UNIVERSIDADE FEDERAL DO PARÁ
INSTITUTO DE TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

5G MIMO and LIDAR Data for Machine Learning: mmWave Beam-selection using Deep Learning

Marcus Vinicius de Oliveira Dias

DM: 22/2019

UFPA / ITEC / PPGEE

Campus Universitário do Guamá

Belém-Pará-Brasil

2019

UNIVERSIDADE FEDERAL DO PARÁ
INSTITUTO DE TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

Marcus Vinicius de Oliveira Dias

5G MIMO and LIDAR Data for Machine Learning: mmWave Beam-selection using Deep Learning

DM: 22/2019

UFPA / ITEC / PPGEE

Campus Universitário do Guamá

Belém-Pará-Brasil

2019

UNIVERSIDADE FEDERAL DO PARÁ
INSTITUTO DE TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

Marcus Vinicius de Oliveira Dias

5G MIMO and LIDAR Data for Machine Learning: mmWave Beam-selection using Deep Learning

A dissertation submitted to the examination committee in the graduate department of Electrical Engineering at the Federal University of Pará in partial fulfillment of the requirements for the degree of Master of Science in Electrical Engineering with emphasis in Telecommunications.

UFPA / ITEC / PPGEE
Campus Universitário do Guamá
Belém-Pará-Brasil
2019

Dados Internacionais de Catalogação - na – Publicação (CIP) Sistema de Bibliotecas da UFPA

D541q Dias, Marcus Vinicius de Oliveira, 1992-
5G MIMO and LIDAR data for machine learning: mmWave Beam-selection using deep learning / Marcus Vinicius de Oliveira Dias.- 2019.
Orientador: Aldebaro Barreto da Rocha Klautau Júnior
Dissertação (Mestrado) - Universidade Federal do Pará, Instituto de Tecnologia, Programa de Pós-Graduação em Engenharia elétrica, Belém, 2019.

1. Sistemas de comunicação sem fio . 2. Teoria dos Sinais (Telecomunicações). 3. Redes neurais (Computação). 4. Detectores. I. Título.

CDD 23. ed. 621.38456

Elaborada por Lucicléa S. de Oliveira – CRB -2/648

5G MIMO and LIDAR Data for Machine Learning: mmWave Beam-selection using Deep Learning

Dissertação de mestrado submetida à avaliação da banca examinadora aprovada pelo colegiado do Programa de Pós-Graduação em Engenharia Elétrica da Universidade Federal do Pará e julgada adequada para obtenção do Grau de Mestre em Engenharia Elétrica na área de Telecomunicações.

Aprovada em _____ / _____ / _____

Acknowledgments

First, I would like to thank my advisor, Prof. Aldebaro Klautau, for the intellectual guidance on this work, for the motivational support during my masters degree program and for the knowledge provided during this interaction over the past years.

Secondly, I wish to thank Prof. Carlos Renato Francês, Prof. Diego Gomes and Dr. André Cavalcante for serving as readers and examiners of this dissertation.

Third, I would like to thank the colleagues that collaborated to this work. Special thanks to Pedro Batista, Joary Fortuna and Marcos Yuichi Takeda, who collaborated with solutions on the issues during the development presented in this work. I was privileged to have received insights from their expertise in the field. Also thanks to Ailton Oliveira and Isabela Trindade, who helped on the simulation tasks and the elaboration of the methodology of creation of realistic environments.

Furthermore, I am grateful to the colleagues from LASSE at the Federal University of Pará who made the past years of work enjoyable.

Last and most importantly, I wish to thank to my parents Augusto Dias and Edilma Dias, my brothers Rodrigo Dias and Tiago Dias, and my girlfriend Yuki Murakami for all their love and support that motivates me to pursue excellence as a professional and a human being.

Marcus Vinicius de Oliveira Dias

August 2019

Those who are quite satisfied sit still and do nothing; those who are not quite satisfied are the sole benefactors of the world.

Walter S. Landor

List of Symbols

BS	Base Station
CAD	Computer-Aided Design
CC	Control Channel
CNN	Convolutional Neural Network
DFT	Discrete Fourier Transform
DNN	Deep Neural Network
DS	Diffuse Scattering
DXF	Drawing eXchange Format
GPS	Global Positioning System
IJEE	International Journal of Engineering Education
LIDAR	LIght Detection And Ranging
LOS	Line Of Sight
MIMO	Multiple-Input Multiple-Output
ML	Machine Learning
mmWave	millimiter-Wave
NLOS	Non Line Of Sight
OFDM	Orthogonal Frequency-Division Multiplexing
PHY	Physical Layer

PO	Python Orchestrator
RF	Radio Frequency
RT	Rate Throughtput
RWI	Remcom’s Wireless InSite
SBRC	Brazilian Symposium on Computer Networks and Distributed Systems
SBrT	Brazilian Symposium on Telecommunications and Signal Processing
SPAWC	Signal Processing Advances in Wireless Communications
SUMO	Simulation of Urban Mobility
UPA	Uniform Planar Arrays
USGS	United States Geological Survey

List of Figures

List of Tables

Contents

Abstract

Modern communication systems can exploit the increasing number of sensor data currently used in advanced equipment and reduce the overhead associated with link configuration. Also, the increasing complexity of networks suggests that machine learning (ML), such as deep neural networks, can effectively improve 5G technologies. The lack of large datasets make harder to investigate the application of deep learning in wireless communication. This work presents a simulation methodology (RayMobTime) that combines a vehicle traffic simulation (SUMO) with a ray-tracing simulator (Remcom's Wireless InSite), to generate channels that represents realistic 5G scenarios, as well as the creation of LIDAR sensor data (via Blensor). The created dataset is utilized to investigate beam-selection techniques on vehicle-to-infrastructure using millimeter waves on different architectures, such as distributed architecture (usage of the information of only a selected vehicle, and processing of data on the vehicle) and centralized architectures (usage of all present information provided by the sensors in a given moment, processing at the base station). The results indicate that deep convolutional neural networks can be utilized to select beams under a top-M classification framework. It also shows that a distributed LIDAR-based architecture provides robust performance irrespective of car penetration rate, outperforming other architectures, as well as can be used to detect line-of-sight (LOS) with reasonable accuracy.

Index terms — 5G, Ray-tracing, MIMO, Beam selection, Machine Learning, Convolutional networks, Deep neural networks.

Resumo

Sistemas de comunicação modernos podem explorar o crescente número de dados de sensores usados atualmente em equipamentos avançados e reduzir a sobrecarga associada à configuração de links. Além disso, a crescente complexidade das redes sugere que o aprendizado de máquina, como redes neurais profundas, podem ser utilizadas efetivamente para melhorar as tecnologias 5G. A falta de grandes conjuntos de dados dificulta a investigação da aplicação de aprendizado profundo na comunicação sem fio. Este trabalho apresenta uma metodologia de simulação (RayMobTime) que combina um simulador de tráfego de veículos (SUMO) com um simulador de *ray-tracing* (Remcom's Wireless InSite), para gerar canais que representem cenários 5G realísticos, bem como a criação de dados de sensores LIDAR (através do Blensor). O conjunto de dados criado é utilizado para investigar técnicas de *beam selection* de veículo para infraestrutura usando ondas milimétricas em diferentes arquiteturas, como arquitetura distribuída (uso das informações de apenas um veículo selecionado e processamento de dados no veículo) e arquiteturas centralizadas (uso de todas as informações presentes fornecidas pelos sensores em um dado momento, processando na estação base). Os resultados indicam que redes neurais profundas convolucionais podem ser utilizadas para *beam selection* sob uma estrutura de classificação de top-M. Também mostra que uma arquitetura distribuída baseada em LIDAR fornece desempenho robusto independentemente da taxa de penetração de veículos, superando outras arquiteturas, bem como pode ser usada para detecção de visada direta com precisão razoável.

***Palavras-chave* —** 5G, Ray-tracing, MIMO, Beam selection, Aprendizado de máquina, Redes convolucionais, Redes neurais profundas

Chapter 1

Introduction

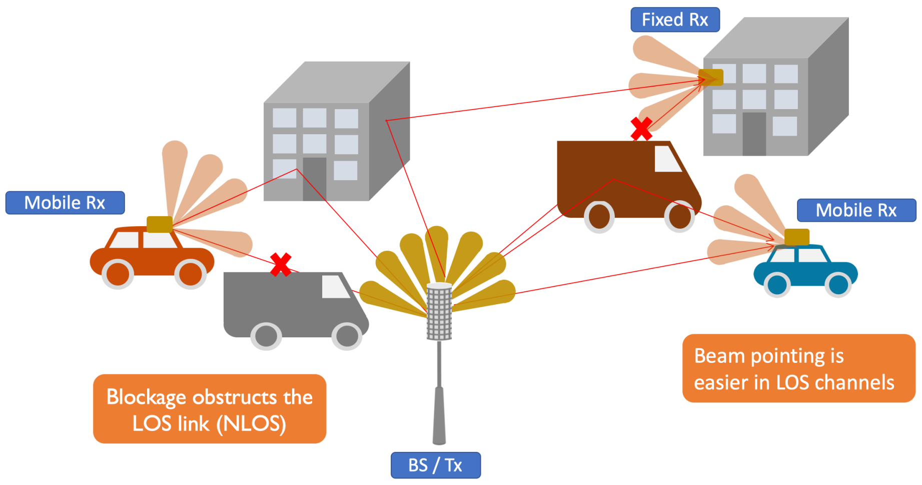
The growth in computational power accompanied by a faster and increased data storage have already allowed the application of machine learning (ML) techniques on large variety of datasets that were previously intractable due to their size and complexity. In telecommunications, ML has been applied in problems such as network management, self-organization, self-healing, and physical layer (PHY) optimizations [1, 2].

Deep learning (DL), a special case of ML, is responsible for several recent performance breakthroughs in areas such as speech processing, computational vision and even biological data [3–5]. One of the keys for the success of DL in many application is abundant data availability or a low cost on its generation. The reason is that DL performance scales well with the amount of data and model complexity [4].

In parallel, the growing demand for bandwidth, low latency connections, and the scarcity of spectrum available in sub-6 GHz bands suggest a requirement for new technology. Despite the usage of signal processing approaches like cognitive radio free more spectrum, it still is not enough to supply gigabit-per-second data rates and lower latencies. MmWave address these demands by using higher carrier frequencies, where channels with larger bandwidth can take place, which means higher data rates. For example, channels with 2 GHz of bandwidth are common for systems operating in the 60 GHz unlicensed millimeter wave (mmWave) band. Multiple-input multiple-output (MIMO) communication also can be applied to this transmission technology to achieve higher data rates.

In environments such as vehicle-to-infrastructure (V2I), mmWave is utilized for sharing high rate sensor data for connected and automated vehicles [6]. As mmWave uses higher carrier frequencies, the reach of the propagation gets lower. To bypass this problem, beam-pointing is utilized to extend the propagation range. However, select the appropriate beam can associate a high overhead to the system, given that vehicles continually change its position, and blockages may appear at every moment. Figure 1.1 shows the overview of the beam-selection problem, where beam-poiting in receivers with line-of-sight (LOS) is easier than receivers with non-LOS (NLOS).

Figure 1.1: Overview of the beam-selection problem. NLOS beam-selection is harder because it has to select a beam that points to other position than the receiver's position.


Fixed Rx
Mobile Rx
Mobile Rx
Beam pointing is easier in LOS channels
Blockage obstructs the LOS link (NLOS)
BS / Tx

Source: Author.

Reducing the beam-selection overhead is important in cellular and WiFi systems operating at mmWave frequencies [7–9]. Prior work has shown that out-of-band information, such as position obtained from vehicles, can be used to reduce the overhead required to establish mmWave links [6, 10], and that ML can be applied to these problems [11]. The usage of sensors like LIDAR, which is already used on automated vehicles for mapping, positioning, or obstacle detection, can provide an additional source of information that can be taken advantage without cost to reduce communication overhead.

This work concentrates on the challenges of channel data generation in 5G mmWave MIMO scenarios as well as LIDAR data generation. In parallel, it proposes a ML-based beam-selection strategy for V2I mmWave cellular communication system which utilizes the datasets created. In this context, it is assumed that the base station (BS) broadcasts its position via a low-frequency control channel (CC), and given the receiver information (e.g., GPS position and

LIDAR data), the ML is utilized to tackle two key problems in the mmWave system. First, a predictor assess whether the channel is in LOS or NLOS. LOS detection is useful because beam-selection is easier in the LOS setting. Second, to recommend a set of beams, a deep learning (DL) [4] with a convolutional neural network (CNN) is trained to perform top-M classification [12] conditioned on LOS and NLOS state estimated.

The processing location (whether it is the BS or the connected vehicle) and which information (GPS position and / or LIDAR data) depends on the DL architecture. The DL architectures proposed fall into two main cathegories: *distributed* architecture and *centralized* architecture. The main difference is that the *distributed* the processing is performed by the connected vehicle while the *centralized* the processing is performed by the base station (BS) and usually aggregates information of others connected vehicles to perform its decision.

This work presents simulation results obtained with a methodology that combines a traffic simulator to model realistic mobility scenarios with *paired* data for ray-tracing (for estimating mmWave channels) and LIDAR simulators.

An advantage of distributed architecture over centralized is that it only utilizes the position and LIDAR of the connecting vehicle, not depending on the penetration rate (percentage of connected vehicles in relation to the total number of vehicles present in the simulation) of connected vehicles. Since the BS does not have information of unconnected cars, the beam-selection performance in a centralized architecture decreases with the number of unconnected vehicles [11].

1.1 Research Contributions

The contributions of this work fall generically in: (1) on the methodology of creation of realistic 5G mmWave MIMO datasets, softening the data scarcity, thus facilitating the investigation of ML-based problems related to that area; (2) creation of LIDAR datasets, turning possible to assess its benefits on the communication area; and (3) proposal of deep learning architectures that utilize the datasets created to tackle the beam-selection problem;

This work also provides a comprehensive overview of software and technologies associated to the datasets generation, as well as shows the power and application of deep neural networks.

1.2 Dissertation Outline

The following summarizes the organization of this work and its main contributions.

Chapter 2: provides an overview of the tecnologies utilized as well as the software choosen to be utilized for the dataset creation.

Chapter 3: introduces the elementary background with respect to machine learning and deep neural networks.

Chapter 4: presents the data creation methodology and post processing of the data.

Chapter 5: presents the results obtained in different deep learning architectures.

Chapter 6: presents the conclusions and describes possible future extensions of the work.

Chapter 2

Software Tools

Millimeter-wave communication is one of 5G networks strategies to achieve higher bit rates. Measurement campaigns with these signals are difficult and require expensive equipment [13]. Since research and development of lower 5G layers deal with a relatively limited amount of channel data, generating data is a way to alleviate data scarcity while benefiting from an accuracy associated with ray-tracing.

This chapter shows the technologies and software utilized to generate 5G and LIDAR datasets in realistic scenarios where the mobility vary over time. These simulated datasets do not replace, but complement measurement data, which can improve and validate simulated data and statistical channel models as they become available.

2.1 Ray-tracing and Remcom's Wireless InSite

Ray-tracing is an approach to calculate the path of the signals through a system with different absorption characteristics and reflecting surfaces. Ray-tracing is considered one of the most effective methods for estimating the propagation characteristics in MIMO systems, and provide very accurate results [14, 15]. An issue of ray-tracing is that the generation of channels depends on the specific propagation environment, and its accuracy is strictly related to the scenario's detail. For example, in an outdoor scenario, a detailed specification (such as size, material, and electromagnetic parameters) of buildings, pedestrians, vehicles and other objects of interest, is required to obtain an accurate result.

Given the simulation scenario, basic configurations such as the position of transmitter and receiver, wave format, frequencies, and other electromagnetic parameters have to be set.

Ray-tracing calculations can be performed by method of images, shooting rays, etc, where the shooting rays is the most tradicional method. On the shooting rays technique, the rays are shoot from the transmitters and propagating them through the scenario. These rays interact with the present 3D features and make their way to receiver locations. The ray interactions include reflections from feature faces, diffractions around feature edges, and transmissions through features faces. The ray interactions may also include diffuse scattering and interactions with the vegetation.

At each receiver, arriving ray paths are combined and evaluated to determine predicted quantities such as electric and magnetic field strength, received power, interference measures, path loss, delay spread, the direction of arrival, impulse response, electric field versus time, electric field versus frequency, and power delay profile.

Remcom's Wireless InSite (RWI) is the software utilized to perform the ray-tracing on this work. RWI enables the creation of scenarios (including buildings, terrain and foliage) using its editing tools or importing from several file formats, such as *DXF*, *shapefile*, *DTED* and *USGS*. Transmitter and receiver locations can be specified using RWI's site-defining tools, or even imported from an external data file. It is also possible to create new materials on RWI by providing their properties such as electrical permissiveness, electrical conductivity, hardness, and thickness, which vary according to the frequency of the wave used in the simulation.

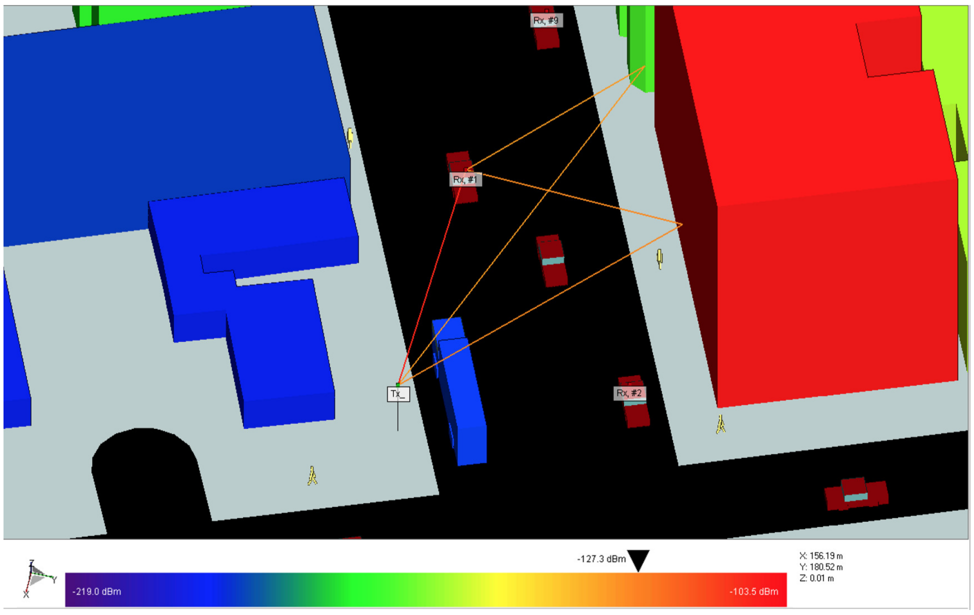
The RWI X3D propagation model includes atmospheric absorption, extending the validity of its propagation calculations up to mmWave frequencies. All RWI output files produced by Wireless InSite are in a readable *ASCII* format, which turns to be an easy file to read by other programs. Figure 2.1 depicts a ray-tracing performed using RWI.

2.2 LIDAR and Blensor

LIDAR (Light Detection and Ranging) devices are the key sensor technology in today's autonomous systems. It originated in the early 1960s, shortly after the invention of the laser. LIDAR is a remote sensing optical technology that measures properties of reflected light to obtain distance of a particular distant object.

LIDAR works emitting laser pulses that reflect into any object on its surrounding. Then the distance is obtained by calculating the time elapsed from the emission of a laser pulse and the time of return to the sensor. Time is converted into distance, and it is associated with

Figure 2.1: RWI ray-tracing performed in a 3D urban canyon scenario. The color of the rays represents its strength. The stronger ray (red) is in LOS case, while the others are reflecting in a building before reaching the receiver, therefore a NLOS case.



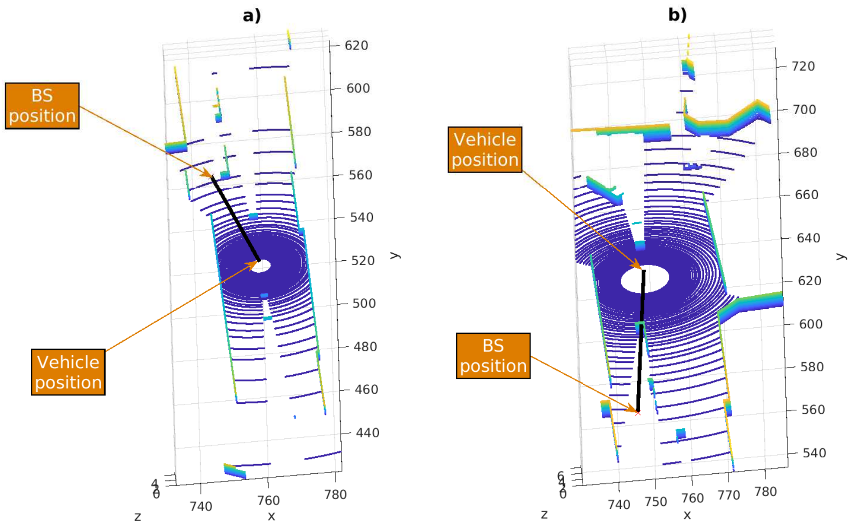
Source: Author.

positioning information. LIDAR's output is a point cloud data, which can be interpreted as three-dimensional (3D) images with pixels indicating relative positions from the sensor [8]. LIDAR data can be utilized for obstacle detection, tracking, surface reconstruction or object segmentation. Figure 2.2 depicts how the LIDAR scan could help on identify line-of-sight (LOS) or non-LOS (NLOS) cases.

Many algorithms exist to process and analyze LIDAR data. However, most of those algorithms are tested on recorded (usually not publicly available) sensor data, and algorithmic evaluations rely on visual inspection of the results, mainly due to the lack of available ground truth.

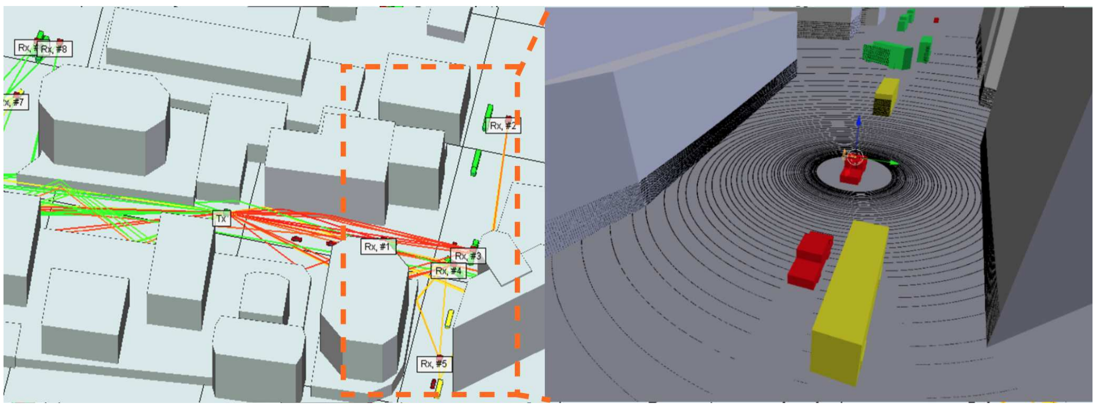
Blensor, a toolkit completely integrated with Blender (an open source 3D creation suite) [16], allows researchers to develop algorithms without the need to possess a sensor physically, providing an exact per measurement ground truth. The integration with Blender provides a unified simulation and modeling environment, being capable of model scenarios with an arbitrary level of detail and immediately simulate the sensor output directly within the modeled environment. It also allows Python code to run the simulation, turning the scalability of the simulations easier.

Figure 2.2: Two examples of point clouds from a LIDAR located on a vehicle. The line between the LIDAR and the mmWave radio at the BS is also shown. In a) this line does not encounter any LIDAR point. This suggests a mmWave LOS link, which may be checked using a ray-tracing simulation. In b) the line traversing points detected by the LIDAR indicates a NLOS condition.



Source: Author.

Figure 2.3: Example of a Blensor LIDAR scan (right) performed in one of the scenarios of the simulation (left).



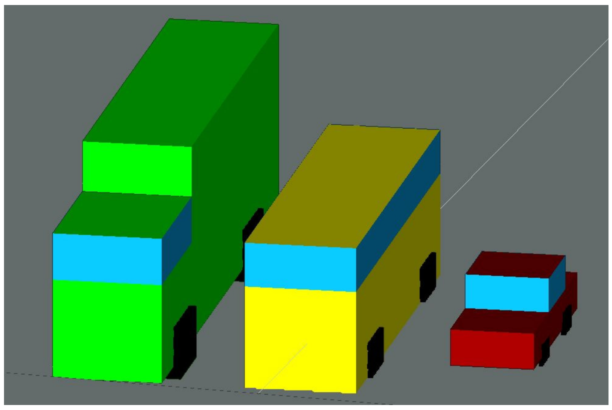
Source: Author.

Blensor has a variety of LIDAR sensors available. Among them, there is even a commercial product named Velodyne. Velodyne HDL-64E is a commercial product mainly designed for obstacle detection and autonomous vehicles navigation. This unit features a unique combination of high resolution, broad field of view, high point cloud density, and an output data rate, up to one million points per second. Figure 2.3 depicts a Blensor LIDAR scan using Velodyne HDL-64E.

2.3 Blender

Blender supports the entirety of the 3D pipeline: modeling, rigging, animation, simulation, rendering, compositing and motion tracking, even video editing and game creation [16]. Figure 2.4 shows the 3D models created by Blender that will be utilized by both the LIDAR and ray-tracing simulation. As mentioned in the Section 2.2, Blender takes great importance in the LIDAR simulation, since it is responsible for building / replicating the same 3D environment created on RWI, then creating a *paired* simulation. An example of a *paired* simulation is shown on Figure 2.5.

Figure 2.4: Blender models exported to RWI.

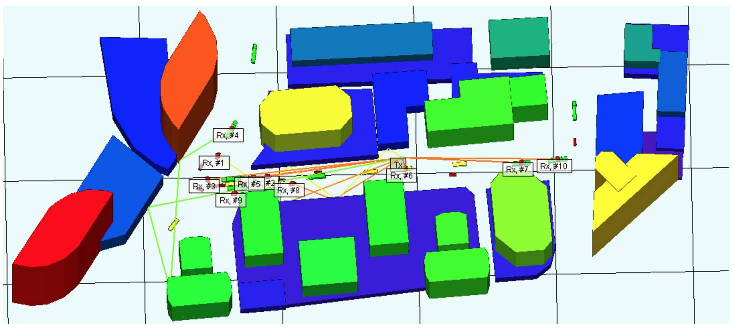


Source: Author.

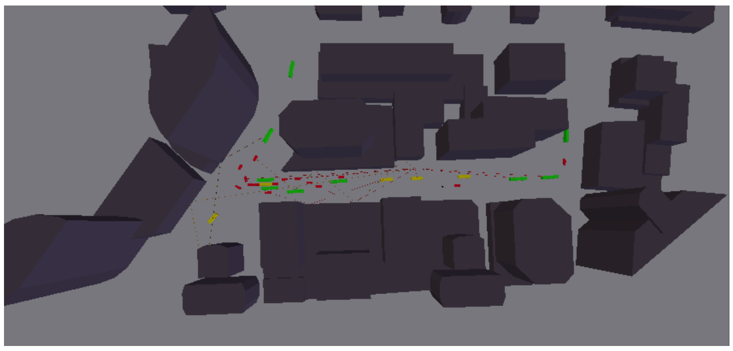
2.4 Simulation of Urban Mobility (SUMO)

SUMO [17] is introduced with the purpose of creating an environment which represents the mobility found in outdoor scenarios. SUMO is a free and open source software designed for traffic simulation. In addition, it can also be utilized to evaluate infrastructure changes as well

Figure 2.5: A paired simulation on Rosslyn city, where every 3D object is positioned in the same place.



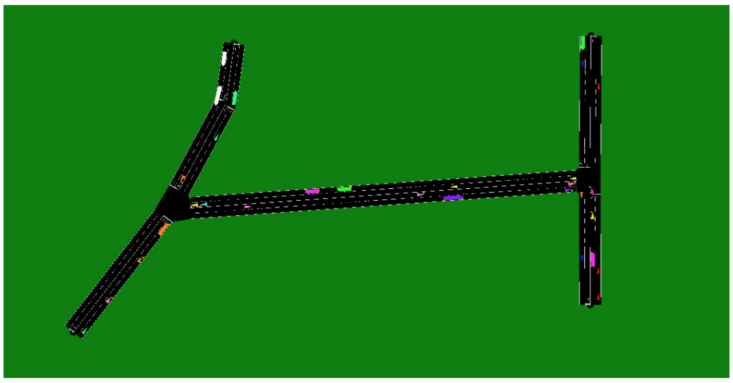
(a) Remcom's Wireless InSite



(b) Blender

Source: Author.

Figure 2.6: SUMO network of a study area in Rosslyn city.



Source: Author.

as policy changes before implementing them in the real world. For example, traffic light control algorithms can be tested and optimized in a simulation before being deployed in the real world.

In this work, SUMO is utilized to generate a traffic demand, which basically consists in create mobility of vehicles and pedestrians through a given road network, to enrich the simulation environment. Figure 2.6 shows a SUMO network designed to handle mobility.

2.5 OpenStreetMap and CadMapper

OpenStreetMap (OSM) is a collaborative project to create a free editable map of the world. The geodata underlying the map is considered the primary output of the project. The creation and growth of OSM has been motivated by restrictions on use or availability of map data across much of the world. CadMapper is an online tool that transform data from public sources like OSM, *NASA* and *USGS* in organized *CAD* files.

The data from OSM and CadMapper is utilized to create virtual environments based on places of the real world. OSM is mainly utilized to obtain the streets and lane network utiized on SUMO while CadMapper gives the 3D buildings. This process is better described on section 4.1.1.

Chapter 3

Machine Learning

A machine learning algorithm is an algorithm that is able to learn from data. ML allows us to perform tasks that are too difficult to solve with fixed programs written and designed by human beings. In this work, it is utilized a specific type of ML, called deep convolutional neural network, to tackle both the LOS detection and beam-selection problems. This chapter briefly introduces neural networks, deep learning, and convolutional networks, as well as the concept of overfitting, underfitting, and regularization.

3.1 Neural Network

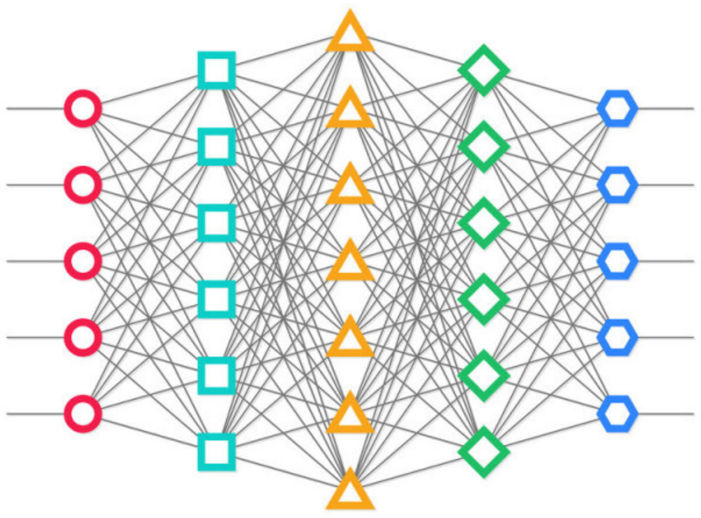
Neural networks are a type of machine learning models which are designed to operate similar to biological neurons and the human nervous system. These models are used to recognize complex patterns and relationships that exist within a labeled dataset.

The core architecture of a neural network model is comprised of a large number of simple processing nodes called *neurons*, which are interconnected and organized in different *layers*. An individual node in a layer is connected to several other nodes in the previous and the next layer. The inputs form one layer are received and processed to generate the output which is passed to the next layer.

Figure 3.1 represents a neural network. The first layer of this architecture is known as *input layer* which accepts the inputs, the last layer is named as the emphoutput layer which produces the output and every other layer between input and output layer is named is *hidden layers*.

The *neuron* is a single processing unit of a neural network which is connected to different

Figure 3.1: Neural network. The circles represents the *input layer*, the hexagons represents the *output layer* and the squares, triangles and rhombus represent *hidden layers*, while each of these geometric figures represent a neuron.



Source: [18].

other neurons in the network. These connections represent inputs and outputs from a neuron. To each of its connections, the neuron assigns a *weight* (W) which signifies the importance the input and adds a *bias* (B) term.

Each *hidden layer* and *output* has an activation function, which is used to apply a non-linear transformation on an input to map it to output. Activation functions aim to predict the right class of the target variable based on the input combination of variables. Some of the popular activation functions are Relu, Sigmoid, and TanH.

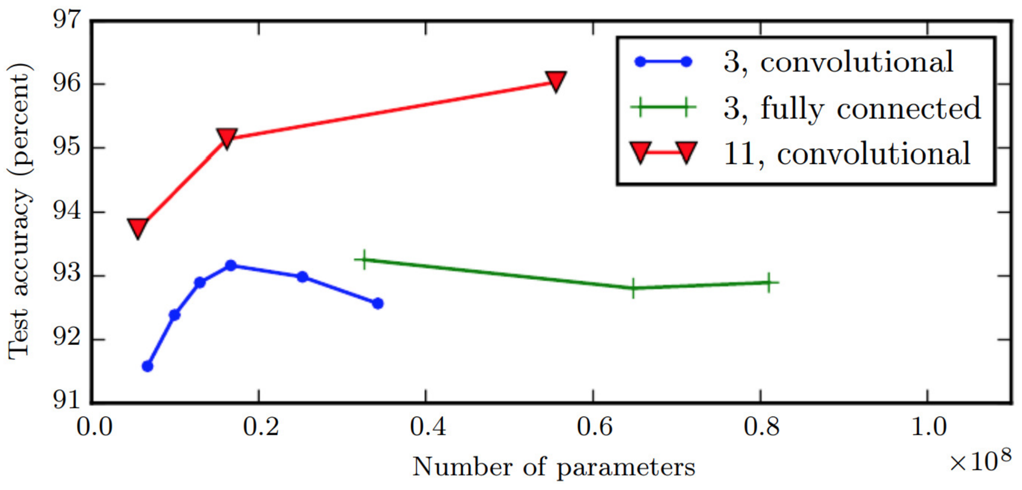
Most of ML based on a neural network is trained based in two phases: feed forward and back propagation. In the feed forward part of a neural network, predictions are made based on the values in the input nodes and the weights. In the beginning, these weights are randomly initialized, thus causing some random predictions. Then the backpropagation part begins, the predicted output is compared with the actual output computing the error.

The error is passed to backward layers so that those layers can also make small adjustments the weights and bias. It uses the algorithm called gradient descent in which the error is minimized, and optimal values of weights and bias are obtained. This weights and bias adjustment is done by computing the derivative of error, the derivative of weights, bias, and subtracting them from the original values. Loss function measures the error in the final layer, and cost function measures the total error of the network.

3.2 Deep Learning and Convolutional Neural Networks

Deep learning is a particular case of machine learning which inherits several characteristics from neural networks. Deep neural networks (DNN) use multiple layers to extract higher level features from raw input progressively. Deeper models tend to perform better, but not merely because the model is larger. Figure 3.2 shows that increasing the number of parameters in layers of convolutional networks without increasing their depth is not nearly as effective at increasing test set performance [19].

Figure 3.2: Comparison between shallow models and deep models. Shallow models in this graph overfit at around 20 million parameters while deep ones can benefit from having over 60 million.

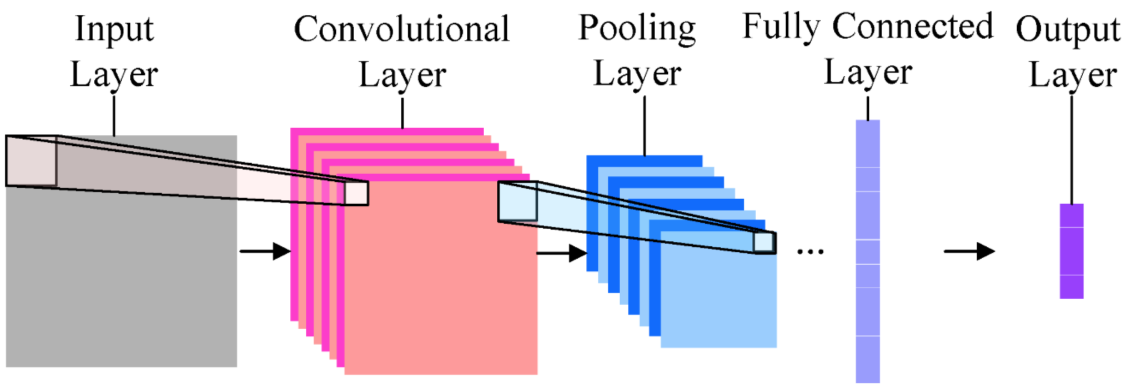


Source: [19].

As mentioned, this work utilizes a convolutional neural network (CNN). Despite being used mainly in imaging tasks, CNNs are also successful in other tasks such as speech recognition [20]. CNN learns to extract features automatically, which comes in handy when dealing with complex tasks. Figure 3.3 shows the layers that are often part of a convolutional network.

Convolutions layers work as filters (matrix/vectors) with learnable parameters that are

Figure 3.3: Convolutional neural network and its layers.



Source: [18].

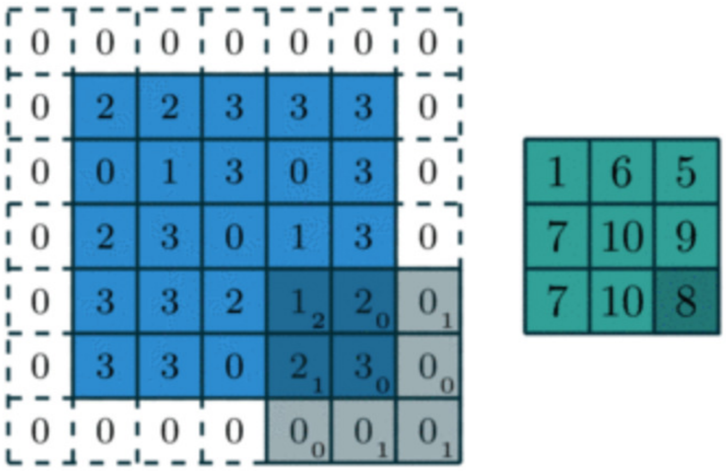
used to extract low-dimensional features from input data. They have the property to preserve the spatial or positional relationships between input data points. CNN exploits the spatially-local correlation by enforcing a local connectivity pattern between neurons of adjacent layers.

The convolution layer has a *weight matrix*, also known as *kernel*, that behaves like a filter with learnable weights that works like a sliding window. This filter slides over the input and extracts features from the original input matrix. The number of steps the weight matrix takes while moving across the entire input moving N pixel at a time is called a *stride*. For example, If the weight matrix moves N pixel at a time, it is called stride of N.

Higher *stride* values cause a large number of pixels to be moved at a time and hence producing smaller output volumes. If it is not desirable to have smaller output, *padding* the input image with zeros across it solves this problem. The technique called *same padding* retains the shape of the input producing an output with the same size as the input. Figure 3.4 shows the addition of padding, zeros around an input matrix. It is also possible to add more than one layer of zeros around the image in case of higher stride values.

When there are multiple convolutional layers, the initial layer extract more generic features, while as the network gets deeper, the features extracted by the weight matrices are more and more complex and more suited to the problem at hand. For example, in the case of an image, a weight combination might be extracting edges, while another one might be a particular color, while another one might blur the unwanted noise. The weights are learned such that the

Figure 3.4: Example of input matrix, kernel and padding. The input is represented by the blue squares, the kernel by the blurred squares on the bottom right, and the padding are the zeros added around the input matrix. The output produced is the green squares.



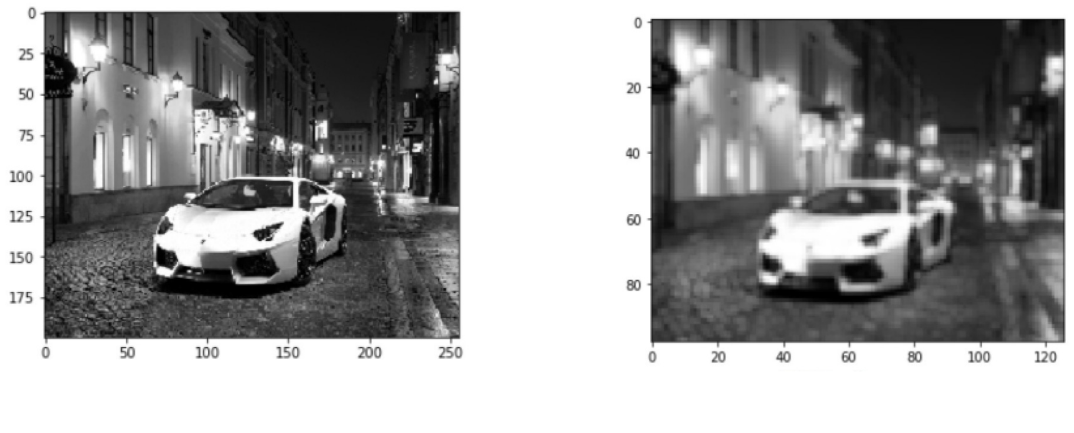
Source: [18].

loss function is minimized similar to a conventional neural network.

Pooling layers are introduced when it is required to reduce the number of trainable parameters, thus reducing the computational cost. This layer is added periodically between subsequent convolution layers. Pooling is done to reduce the spatial size of the input. Pooling is done independently on each depth dimension, therefore the depth of the input remains unchanged. The most common form of pooling layer generally applied is the *max pooling*. A max pooled input still retains all the information, for example, in the Figure 3.5b it is possible to see the information (car on the street), but the dimensions of the image are reduced by half if compared to the original image as shown in the Figure 3.5a, which helps to reduce the number of parameters.

The convolution and pooling layers would only be able to extract features and reduce the number of parameters from the original inputs. However, to obtain some sort of classification it is required to apply a fully connected layer as an output layer to generate an output equal to the number of classes we need. Note that the input of the fully connected layer has to be flattened. The output layer has a loss function like categorical cross-entropy, to compute the error in prediction. Once the forward pass is complete, the backpropagation begins to update

Figure 3.5: Original image versus image with max pooling.



(a) Original image. **(b)** Max pooled image.

Source: [21].

the weight and biases for error and loss reduction. One training cycle, known as *epoch*, is completed in a single forward and backward pass.

3.3 Overfitting, Underfitting and Regularization

The central challenge in machine learning is that we must perform well on new, previously unseen inputs, not just those on which the model was trained. The ability to perform well on previously unobserved inputs is called generalization.

The factors determining how well a machine learning algorithm will perform are its ability to: Make the training error small; and make the gap between training and test error small. Machine learning algorithms will always try to achieve the best accuracy as possible, which leads to two central challenges when trying to build a model: *underfitting* and *overfitting*. Models with low capacity may struggle to fit the training set, being not able to obtain a sufficiently low error value on the training set, what characterizes the *underfitting*. On the other hand, models with high capacity can overfit by memorizing properties of the training set that do not serve them well on the test set, having a large gap between the training error and test error.

Regularization strategies as *dropout* can be added to the model to soften this problem. Regularization is any modification made to a learning algorithm that is intended to reduce its generalization error but not its training error. An effective regularizer is one that makes a profitable trade, reducing variance significantly while not overly increasing the bias. In this work, it is utilized *dropout* as a regulation strategy.

Dropout is a technique where randomly selected neurons are ignored during training. This regularization technique comes with a price, it also reduces the effective capacity of the model. To offset this effect, the size of the model must be increased, and possibly need many more iterations of the training algorithm. *Dropout* is more effective in ML models that deal with a limited size dataset. For very large datasets, regularization confers a little reduction in generalization error. In these cases, the computational cost of using dropout and larger models may outweigh the benefit of regularization.

Chapter 4

Data creation methodology

This chapter focus on the dataset creation methodology of 5G mmWave and LIDAR, where a Python orchestrator puts together the software mentioned on Chapter 2.

4.1 RayMobTime

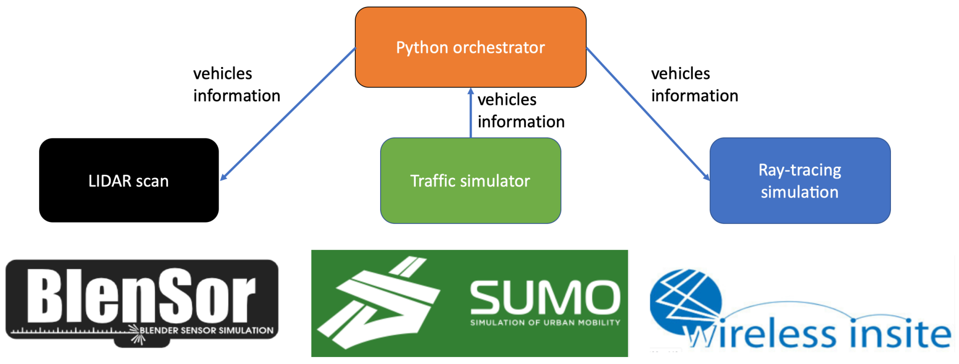
RayMobTime was created aiming to diminish the scarcity of 5G mmWave datasets. Currently, this methodology is capable of creating ray-tracing datasets with the mobility of vehicles, pedestrians, and drones. The created datasets can be classified into two types: *fixed* and emphmobile. The distinction of these datasets is defined by if whether the transceivers change location overtime or not.

Figure 4.1 shows the software which RayMobTime relies on. SUMO to be generating the urban mobility of vehicles, RWI to perform the ray-tracing, and a Python orchestrator (PO). The central role of the PO is to invoke SUMO and use its outputs (pedestrians, vehicles and drones positions, orientations, etc.) to create folders with RWI simulations. SUMO's outputs are also saved into a file to be utilized by other possible simulators. In this work, this file is utilized by the LIDAR simulator (Blensor) to obtain paired results. Blensor data is later utilized to help on the LOS predictor and beam-selection problems.

4.1.1 Methodology of base files creation

In order to create an environment where RayMobTime will take place, several configurations have to be done, as well as multiple base files have to be created. This section goes over the process of creation of these base files.

Figure 4.1: Data creation methodology. RayMobTime is composed by a Python orchestrator, a traffic simulator and a ray-tracing simulator. Python orchestrator gets vehicles information from the SUMO and send this information to both RWI and Blensor to obtain paired simulations.



Source: Author.

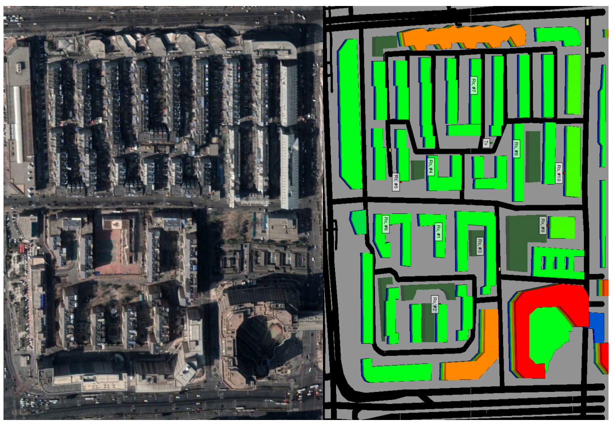
Blender and RWI as mentioned in Chapter 2 have the ability to import 3D scenarios into them using a variety of file types. This ability is exploited to import places from the real world, then possibly obtaining results closer to the real as possible.

The first step to import an outdoor place from the real world to a virtual environment, it is necessary to choose a study area. Figure 4.2 shows a *study area* where the analyze of signal propagation could take place. Once the *study area* is chosen, *CadMapper* is utilized to import a real scenario. Mainly, the buildings, streets, and terrain are exported from this *CAD* file and converted to a *DXF* file that can be interpreted by both RWI and Blender. It is important to note that big cities usually have better information about places, such as the correct height of the buildings, turning these cities a better target for our simulations.

After the buildings, street and terrain are imported,it is time to configure some propagation properties on RWI. Carrier frequency, type of antenna used on the transceivers, transceivers locations, propagation model, number of rays that will be propagated are some examples of ray-tracing parameters/properties contained on RWI's base files.

RWI's base files can be classified into two types of files: *fixed* and *changeable*. The *fixed* are the ones the hold features that are static, such as city's terrain, vegetation, road and buildings, and will remain untouched by the Python orchestrator. The *changeable* are the ones that hold information about things that change overtime such as pedestrians, vehicles, drones

Figure 4.2: Selected study area: satellite image taken from Google Maps at left, and the DXF file imported to RWI at right. The colors represents different building heights.



Source: Author.

and transceivers, and will require some updates. However, there are exceptions; for example, it can be chosen that the transceivers do not change its locations, generating the *fixed simulations*.

To inject mobility to this created 3D environment, it is needed to import SUMO's road network file, which contains the paths where our mobile objects can travel. Once provided the global coordinates of the *study area* to *OpenStreetMaps*, it gives a file with information about the location's traffic lanes, which is then utilized to create SUMO's road network. It is also necessary to generate traffic planning for vehicles and pedestrians on these scenarios. The traffic planning is done in such a way which improves the diversity of results in the simulations. For example, a seasonal plan for traffic is utilized to increase the number of vehicles on a period (day) and a decrease in another (night).

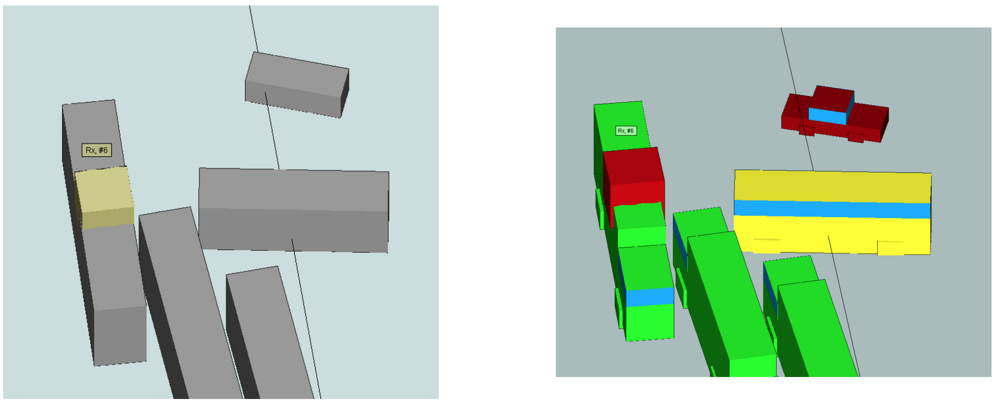
By adapting the software tools, it was possible to even include drones mobility. From standard street generation techniques in the SUMO software, airways can be created so that it is possible to choose routes for drones, enabling them to land, take off, or just fly over the *study area*.

There is another crucial SUMO configuration which holds the probabilities of an object appear in some routes in a given timestamp and the characteristics of mobile objects (sizes, acceleration, max speed, probabilities of being chosen, etc.).

As mentioned on Chapter 2, the realism of the ray-tracing simulation depends on the details of the scenario, which include geometric aspects (number of object faces, etc.) and materials (electromagnetic parameters, etc.). To improve even more our simulations, instead of representing vehicles with boxes like in [11], it was added an option to utilize 3D mobile objects created with Blender. Figure 4.3 shows the difference between the boxes representation, where only one material could be assigned, and the created 3D mobile objects, where multiple materials could be assigned.

In their corresponding RWI's configuration, these 3D models enable multiple materials to be associated to these objects, for example in the car model, the region of the windshield will be associated the glass material while most of the vehicle will be made of metal, as can be seen in Figure 4.3b. It should be noted that the new (more realistic) models increase the ray-tracing simulation time due to the larger number of 3D faces per object.

Figure 4.3: 3D objects that can be utilized in RayMobTime simulations.



(a) Box 3D model.

(b) Blender 3D models.

Source: Author.

RayMobTime has several configuration parameters, among them, there is the use of the concepts such as episodes and scenes. Episode is utilized to keep track of a set of receivers position (in the case of mobile simulations). An episode may have several scenes, where each scene represent a *snapshot* of the simulation. RayMobTime parameters have a significant effect on the simulation. For example, it is possible to choose: if the simulation will utilize whether the 3D models or the boxes representing the objects; the type of the simulations (*fixed* or *mobile*); number of scenes per episode; number of vehicles with receivers; sampling period of the scenes; jump in time on the end of an episode (which becomes the time sampling if the number of scenes per episode is equal to one); etc.

4.1.2 Python Orchestrator

After the creation of all base files, the configuration of RayMobTime and SUMO's files, the PO is ready to take place. The workflow of the PO can be seen in the Appendix A, where Algorithm 1 is the workflow for fixed simulations and Algorithm 2 for *mobile simulations*. We first describe the *mobile simulations*.

At the beginning of every episode, PO gets SUMO's outputs and check if the number of vehicles (*num_veh*) on the simulation is greater or equal the number of connected vehicles (*num_veh_antenna*) required for that simulation. A step on the simulation time is taken until this condition is met. Then, there is the selection of the vehicles that will own an antenna, which means that it will choose which vehicles are connected. In addition, the scene variable is reset to 0. An episode ends when all the scenes of this episode are done, or it can finish prematurely if all connected vehicles leave the *study area*.

After the connected vehicles are chosen, the PO uses SUMO's outputs to do the placement of the 3D objects on RWI. Then, there is the execution of ray-tracing for that scene, and a step on the simulation time to move the objects. This process repeats until the end of the simulation. Each scene creates an RWI *run* folder, which contains the information utilized for positioning the 3D objects, the modified RWI's base files, and the results of the ray-tracing, for a given snapshot.

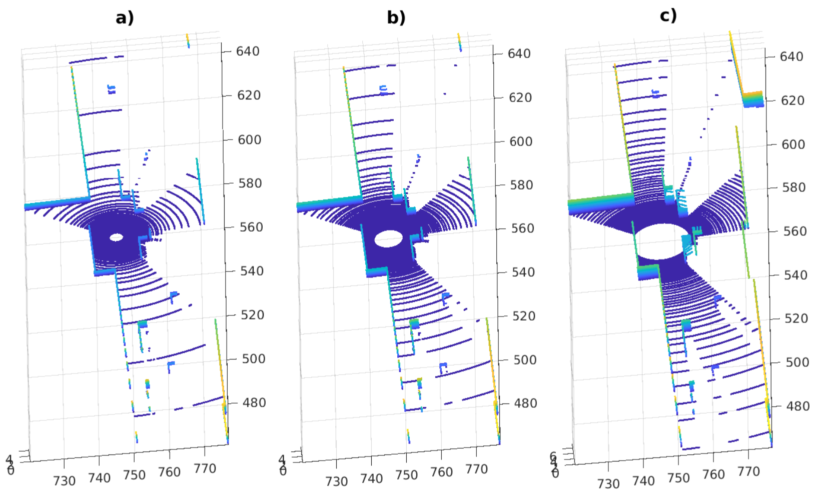
For the *fixed simulations*, the receiver antennas are not positioned on mobile objects, they are fixed. As can be seen on Algorithm 1, there is no step for antennas position updates and it is not required to verify if the receivers left the *study area*.

When creating the LIDAR dataset, a *paired simulation* is created on Blender using the information file contained on each *run* folder. Then, to create the LIDAR data it is required to position the sensor and perform the scan utilizing the Blensor toolkit.

In this work, there are two different LIDAR datasets utilized by the ML architectures. The first one is created assuming that every connected vehicle (has an antenna), also have a LIDAR giving environment information of its surroundings. The second is created equipping a LIDAR at the base station. Properly positioning the LIDAR at the BS and vehicles is important. Figure 4.4 depicts the results considering a LIDAR at a pole with three distinct heights (1, 2 or 4 m). As expected, the LIDAR position significantly impacts the resulting scans, and the number of points (obstacles) obtained. It is out of the scope of this work to optimize this height, and we assumed that the LIDAR at the BS is located at the same height of the antenna array, at $z = 4$ m. The LIDAR on a vehicle is on the top of its roof.

It is important to note that it was assumed that: the LIDAR is able to make a full rotation in every snapshot, which means that the moving objects are treated as static on each *snapshot*; and the LIDAR will not scan the vehicle itself, in the case of the LIDAR on the vehicles.

Figure 4.4: Scans for a LIDAR at a BS assuming the LIDAR is located (in the center of the circles) at different heights z: a) 1 m, b) 2 m and c) 4 m.



Source: Author.

4.2 Post-processing data

After the creation of the ray-tracing and LIDAR raw data, a post-processing phase is required to transform this raw data into something that a neural network can learn. In the LIDAR data, it is known that every scan returns a number of points which is associated with the number of objects surrounding the sensor. To become an input, this data must transform this variable number of points into a fixed shape. In the case of the ray-tracing data, its output should be transformed into labels, such that the DNN could suggest a set of beams.

4.2.1 mmWave communication model

It is assumed a downlink orthogonal frequency-division multiplexing (OFDM) mmWave system with analog beamforming [10]. Both transmitter and receivers have antenna arrays with only one radio frequency (RF) chain and fixed beam codebooks. Ray-tracing data is utilized to simulate the channel and combine the ray-tracing output with a *wideband* mmWave *geometric channel model* as in [22].Assuming R_c multipath components (MPC) per transmitter / receiver pair, the information collected from the ray-tracing outputs for the r-th ray of a given pair is: complex path gain α_r, time delay τ_r and angles ϕ_r^D, θ_r^D, ϕ_r^A, θ_r^A, corresponding respectively to azimuth and elevation for departure and arrival, and whether it is a LOS or NLOS situation.

To describe the channel model at the time instant corresponding to the n-th symbol vector following [10]. N_t and N_r are the numbers of antennas at the transmitter and receiver, respectively, the normalization factor $\sqrt{N_t N_r}$, $g(\tau)$ is the shaping pulse (a raised cosine with roll-off of 0.1), $T = 1/B$ the symbol period, B the bandwidth, $\mathbf{a}_r(\phi_r^A, \theta_r^A)$ and $\mathbf{a}_t^*(\phi_r^D, \theta_r^D)$ are the array steering vectors at the receiver and transmitter for the r-th MPC, respectively. Therefore, the sampled channel is

$$\mathbf{H}[n] = \sqrt{N_t N_r} \sum_{r=0}^{R_c-1} \alpha_r g(nT - \tau_r) \mathbf{a}_r(\phi_r^A, \theta_r^A) \mathbf{a}_t^*(\phi_r^D, \theta_r^D), \tag{4.1}$$

To simplify notation, when using uniform planar arrays (UPAs), is adopted the corresponding Kronecker products of steering, precoding and combining vectors to represent the 2D arrays as 1D vectors [22].

Assuming OFDM with K subcarriers and that $\mathbf{H}[n]$ can be fairly represented by its first L taps, the frequency-domain channel at subcarrier k is the discrete fourier transform (DFT) of (4.1)

$$\mathsf{H}[k] = \sum_{n=0}^{L-1} \mathbf{H}[n] e^{-\mathrm{j}\frac{2\pi k}{K} n}. \tag{4.2}$$

Assuming beam codebooks $\mathcal{C}_t = \{\mathbf{f}_1, \cdots, \mathbf{f}_{|\mathcal{C}_t|}\}$ and $\mathcal{C}_r = \{\mathbf{w}_1, \cdots, \mathbf{w}_{|\mathcal{C}_r|}\}$ at the transmitter and the receiver sides, with no restriction on the codebook size, they do not have to be DFT codebooks, where $|\mathcal{C}_t|$ and $|\mathcal{C}_r|$ are their corresponding cardinalities, which are not imposed by the number of antennas as in a conventional DFT codebook. For a given pair (p, q) of vectors, representing precoder $\mathbf{f}_p$ and combiner $\mathbf{w}_q$, the received signal at subcarrier k is

$$\mathbf{s}[k] = \mathbf{w}_q^H \mathbf{H}[k] \mathbf{f}_p, \tag{4.3}$$

where H denotes conjugate transpose. The beam-selection is guided by the normalized signal power

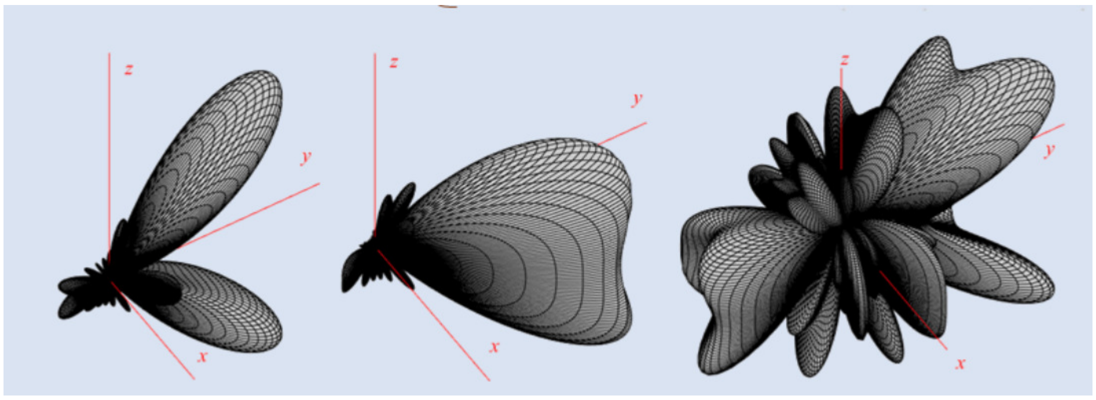
$$y_{(p,q)} = \sum_{k=0}^{K-1} |\mathbf{w}_q^* \mathbf{H}[k] \mathbf{f}_p|^2 \tag{4.4}$$

and the *optimum* beam pair is

$$\widehat{(p,q)} = \arg\max_{(p,q)} \mathbf{y}_{(p,q)}. \tag{4.5}$$

In this work, the goal of beam selection is to recommend a set $\mathcal{B} = \{(p_i, q_i)\}_{i=1}^{M}$ such that $\widehat{(p,q)} \in \mathcal{B}$, which means that our the beam-selection strategy is not required to always pick the optimum beam pair but select a subset of M beams in which $\widehat{(p,q)}$ belongs [22]. This post-processing and data representations allow the formulation of several ML problems for beam-selection. Figure 4.5 shows exmaples of adopted beams.

Figure 4.5: Uniform planar array (UPA) MIMO antenna patterns for three different beam configurations.

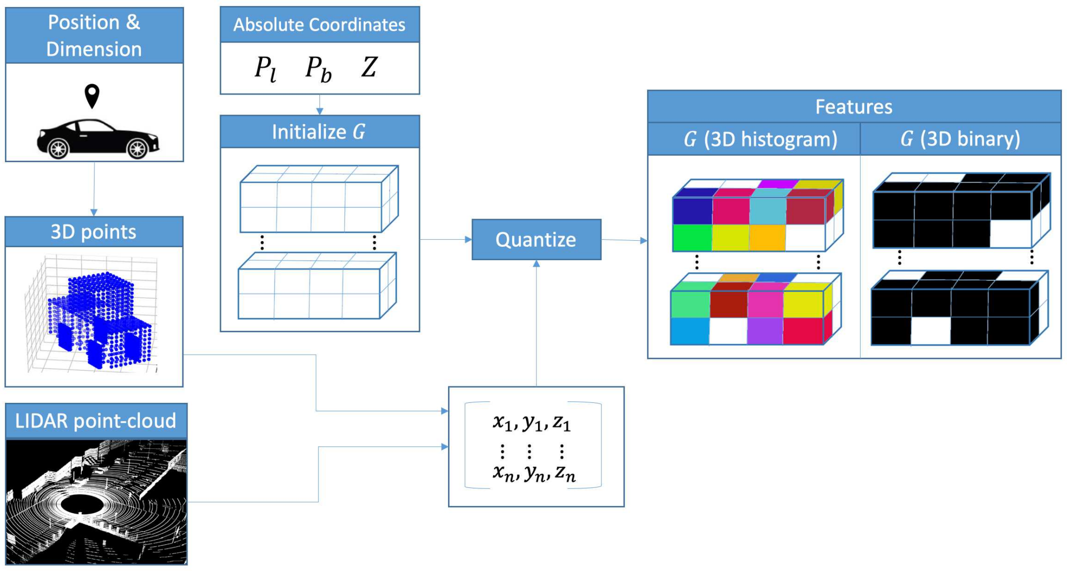


Source: Author.

4.2.2 Position and LIDAR

In this work, it is proposed ML architectures that utilize position and LIDAR features. All ML architectures proposed use input features that consist of a binary 3D array of dimension $\mathbf{G} = 20 \times 200 \times 10$ points. Figure 4.6 depicts the extraction of features for the LIDAR and Position architectures. In the case of the LIDAR architectures, the elements of this arrays are derived from the point cloud $\mathbf{C}$ collected by the LIDAR. For the Position architecture, the array

Figure 4.6: Feature extraction for the Position and LIDAR architectures. The n 3D points correspond to a point-cloud (LIDAR architectures) or were created from information about vehicles (position, size and orientation). The features are represented by a 3D array $\mathbf{G}$.



Source: Author.

with features is created based on the positions and dimensions reported by the vehicles. We first describe the creation of features to the LIDAR architectures.

The point cloud $\mathbf{C}$ provides obstacle distances with respect to the LIDAR position P_r.The absolute positions $\mathbf{A} = \mathbf{C} + P_r$ are quantized into a grid representing the BS coverage zone $Z = (x_z^b, y_z^b, x_z^e, y_z^e, h)$.In summary, the absolute positions are mapped to a grid $\mathbf{G}$ using the provided absolute coordinates.

Then the 3D histogram elements is converted to a binary histogram, where 1 is set whenever the histogram counter was larger than 0 or, otherwise, keeping the 0 value. This convertion is done because binary histogram representation has lower requirement of memory for storage.

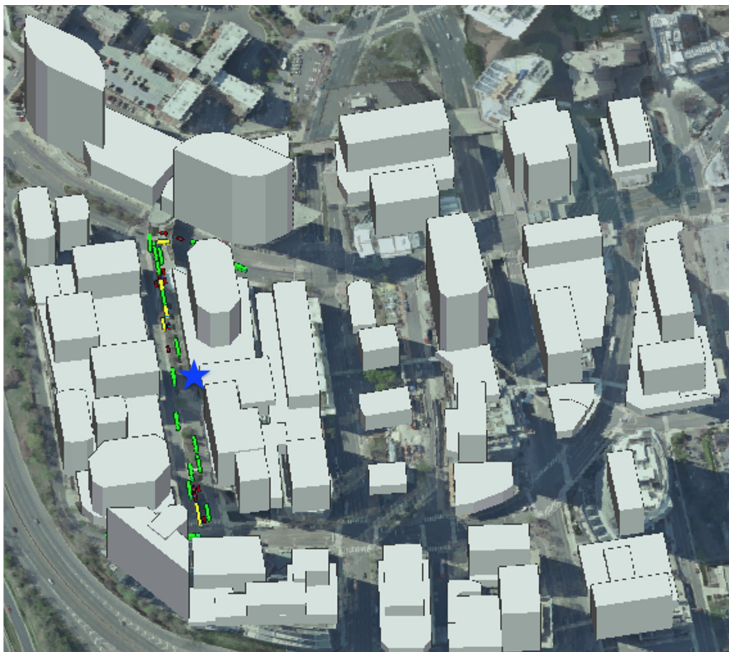
For the Position architecture, a similar process is done. However it utilizes the position, orientation and dimension of the present vehicles (based on the penetration rate), instead of LIDAR data. Given this data, the corresponding elements of $\mathbf{G}$ are represented with 1 where there is the presence of vehicles, and 0 elsewhere.

Chapter 5

Results

This chapter shows scenario and configurations utilized to create the dataset utilized in this work, and the numerical results obtained with deep learning applied to LOS detection and beam-selection. Figure 5.1 depicts the urban canyon 3D scenario utilized, which is part of RWI's examples and represents a region of Rosslyn, Virginia, which was also used e. g. in [23, 24]. The chosen *study area* is a street corresponding to a rectangle of approximately 23×250 m^2.

Figure 5.1: 3D urban canyon scenario used in this work for the InSite ray-tracing simulations.



Source: Author.

Table 5.1: Simulation parameters.

RayMobTime parameters	
Type of simulation	Mobile
Type of mobile object	Realistic
Scenes per episode	1
Number of connected vehicles	10
Sampling period T_{sam} (s)	1
Ray-tracing parameters	
Carrier frequency	60 GHz
BS transmitted power	0 dBm
BS antenna height	4 m
Antenna (Tx and Rx)	Isotropic
Propagation model	X3D
Terrain and city material	ITU concrete 60 GHz
Vehicle material	Metal and glass
Ray spacing (degrees)	1
Num. L of strongest rays	100
Diffuse scattering (DS) model	Lambertian
DS max. reflections ($N_{\text{max}}^{\text{DS}}$)	2
DS coefficients (S)	0.4 (concrete), 0.2 (metal)
Traffic parameters	
Number of lanes	4
Vehicles	car, truck, bus
Lengths, respectively (m)	4.645, 12.5, 9.0
Heights, respectively (m)	1.59, 4.3, 3.2
Probabilities, respectively	0.45, 0.3, 0.25
Average speed (m/s)	8.2
LIDAR parameters	
Model	Velodyne HDL-64E2
Scan distance (m)	120
Scan angle resolution	0.17
Angle of vision	360 degrees

Source: Author.

The *road-side unit* (RSU) or base station is located in the middle of the main street, right where the blue star appears in Figure 5.1. The Table 5.1 contains the most important parameters utilized in the creation of the dataset utilized in this work: RayMobTime parameters were chosen to create as much diversity as possible in the simulations; The ray-tracing parameters were chosen to match an environment of mmWave propagation, and the height of the RSU is 4 meters to obtain more NLOS cases; The vehicles sizes contained on the traffic parameters, were retrieved from real vehicles. The LIDAR parameters utilized are the Blensor default from the model utilized. As can be seen, the simulation is configured to have only one scene per episode. Thus, the connected vehicles are likely to be different in each episode, which means that among this simulation, it will have a great amount of variety of the channels created.

Note that the antennas utilized in the simulation are isotropic instead of expected MIMO UPA antennas. This is done as a strategy to have a high degree of freedom when choosing the antennas. The rays are thrown in all directions, and then the type and number of antennas can be chosen in the post-processing phase where the channel approximation is made.

LOS detection and beam selection are evaluated using the following ML architectures:

***LIDAR distributed* [25]:** each connected vehicle runs a ML algorithm and makes its decision based on its own LIDAR data and position information broadcasted by BS;

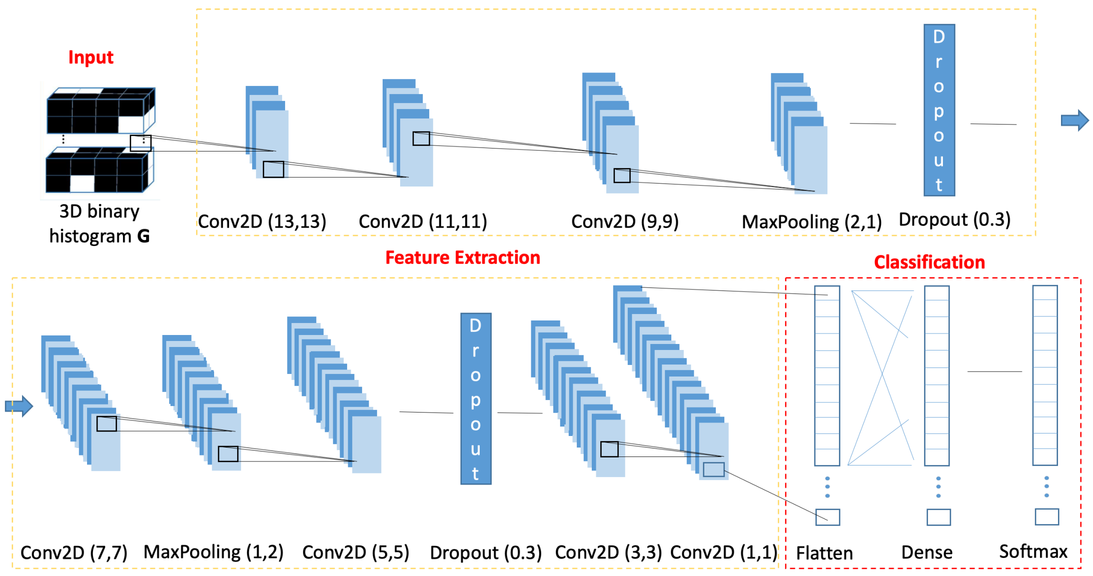
***LIDAR centralized*:** the BS runs the ML algorithm with data sent by connected vehicles and makes its decision based on the fused data;

***LIDAR@BS*:** there is a single LIDAR at the BS, and the BS runs a ML algorithm to makes decision based on its own LIDAR data;

***Position*:** the BS receives the positions of connected vehicles, executes the ML algorithm and decision.

Remember that in this work, the beam-selection is done in two phases: LOS detection and then beam-selection conditioned to the result obtained by the LOS detection, which means that there are two beam-selection DNN models, one specialized in LOS and another in NLOS. Figure 5.2 shows the DNN model utilized in both problems, LOS detection and beam selection. The model is composed of 13 layers trained with Keras' *Adadelta* optimizer, in which 7 of them are 2D *convolutional* layers with decreasing kernel sizes, from 13×13 to 1×1. Regularization methods and *dropout* are utilized to decrease the effects of overfitting. For beam selection, a top-M classification is utilized, then the output layer have a *softmax* activation function and a *categorical cross-entropy* as loss function [12]. For LOS detection, a binary classification is utilized, then the activation function of the output layer and the loss function utilized were *sigmoid* and *binary cross-entropy*, respectively [12].

Figure 5.2: The DNN model utilized in this work.



Source: Author.

5.1 Numerical results

The mmWave data [26] is composed by 3582 snapshots, each with 10 receivers. From this data, it was selected $N_L = 11,691$ LOS and $N_N = 4,048$ NLOS channel examples. For the beam selection experiments, as in [25], were chosen kept only the codevectors that were chosen as $\widehat{(p,q)}$ more than 100 times in the training set. This procedure led to $|\mathcal{C}_t| = 24$ and $|\mathcal{C}_r| = 11$, respectively. Hence, the number of classes for top-M classification is 264.

The beam selection experiments used N_L and N_N examples in the LOS and NLOS evaluations, respectively, while LOS detection used all $N_L + N_N$ examples. For all experiments we created disjoint test and training sets with 20% and 80% of the examples, respectively. The following paragraphs present results for the two LIDAR architectures, distributed and centralized, and for Position using 100% of penetration rate. Note that penetration rate is the percentage of connected vehicles in relation to the total number of vehicles present in the simulation, which means that the architecture using Position with 100% of penetration rate knows the position of every car in the simulation, but do not have additional information about the environment (buildings).

Table 5.2 shows the accuracy (fraction of correct decisions) of the ML systems for LOS detection, the problem of identifying if a receiver is in LOS or NLOS situation. The distributed architecture outperformed the other methods. The results indicate that the quantization using a grid hurts the performance of the LIDAR centralized architecture, given that the adopted representation accumulates the quantization errors into the final feature array. It can also be observed that the area covered by the LIDAR@BS architecture is not big enough, which leads to a decreasing performance. Despite the penetration rate of 100%, the Position architecture have no information about buildings, leading to a lower performance.

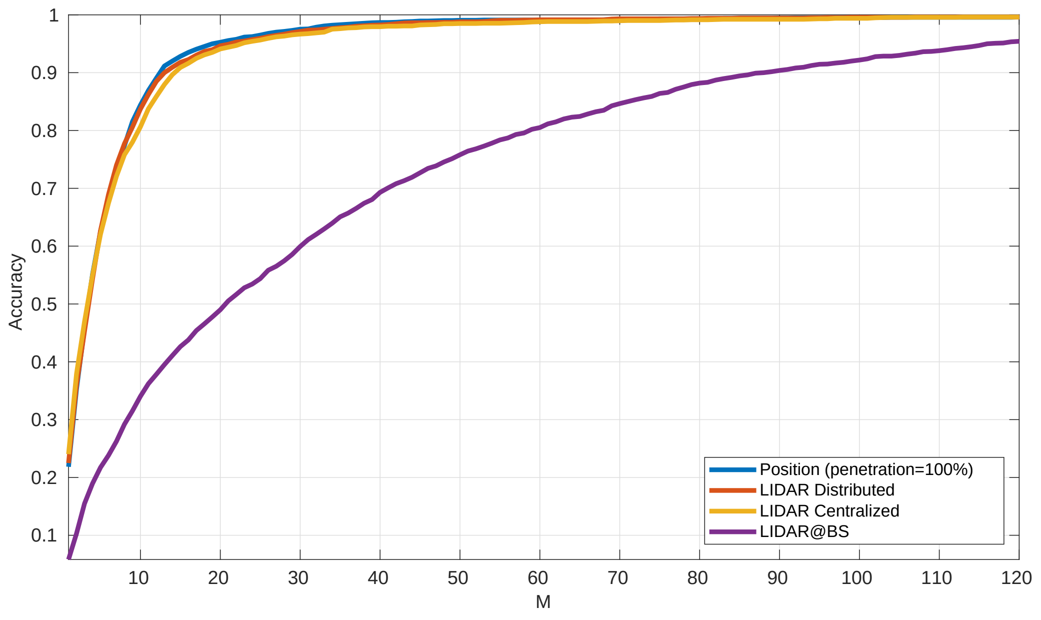
Table 5.2: Simulation results for LOS vs NLOS binary classification.

Architecture	Accuracy
LIDAR distributed	0.912
LIDAR centralized	0.799
Position (penetration=100%)	0.797
LIDAR@BS	0.702

Source: Author.

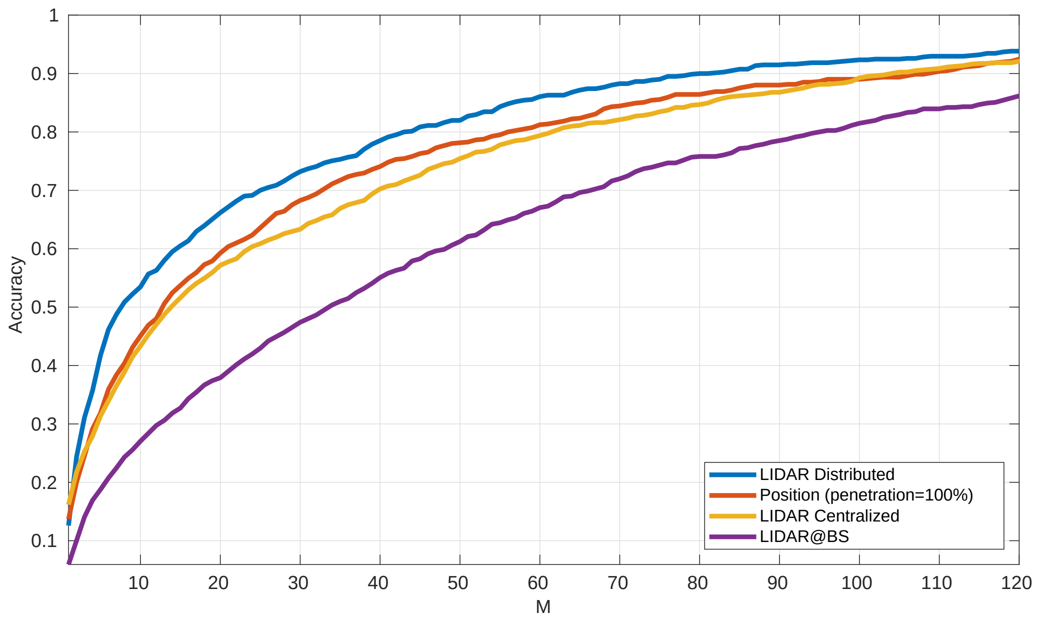
Figure 5.3 and 5.4 show the beam selection top-M accuracy for LOS and NLOS, respectively. In LOS cases, all architectures other than LIDAR@BS perform equivalently, given that the best beam primarily depends on the position of the receiver with respect to the BS. However, in NLOS and LOS detection in Table 5.2, even considering a 100% penetration rate, the distributed outperformed the Position architecture. This may be related to the fact that the Position takes in account only the (connected) vehicles while a LIDAR is capable of incorporating information about the surroundings (buildings, etc.). For the LIDAR@BS architecture, the BS only collects information about the environment close to it. Vehicles far from the BS may be poorly represented or even invisible to the system.

Figure 5.3: LOS top-M classification for beam selection with 264 beam-pairs, for $M = 1, \ldots, 120$.



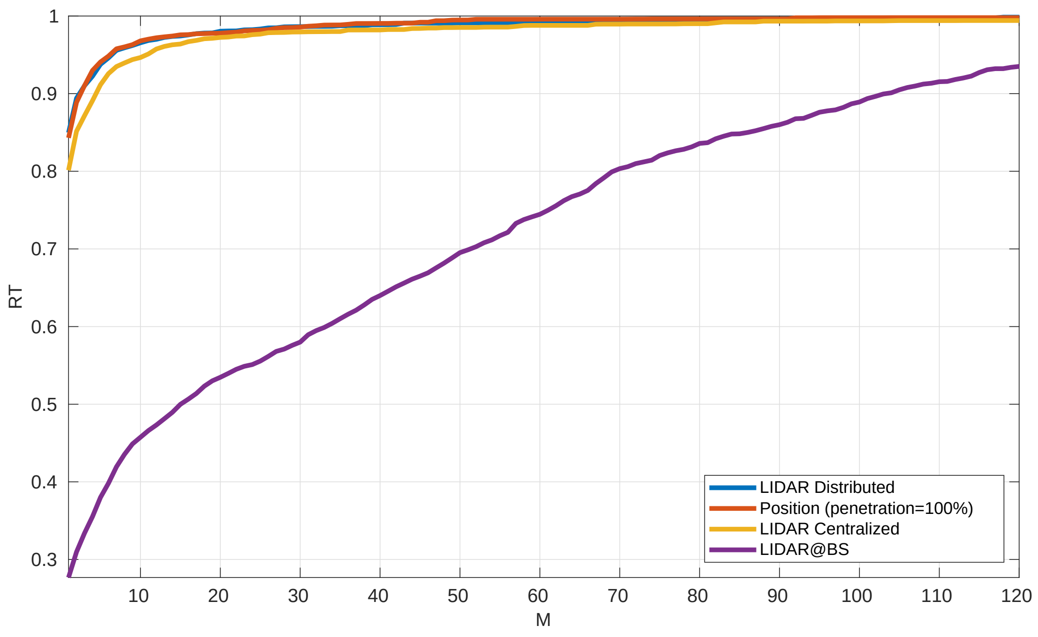
Source: Author.

Figure 5.4: NLOS top-M classification for beam selection with 264 beam-pairs.



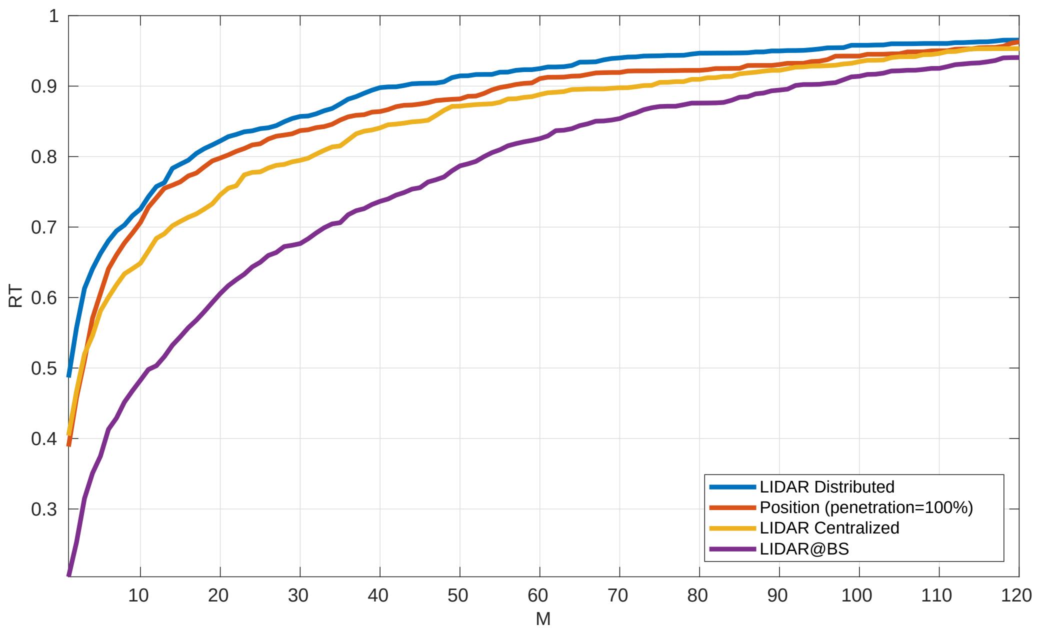
Source: Author.

Figure 5.5: LOS throughput RT for beam selection with 264 beam pairs.



Source: Author.

Figure 5.6: NLOS throughput RT for beam selection with 264 beam pairs.



Source: Author.

Fig. 5.5 and Fig. 5.6 depict the corresponding *achieved throughput ratio* (RT)

$$\text{RT} = \frac{\sum_{i=1}^{N} \log_2(1 + y_{\widetilde{(p,q)}})}{\sum_{i=1}^{N} \log_2(1 + y_{\widehat{(p,q)}})}, \tag{5.1}$$

where N is the number of test examples and $\widetilde{(p,q)}$ is the best beam pair in $\mathcal{B}$. RT represents how close the best beam pair throughput chosen by the DNN is when compared to the maximum throughput obtained among the possible pairs. The overhead from beam selection for $M = 10$, decreases by a factor of 26.4, however the RT indicates a reduction to 64% of the achievable throughput for NLOS in the LIDAR centralized architecture. For NLOS and the Position architecture, RT reaches e. g. 92% for $M = 75$.

Chapter 6

Conclusions

Deep learning is a powerful tool that can be leveraged to recommend mmWave beam pairs based on sensor and location data. In this work, we used deep learning to develop a recommendation engine that used data from LIDAR and position.

The main conclusions are that, in spite of their lower cost, the single LIDAR at BS and position-based solutions are outperformed by distributed LIDAR-based architecture in the two investigated problems. For the NLOS case, the distributed LIDAR-based architecture outperforms the centralized and position-based alternatives. Essentially, the same beam recommendation probability is achieved in the distributed case using a smaller recommendation subset size.

A supposition for the achieved results for the LIDAR at BS architecture is that the most important blockages are the ones closer to the receiver, and not the ones close to the base station. The performance of the centralized data fusion did not bring significant improvements over the distributed processing and may had been limited by the adopted ML technique and coarse resolution used for the 3D grid.

6.1 Publications

The results presented in this work were published in the IEEE International Workshop on Signal Processing Advances in Wireless Communications (SPAWC) in 2019, with the paper named "Position and LIDAR-Aided mmWave Beam Selection using Deep Learning". The methodology of creation of realistic scenarios shown on Section 4.1.1 was published on the Brazilian Symposium on Telecommunications and Signal Processing (SBrT) in 2019, with the paper named "Ray-Tracing 5G Channels from Scenarios with Mobility Control of Vehicles and Pedestrians".

During the pursuit of the masters degree, other publications were done in topics related to deep learning and other to community networks, such as: "Deep Learning in RAT and Modulation Classification with a New Radio Signals Dataset" published on SBrT 2018, "Techno-Economic Studies to Connect Amazon Using Community Telephony" on the Workshop on Information and Communication Technologies (ICT) for Development (WTICp/D) which was part of the Brazilian Symposium on Computer Networks and Distributed Systems (SBRC) in 2017; "Use of Community Mobile Phone Network for Social and Digital Inclusion in the Amazon" on SBrT 2019; and "CELCOM Project: Engineering Practice via Community Networks in Amazon" on the International Journal of Engineering Education (IJEE) in 2019.

6.2 Future Work

Building a model for a specific scenario is great, and can be very powerful. But when dealing a good amount of scenarios, the data scarcity can turn the same problem approached in this work much harder. As not every scenario has massive datasets or in a way to save compute power that's needed to effectively train DNNs. Transfer learning can help solve this, where it is possible to use models trained on large datasets, so that it is possible to either use them directly, or, use the features that they have learned and apply them in other scenario, only training the fully connected dense layers, where the classification takes place.

Future work includes developing and testing alternative feature representations and neural network models for fusing LIDAR, position and other information sources, such as camera images. An alternative architecture could utilize multiple fixed LIDAR at different points of the street, instead of only at BS, so more information about the environment would be sent as input for the DNN.

Another issue that could be settled is that SUMO restricts the minimum time sample to one millisecond. This time sample for telecommunication study is still considerably high. The use of interpolation techniques would cover these gaps between the one millisecond scenes, predicting the rays that may appear or disappear in this small interval, thus creating a time-domain sampling period, more consistent with the demands of studies in the telecommunications area.

Bibliography

[1] C. Jiang, H. Zhang, Y. Ren, Z. Han, K. C. Chen, and L. Hanzo, “Machine Learning Paradigms for Next-Generation Wireless Networks,” vol. 24, no. 2, pp. 98–105, Apr. 2017.

[2] P. V. Klaine, M. A. Imran, O. Onireti, and R. D. Souza, “A Survey of Machine Learning Techniques Applied to Self-Organizing Cellular Networks,” vol. 19, no. 4, pp. 2392–2431, 2017.

[3] J. S. et al, “Natural TTS synthesis by conditioning WaveNet on mel spectrogram predictions,” in *https://arxiv.org/abs/1712.05884*, 2018.

[4] Y. LeCun, Y. Bengio, and G. Hinton, “Deep learning,” *Nature*, vol. 521, pp. 436–444, 2015.

[5] M. Mahmud, M. S. Kaiser, A. Hussain, and S. Vassanelli, “Applications of deep learning and reinforcement learning to biological data,” *IEEE Transactions on Neural Networks and Learning Systems*, vol. 29, no. 6, pp. 2063–2079, June 2018.

[6] N. González-Prelcic, A. Ali, V. Va, and R. W. Heath, “Millimeter-wave communication with out-of-band information,” vol. 55, no. 12, pp. 140–146, Dec. 2017.

[7] J. Kim and A. F. Molisch, “Fast millimeter-wave beam training with receive beamforming,” *Journal of Comm. and Networks*, vol. 16, no. 5, pp. 512–522, Oct. 2014.

[8] J. Choi, V. Va, N. González-Prelcic, R. Daniels, C. R. Bhat, and R. W. Heath, “Millimeter-Wave Vehicular Communication to Support Massive Automotive Sensing,” vol. 54, no. 12, pp. 160–167, Dec. 2016.

[9] P. Zhou, X. Fang, Y. Fang, Y. Long, R. He, and X. Han, “Enhanced Random Access and Beam Training for Millimeter Wave Wireless Local Networks With High User Density,” vol. 16, no. 12, pp. 7760–7773, Dec. 2017.

[10] A. Ali, N. González-Prelcic, and R. W. Heath, "Millimeter wave beam-selection using out-of-band spatial information," vol. 17, no. 2, pp. 1038–1052, Nov. 2017.

[11] Y. Wang, A. Klautau, M. Ribero, M. Narasimha, and R. Heath, "Mmwave vehicular beam training with situational awareness by machine learning," in *Proc. of IEEE GLOBECOM*, Dec. 2018.

[12] A. Géron, *Hands-On Machine Learning with Scikit-Learn and TensorFlow*. O'Reilly Media, 2017.

[13] G. R. MacCartney, H. Yan, S. Sun, and T. S. Rappaport, "A flexible wideband millimeter-wave channel sounder with local area and NLOS to LOS transition measurements," in *Proc. IEEE Int. Conf. Communications (ICC)*, May 2017, pp. 1–7.

[14] F. Fuschini, E. M. Vitucci, M. Barbiroli, G. Falciasecca, and V. Degli-Esposti, "Ray tracing propagation modeling for future small-cell and indoor applications: A review of current techniques," *Radio Sci.*, vol. 50, no. 6, p. 2015RS005659, Jun. 2015. [Online]. Available: http://onlinelibrary.wiley.com/doi/10.1002/2015RS005659/abstract (Accessed 2018-02-04).

[15] T. S. Rappaport, R. W. Heath, R. C. Daniels, and J. N. Murdock, *Millimeter Wave Wireless Communications*. Prentice Hall, 2014.

[16] "Blender - Open source 3D creation. Free to use for any purpose, forever." [Online]. Available: https://www.blender.org (Accessed 2019-08-15).

[17] D. Krajzewicz, J. Erdmann, M. Behrisch, and L. Bieker, "Recent development and applications of SUMO - Simulation of Urban MObility," *International Journal On Advances in Systems and Measurements*, vol. 5, no. 3&4, pp. 128–138, Dec. 2012.

[18] "A Very Comprehensive Tutorial : NN + CNN." [Online]. Available: https://www.kaggle.com/shivamb/a-very-comprehensive-tutorial-nn-cnn (Accessed 2019-08-15).

[19] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[20] G. Hinton, L. Deng, D. Yu, G. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, B. Kingsbury, and T. Sainath, "Deep neural networks for acoustic modeling in speech recognition," *IEEE Signal Processing Magazine*, vol. 29, pp. 82–97, November 2012. [Online]. Available: https://www.microsoft.com/en-us/research/publication/deep-neural-networks-for-acoustic-modeling-in-speech-recognition/

[21] "Architecture of Convolutional Neural Networks (CNNs) demystified." [Online]. Available: https://www.analyticsvidhya.com/blog/2017/06/architecture-of-convolutional-neural-networks-simplified-demystified/ (Accessed 2019-08-15).

[22] V. Va, J. Choi, T. Shimizu, G. Bansal, and R. W. Heath, "Inverse multipath fingerprinting for millimeter wave V2I beam alignment," vol. 67, no. 5, pp. 4042–4058, May 2018.

[23] S.-C. Kim, B. J. Guarino, T. M. Willis, V. Erceg, S. J. Fortune, R. A. Valenzuela, L. W. Thomas, J. Ling, and J. D. Moore, "Radio propagation measurements and prediction using three-dimensional ray tracing in urban environments at 908 MHz and 1.9 GHz," vol. 48, no. 3, pp. 931–946, May 1999.

[24] J. C. Aviles and A. Kouki, "Position-Aided mm-Wave Beam Training Under NLOS Conditions," *IEEE Access*, vol. 4, pp. 8703–8714, 2016.

[25] A. Klautau, N. González-Prelcic, and R. W. Heath, "LIDAR data for deep learning-based mmWave beam-selection," *IEEE Wireless Communications Letters*, pp. 1–1, Feb. 2019.

[26] "RayMobTime dataset." [Online]. Available: https://www.lasse.ufpa.br/raymobtime (Accessed 2019-08-15).

Appendices

Appendix A

Python Orchestrator Algorithms

Algorithm 1: Python orchestrator algorithm for *fixed simulations*.

```
Result: RWI simulation folders with its ray-tracing results
/* The information utilized to create the RWI simulation
   folders is also saved in text file inside that folders
   to be utilized by other simulators */
episode = 0;
while number of simulations is not over do
    Get sumo info;
    if new episode then
        scene = 0;
    end
    if $scene < n\_scenes\_per\_episode$ then /* episode is not over */
        Update 3D objects position;
        Create RWI run folder;
        Execute ray-tracing;
        scene = scene + 1;
        Simulation step(Tsam_scene);
    else
        Simulation step(Tsam_episode);
        episode = episode + 1;
    end
end
```

Algorithm 2: Python orchestrator algorithm for *mobility simulations*.

```
Result: RWI simulation folders with its ray-tracing results
/* The information utilized to create the RWI simulation
   folders is also saved in text file inside that folders
   to be utilized by other simulators */
episode = 0;
while number of simulations is not over do
    Get sumo info;
    if new episode then
        scene = 0;
        while $n\_veh < n\_veh\_antenna$ do
            Simulation step(Tsam_scene);
            Get sumo info;
        end
        Randomly Choose Connected Vehicles(n_veh_antenna);
    end
    if $scene < n\_scenes\_per\_episode$ then /* episode is not over */
        if Vehicles with antenna are present on the simulation then
            Update 3D objects and antennas position;
            Create RWI run folder;
            Execute ray-tracing;
            scene = scene + 1;
            Simulation step(Tsam_scene);
        else
            scene = $Inf$
        end
    else
        Simulation step(Tsam_episode);
        episode = episode + 1;
    end
end
```