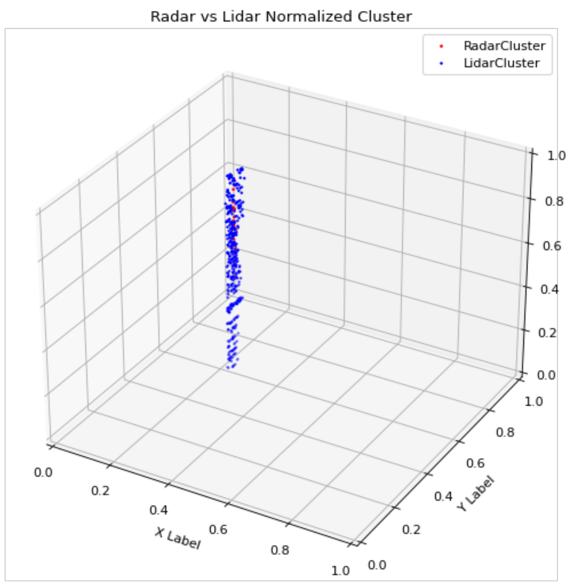
Figure 3.23 (a-b): Final clusters

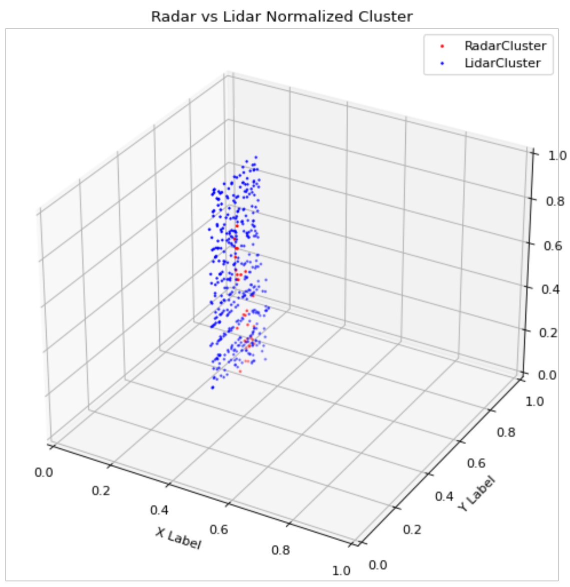
Now, applying the normalization algorithm to both radars and lidar patches, the clusters are ready to be used as training samples for the Neural Network (Figures 3.24). It has to be noticed here, the differences in the X, Y and Z values:




Radar vs Lidar Normalized Cluster
RadarCluster
LidarCluster
X Label
Y Label
Z Label


Radar vs Lidar Normalized Cluster
RadarCluster
LidarCluster
X Label
Y Label
Z Label


Radar vs Lidar Normalized Cluster
RadarCluster
LidarCluster
X Label
Y Label
Z Label


Radar vs Lidar Normalized Cluster
RadarCluster
LidarCluster
X Label
Y Label
Z Label

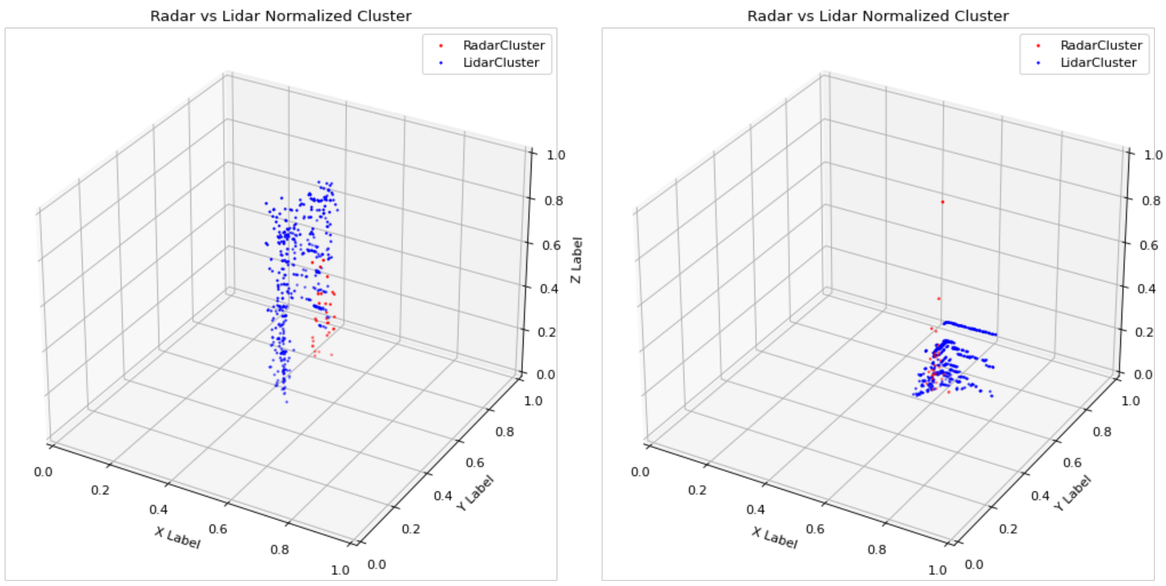


Figure 3.24 (a-b-c-d-e-f-g-h): Examples of training clusters

3.2.7 Neural Network Training and Up-Sampling

The neural network is a Multilayer Perceptron. It is composed of an input layer made of 90 nodes, three hidden layer of 1200 nodes and an output layer of 1200 (Figure 3.25). The information propagates from the input layer thanks to linear transformations and the result of this operation at each node is passed through an activation function that activates the node giving that nonlinearity property to the model. Once the values on the output layer are found, those values are compared with the ground truth values i.e., the coordinates of the point of the lidar cloud which are the same number of the output layer nodes. The comparison generates the delta differences that is then backpropagated through all the network to update each node weight.

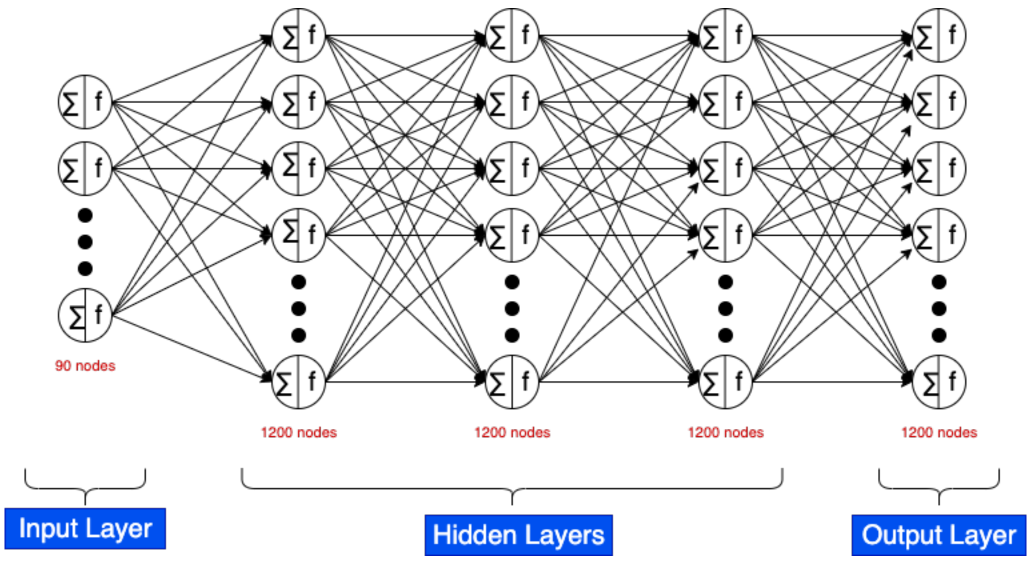


Figure 3.25: Neural Network structure

The 90 nodes of the input layer are given by the 30 points by 3 coordinates. The same applies for the output layer that has 1200 values composed by 400 points with 3 coordinates each.
The Multilayer Perceptron make use of the Mean Squared Error function as loss function to be minimized, which is widely used in this sector. Moreover, the optimizer chosen is Adam, also very commonly employed and considered one of the most stable optimizers available. The optimizer takes as input the value of the learning rate, which is 5e-5 in this case, while all the rest is left to its default value.
As additional source of stability, the initial weights have been randomly initialized considering a uniform distribution with the Xavier Golorot Method (Golorot & Bengio, 2010).

3.2.8 The GPS issue and dataset enhancement

During the samples collection arose an issue related with the GPS module. As previously mentioned, the localization signal provides the pose information at the *octomap_server* package for the cloud conversion into map. Unfortunately, due to the presence of trees, walls, and more in general disturbing factors, often the localization signal dropped or was completely misplaced. In turn, this caused the loss of several parts of the experimental campaign.

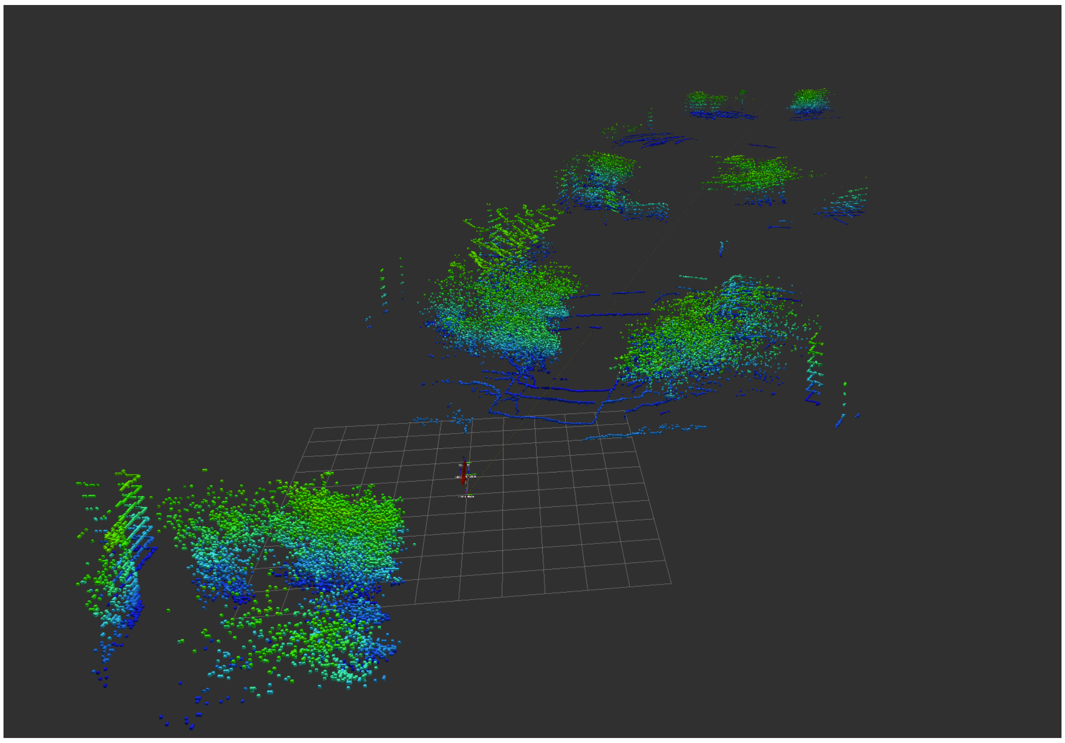
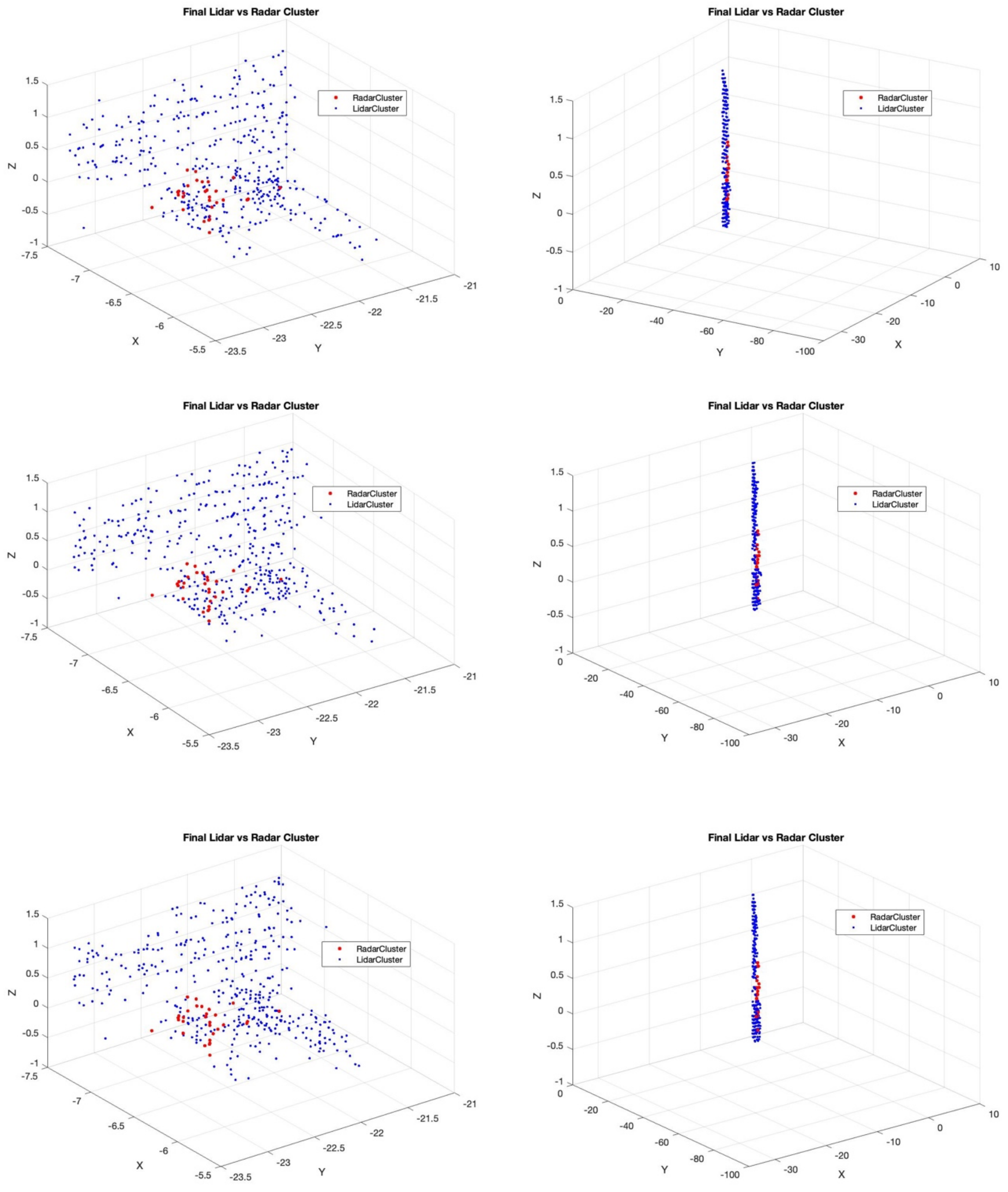


Figure 3.26: GPS module issue

The picture above (Figure 3.26) is representative of a case in which the GPS signal is poor: the robot is moving in the real world following a straight line, but the produced lidar map misses several parts of the real environment, and the few regions collected are also badly modeled, as the orientation quaternion defining the view direction of the robot keeps changing due to the signal instability. Considering few minutes of sampling at a time, among all the different sessions started, many of them contains only few seconds of actual useful sampling. Those had to be thrown away, as there is no way to recover missing data. Additionally, many others had minor issue, with just little missing spots, causing the lack of uniformity of the information and thus the necessity to manually intervene from time to time on the process.
As a consequence, when all the radar points samplings have been ultimately transformed into clusters for the training dataset, the final total amount was only 167 samples.

Moving now to the lidar cluster generation, this was performed through the procedure explained in the last paragraph: in brief, the procedure is based on manual extrapolation through visual overlapping using the *brush* feature available in the MATLAB plot environment. In this way, the coherent lidar cluster of each of those 167 radar clusters can be extrapolated. In that specific moment though, the cluster is not yet ready to be integrated into the dataset because it does not have a standardized number of points. Therefore here, it is possible to take advantage of the high density

of the lidar cluster, and by reducing multiple times the lidar clusters with a sufficiently large number of points, the result is a multiplication of the total number of samples. The performance of this procedure brought the dataset dimension up to 1263 unique samples couple.
Below we show the same radar cluster matched with the different combination of the same reduced lidar cluster (Figures 3.27):



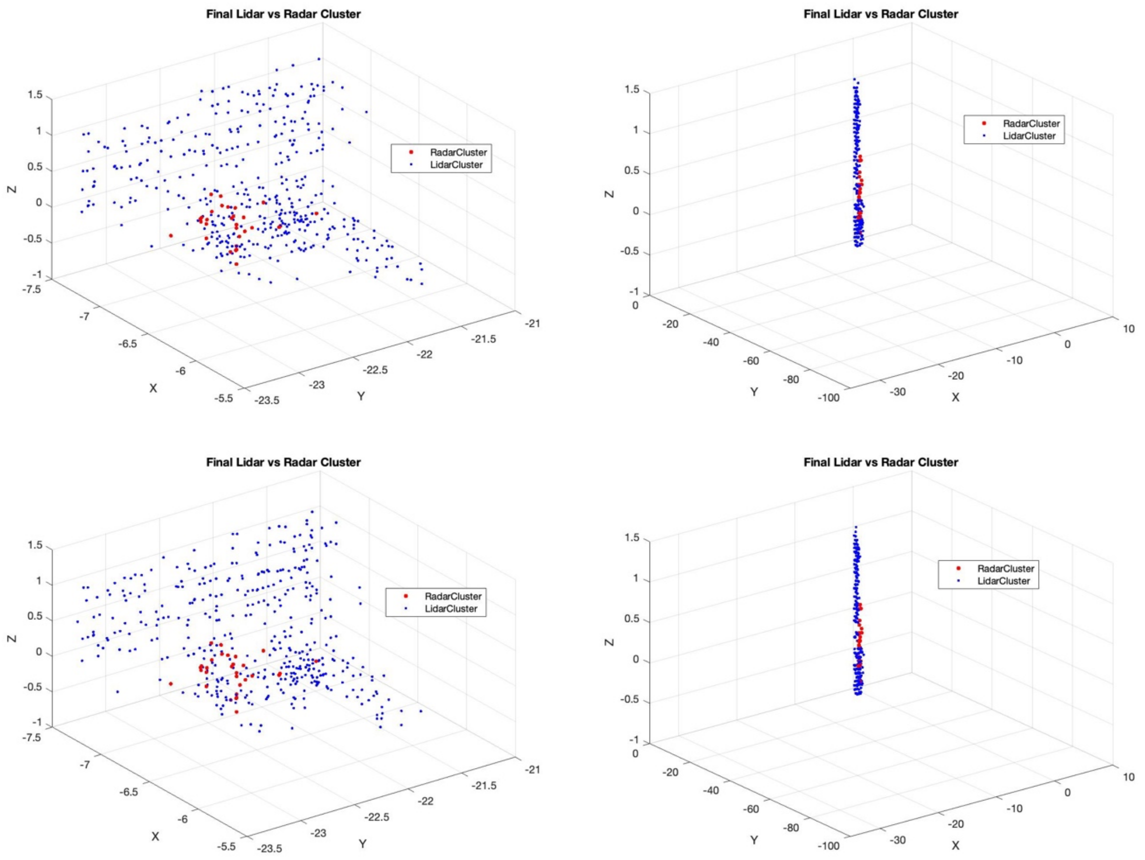
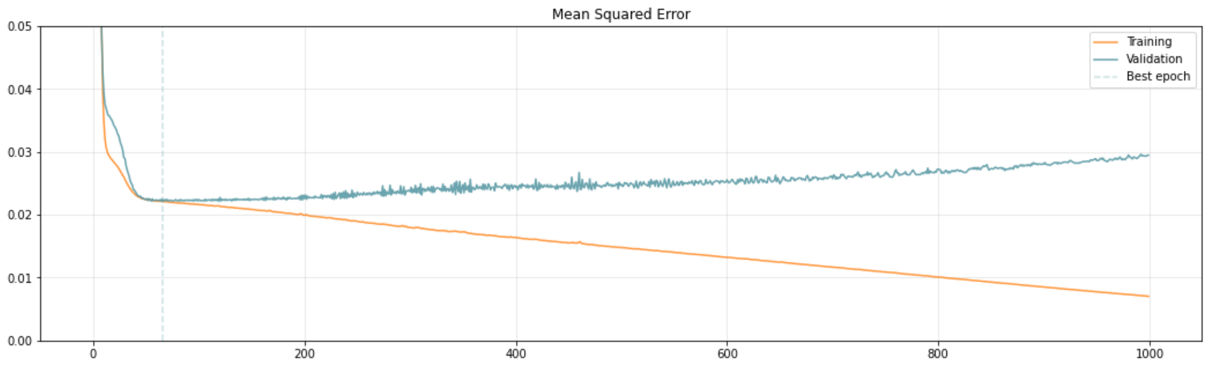


Figure 3.27 (a-b-c-d-e-f-g-h-i-j): Different normalized clusters from a single lidar cloud

3.2.9 Training

The training presented in this paragraph concerns the same multilayer perceptron model presented beforehand, but in two different contexts: in one, the model was trained using the complete dataset, thus 1263 points; the other makes use of only the original 167 radar samples at which only one coherent lidar cluster gets associated.
The training curves are displayed below (Figure 3.28):



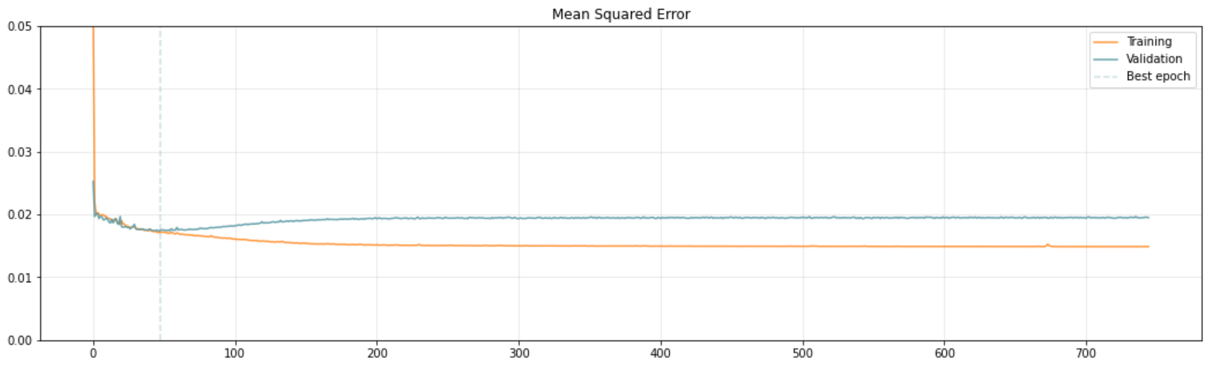
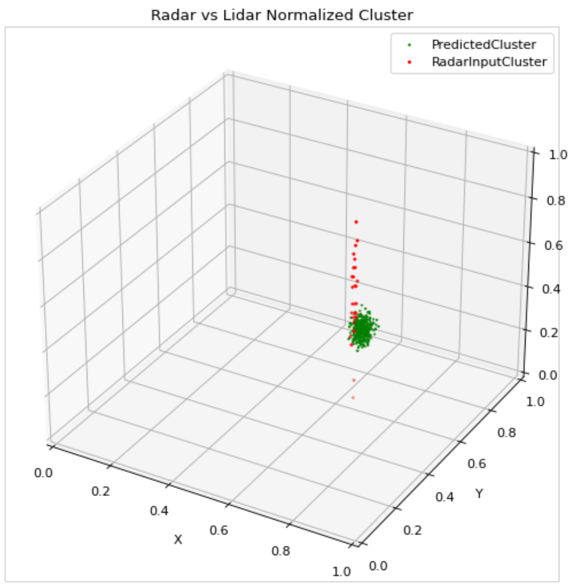
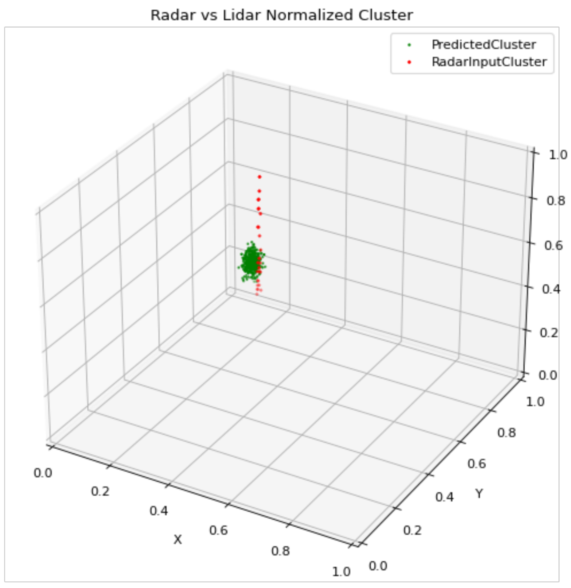
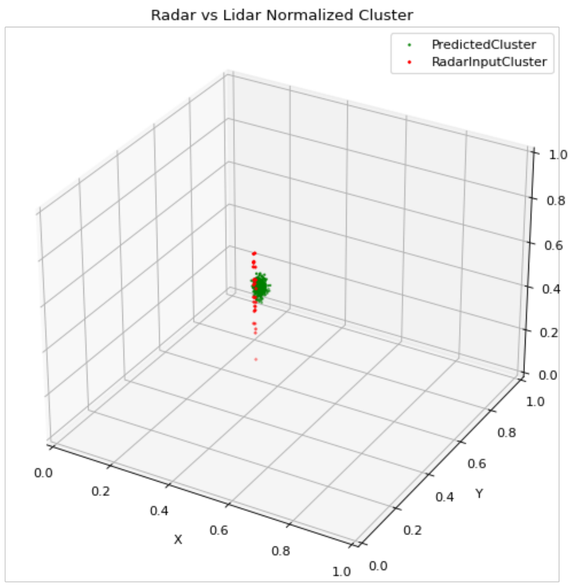


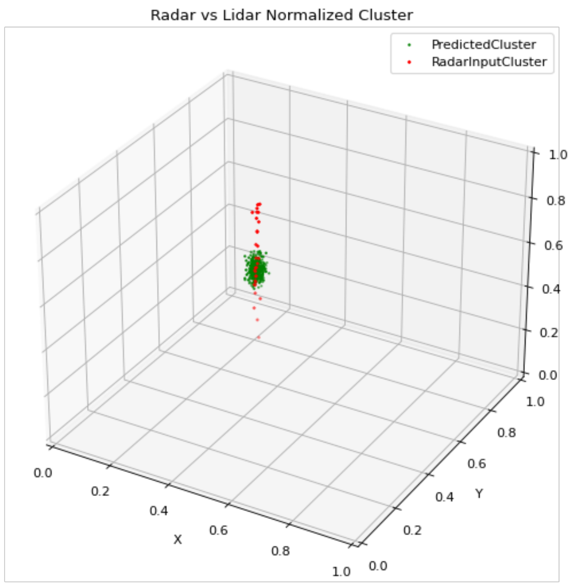
Figure 3.28 (a-b): Training curves of small and enhanced dataset

It is possible to notice that the optimal validation error of the dataset trained with the enhanced dataset is lower than the other. Aside from the performances highlighted by the evaluating metrics of the Neural Network, it is better to visually check the actual clouds, to see if this optimum is effectively coherent with our problem. Therefore, to assess the quality of the predicted clusters it is necessary a test set, different from the training set. The radar samples of this set are inputted inside the trained model to output their predicted clusters. Below, input and output data are plotted together (Figures 3.29):


Radar vs Lidar Normalized Cluster
PredictedCluster
RadarInputCluster
X
Y
Z


Radar vs Lidar Normalized Cluster
PredictedCluster
RadarInputCluster
X
Y
Z


Radar vs Lidar Normalized Cluster
PredictedCluster
RadarInputCluster
X
Y
Z


Radar vs Lidar Normalized Cluster
PredictedCluster
RadarInputCluster
X
Y
Z

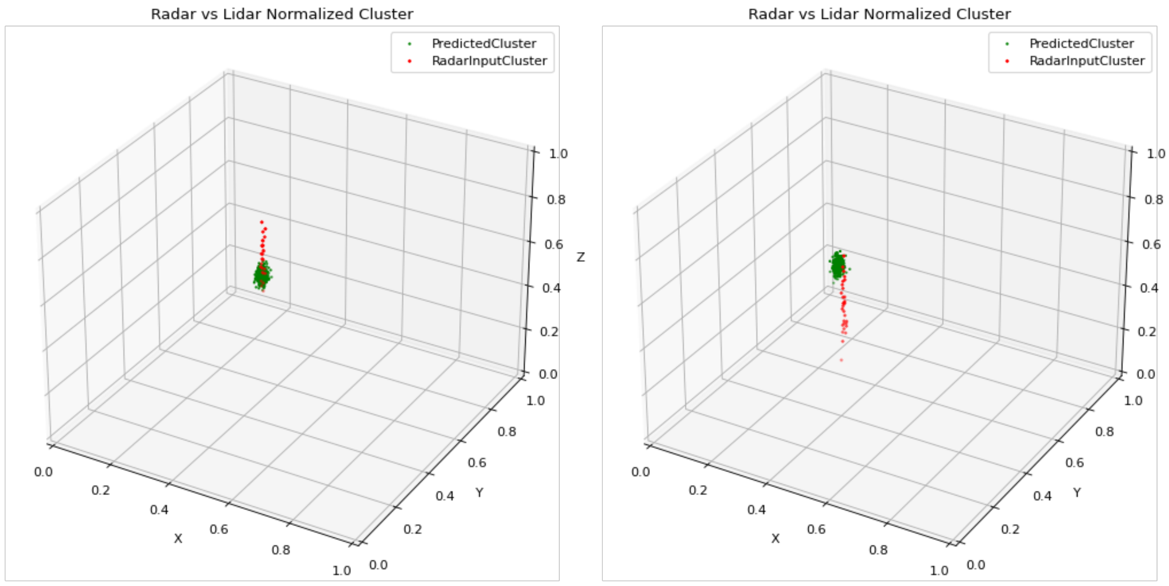


Figure 3.29 (a-b-c-d-e-f): Enhanced dataset network results

As one can notice, even if the validation error is lower for the first model, the actual cluster transformation for both models provide similar results. Unfortunately, the predicted clusters are completely concentrated around the centroid and does not make a good job in shaping the original cluster.

Something noticeable when looking at the training curve is the different shape of the two cases. The one that considers the smaller but unique set is a more natural shape with respect to the other. As explained in the Fundamental section, the typical training curve features a common descending trend at the very beginning up until the optimal solution, then the two curves separate as the training loss keeps decreasing and the validation loss start increasing, victim of the overfitting issue. This is fully coherent with the small set training, but it is not entirely true with the augmented dataset. With this latest, well beyond the optimum point, the curve flattens, and the network lies steadily, independently on how long the training keeps going. This behavior shows that this perceptron will never perform any different from the optimal performances, as the curve flattens immediately after.
Considering now the model trained with the unique dataset, its curve clearly shows that, after the optimal solution, it keeps learning from the training set, losing generalization capabilities. However, having in mind the last results obtained with the optimal networks, it is possible that the optimization task solved by the Adam optimizer does not represent the problem with effectiveness: in other terms, finding the minimum loss in the validation set does not finally provide a model capable of performing well in cluster transformation. It is rather possible that the algorithm is

learning to solve a problem somewhat slightly different, providing thus solutions that are numerically optimized but that are not visually satisfactory. Therefore, considering the strong visual component of the data that the algorithm tries to model, it is worth to try to forget about the evaluation of the machine learning training module, letting in such way the model be overtrained and directly check how the final clusters look like. In order to properly test different levels of training, the same identical model has been trained at: minimum validation error (Figures 3.30), 500 (Figures 3.31), 700 (Figures 3.32), 900 (Figures 3.33) and 1000 epochs (Figures 3.34).
Here are reported some clusters for comparisons purposes from each of these five nets:

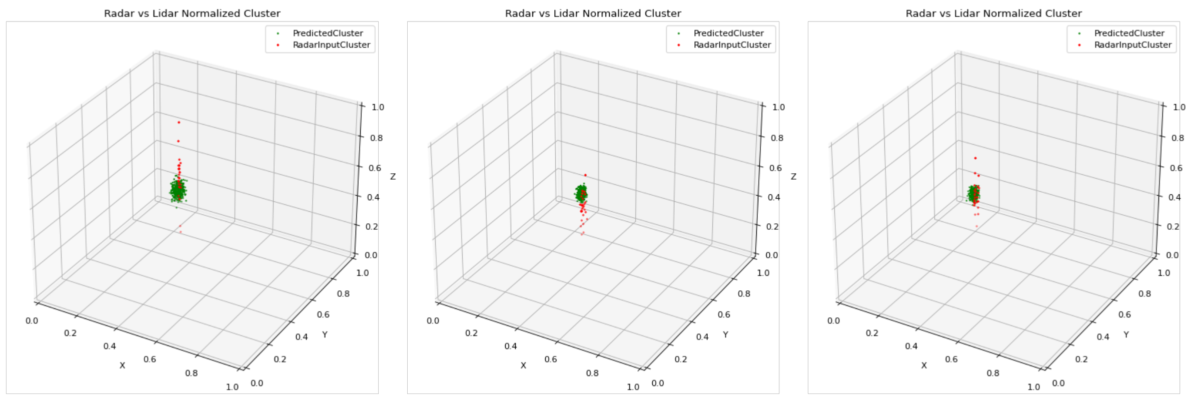


Figure 3.30 (a-b-c): Minimum validation error training results

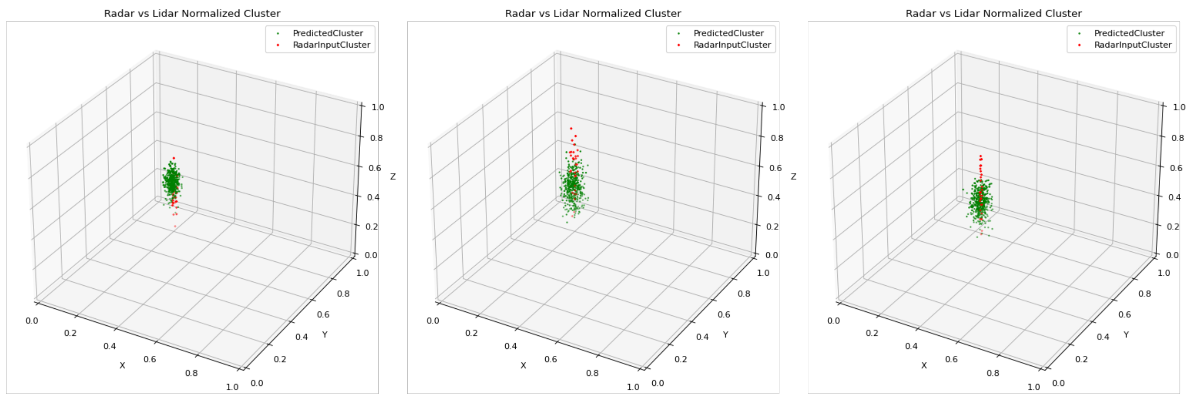


Figure 3.31 (a-b-c): 500 epochs training results

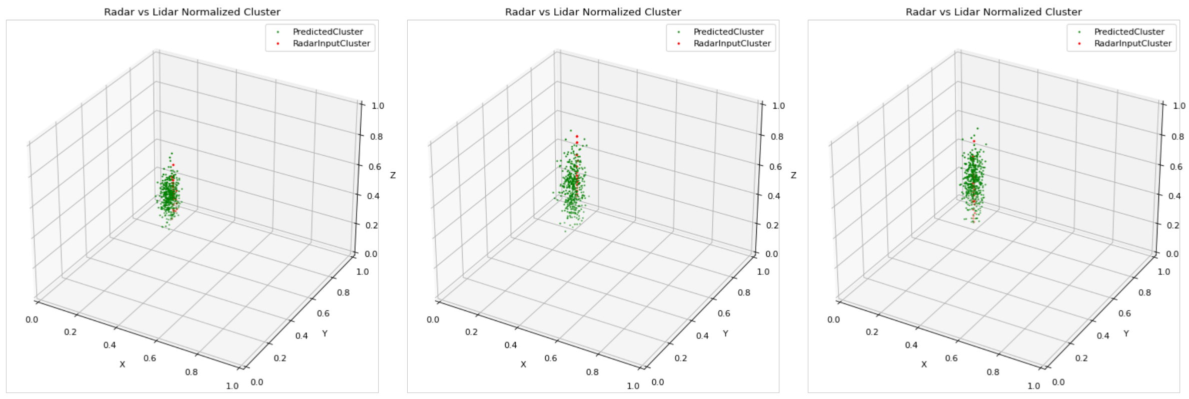


Figure 3.32 (a-b-c): 700 epochs training results

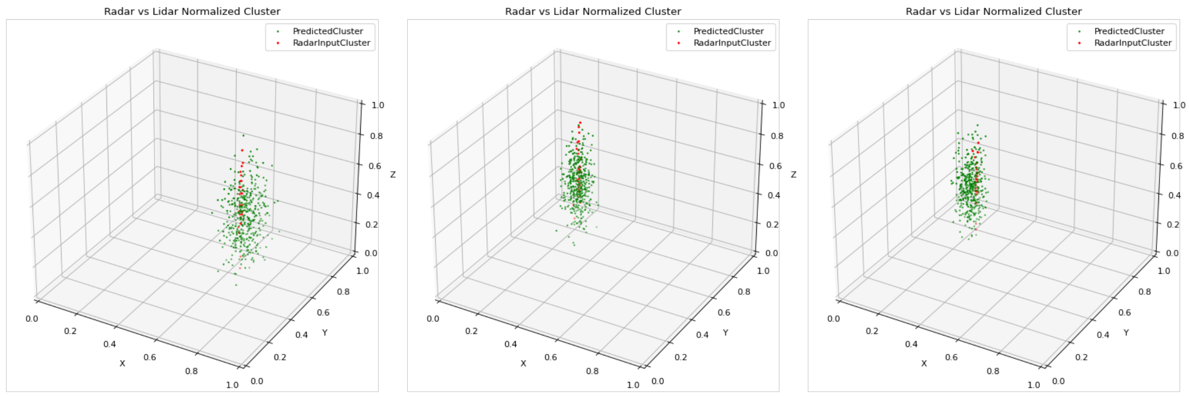


Figure 3.33 (a-b-c): 900 epochs training results

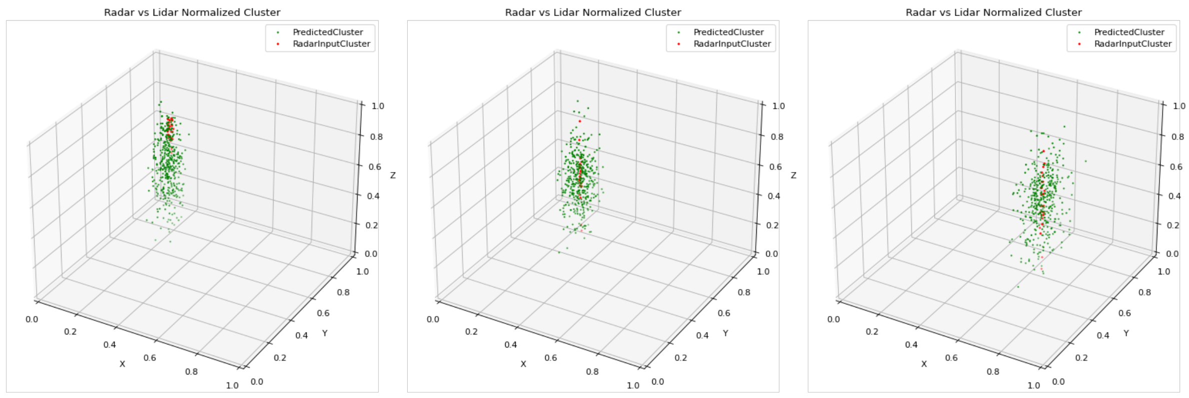


Figure 3.34 (a-b-c): 1000 epochs training results

Analyzing the results, one can easily understand how the model operation is changing along the training. From the clusters generated by the optimal model, occurring at the ninetieth epoch, the overtraining helps the predicted cluster in gaining more and more variance, especially along the z direction. Visually the overfitting seems to be capable of improving the quality of the predictions up to a certain training level. After that,

they become too sparse as effectively the problem is overfitting. The best solution found here is the model trained at 700 epochs, as it represents the right tradeoff of what has been explained up to now.
It is important also to notice that the model is very accurate in collocating the points in the right portion of space without moving them elsewhere. This implicit learning was made possible only by the normalization algorithm, generating those large patches previously discussed.

3.2.10 Map Reconstruction

Once the cluster is transformed, the map needs to be assembled back into its full shape. In order to do so, we need all the final clusters produced and the array containing the position coordinates and the derivatives of the clusters centroid that we have created and saved beforehand, during the cluster normalization. The reconstruction works exactly like the reverse of the normalization as there is no need for any adjustment to be made in terms of cluster position. The final reconstructed map is shown in the following pictures (Figures 3.35):

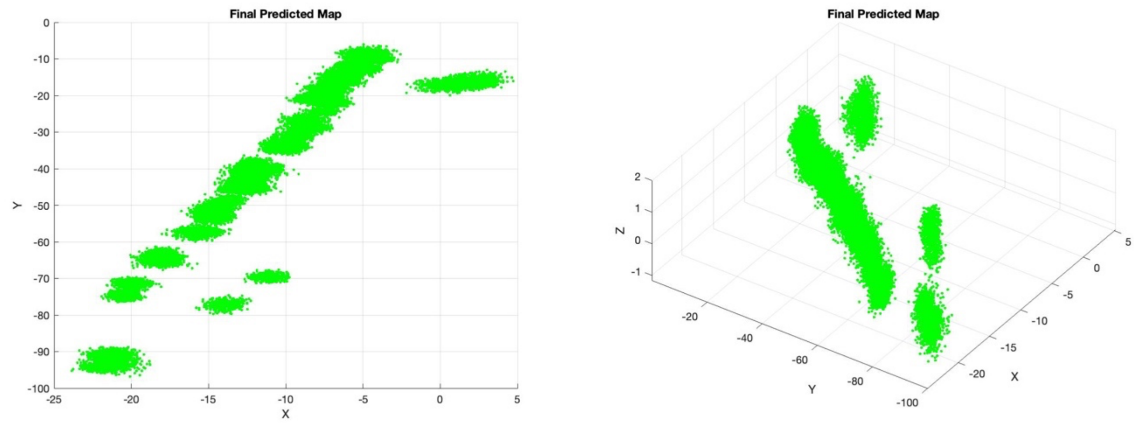


Figure 3.35 (a-b): Final predicted map

4. Results

In this chapter, the analysis of the developed algorithm described in the previous chapter will be carried out. This fourth chapter is divided into two sections, one introducing the metrics and the other one providing the numerical results and the comments associated.

4.1 Introduction to the metrics

In order to properly evaluate the final results, it is necessary to introduce an evaluation metric. This score aims at evaluating the similarity between two clouds, taking the first one as a reference cloud and making the evaluation on the second one. In this case, the reference cloud will be in all cases the lidar map. As already said, in this work the lidar is considered the ground truth because of its performances in terms of accuracy and density. Therefore, it is the most suitable candidate as reference cloud for the metrics.

The evaluation metric (Lethola et al., 2017) takes as input the point cloud to be evaluated, the reference point cloud and a parameter r_c. The working principle is to further consider one point p_i of the cloud to be evaluated at each time and find its closest point $p_{nn(i)}$ belonging to the reference cloud. Once the two are found, the algorithm computes the distance and compare it with the r_c value. If this distance is higher than the parameter, the point is completely discarded (thus ω_i takes value 0). In the other scenario, the point i is considered inside the metric, thus being part of the summation of the L2 defined norm and belonging to the N points value:

$$C = \frac{1}{2} w(\|y(x) - a\|)^2 \tag{4.1}$$

$$E(L_p) = \left(\frac{1}{N} \sum_{i=1}^{N} \omega_i |p_i - p_{nn(i)}|^p \right)^{1/p} \tag{4.2}$$

$$p = 2 \quad (4.3)$$

$$\omega_i = \begin{cases} 0, & if\ d_i > r_c \\ 1, & otherwise \end{cases} \quad (4.4)$$

The algorithm proceeds until it has evaluated all the points of the cloud.
This procedure is carried out for different values of r_c, such that it is possible to see the evolution of the metric depending on this threshold value and draw conclusion based on it.

The other metric considered is the evaluation of the coverage, or how the evaluated map is able to occupy the same space of the reference map.
To compute this metric, the algorithm proposed takes into consideration the clusters of the predicted map, and for each of those it creates a box perfectly enclosing the cluster and counting how many lidar map points are within the box. At each iteration, the lidar points already considered are removed from the map such that it is not possible to consider them twice. Finally, all the points counted are summed together and this value is divided by the total amount of initial lidar points

The third metrics is instead related with another issue that is considerably improved, which is map sparsity. A comparison between the number of points composing the maps along the different steps of the algorithm is provided to shown how the final predicted map behave with respect also to the reference ground truth map.

4.2 Metric results

Here the actual results of the evaluation introduced in the previous section are shown. In the figure below are displayed the plots of the evaluation metric (Figure 4.1):

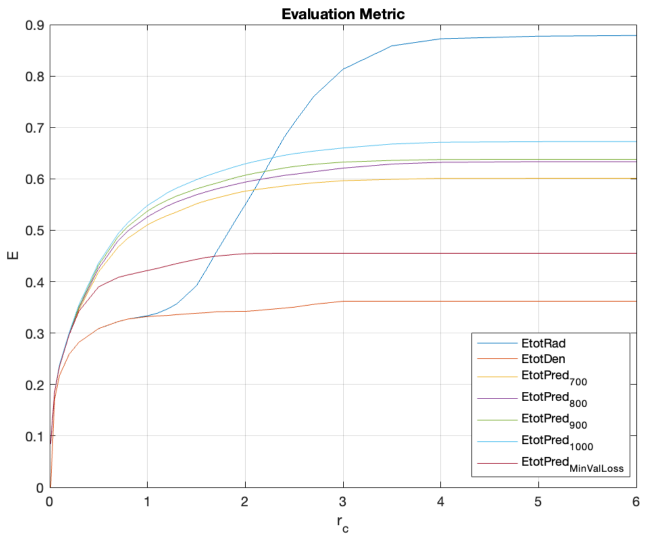


Figure 4.1: Evaluation metric result

As one can see from the picture, the curve associated with the original radar map (in blue) is the one having the highest value. Not only that, but it is also the only curve characterized by two strong positive slopes: this has to be related with the high presence of outliers in the cloud, as they are taken more and more into consideration through the increasing r_c value.
Here one can appreciate the quality of the job made by the denoising phase, as the metric shows how the plot of the denoised map curve is the one having the lowest metric value, highlighting thus the elimination of the greatest part of the outliers of the original map (in orange).
Moving now to the predicted map curves of the different model, one can notice how the evaluation metric favors the model trained at minimum validation loss instead of any other net. The reasons for this are made clear by observing the two maps directly compared below (Figure 4.2):

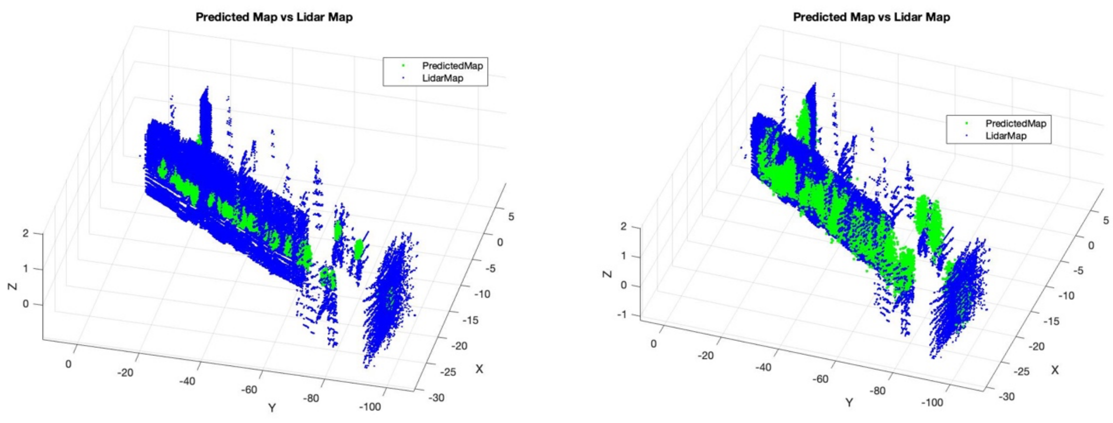


Figure 4.2 (a-b): Comparison between minimum validation loss and 700 epochs training model

The minimum validation model (on the left) generates clusters very dense and concentrated around the centroid of its respective input. This is numerically advantageous for what concerns this metric, especially considering the map aside that have a much higher variance, because of the actual proximity to the points of the lidar map in the same region. Visually though, it is clear how the performances of the first model are not at all acceptable and the proper metric to prove this incapability is the map coverage.

Indeed, in the table below (Table 4.1). are provided the results of the coverage metric applied to the network trained at minimum validation loss and to the one trained at 700 epochs:

MinValLoss Network Coverage	700_Epochs Network Coverage
13,7 %	68,2%

Table 4.1: Coverage metric results

As predicted, the model trained at optimum performed very poorly, by only covering the 13% of the lidar map, while the 700 epochs training model took a score of 68%, highlighting thus what was previously said. The 700 epochs training model is the one that has the best tradeoff between the evaluation metric, the map coverage metric, and the visual similarity validation.

Another fundamental thing to be evaluated is how the algorithm performs against the initial sparsity. From this perspective, the final amount of point belonging to the predicted map is high. In average, the denoised map loses around 30% of the initial points, and in turns the predicted map increases directly the denoised map by a factor slightly higher than 7. Compared to the original on the other hand, the factor is around 5. Conversely compared to the Lidar Map, there exist a factor difference of less than 5 (Table 4.2).

Lidar Map	Radar Original Map	Radar Denoised Map	Predicted Map
135666	5409	3764	28800

Table 4.2: Sparsity Metric

Lastly, below are depicted the final 3D maps, comparing the original Radar, the Predicted one and the Lidar Ground Truth map. This comparison is shown here only to visually appreciate the quality of the algorithm (Figures 4.3):

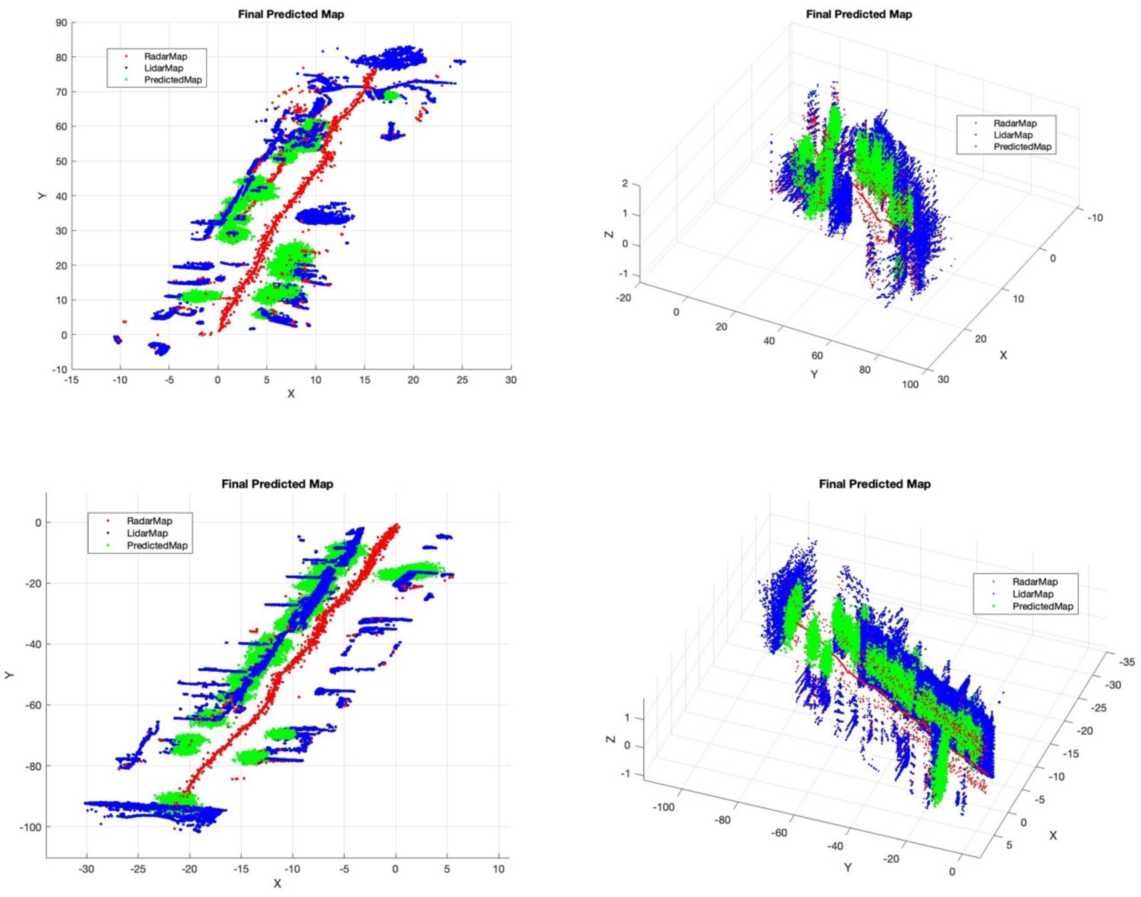


Figure 4.3 (a-b-c-d): Comparison of final map and original radar and lidar maps

5. Commercial radar analysis

Here is proposed a brief discussion about the radar that was used to carry out this project, possible alternatives and, in consideration of the behavior of the algorithm, what could be the consequences generated by making a different choice.

5.1 Commercial Radars

As previously stated, the original performances of the radar are poor either in terms of accuracy, noise production and sparsity. In particular, by making reference to Figure 3.9, the objects that the AWR1843BOOST sensor tries to model are not even recognizable by humans. Only by looking at the equivalent lidar map underneath (Figure 3.10), one can effectively tell which points model real structures of the environment and which are associated with complete noise. This level of scarcity of the information is not that common among commercial radars, but can be considered standard for cheaper off-the-shelf model like this one.
For this reason, it is worth to carry out a brief analysis of other employable radar sensors, accounting also for their price as one of the goals of this work is to improve sensor's quality without bringing heavy financial commitments.

5.1.1 ARS408

Figure 5.1: Millimeter wave sensor device ARS408-21

The ARS408 (Figure 5.1) is a long-distance radar working at 77GHz frequency. The sensor range of operation begins at 0.2 meters up to 250 meters The downloadable software distributed with the hardware is built for automotive industry: the main application is for detections of different objects like trucks, pedestrians etc. The sensor azimuth field of view is ±60 degrees and ±20 degrees in vertical for close to medium range application. The official price for this sensor is 844 USD.

5.1.2 The Sensor Fusion kit

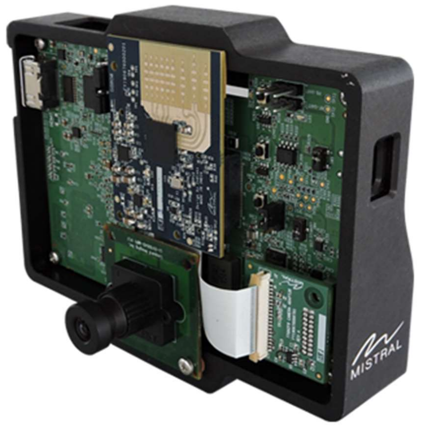


Figure 5.2: Millimeter wave radar integration Sensor Fusion Kit

The Sensor Fusion Kit (Figure 5.2) is a combination of camera and millimeter wave radar integrated which uses Texas Instruments single chip 77 GHz millimeter wave sensors like the AWR1843BOOST exploited in this work. Its frequency bandwidth ranges from 76 to 81 GHz. Released software for the kit offers classification and identification of sampled objects and streams radars data as video clip. Also, different customization services are provided in order to properly set the parameters of millimeter wave radar and camera's video. The official price for the Sensor Fusion Kit is 1499 USD.

5.1.3 IWR6843ISK with platform MMWAVEICBOOST

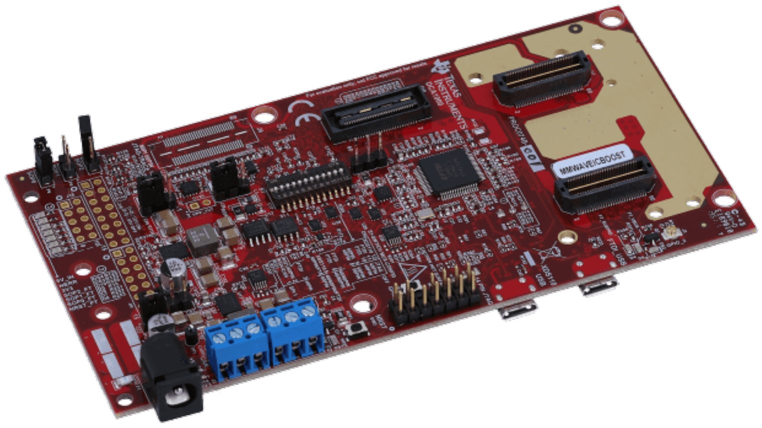


Figure 5.3: IWR6843ISK millimeter wave sensor on its carrier platform MMWAVEICBOOST

IWR6843ISK (Figure 5.3) is a millimeter wave sensor with a long-range antenna enabling direct connectivity to its carrier MMWAVEICBOOST. The sensor enables pointcloud data through a USB and raw ADC data through a 60-pin connector. The radar has 4 RX and 3 TX antenna with 108 degrees azimuth field of view angle and 44 degrees on elevation. The operating frequency ranges from 60 to 64 GHz millimeter wave sensor. The max range for vehicle detection is 114.43 meters. The price for the sensor module is 125 USD while the carrier platform is sold for 199 USD for a total of 324 USD. There exists a unified software platform mmWave SDK for TI radars which supports evaluation and development of the device.

5.2 Possible algorithm behavior

Those three radars have been chosen because they are considered to have better performances compared to the AWR1843BOOST. Even though it was not possible to carry out any experiment with such radars, one can imagine that if the quality of the initial information rises, then the quality of all transformations and consequently of the output, increases as well. The denoising phase becomes useless as all the central noise does not appear in the first place, while all the rest is kept as before.
During the clustering phase, if the information is more packed, the outcome of clustering, densification and reconstruction will surely better approximate the overall map, because the density of the clusters would be higher and more accurate. It is indeed expected that the overall number of clusters that would move to the subsequent normalization phase is much higher. This effect, in turn, makes the upsampler work on a higher number of provided clusters and thus the final map has an even larger number of points.

6. Conclusions

In this chapter are presented the conclusions of this Master Thesis, in terms of what has been achieved through the algorithm developed, what are its limitations and finally the possible future developments.

First of all, in comparison with the original radar, it is clear that, as shown in the final plots of the Results chapter, that the algorithm dramatically improves the mapping capabilities. In particular, the performances improve in terms of reliability, as the algorithm retains only very packed clusters, related, with very high probability, to real object observed. This guarantees the consistency of the data transformed with respect to the world structure, avoiding noise transformation.

Furthermore, another relevant aspect greatly improved is the noise reduction: in the Results chapter, the metric shows how the evaluation greatly improves thanks to the denoising phase of the algorithm. Even though in the latest cluster transformation some variance is added into the clouds, the typical noise associated with the radar signal has been correctly removed.

Finally, another key point that received major improvement is related with the cloud sparsity issue. As already stated, the lidar cloud on which this algorithm tries to learn from has more than 20 times the level of points of the original radar sampling. Moreover, the denoising phase removes almost the 40% of the original radar points. The final up-sampling phase increases the number of points composing a map up to having more than 5 times the radar original points, thus only 4 times less than the original lidar map on average.

Other than improving the capabilities of its mapping, it is very important to remember that the radar keeps unaltered its intrinsic properties. As mentioned in the Fundamentals section, those are being invariant with respect to weather and well performing in dynamically changing environment without losing relevant information. Both those characteristics are of fundamental importance for outdoor autonomous navigation.

6.1 Limitations and Future development

In this paragraph are presented the limitations that this work faces and most importantly what are the future research areas where to focus the attention for further development.

Firstly, about the training, the choices made were based on the direct evaluation of the results, rather than on the learning loss function. This is clearly problematic, as the aim of the optimization algorithm is to minimize the loss function. In this case, the loss function applied is not fully representative of the problem, and the converging solutions are not satisfying from that perspective. In this context, the final algorithm takes into consideration the model providing the best visual results, thus with a sub-optimal goal-oriented approach. The perceptron model is used as a way of achieving what is considered to be a sufficiently good result. The limitation here stands obviously in the lack of comprehension of this problem. The need is to define a different and more precise loss function, capable of finding a solution which is consistent with the problem to be solved.

Connected to this last point, the second relevant issue to be risen here is the too small training dataset. The one employed for the training contains only 167 samples, not enough for a proper training. Generally, this level strictly depends on the type of problem one aims at solving. In this case even more importantly, the complexity of the task is huge, the combinations of the various cluster shapes are many, even in terms of implicitly understandable semantic. Also, one should consider that different point combinations might be modelling the same object, and vice versa same point sets shaping different objects. In this context, the higher the number of collected samples, the more generalization capability can be provided to the model with effectiveness.

Always referring to the multilayer perceptron, which has been the most critical part of the work, one should consider the possibility to move towards different models. As already stated, this work makes use of a Neural Network in an exclusively goal-oriented manner. It was thus without the scope of this work try and assess other more complex models. Nevertheless, the multilayer perceptron, as well as many other models, has the constraints on the dimensionality of input and output information, which has been driver of many relevant choices in the algorithm. There are not many models capable of providing such flexibility, and the few existing require a certain level of expertise because of their complexity. One example of those Machine Learning models is the class of *seq2seq* algorithms (Sustkever et al., 2014): those kind of algorithms are mainly employed to perform languages translations, as they are capable of taking one input at time, in that case each word would be an input, and transform this data into a new one considering the past transformations.

By using this kind of model, many phases of the algorithm in this case could be simplified and even more importantly, the cluster passed to the net would have a certain dimensionality depending on the characteristic of the sampled object, giving thus further information to improve the densification. This strategy could be matched with those algorithms present in the literature performing object recognition based on its radar signature.

Some concerns, on the sampling procedure: the dimensions of the map sampled is always relatively limited in this work, therefore this has no major implication and negligible effects. It has to be noticed that, if this algorithm is ever used for creating larger maps, it definitely should be considered a modification in order to deal with this unrestrained memory resource allocation issue. Python and MATLAB are very strong and high-level programming tool, but especially for this reason, the user always has to keep in mind the dimensions of the data collected.

Another limitation to be mentioned in this paragraph refers to the idea that this work is a union of separated pieces of code, ran on different platforms, with few manual operations inside, mainly linked with data movement cross-platform. With the aim of providing something useful for the autonomous navigation, all the algorithms should be fully automated. Certain phases of the algorithm are already implemented with a good level of automation and could be just integrated, for instance the clustering algorithm, the normalization or the up-sampling phase. On the other hand, the denoising phase and the dataset generation are mainly user defined: the denoising consists in a segmentation algorithm, but its parameters depend on the map, and each map has different parameters because of its spatial orientation, the dimensions of the environment etc. In the same fashion, the dataset generation is almost fully manual, as the lidar cluster is cutted out of the whole lidar map around the associated radar cluster. A possible solution for the automation of the denoising phase is the ROS node called *crop_box*. It is stored into the extensive library called *pcl* born to provide support for 3D pointcloud operations (Rusu et al., 2011). This takes the *PointCloud2* message as input and it outputs the same cloud out of which the points laying inside a user defined box are removed. Correctly tuning the dimensions and the position of this box, the noise could be removed before the generation of the map.

Finally, we highlight an issue on which further research should be considered, which is map coverage. As already said, the algorithm is reliable, as the clusters that are found and transformed are highly packed and thus representing with high probability parts of real obstacles. It is clear, on the other hand, that to obtain this level of safety, the clustering algorithm is unable to model those regions where some points have been collected properly, but not enough to generate a cluster. The cluster thus considers all these points as outliers incorrectly. Future research should try to instruct the cluster algorithm to look for more than only the point density, such that the shaping can improve and thus start being reliable even in the reversed direction i.e., not only providing certainty in what is brought to be transformed but also, certainty in the reconstruction of the complete environment.

Bibliography

Bouzouraa, M. E., & Hofmann, U. (2010). Fusion of occupancy grid mapping and model based object tracking for driver assistance systems using laser and radar sensors. *2010 IEEE Intelligent Vehicles Symposium*. https://doi.org/10.1109/ivs.2010.5548106

Coughlan, J., Yuille, A.. (1970). The Manhattan World Assumption: Regularities in scene statistics which enable Bayesian inference.. *NIPS*.

Diehl, C., Feicho, E., Schwambach, A., Dammeier, T., Mares, E., & Bertram, T. (2020). Radar-based dynamic occupancy grid mapping and Object Detection. *2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC)*. https://doi.org/10.1109/itsc45102.2020.9294626

Dogru, S., & Marques, L. (2019). Grid based indoor mapping using radar. *2019 Third IEEE International Conference on Robotic Computing (IRC)*. https://doi.org/10.1109/irc.2019.00095

Dube, R., Hahn, M., Schutz, M., Dickmann, J., & Gingras, D. (2014). Detection of parked vehicles from a radar based occupancy grid. *2014 IEEE Intelligent Vehicles Symposium Proceedings*. https://doi.org/10.1109/ivs.2014.6856568

Elfes, A. (1989). Using occupancy grids for mobile robot perception and Navigation. *Computer*, *22*(6), 46–57. https://doi.org/10.1109/2.30720

Foote, T. (2013). TF: The transform library. *2013 IEEE Conference on Technologies for Practical Robot Applications (TePRA)*. https://doi.org/10.1109/tepra.2013.6556373

Glorot, X., Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. *Journal of Machine Learning Research - Proceedings Track. 9. 249-256.*

Gottinger, M., Hoffmann, M., Christmann, M., Schutz, M., Kirsch, F., Gulden, P., & Vossiek, M. (2021). Coherent Automotive Radar Networks: The next generation of radar-based imaging and mapping. *IEEE Journal of Microwaves*, *1*(1), 149–163. https://doi.org/10.1109/jmw.2020.3034475

Guan, J., Madani, S., Jog, S., Gupta, S., & Hassanieh, H. (2020). Through fog high-resolution imaging using Millimeter Wave Radar. *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. https://doi.org/10.1109/cvpr42600.2020.01148

Hornung, A., Wurm, K. M., Bennewitz, M., Stachniss, C., & Burgard, W. (2013). OctoMap: An efficient probabilistic 3D mapping Framework based on octrees. *Autonomous Robots*, *34*(3), 189–206. https://doi.org/10.1007/s10514-012-9321-0

Lee, S., Kwon, S.-Y., Kim, B.-J., Lim, H.-S., & Lee, J.-E. (2021). Dual-mode radar sensor for indoor environment mapping. *Sensors*, *21*(7), 2469. https://doi.org/10.3390/s21072469

Lehtola, V., Kaartinen, H., Nüchter, A., Kaijaluoto, R., Kukko, A., Litkey, P., Honkavaara, E., Rosnell, T., Vaaja, M., Virtanen, J.-P., Kurkela, M., El Issaoui, A., Zhu, L., Jaakkola, A., & Hyyppä, J. (2017). Comparison of the selected state-of-the-art 3D indoor scanning and Point Cloud Generation Methods. *Remote Sensing*, *9*(8), 796. https://doi.org/10.3390/rs9080796

Li, Larry. “Time-of-Flight Camera - An Introduction.” (2014).

Lin, C.-H., Kong, C., & Lucey, S. (2018). Learning efficient point cloud generation for dense 3D object reconstruction. *Proceedings of the AAAI Conference on Artificial Intelligence*, *32*(1). https://doi.org/10.1609/aaai.v32i1.12278

Lu, C. X., Rosa, S., Zhao, P., Wang, B., Chen, C., Stankovic, J. A., Trigoni, N., & Markham, A. (2020). See through smoke. *Proceedings of the 18th International Conference on Mobile Systems, Applications, and Services*. https://doi.org/10.1145/3386901.3388945

Lu, C. X., Saputra, M. R., Zhao, P., Almalioglu, Y., de Gusmao, P. P., Chen, C., Sun, K., Trigoni, N., & Markham, A. (2020). Milliego. *Proceedings of the 18th Conference on Embedded Networked Sensor Systems*. https://doi.org/10.1145/3384419.3430776

Mahesh, Batta. (2019). Machine Learning Algorithms -A Review. 10.21275/ART20203995.

Marck, J. W., Mohamoud, A., Houwen, E., van Heijster, R. (2020) Indoor radar SLAM A radar application for vision and GPS denied environments. *European Radar Conference, 2013, pp. 471-474.*

Milz, S., Simon, M., Fischer, K., Pöpperl, M., & Gross, H.-M. (2019). Points2Pix: 3D point-cloud to image translation using conditional gans. *Lecture Notes in Computer Science*, 387–400. https://doi.org/10.1007/978-3-030-33676-9_27

Mirza, M., Osindero, S. (2014). Conditional Generative Adversarial Nets.

Nielsen, M. L. (2015). Neural Networks and Deep Learning, *Determination Press.* http://neuralnetworksanddeeplearning.com/index.html

Nobis, F., Geisslinger, M., Weber, M., Betz, J., & Lienkamp, M. (2019). A deep learning-based radar and camera sensor fusion architecture for object detection. *2019 Sensor Data Fusion: Trends, Solutions, Applications (SDF).* https://doi.org/10.1109/sdf.2019.8916629

Park, Y. S., Kim, J., & Kim, A. (2019). Radar localization and mapping for indoor disaster environments via multi-modal registration to prior lidar map. *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS).* https://doi.org/10.1109/iros40897.2019.8967633

Quigley, M., Gerkey, B., Conley, K., Faust, J., Foote, T., Leibs, J., Berger, E., Wheeler, R., Ng, A.. (2009). ROS: An open-source Robot Operating System. *ICRA Workshop on Open Source Software. 3. 1-6.*

Rablau, C. I. (2019). LIDAR: A new self-driving vehicle for introducing optics to broader engineering and non-engineering audiences. *Fifteenth Conference on Education and Training in Optics and Photonics: ETOP 2019*. https://doi.org/10.1117/12.2523863

Ros-drivers/Velodyne: ROS support for Velodyne 3D lidars. GitHub. https://github.com/ros-drivers/velodyne

Rosenblatt, F. (1958). The Perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, *65*(6), 386–408. https://doi.org/10.1037/h0042519

Rousseeuw, P. J. (1987). Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. *Journal of Computational and Applied Mathematics*, *20*, 53–65. https://doi.org/10.1016/0377-0427(87)90125-7

Rusu, R. B., & Cousins, S. (2011). 3D is here: Point cloud library (PCL). *2011 IEEE International Conference on Robotics and Automation*. https://doi.org/10.1109/icra.2011.5980567

RViz: http://ros.org/wiki/rviz

Sutskever, I., Vinyals, O. Le, Q. (2014). Sequence to Sequence Learning with Neural Networks. *Advances in Neural Information Processing Systems. 4.*

Weston, R., Cen, S., Newman, P., & Posner, I. (2019). Probably unknown: Deep Inverse Sensor Modelling radar. *2019 International Conference on Robotics and Automation (ICRA)*. https://doi.org/10.1109/icra.2019.8793263

Yifan, W., Wu, S., Huang, H., Cohen-Or, D., & Sorkine-Hornung, O. (2019). Patch-based progressive 3D point set Upsampling. *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR).* https://doi.org/10.1109/cvpr.2019.00611

Zhang, L. (2019). *Radar-lab/ti_mmwave_rospkg: Ti mmwave radar ROS driver (with sensor fusion and hybrid)*. GitHub. Retrieved June 26, 2022, from https://github.com/radar-lab/ti_mmwave_rospkg

Appendix

Below, one can find the most relevant parts of the source code used for this work. In this appendix, the Python code will be shown first, then the MATLAB code. Before each part, a brief comment on its behavior will be given.

Python codes

1) Firstly, below it is shown one of the ROS nodes launched in the phase of map transformation. Basically, Octomap is not capable of working with a Pointcloud message which has been registered and stored. In brief, the reason is related with the time stamp inside the header of each message that is not coherent with the current time stamp when the messages are played back.
Thus, this node subscribes to the topic providing the input cloud message (***/velodyne_points*** for the lidarar, ***/ti_mmwave/radar_scan_pcl*** topic for the radar) and it modifies it such that the header now contains the current time information. This transformation occurs in the callback function, which is called whenever a new message on the topic. The new cloud (which is completely identical to the input one, header stamp aside) is then published to a new topic called ***/rebroadcast_pcl***. Octomap will then take this cloud, subscribing to this topic, and transform it into a map.

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

import rospy
from sensor_msgs.msg import PointCloud2
rospy.init_node("rebroadcaster_pcl")

rebroad_pub = rospy.Publisher("/rebroadcast_pcl", PointCloud2,
queue_size=1)
rate = rospy.Rate(10)

def pcl_callback(input_pcl):
    input_pcl.header.stamp = rospy.Time.now()

    rebroad_pub.publish(input_pcl)
```

```
        # type depends on your data type, first three entries are
probably x,y,z

# velodyne_sub = rospy.Subscriber("/ti_mmwave/radar_scan_pcl",
PointCloud2, pcl_callback)
velodyne_sub = rospy.Subscriber("/velodyne_points", PointCloud2,
pcl_callback)

rospy.spin()
```

2) This code represents the full clustering phase. The input value here are ***Xr_3d*** which is a 3d numpy array. As specified in the Map Transformation paragraph, octomap always outputs the full map, and in this variable the first dimension represents the time step. Since the map grows at each step, to keep dimension coherency the 2d layers above the new one is filled with zeros that are removed with: `X = Xr_3d[i,:-count,:]`. The other two dimensions contains the coordinates of the points of the map. The other input necessary for the code to work is the trajectory matrix ***pose***, having in the first 2 column the X and Y position of the points composing the trajectory and the third column the values of the first derivative of the trajectory curve.

```
thresh = 0.3
n_samp = 30
def_clus = np.empty((0,3))
mat_clus = np.empty((0,n_samp,3))
pbar = tqdm(total=iterat)

for i in range(iterat):
    pbar.update(n=1)
    count = sum (Xr_3d[i,:,1]==0)
    X = Xr_3d[i,:-count,:]
    m = - 1 / np.tan(pose[2,i])
    X_beh = X[X[:,1] > m * (X[:,0]-pose[0,i]) + pose[1,i],:]
    # X = np.array(list(set(tuple(map(tuple, X)))^set(tuple(map(tuple, X_beh)))))
    X_beh = np.array(list(set(tuple(map(tuple, X_beh)))^set(tuple(map(tuple,
def_clus)))))

    if len(X_beh) <= 50:
        continue

    clusters = hcluster.fclusterdata(X_beh[:,:2], thresh, criterion="distance")
    num_clus = list(set(clusters))

    sample_silhouette_values = silhouette_samples(X_beh[:,:2], clusters)
    if len(num_clus) == 1:
        if len(X_beh) >= n_samp:
            ind = np.argpartition(sample_silhouette_values, -n_samp)[-n_samp:]
            X_clus = X_beh[ind,:]
            mat_clus = np.concatenate((mat_clus,X_clus.reshape((1,n_samp,-1))),
axis = 0)
            def_clus = np.concatenate((def_clus,X_clus), axis = 0)


    for num in num_clus:
        if sum (num == clusters) >= n_samp:
            clu_sample_silhouette_values = sample_silhouette_values[num ==
clusters]
            X_clu_samp = X_beh [num == clusters,:]
            ind = np.argpartition(clu_sample_silhouette_values, -n_samp)[-n_samp:]
            X_clus = X_clu_samp[ind,:]
            mat_clus = np.concatenate((mat_clus,X_clus.reshape((1,n_samp,-1))),
axis = 0)
            def_clus = np.concatenate((def_clus,X_beh), axis = 0)
```

3) This is the coded machine learning model. The Keras toolbox, now part of Tensorflow, provides all the necessary functions to create and customize a network and train it. ***input_shape*** and ***output_shape*** are problem defined and they take values 90 and 1200 respectively. This is explained in the Upsampling section.
The variable ***mat_clus_norm*** contains the test set to be predicted in order to evaluate the model capabilities.

```
def tesinn_xyz (input_shape,output_shape):

    input_layer = tf.keras.layers.Input(shape=input_shape, name='input')

    hidden_layer1 = tf.keras.layers.Dense(units=1200, activation='relu',
name='Hidden1',
kernel_initializer=tfk.initializers.GlorotUniform(seed=seed))(input_layer)

    hidden_layer2 = tf.keras.layers.Dense(units=1200, activation='relu',
name='Hidden2',
kernel_initializer=tfk.initializers.GlorotUniform(seed=seed))(hidden_layer1)

    hidden_layer3 = tf.keras.layers.Dense(units=1200, activation='relu',
name='Hidden3',
kernel_initializer=tfk.initializers.GlorotUniform(seed=seed))(hidden_layer2)

    output_layer = tf.keras.layers.Dense(units=output_shape, activation='linear',
name='Output')(hidden_layer3)

    model = tf.keras.Model(inputs=input_layer, outputs=output_layer)
    learning_rate = 5e-5
    opt = tfk.optimizers.Adam(learning_rate)
    loss = tfk.losses.MeanSquaredError()
    metrics=['mse', 'mae']
    model.compile(loss=loss, optimizer=opt, metrics=metrics)

    return model

tesi_nn_xyz = tesinn_xyz(input_shape, output_shape)
tesi_nn_xyz.summary()

tfk.utils.plot_model(tesi_nn_xyz, show_shapes=True, expand_nested=True)

history_yz = tesi_nn_xyz.fit(
    X_xyz,
    Y_xyz,
    batch_size=30,
    epochs=700,
    validation_split=0.1,
    callbacks=[
        tfk.callbacks.EarlyStopping(monitor='loss', patience=50,
restore_best_weights=True),
        tfk.callbacks.ReduceLROnPlateau(monitor='loss', patience=10, factor=0.5,
min_lr=1e-5),
    ]
).history

predictions = tesi_nn_xyz.predict(mat_clus_norm)
```

MATLAB codes:

1) Below, the code for the patch normalization described in the Normalization section. ***cluster_map_csv*** is the input variable containing all the clusters found of a certain map (map 5 in this case). For this example, the trajectory has been approximated to a straight line defined by starting point (x1_discr, y1_discr) and ending point (x2_discr, y2_discr). Within the for-loop, iteratively considering a different patch, the procedures of frame rotation and the scaling are carried out as described. Commented also the code related to the lidar patch normalization, which is anyway completely equivalent to the radar patch.

```
cluster_map_csv = readmatrix("csv/rad_cluster_5.csv");
```

```
x_dist = 27;
y_dist = 20;
z_dist = 2.7;
z_min = -0.7;
x_traj = linspace(x1_discr,x2_discr,1000);
y_traj = linspace(y1_discr,y2_discr,1000);

% cluster_mat_sc = zeros(1,400,3);
% rcluster_mat_sc = zeros(1,30,3);

for i = 1:length(cluster_map_csv(:,1))
    x = cluster_map_csv(i,1:30);
    y = cluster_map_csv(i,31:60);
    z = cluster_map_csv(i,61:end);
    x0=mean(x);
    y0=mean(y);

    [min_dist,pos] = min(sqrt((x_traj-x0).^2+(y_traj-y0).^2));
    xcent = x_traj(pos);
    ycent = y_traj(pos);

    der = (y_traj(pos)-y_traj(pos+1))/(x_traj(pos)-x_traj(pos+1));
    cluster_pos(i,:) = [xcent, ycent, der];

    alpha = pi/2-atan(der);

    x_sc = [cos(alpha) -sin(alpha)]*[x-xcent;y-ycent]+xcent;
    y_sc = [sin(alpha) cos(alpha)]*[x-xcent;y-ycent]+ycent;
%     x_velo_sc = [cos(alpha) -sin(alpha)]*[x_velo-xcent;y_velo-
ycent]+xcent;
%     y_velo_sc = [sin(alpha) cos(alpha)]*[x_velo-xcent;y_velo-
ycent]+ycent;

    % x_ntraj = [cos(alpha) -sin(alpha)]*[x_traj-xcent;y_traj-
ycent]+xcent;
    % y_ntraj = [sin(alpha) cos(alpha)]*[x_traj-xcent;y_traj-
ycent]+ycent;

    x_fin_sc = (x_sc-xcent)/x_dist + 0.5;
    y_fin_sc = (y_sc-ycent)/y_dist + 0.5;
    z_fin_sc = (z-z_min)/z_dist;
%     x_velo_fin_sc = (x_velo_sc-xcent)/x_dist + 0.5;
%     y_velo_fin_sc = (y_velo_sc-ycent)/y_dist + 0.5;
%     z_velo_fin_sc = (z_velo-z_min)/z_dist;
```

```
    cluster_map_norm(i,:,1) = x_fin_sc;
    cluster_map_norm(i,:,2) = y_fin_sc;
    cluster_map_norm(i,:,3) = z_fin_sc;
%     cluster_mat_sc(i,:,1) = x_velo_fin_sc;
%     cluster_mat_sc(i,:,2) = y_velo_fin_sc;
%     cluster_mat_sc(i,:,3) = z_velo_fin_sc;
end

cluster_map_norm_csv = [];
cluster_map_norm_csv(:,1:length(cluster_map_norm(1,:,1))) =
cluster_map_norm(:,:,1);
cluster_map_norm_csv(:,end+1:length(cluster_map_norm(1,:,1))*2) =
cluster_map_norm(:,:,2);
cluster_map_norm_csv(:,end+1:length(cluster_map_norm(1,:,1))*3) =
cluster_map_norm(:,:,3);
cluster_map_norm_csv(cluster_map_norm_csv(end,:,:)==0,:,:) =  [];
```

The code for the computation of the evaluation metric, as described in the Results chapter is shown below:

```
for t = 1:length(rc)
    E = 0;
    N = 0;
    pos = randperm(length(lidar_map),length(lidar_map)/2);
    lidar_map_mod = lidar_map(pos,:);
    for j = 1:length(pred_map)
        d0 = 1000;
%         posx = lidar_map(:,1) < pred_map(j,1)+4 & lidar_map(:,1) >
pred_map(j,1)-4;
%         posy = lidar_map(:,2) < pred_map(j,2)+4 & lidar_map(:,2) >
pred_map(j,2)-4;
%         posz = lidar_map(:,3) < pred_map(j,3)+4 & lidar_map(:,3) >
pred_map(j,3)-4;
%         pos = posx & posy & posz;
%         lid_filtered = lidar_map(pos,:);
        for i=1:length(lidar_map_mod(:,1))
            %minimum distance S(j) to R
            d1 = sqrt((pred_map(j,1)-lidar_map_mod(i,1))^2 +
(pred_map(j,2)-lidar_map_mod(i,2))^2 + (pred_map(j,3)-
lidar_map_mod(i,3))^2);
            if d1 < d0
                d0 = d1;
                k = i;
            end
        end
        lidar_map_mod(k,:) = [];
        if d0 < rc(t)
            E = E + d0;
            N = N + 1;
        end
    end
    Etot_pred_minValLoss_half(t) = sqrt(E/N);
end
```

2) Finally, described below the code related with the computation of the coverage metric also described in the Results section.

```
total = 0;
lidar_map_new = lidar_map;
for i = 1:length(pred_map_fin(:,1,1))
    pos = ((lidar_map_new(:,1) <= max(pred_map_fin(i,:,1))) &
(lidar_map_new(:,1) >= min(pred_map_fin(i,:,1))) &...
            (lidar_map_new(:,2) <= max(pred_map_fin(i,:,2))) &
(lidar_map_new(:,2) >= min(pred_map_fin(i,:,2))) &...
            (lidar_map_new(:,3) <= max(pred_map_fin(i,:,3))) &
(lidar_map_new(:,3) >= min(pred_map_fin(i,:,3))));
    lidar_map_new(pos,:) = [];
    total = total + sum(pos);
end

coverage = total / length(lidar_map) * 100;
```

List of Figures

List of Tables

Acknowledgements

In the first place, I would like to thank the ATB institute, in particular Professor Cornelia Weltzien, Dr. Dirk-Niklas Müller and Ph.D. candidate Tjark Schütte, for hosting, helping and giving me access to all the precious materials without which this work could have never seen the light of the day. I would like also to thank Professor Emanuele Zappa for his contribution, his time and his knowledge that I deem they have played a key role for this Master Thesis.

To my family and my wonderful parents: there are neither enough nor the proper words in any dictionary to express how much your love and support have been of fundamental relevance during these years. You have been, are and forever will be, my guidance and the models I take inspiration from. You gave me the wisdom, the cheering, the chances, everything that was necessary and so much more. I cannot add anything else except to thank you for everything you represent from the bottom of my heart.

A deep and immense thank you goes to all my friends: from time to time, I use to say out loud how lucky I am. You have always been kind to me, supportive in all of my decisions and made me feel I was in the right place. In brief, you are the reasons why I feel lucky. The intense joy that I feel every time I see you can turn any day into a special one. I love living my life because of you and I will always be thankful for having the chance to share it with all of you.
For the future: as fast as we have bumped into each other, life will might bring us apart. I would like you to know that wherever you are or will be, I will never forget what you mean to me and what you have done for me.

A special mention to my girlfriend Federica: from the moment I met you I knew you were somebody special. I have always truly admired your strength, persistence and intelligence. Having you at my side has been the strongest motivation to move forward when things just were not going the right way. You have inspired in me the motivation I try to put in everything I do, the ambition to always be better and the bravery to put myself on the line. You know that this is only a small portion of what you represent to me now, and therefore I would like to thank you for the person you are and the love that lights up all my days.

I would like now to mention Professor Mauro Filippini: I recall myself in participating at few of the presentations you were hosting, and I do not remember a single time I got out of that room, after your speech, without an indescribable desire and excitement to leave the next day (if only I could). The sense of adventure and curiosity you were able to transmit in every single word, the connection you were capable of establish with me meant so much that it pushed me to take one of the most important decisions of my life: leave for the Alliance4Tech mobility program. And it changed my life.
As one of the many students that has passed through your wise and precious hands, I deeply thank you for the dedication and passion you put in your job every single day.

To my trainers and teammates from the "AS Tennis del Lago" tennis club: even if this is just a hobby, I consider you as a second family and this sport is what connects us as such. I would like to thank all of you for the passion we share for this beautiful sport, for the warm I feel every time I get on the court and the moments we have spent together during trainings and cups. We are a great team, and I love being part of it.

Finally, a huge thank you goes to all the people I met in this life, either big or small their role might have been. Each one of them contributed to build the person I am proud to be today, in all of his good and evil sides, but always willing to grow, to be passionate and curious, to love, to fail and improve, never stopping. This is the way I have learnt to live in this world, the one that better meets the values that I believe to be fundamental. And if I had the chance to grasp this knowledge, this is all thanks to you.