FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Radar based static 3D occupancy map

Gaspar Santos Pinheiro

Mestrado em Engenharia Informática e Computação

Supervisor: Aníbal Matos

Second Supervisor: Sara Noronha

July 26, 2022

Radar based static 3D occupancy map

Gaspar Santos Pinheiro

Mestrado em Engenharia Informática e Computação

July 26, 2022

Abstract

Autonomous driving requires detailed vehicle perception of its surroundings. Sensors, such as radars, are used to capture the world around the vehicle and create a model representation of the environment. For this purpose, the static occupancy grid map approach [1] is frequently used, which models the static environment as a two-dimensional grid of random variables that indicate the occupancy state of each cell.

The Perception Development Kit (PDK) is a multi-sensor prototype system that provides sensor gateways for radars and cameras, and an Application Programming Interface (API) for easy integration into the customer's development environment, with the goal of facilitating and accelerating sensor prototyping, developed by Continental Engineering Services (CES) [2]. The Occupancy Grid Processing (OGP) is a software module built on top of the PDK that implements a two-dimensional static environmental model.

However, modeling the three-dimensional world using only two dimensions can lead to a misrepresentation of particular objects, such as bridges, which, in case they are modeled as an obstacle in the middle of the road, could lead to an accident. The proposed solution consists of implementing a three-dimensional occupancy grid map based on the PDK by using *OctoMap*'s [3] Octree implementation and adapting it to use a more complex radar sensor model. Furthermore, the solution implements two probabilistic low-level sensor fusion methods, the Bayesian Probability Theory, and the Dempster-Shafer Evidence Theory.

The implemented system has been evaluated using datasets recorded with a vehicle installed with two radar sensors and two cameras in real-world scenarios, such as two-lane streets with parked cars and a highway. The results show that the presented solution is able to achieve real-time performance, while requiring low-memory consumption and achieving good accuracy, with the calculations of the free space disabled. Furthermore, it was able to properly model a bridge as an overhanging object. The results also demonstrate that the time performance improves linearly, and the memory consumption decreases exponentially, with the Octree's resolution. It is also shown that incrementing the tree depth by just a few units could be an appropriate approach to significantly increase the size of the mapped area, as each unit increment to this parameter doubles this size, while worsening the performance just slightly. The comparison between the Bayesian theory method and the Dempster-Shafer approach showed barely any difference between the two in terms of accuracy, while the former achieves better memory consumption but worse time performance.

Keywords: Environment Modeling, Radar Perception, Robotic Mapping

Resumo

Condução autónoma requer perceção detalhada por parte do veículo em relação ao seu redor. Sensores, como os radares, são usados para capturar o mundo à volta do veículo e criar um modelo representativo do ambiente. Para este propósito, a framework de mapa estático *occupancy grid* [1] é frequentemente usada, que modela o ambiente estático como uma grelha bidimensional de variáveis aleatórias que indicam o estado de ocupação de cada célula.

O Perception Development Kit (PDK) é um sistema de protótipagem de múltiplos sensors que fornece entradas para radares e cameras, e uma Application Programming Interface (API) para uma integração fácil com o ambiente de desenvolvimento do cliente, com o objetivo de facilitar e acelerar a prototipagem de sensores, desenvolvido pela Continental Engineering Services (CES) [2]. O Occupancy Grid Processing (OGP) é um módulo de software construído sobre o PDK que implementa um modelo ambiental bidimensional estático.

Contudo, modelar o mundo tridimensional usando apenas duas dimensões pode levar à falsa representação de certos objetos, como pontes, o que, no caso de serem models como um obstáculo no meio da rua, pode levar a um acidente. A solução proposta consiste em implementar um mapa occupancy grid tridimensional, usando a implementação de Octree do *OctoMap* e adaptando-a para usar um modelo mais complexo do radar. Para além disso, a solução implementa dois métodos probabilisticos de fusão de sensores de baixo nível, o Bayesian Probability Theory e o Dempster-Shafer Evidence Theory.

O sistema implementado foi avaliado usando datasets gravados com um veiculo instalado com dois sensores radar e duas cameras, em cenários reais, como estradas com duas faixas com carros estacionados e uma autoestrada. Os resultados mostram que a solução apresentada é capaz de alcançar performance em tempo real, enquanto requer baixo consumo de memória e atinge boa precisão, com os calculos do espaço livre desativados. Para além disso, foi capaz de modelar corretamente a ponta como um objecto pendente. Os resultados demonstram também que a performance temporal melhora linearmente, e o consumo de memória diminui exponencialmente, com a resolução da Octree. É também mostrado que incrementar a profoundidade da árvore por apenas algumas unidades pode ser uma abordagem apropriada para aumentar significativamente o tamanho da área mapeada, uma vez que por cada unidade incrementada a este parâmetro este tamanho é duplicado, enquanto que a performance piora apenas um pouco. A comparação entre o método Bayesian Theory e a abordagem Dempster-Shafer mostrou quase nenhuma diferença entre os dois em termos de precisão, enquanto que o primeiro alcançou melhor consumo de memória, mas pior desempenho de tempo.

Keywords: Modelação Ambiental, Perceção de Radar, Mapeamento Robótico

Acknowledgements

I want to express my gratitude to everyone who has helped me, not only throughout the last few months while working on this Dissertation but also during the last five years as a student.

First to Continental Engineering Services (CES) for the opportunity to do my Dissertation at such an incredible company, allowing me to increase further my knowledge and experience within the Advanced Driving Assistance Systems (ADAS) industry, which I am so interested in. Specifically, I would like to thank Sara Noronha, Patrick Reichensperger, Sylvain Roy, and Alexander Stoff.

I would also like to thank Professor Aníbal Matos for agreeing to supervise my Dissertation and helping me with meaningful insights.

I am also grateful to the Faculty of Engineering of University of Porto (FEUP) for the last five incredible years as a student.

Last but not least, I am thankful for all the support from my family, especially my parents, who always encourage me to do my best, and to my friends, who have made fun of every challenge we shared over the last five years.

Gaspar Santos Pinheiro

“I’m a greater believer in luck,
and I find the harder I work the more I have of it”

Thomas Jefferson

Contents

List of Figures

List of Tables

Abbreviations and Symbols

ACC	Adaptive Cruise Control
ADAS	Advanced Driving Assistance Systems
ADNN	Average Distance to the Nearest Neighbor
ADS	Automated Driving System
API	Application Programming Interface
BPA	Basic Probability Assignment
CDF	Cumulative Distribution Function
CES	Continental Engineering Services
CPU	Central Processing Unit
DDT	Dynamic Driving Task
DOT	Dynamic Object Tracking
DSET	Dempster-Shafer Evidence Theory
EME	Ego Motion Estimation
EMI	Electromagnetic Interference
FOD	Frame of discernment
FOV	Field of View
GLSL	OpenGL Shading Language
GPU	Graphics Processing Unit
GUI	Graphical User Interface
ICP	Iterative Closest Point
ODD	Operational Design Domain
OGP	Occupancy Grid Processing
PDF	Probability Density Function
PDK	Perception Development Kit
RANSAC	RANdom SAmple Consensus
RCS	Radar Cross-Section
SAE	Society for Automotive Engineers

Chapter 1

Introduction

Throughout the last few decades, transportation safety has improved significantly, however, traffic accidents are still a major cause of death. A recent report from the World Health Organization [4] indicates that road traffic accidents are responsible for approximately 1.3 million deaths each year and between 20 to 50 million more non-fatal injuries. Most of the identified factors that caused these accidents are related to human error.

1.1 Context

During the last decade, the automotive industry has been focused on improving road safety through an increase in the level of driver assistance, by incorporating innovative technology into the vehicle, such as high-end sensors and complex algorithms into sophisticated computational units. The goal is to make driving a safer activity by facilitating the work of the driver, which can be achieved by increasing the driving automation level of the vehicle. According to the Society of Automotive Engineers (SAE), Automated Driving Systems (ADSs) refer to hardware-software systems that are capable of executing Dynamic Driving Tasks (DDTs) in a sustainable way. These are commonly referred to as "autonomous driving" or "self-driving cars". Although SEA advises to not use these terms as it considers them unclear and misleading, in this dissertation, all of these terms are used interchangeably to simplify the text and considering that the scientific community and automotive companies regularly use these more common terms.

With the promise of preventing accidents, reducing emissions, and increasing productivity by reducing the time spent driving, among other benefits, ADS systems are being intensely researched and developed by the automotive industry. By reducing the responsibilities of the driver, human errors can be avoided, which are a major cause of traffic accidents, thus increasing road safety. Furthermore, with fully automated driving, transportation becomes more efficient, both in time and in energy consumption.

Reaching fully automated driving is an open and challenging problem with many obstacles in its path. One of these obstacles comes from ethical dilemmas, where questions are placed regarding how the system should behave in case of an inevitable accident situation or who or what entity should be responsible in such situations. Furthermore, the non-deterministic environment in which ADS systems operate, from variable weather conditions to unpredictable pedestrians, poses a major technical challenge. Many of the current system failures that lead to accidents are related to inaccurate environment perception. An error in the recognition of another vehicle or of a lane line can have catastrophic effects.

1.2 Motivation

As the context given in section 1.1 explains, the improvement and consequent success of ADS systems is an important challenge that can bring significant benefits to society. As also mentioned, one of its major challenges is environment perception. As such, this dissertation focuses on tackling this problem by focusing on the use of radars capable of retrieving 3D perception, which are reliable sensors under any weather condition, to create a three-dimensional model of the environment, efficient both in time and space, allowing for detailed and reliable vehicle perception.

This dissertation will be done with the support of Continental Engineering Services (CES), and its work will be integrated into the Perception Development Kit (PDK), which is a multiple sensor prototype system that provides different hardware and software modules with the goal of facilitating and accelerating sensor prototyping, developed by CES.

1.3 Goals

The main goal of this dissertation is to analyze the required adaptation of the traditional two-dimensional static occupancy grid mapping approach into the three-dimensional space, using data from radar sensors capable of retrieving three-dimensional information from the surroundings of the vehicle. With the intention of achieving the best performance in terms of computation time and memory consumption while maintaining good accuracy, different mapping techniques are compared based on these metrics.

Considering the integration into the software module of the PDK, another goal consists of contributing to the acceleration of radar sensors prototyping.

1.4 Document Structure

After introducing this dissertation's subject in chapter 1, describing the context of the problem, its motivation, the proposed goals, and the document structure, this report is organized as follows:

- Chapter 2 presents the State of the Art, focusing on a review of the current state of autonomous driving research and development, from common methodologies to sensor technology employed and occupancy mapping techniques, such as environmental modeling methods, dynamic environment handling, sensor fusion, sensor modeling, evaluation, and visualization.

- Chapter 3 presents an overview of the problem, by describing it, listing the requirements and assumptions of the system, and presenting the proposed solution.
- Chapter 4 details the implementation regarding the proposed adaptation of the occupancy mapping techniques, such as sensor modeling and sensor fusion, to the presented problem, alongside a description of the evaluation method and its results.
- Chapter 5 presents the implementation details of the different visualization methods which were implemented for visual evaluation of the proposed solution, alongside the evaluation method for the different approaches and the corresponding results.
- Finally, chapter 6 concludes the work of this dissertation, presenting a summary of the achieved results and listing the possibilities for the continuation and improvements of this work.

Chapter 2

State of the Art

To tackle the problem of creating a three-dimensional environmental model, as an important step in Automated Driving Systems (ADSs), based on data captured by radars able to obtain 3D information from the scene, this chapter presents an analysis of what the scientific community has to offer to this work.

The Society for Automotive Engineers (SAE) divides driving automation into six sequential levels [5], as shown in figure 2.1. Level zero corresponds to no driving automation, meaning the driver is responsible for all motion control of the vehicle. At level one, primitive driver assistance systems are integrated, such as Adaptive Cruise Control (ACC), anti-lock braking systems, and stability control. With level two comes advanced assistance systems such as emergency braking or collision avoidance. From levels three to five, an Automated Driving System (ADS) takes over the monitoring task. Level three is conditional automation, meaning that the driver needs to be ready to take over in case of an emergency alert from the vehicle. At level four no human interaction is necessary. Both at levels three and four, the system can only operate in limited operational design domains (ODDs) such as highways. Finally, at level five, full automation is reached that operates under any environmental conditions. Achieving the last two levels of driving automation is a challenging problem that is yet to be solved.

In recent years, the research and development of ADSs have gained more and more focus by the industry due to the accumulated knowledge in vehicle dynamics, advancements in Computer Vision caused by the advent of Deep Learning, and the availability of new sensors, such as the lidar [7]. Furthermore, the success of ADSs could lead to several significant benefits for society due to the increase in transportation efficiency and avoidance of human error, including accident prevention, emissions reduction, congestion mitigation, transportation of the mobility-impaired, and reduction of driving-related stress [8] [9].

Regarding ADSs implementation, common architectures have been established over the years. Two main philosophies are defined for ADSs operation [7]. With an ego-only system, a single self-sufficient vehicle carries out all necessary automated driving operations. On the other hand, in a connected system vehicles may depend on each other or on infrastructure elements. Furthermore, each of these architectures is performed with a modular or an end-

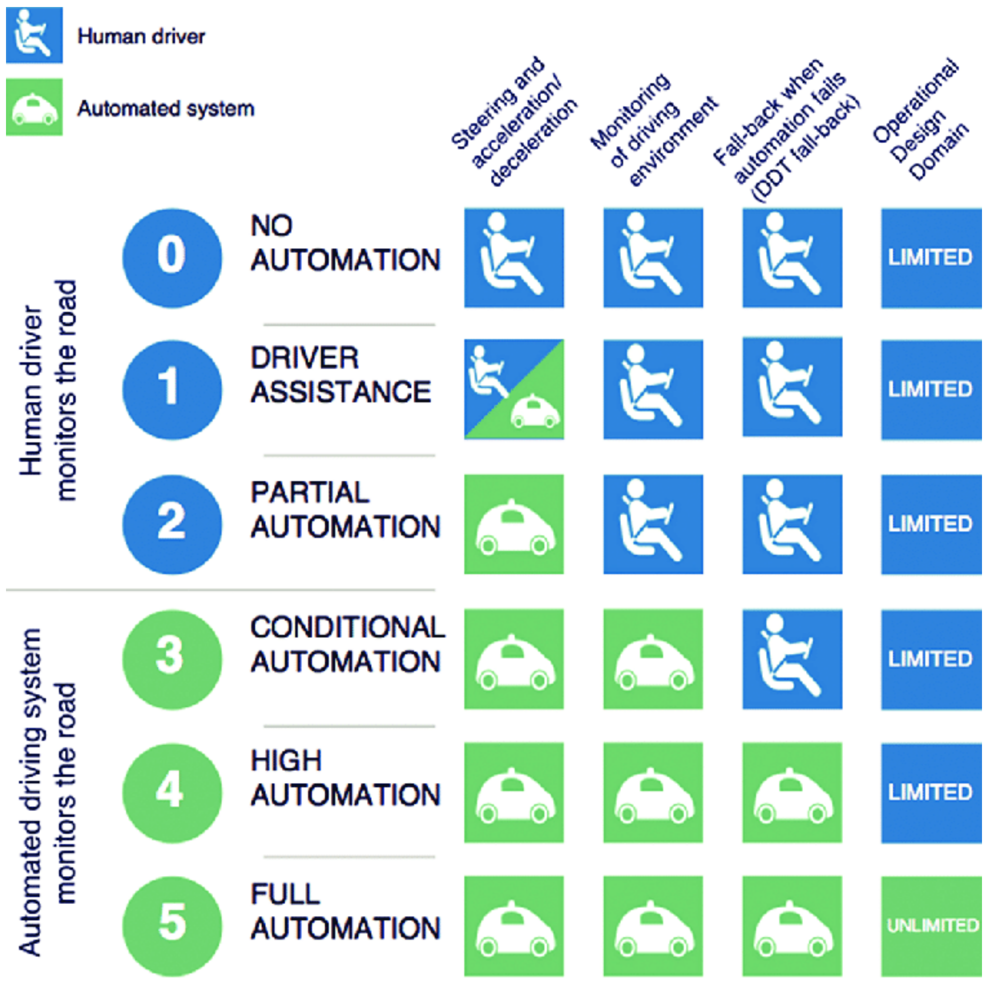

Human driver
Automated system
Steering and acceleration/ deceleration
Monitoring of driving environment
Fall-back when automation fails (DDT fall-back)
Operational Design Domain
Human driver monitors the road
Automated driving system monitors the road
0
NO AUTOMATION
1
DRIVER ASSISTANCE
2
PARTIAL AUTOMATION
3
CONDITIONAL AUTOMATION
4
HIGH AUTOMATION
5
FULL AUTOMATION
LIMITED
LIMITED
LIMITED
LIMITED
LIMITED
UNLIMITED

Figure 2.1: The six levels of driving automation according to SAE. Image from [6].

to-end approach. Recently, the end-to-end approach, where ego-motion is generated directly from sensory inputs, emerged as a result of advancements in Deep Learning [7]. However, most current ADSs applications use the modular approach, where the challenging task of automated driving is divided into sub-tasks structured as a pipeline starting from sensory inputs and ending at actuator outputs. This pipeline usually consists of the following modules: environment perception and modeling, localization and mapping, planning and decision making, and vehicle control [7]. This divide-and-conquer approach facilitates the bigger automated driving task by first tackling the easier-to-solve sub-tasks.

The perception module collects information from the surrounding environment that is essential for safe navigation. It is composed of a variety of sensors that enable the vehicle to understand its surroundings, which is crucial for autonomous driving. Furthermore, it is the first component in the pipeline of the modular approach, thus the correct functionality of the whole system depends on the high accuracy and reliability of this module. Core tasks of the perception module include object detection, semantic segmentation, road and lane detection, and object tracking [7]. For these purposes, the vehicle is usually equipped with data acquisition sensors, such as stereo cameras, lidar, and radar sensors, which are described in more detail and compared in section 2.1. Following the pipeline of the modular approach, the sensory data needs to be interpreted by the subsequent modules to enable important tasks, such as obstacle avoidance in the planning module. However, data obtained directly from sensors, which is just a point cloud, has several limitations, such as the fact that it is unstructured, does not directly represent unoccupied or unknown areas, may require a huge memory space, and does not take into account measurement uncertainty [3]. These problems can be handled by building a model representation of the environment based on sensor data, that is efficient both in runtime and memory usage, while also being probabilistic to handle uncertainty. An overview and comparison of environment modeling methods are presented in section 2.2.

2.1 3D Sensors

Sensors have been part of the vehicle for some decades now, and their role only grows more important as their complexity increases and are able to retrieve more detailed information from the scene around the vehicle. Automotive sensors can be divided into two categories [10]:

- **Proprioceptive**: monitor the internal state of the vehicle, such as Inertial Measurement Units (IMUs), which combine accelerometers, gyroscopes, and sometimes magnetometers to measure acceleration and orientation.

- **Exteroceptive**: perceive the exterior environment, including both static and dynamic objects, such as buildings, traffic lights, and pedestrians.

In particular, **3D sensors** are exteroceptive sensors capable of capturing the three-dimensional structure of the environment. The extra detail obtained from these sensors allows for a more accurate perception module, and, therefore, better decisions can be made during the planning phase of the ADS system. Nowadays, different types of sensors are installed in a vehicle, merging their data through a sensor fusion process, which is explained in more detail in section 2.4, with the goal of achieving redundancy, leading to an increase in robustness and reliability, as it allows sensors to complement each other's weaknesses. This section starts by listing and describing the different types of 3D sensors used in the automotive industry and then presents a comparison between these sensors, which is summarized in table 2.1.

The perception module is composed of passive and active exteroceptive sensors. Passive sensors are devices that measure reflected light emitted from the sun, whereas active sensors have their own source of energy or illumination, sending a pulse to the environment and measuring the reflection of that pulse back to the sensor. The most commonly used sensors for environment perception in the automotive industry are cameras, lidars, and radars. Table 2.1 shows a comparison between these sensors.

Cameras are passive sensors that sense light reflections. For perception, a single or monocular camera or a stereo camera system, i.e. using more than one camera, can be used. Many

ADSs functions are possible with only one camera due to sophisticated algorithms that have emerged through advancements in the field of Computer Vision, such as object tracking and semantic segmentation [7]. Although depth perception is possible with just one camera, as it was done by W. Aguilar [11] using a Convolution Neural Networks (CNNs), its results are very limited, with lower accuracy when compared to the use of a stereo camera system [12], which, on the other hand, computes the distance using the disparity of the objects between the different cameras [13]. This can be a complex process, involving calibration and syncing of the cameras, and retrieval of features from the captured images to then perform a matching algorithm and, finally, obtain the disparity between matching objects [14]. Salman et al. [15], presented a method to compute the distance using trigonometric equations. In [16], a stereo vision system is used, creating disparity maps which are then used to perform three-dimensional reconstruction, creating a point cloud map.

Both radars and lidars are active sensors that work according to the time-of-flight principle, where a signal is transmitted and the time it takes for the reflection to return is measured. The distance to objects, d, is then calculated as half of this time, t, multiplied by the speed of the signal, v, as shown in the following equation:

$$d = \frac{t * v}{2} \tag{2.1}$$

The main difference between the two systems is in the transmitted signal. Whereas lidar emits infrared light waves, radar uses radio waves [7]. This difference has a significant impact on the 3D information obtained with these sensors, since light cannot pass through dense smoke, fog, or dust, making lidars unreliable under certain environmental conditions. On the other hand, radars are much less affected by these effects [17]. However, radars have limited resolution, resulting in lower accuracy maps, when compared to lidars. This impacts the density of the point clouds generated by these sensors, as the ones obtained from the radar can be much less dense compared to a point cloud from a lidar scan. However, in certain scenarios, radars may be able to retrieve detections from objects behind other obstacles, due to the reflectivity of the radio wave. In [18], a quantitative comparison between these two sensors is presented based on Simultaneous Localization and Mapping (SLAM) accuracy. The results show that the radar used is capable of accurately performing SLAM and building a detailed map of the environment, however with less accuracy compared to the one obtained using a lidar. With regards to range, radars can detect objects at longer distances, whereas lidars have much better accuracy under 200 meters [7]. Size and cost are other important metrics to consider about these sensors. Radars are cheaper and have a small size, being able to fit inside side-mirrors. On the other hand, lidars are expensive and have a large size. However, in recent years both the size and cost of lidar sensors have been decreasing [19].

When comparing the stereo camera system with radar and lidar, stereo cameras have the downside of being affected by illumination and have a lower range and accuracy. However, they have a reduced cost and their size is smaller in relation to lidar [7].

Table 2.1 presents a summary of the comparison made between the mentioned 3D sensors, describing for each one whether they are affected by illumination and/or weather, and their relative range, accuracy, size, and cost, as shown in [7].

Table 2.1: 3D Sensors Comparison

3D Sensor	**Affected by Illumination**	**Affected by Weather**	**Range**	**Accuracy**	**Size**	**Cost**
Stereo Camera	✓	✓	medium (< 100m)	low	medium	low
Lidar	-	✓	medium (< 200m)	high	large	high
Radar	-	-	high	medium	small	medium

As can be observed in Table 2.1, each sensor is unique, offering its own set of advantages and disadvantages. This is why the data from these different sensors are usually combined through a sensor fusion process, explained in section 2.4. However, it is important to note two other unique measuring capabilities of the radar sensor which offer an advantage over the other type of sensors and, therefore, are relevant in the context of this dissertation:

- **Doppler velocity**. Radars are capable of detecting the range rate of targets in the radial direction, based on the Doppler effect [20]: when a moving object reflects the radio wave sent by the radar, its frequency changes. By measuring the difference between the frequencies of the transmitted and the received radio waves, the velocity of the moving object can be measured.

- **Radar Cross-Section** (RCS). As explained before in this section, a radar detects the distance to targets by measuring the time it takes for the radar energy to be reflected from the object back to the radar. However, due to some properties of the detected object, such as its size and material, the amount of reflected energy may vary. Considering this, the RCS is a measure of how detectable an object is by the radar, based on different factors, such as the aforementioned object properties and the incident or reflected angles [21].

2.2 Environment Modeling

As the data from 3D sensors is unstructured and requires huge memory space, creating a model that represents the free and occupied areas of the environment in a space-efficient way while enabling real-time measurements insertions, allows the subsequent ADS system modules to more easily obtain relevant and accurate information of the vehicle's surroundings. Furthermore, by representing unoccupied areas correctly, the model serves as a map to the vehicle, which enables key autonomous driving capabilities such as obstacle avoidance. Localization is another fundamental task of an ADS system that can be performed using a pre-built map of the environment. This function consists of finding ego-position relative to a reference frame in an environment [22].

The occupancy grid map, first introduced by A. Elfes et al. [23] [24] using sonar sensors, is the most commonly used paradigm to represent the environment in the context of robotic mapping. With this approach a discretization of the world is performed, resulting in a grid of equal-sized cells, where each cell is described by a value following a binomial distribution that represents the probability of it being occupied. In most applications, a threshold is applied to this probability to determine the actual occupancy of the cell, so that the map can then be used for ADS applications such as obstacle detection, as shown in figure 2.2.

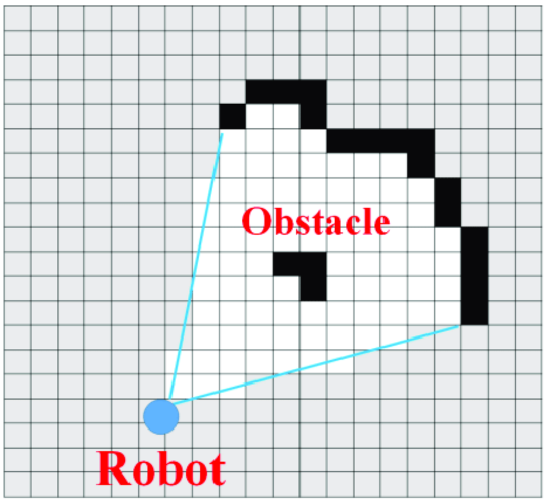


Figure 2.2: Traditional 2D occupancy grid map, where obstacles are represented as a set of occupied cells (shaded black) and unoccupied space as free cells (white). Image from [6].

This approach poses several benefits regarding its probabilistic representation, since it enables the handling of uncertainty that comes from sensor noise or dynamic objects in the environment, facilitating sensor modeling and sensor fusion, both described in more detail in sections 2.5 and 2.4, respectively. However, the two-dimensionality of this approach does not allow for the correct representation of over-hanging objects, such as bridges. Therefore, it is necessary to use a different technique to represent the environment, which is able to integrate the benefits of the occupancy grid, while also adding the modeling capability of over-hanging objects. Furthermore, any ADS system which uses 3D sensors should also implement an environment model with a three-dimensional structure, so that important information on the scene structure is not lost.

This section describes and compares methods that have been used in the literature to model the environment in a three-dimensional way.

A **point cloud** is the most low-level representation of an environment, consisting only of a set of points in space, where each one has its own set of 3D coordinates, either using a Cartesian coordinate system (X, Y, Z) or a spherical one (radial distance, polar angle, azimuth angle). Some sensors, such as lidars [25] or radars [26], directly produce 3D point clouds. This model can also be reconstructed from raw data of other sensors, like an RGB-D camera [27] or a stereo camera [16]. This method has been used in several systems for SLAM purposes, such as in [28] and [29]. Although this model offers an advantage in high accuracy and update-time efficiency, it is limited by the huge memory requirement, which only increases with each new measurement. It is also unable to represent free space and unmapped areas and does not deal with sensor noise.

A common approach to overcome the limitations of point clouds consists of performing a discretization of the world into equal-sized cubic volumes, called **voxels**, creating a **three-dimensional grid** [30]. By storing an occupancy probability on each voxel it is possible to correctly differentiate between free and occupied space by thresholding these values. With this probabilistic approach, the uncertainty that comes from sensor noise or detections which result from dynamic changes in the environment, like dynamic objects, is handled. However, one of the major drawbacks of fixed grid models, such as this one, is that the dimensions of the area to be mapped have to be known beforehand so that the required memory is properly allocated, which in cases of high-resolution mapping of large areas requires huge memory consumption. This model is also unable to correctly represent unmapped areas, which may be important for some autonomous driving systems.

To overcome the limitations of fixed three-dimensional occupancy grids, an **Octree** [31] is often used, which represents the occupancy grid as a hierarchical tree data structure, where each node corresponds to the space contained in a voxel, which stores its occupancy state, as free or as occupied. This volume is recursively subdivided into eight sub-volumes until a minimum voxel size is achieved, which determines the resolution of the model. Figure 2.3 shows a visual representation of the correspondence between a three-dimensional occupancy grid and an Octree model.

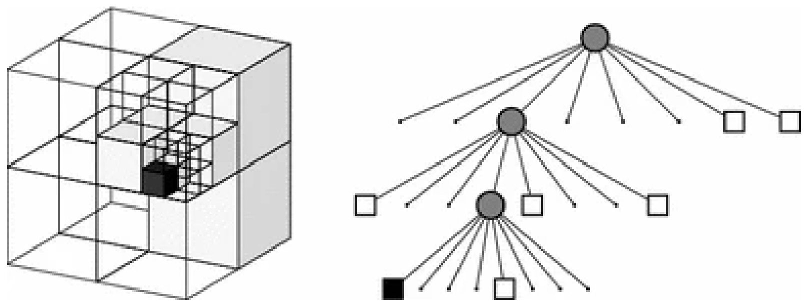


Figure 2.3: Visual representation of a 3D occupancy grid on the left and the corresponding Octree, on the right, storing free (shaded white) and occupied (black) voxels. Image from [3].

This structure results in a more memory-efficient model, since, at any tree depth, the further division of a node into eight children nodes does not have to occur if all of them have the same occupancy state. Furthermore, with each new sensor measurement insertion into the Octree, only the corresponding affected voxels are created, through dynamic memory allocation, which further increases memory consumption efficiency while also allowing for implicit representation of unmapped areas. However, in embedded systems, only static memory allocation is appropriate, which limits the application of this model. The requirement of before-hand knowledge of the mapped area size is also handled with this model, however, it still imposes a maximum limit, lim, on each map dimension, depending on the set map resolution, res, and maximum tree depth, $depth$, according to the following equation:

$$lim = res.2^{depth} \tag{2.2}$$

OctoMap [3] is a probabilistic occupancy grid mapping framework that uses an Octree representation, focusing on offering a flexible mapping tool regarding the mapped area dimensions and resolution. This tool also deals with sensor noise through a probability occupancy estimation for each voxel and uses compression methods for a more memory-efficient model. It provides an Octree data structure with an easy-to-use interface for point cloud insertion, map queries, tree traversal, ray-casting, and more. Regarding the probabilistic map update method, which is described in more detail in section 2.4, *OctoMap* uses a Bayesian approach, based on occupancy grid mapping introduced in [23]. Even though this tool can be used with any kind of sensor, as long as a corresponding sensor model is provided, its base implementation is based on the laser range sensor, as it was used for this tool's experiments. Therefore, it uses a beam-based inverse sensor model, explained in more detail in section 2.4, in which the endpoints of each sensor measurement are considered to contain an obstacle and the space between the sensor and the endpoint is updated as being free. After each node update, a clamping method is applied to limit the minimum and maximum occupancy probability values of each voxel. This approach has two main advantages: first, it handles the problem of an overconfident model, and second, it enhances the memory efficiency of the model, considering that in case all eight children of a node have reached a stable occupancy probability i.e. a value below the minimum threshold or above the maximum threshold, then the node can be pruned and all eight children are removed from memory. To determine the free voxels along the sensor beam, *OctoMap* implements a ray-casting mechanism, based on the three-dimensional variant of the Bresenham algorithm [32] to approximate the beam. Another interesting feature of this tool is multi-resolution queries. During the insertion of a set of measurements into the map structure, besides the probabilistic update of the leaf nodes, inner nodes may also be updated. This way, varied applications with different resolution requirements are able to traverse the tree at different maximum depths, obtaining a coarser or denser model, according to the application's needs. To showcase the compactness of this tool's Octree implementation, the memory consumption mem_{Octree} results presented in this paper were calculated as shown in equation 2.3, based on the number of inner nodes n_{inner} and on the number of leaf nodes n_{leaf}, and considering an inner node occupies $40B$ and a leaf node consumes $8B$, assuming a 32-bit architecture.

$$mem_{Octree} = n_{inner} \times 40B + n_{leaf} \times 8B \tag{2.3}$$

Due to its easy-of-use Octree structure, OctoMap has been used in a lot of systems. However, it still has some limitations, primarily related to its time efficiency. Considering this, there have been proposals for improvements to the OctoMap tool.

In [33], **UFOMap** is presented as an extension to OctoMap that is more memory efficient, provides methods for faster insertions into the Octree and faster queries based on occupancy state, while also handling the limitation of implicit modeling of free space. To insert a point cloud into the Octree structure, UFOMap presents three different methods, each more time-efficient than the previous. The first two methods, which are also available in OctoMap, consist of increasing the occupancy probability of the voxels corresponding to each point in the point cloud and decreasing the occupancy of the voxels between the sensor origin and the endpoint, through ray-casting. The difference between both methods is that the second one performs a discretization of the point cloud before the ray-casting process, resulting in a lower number of rays being cast. The third approach focuses on improving time efficiency with the downside of decreasing the accuracy of free areas computation, by performing ray-casting at higher resolution values so that the number of casts is further reduced. Furthermore, UFOMap also provides fast Octree iterators which allow for the specification of which type of nodes to traverse so that, for example, only occupied voxels are returned.

The work of L. Zhang et al. [34] focuses on reducing the time it takes to build the map, more specifically, by doing a more effective computation of empty voxels and by reducing the insertion time of new voxels. The improvement is based on the Fast Line Rasterization Algorithm [35]. Thread pools are also used to reduce thread creation. The experimental results show a reduction of more than 50% in map building process time.

J. Chen et al. [36] focus on improving the computational efficiency of map update and on occupancy inquiry while maintaining the memory space compactness of *OctoMap*. The paper introduces a ray-casting method that uses map sparsity with early termination for updating the map. To improve occupancy checking specified by a range on each axis dimension, a divide-and-conquer range inquiry method is presented. The experimental results show a significant decrease in computation time in the map update process and that the proposed ray-casting process is more efficient compared to the one used in *OctoMap*.

To overcome the complexity and the huge memory requirement of three-dimensional grid maps, elevation maps are often used to model the environment. This is a 2.5D approach, in the sense that a two-dimensional grid is used, where each cell also contains the height information of the terrain, alongside the occupancy probability [37]. In [38], an elevation map is used to represent the environment. The height of each cell is determined based on the high surface curvatures of the points in the point cloud. This approach highly reduces the memory space required, while also being time-efficient, considering the calculation of the height of each cell does not require much process time. However, this representation is unfeasible in certain situations since vertical or overhanging objects, such as bridges, cannot be represented appropriately. To overcome this problem, an extension to the elevation map is proposed in [39], where the aforementioned objects are properly classified based on the Iterative Closest Point (ICP) algorithm [40]. In [41], the Multi-Level Surface Map is proposed, consisting of another extension on top of the elevation map method proposed in [39], allowing for compactly representing of multiple surfaces in the environment. With this representation, each cell of the discrete grid stores a list of surfaces, and intervals are used to represent vertical structures.

Figure 2.4 shows a visual comparison between the following modeling methods: point cloud, elevation map, multi-level surface map, and the *OctoMap*'s Octree, obtained from a laser range scan. The Octree model offers a detailed representation of the environment and allows for the modeling of free space, which is not shown in the figure to make it more clear.

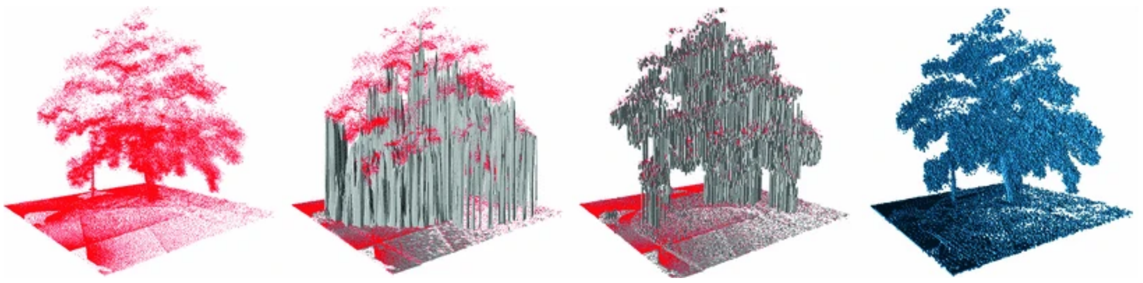


Figure 2.4: Visual representation of the following modeling methods (from left to right): point cloud, elevation map, multi-level surface map, and *OctoMap*'s Octree, obtained from a laser range scan of a tree. Image from [3].

Table 2.2 sums up the previously made comparison between the different environment modeling methods in relation to their accuracy or level of detail, space and time-efficiencies, and whether they are able to represent unmapped areas.

Table 2.2: 3D Environment Modeling Methods Comparison

Model	**Accuracy**	**Space efficiency**	**Time efficiency**	**Unmapped Areas Representation**
Point Cloud	very high	very low	very high	unable
3D Occupancy Grid	high	low	medium	unable
Octree	high	medium	medium	able
Elevation Map	low	very high	high	unable
MLS Map	medium	high	high	unable

2.3 Dynamic Environments

The modeling approaches described in the previous section assume the environment is static. Therefore it is crucial that sensor measurements corresponding to dynamic obstacles are properly detected and filtered out so that noise from these objects is not added to the static environment model. With this in mind, this section describes recent research efforts which handle dynamic obstacles in the environment.

Any world environment is composed of several different objects that can be classified as static, if their pose does not change over time, or as dynamic otherwise. Buildings, bridges, and guardrails are examples of static objects. On the other hand, dynamic obstacles include walking pedestrians and moving vehicles [42]. Furthermore, this kind of classification can be extended to the whole environment, where it is static if only the actions of the robot itself are capable of modifying it or if dynamic objects are ignored. However, in case the vehicle's surroundings change over time or if other agents are assumed to co-exist on it, the environment is classified as dynamic. Most real-world scenarios in which an autonomous vehicle would navigate are complex dynamic environments. For example, in an urban area, an ADS system is faced with moving pedestrians and vehicles. However, if detections of moving objects are ignored, the environment could be treated as static. This is relevant for certain robotic applications in which the information on the location of moving objects is not necessary, such as localization, since information about static landmarks in the environment may be sufficient [43]. For example, most SLAM applications assume the environment is static [44]. On the other hand, modeling dynamic objects correctly is a very complex task, requiring tracking of their poses on each new sensor scan [45]. Detection And Tracking of Moving Objects (DATMO) [46] is a well-studied problem in the field of robotics that handles the dynamic parts of the environment, with the goal of predicting their future poses and, therefore, increasing safety through a more reliable environment model.

As mentioned in section 2.1, radars have a unique advantage over other sensors due to their capability of measuring the radial velocity of detected objects based on the Doppler effect. This property can be specially used to differentiate static and dynamic targets. M. Slutsky et al. [47] perform occupancy grid mapping using radars, where two separate grids are implemented for static and dynamic detections. Considering all detections have a non-zero Doppler velocity when the vehicle is moving, the separation process between the two types of measurements takes into account the current vehicle ego-motion model, specifically, its velocity. Therefore, a detected target is considered dynamic if the difference between these two velocities is above a certain threshold. Otherwise, the detection is classified as static.

2.4 Probabilistic Sensor Fusion

With the goal of increasing the robustness and reliability of the perception module, there has been, in recent years, an effort in research to combine sensory inputs from different sources with the goal of taking advantage of the properties of each kind of sensor. This combination process is called sensor fusion. In the context of robotics, it is often performed in a probabilistic way so that uncertainty that comes from sensor noise or dynamic objects can be taken into account. As stated by Khaleghi et al. [48], which present a review of the main challenges and methods that are currently used for sensor fusion, one of the main difficulties comes from resolving conflicting information from different sources, requiring a common framework that is able to represent data from multiple sensors with varied characteristics.

This section does a review of the most frequently used methods for probabilistic sensor fusion in the context of the perception module for ADS systems.

Regarding its architecture, sensor fusion can be performed at different levels [49]:

- **High-Level**: With this approach, high-level structures of the scenery are used to represent the sensor data, and fusion is performed using these structures, as was done by Perrollaz et al [50], where a laser scanner was used to detect and track obstacles, while a vision system later performed validation on the tracks. In, a [51], millimeter-wave radar and a monocular camera are fused for obstacle detection and tracking, where high-level representations of the tracks are obtained from each sensor, and then only those that match between the two are accepted.

- **Low-Level**: In low-level fusion approaches, the final decision on the map state, including obstacles positions, is performed at the very last phase, meaning that no information is lost during the intermediate stage in which fusion occurs. With this approach, the fusion is performed with low-level representations of the sensory data.

The remanding part of this section focuses on low-level sensor fusion and the different techniques used alongside it, as this is the most commonly used method for robotic mapping.

Regarding the aforementioned common framework capable of representing data from different types of sensors, the occupancy grid [23] [24], aforementioned in section 2.2, is the most common approach in the context of robotic mapping, which performs low-level sensor fusion. This method represents the environment as a grid of equal-sized cells, which store an occupancy probability. With this method, heterogeneous sensor data can be represented in a homogeneous way. Furthermore, this approach facilitates the integration of new sensor measurements, by allowing the translation of sensor beams into a probability distribution over the grid [49], as defined by a sensor model, which is explained in detail in section 2.5.

There are many examples in the literature which apply low-level sensor fusion in an occupancy grid. Kumar et al. [52] were able to obtain a 3D occupancy map by fusing data from a stereo camera set and an infrared sensor, resulting in a smaller global error compared to when a single sensor is used. Paromtchick et al. [53] implement an object detection system in which a stereo camera and a lidar are fused through an occupancy grid.

As the aforementioned examples demonstrate, performing low-level sensor fusion on an occupancy grid is a very versatile approach as it facilitates the combination of different types of sensors. Furthermore, it reduces the problem of determining the probabilities of each cell based on each sensor measurement. Two main approaches are often used in the literature to tackle this problem: the **Bayesian Probability Theory** and the **Dempster-Shafer Evidence Theory** (DSET). The former, which is the most common approach, is based on the Bayes' theorem and was used alongside the occupancy grid paradigm when it was first introduced for map building by A. Elfes et al. [23] [24]. To overcome some limitations of this approach, a generalization was proposed by Dempster [54] and then extended by Shafer [55], resulting in the Dempster-Shafer Evidence Theory.

Using the Bayesian probability theory approach for grid mapping consists of calculating the posterior occupancy probabilities of the map m given the previously obtained sensor measurements $z_{1:t}$ and vehicle poses $x_{1:t}$ [49]:

$$p(m|z_{1:t}, x_{1:t}) \tag{2.4}$$

Most approaches in the literature make a series of assumptions about the environment to facilitate the application of this theory to robotic mapping, namely:

- The map is static, i.e. it does not change over time.
- Each map state update depends only on its previous state and not on the sensor measurements or vehicle poses.
- The current vehicle pose x_t depends only on the pose at the previous time step x_{t-1}.
- There is conditional independence between measurements at different time steps:

  $$p(z_t, z_{t-1}|m) = p(z_t|m).p(z_{t-1}|m) \tag{2.5}$$

  where $p(z_t|m)$ is a forward sensor model, explained in more detail in section 2.5, which represents the probability of obtaining a sensor measurement given the current map state.
- The occupancy probability of each cell in the grid is independent of all other cells.

With the last assumption, the problem is reduced to calculating the occupancy probability of each cell $p(m_{ij}|z_{1:t}, x_{1:t})$, which is solved using the Binary Bayes Filter algorithm [56]. With this approach each cell in the grid stores a *log-odds* ratio $l(m_{ij})$ representation, which is derived from $p(m_{ij}|z_{1:t}, x_{1:t})$, as demonstrated in appendix A.

$$l(m_{ij}) = \log \frac{p(m_{ij}|z_{1:t}, x_{1:t})}{1 - p(m_{ij}|z_{1:t}, x_{1:t})} \tag{2.6}$$

This representation allows the use of a simple iterative formula to update a cell based only on its previous value:

$$l_t(m_{ij}) = l_{t-1}(m_{ij}) + \log \frac{p(m_{ij}|z_t, x_t)}{1 - p(m_{ij}|z_t, x_t)} - \log \frac{p(m_{ij})}{1 - p(m_{ij})} \tag{2.7}$$

where the first term $l_{t-1}(m_{ij})$ is the previous cell state. The second term is the *log-odds* representation of the probability distribution $p(m_{ij}|z_t, x_t)$, which is an inverse sensor model, explained in more detail in section 2.5, representing the probability of cell m_{ij} being occupied given the current sensor measurement z_t and vehicle pose x_t. The last term in equation 2.4 is the initial cell occupancy, which is usually $p(m_{ij}) = 0.5$ to model the unknown state.

Although the application of the Bayesian probability theory to mapping offers a time-efficient map update method, it is limited by its inability to explicitly represent ignorance and to handle conflicting information well. To overcome these limitations, the Dempster-Shafer evidence theory is often used. This approach consists on a generalization of the Bayesian theory, allowing for the distribution of probabilities to the union of events, not only to the events themselves. To explain this method properly, it is important to first define some of its concepts in the context of occupancy grid mapping, as explained by C. Fernández [49]:

- Frame of discernment (FOD), Θ: it is a finite set of all possible states of a random variable. The power set of Θ, described by 2^{Θ}, is the set of all possible subsets of the FOD. In the problem of occupancy grid mapping, this approach, as it is done in the Bayesian method, assumes each cell has a binary state: empty (E) or occupied (O). Therefore, the frame of discernment, Θ, and its power set, 2^{Θ}, are defined as follows:

$$\Theta = \{E, O\} \quad \text{and} \quad 2^{\Theta} = \{\emptyset, E, O, \{E, O\}\} \tag{2.8}$$

  where the sets $\emptyset$ and $\{E, O\}$ represent the *null* set and the *ignorance* set, respectively. The latter set allows for an explicit representation of the unknown state, giving an advantage to this approach compared to the Bayesian theory.

- Basic probability assignment (BPA) function: it is equivalent to the probability value of the classic theory assigned to each element of 2^{Θ}, the set of all possible subsets. In the context of occupancy mapping, this value is directly related to the cell's occupancy and the defined sensor model, defined as follows:

$$m : 2^{\Theta} \rightarrow [0, 1] \tag{2.9}$$

  The BPA function of any subset $A \in 2^{\Theta}$, $m(A)$, which is often called in the literature as *mass of A* or as *A's basic belief number*, must fulfill certain conditions:

$$\sum_{A \in 2^{\Theta}} m(A) = 1 \quad \text{and} \quad m(\emptyset) = 0 \tag{2.10}$$

  In the context of occupancy mapping, the mass values for the empty and occupied states are initially assigned a value of zero to represent that no sensor measurement has yet been incorporated in the grid.

$$m_{ij}(E) = m_{ij}(O) = 0 \quad \forall i, j \tag{2.11}$$

This initialization approach makes it so $m_{ij}(\{E, O\}) = 1$, which explicitly represents complete ignorance about the state of the cell.

- The belief of a subset $X \in 2^\Theta$ is a value representing the degree to which X is believed to be true, accounting for all the evidence that supports it:

$$Bel(X) = \sum_{A \subseteq X} m(A) \tag{2.12}$$

- The Plausibility of a subset $X \in 2^\Theta$ represents the degree to which X is believed not to be false, accounting for all the evidence that does not provide knowledge about X:

$$Pl(X) = 1 - \sum_{A \cap X = \emptyset; A \in 2^\Theta} m(A) \tag{2.13}$$

As noted by A. Hogger [57], these two functions, $Bel(X)$ and $Pl(X)$, define the lower and upper limits of the probability of X in the classical sense:

$$Bel(X) \leq P(X) \leq Pl(X) \tag{2.14}$$

- The Dempster-Shafer combination formula fuses the belief functions of two sources of information, $m_1(X)$ and $m_2(X)$ about a proposition $X \in 2^\Theta$ into a combined value $m_{12}(X)$, represented by operator $\oplus$, as shown in equation 2.15:

$$m_{12}(X) = (m_1 \oplus m_2)(X) = \begin{cases} \dfrac{(m_1 \cap m_2)(X)}{1 - (m_1 \cap m_2)(\emptyset)}, & X \neq \emptyset \\ 0, & X = \emptyset \end{cases} \tag{2.15}$$

This combination formula can be used to update the occupancy grid using a single sensor measurement, the current map m^t and the inverse sensor model, m^s, updating the BPA for the empty (E) and occupied (O) states:

$$m^{t+1}(O) = \frac{m^t(O)m^s(O) + m^t(O)m^s(\{E, O\}) + m^t(\{E, O\})m^s(O)}{1 - m^t(E)m^s(O) - m^t(O)m^s(E)} \tag{2.16}$$

$$m^{t+1}(E) = \frac{m^t(E)m^s(E) + m^t(E)m^s(\{E, O\}) + m^t(\{E, O\})m^s(E)}{1 - m^t(E)m^s(O) - m^t(O)m^s(E)} \tag{2.17}$$

With this approach an inverse sensor model can be used to model independently both probabilities of a cell being empty of occupied, $m^s(E)$ and $m^s(O)$. The value of $m^s(\{E, O\})$ can be obtained by applying equation 2.11:

$$m^s(\{E, O\}) = 1 - m^s(O) - m^s(E) \tag{2.18}$$

- A classical probability measure can be calculated from a mass function using the *pignistic* *transformation* $BetP$, as shown by Smets [58]:

$$BetP(A) = \sum_{X \in 2^{\Theta}} m(X).\frac{|A \cap B|}{|X|} \tag{2.19}$$

In the context of occupancy mapping, the classical probability of occupancy $P(O)$ can be obtained with this formula:

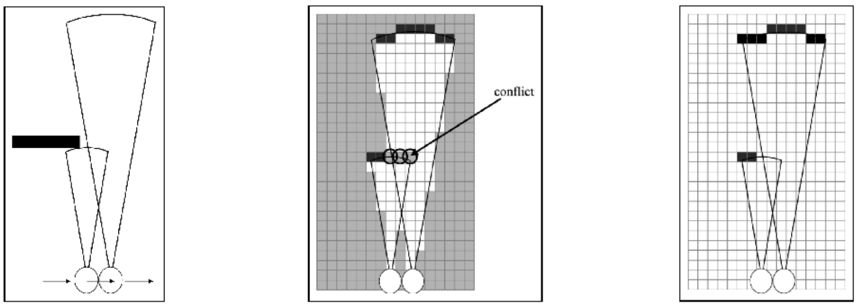
$$P(O) = m(O) + \frac{m(\{E, O\})}{2} \tag{2.20}$$

These two low-level sensor fusion approaches have been used many times in the literature, in the context of occupancy mapping. In [49], a comparison between the two methods is made for obstacle detection using the occupancy grid map to model the environment. With the goal of making an even comparison, the Dempster-Shafer implementation was made as equivalent as possible to the Bayesian formulation, by modeling the masses of the free and occupied states based on the occupancy probability distribution function used in the Bayesian theory. The results show that both methods obtain similar accuracy results, however, the obstacle detection rate is better in the Dempster-Shafer approach compared to the Bayesian method, since it is able to handle conflict between different sensors. However, regarding computation time, the Bayesian method was able to compute the map update twice as fast as the Dempster-Shafer method, considering the expensive combination formula of this approach, whereas the log-odds representation of the Bayesian filter is much more efficient.

2.5 Sensor Modeling

A sensor model describes the limitations of the sensor and its performance regarding measurement uncertainty. In particular, a probabilistic sensor model represents the characteristics of the sensor data in a statistical form [52], facilitating data operations such as sensor fusion, as mentioned in section 2.4. Furthermore, the capability of a probabilistic sensor model to handle uncertainty is crucial, considering any data obtained from sensors may contain incorrect detections or misinterpretations, known as artifacts [59], due to beam reflections or dynamic objects. Therefore, mitigating the influence these artifacts may have on the results of the task-at-hand, such as obstacle detection is an important feature of any environment model. When used alongside an occupancy grid map, a sensor model provides a correspondence between sensor readings and the likelihood of a grid cell being occupied. For this purpose, either a forward or an inverse sensor can be used, which are both described further in this section.

A forward sensor model is concerned with finding the probability of getting a certain sensor measurement z_t, given the map state m and the current position x_t, defined as $p(z_t|m, x_t)$. This direct causal relationship from the map to the sensor measurements allows for the creation of more complex and, therefore, more accurate sensor modeling [49]. Furthermore, this approach does not require the assumption that the occupancy of each cell in the grid is independent of all the other cells' states, which benefits situations in which there is a conflict between different sensor measurements, as shown in figure 2.5. However, this method requires the map state m to be known or estimated. Therefore, to use this approach, the problem of finding the most likely map state m needs to be solved, which increases the computational load of the occupancy mapping problem. S. Thrun [1] proposes a new method for occupancy grid mapping to handle map inconsistencies that come from conflicting information between different sensors, by using forward sensor models and, therefore, maintaining all dependencies between neighboring cells, resulting in more accurate maps. The proposed approach uses the expectation-maximization algorithm to obtain the map which maximizes the likelihood of the measurements.



(a) Environment (b) Inverse sensor model (c) Forward sensor model

Figure 2.5: Conflict scenario comparison between forward and inverse sensor models using a sonar sensor. In figure (a), the real environment is presented. In (b) an inverse sensor model is applied and a conflict of measurements occurs. In (c), a forward sensor model is used and the conflict is resolved. Images from [56].

An inverse sensor model is focused on obtaining the map state m based on the sensor measurements $z_{1:t}$ and vehicle poses $x_{1:t}$, defined as $p(m|z_{1:t}, x_{1:t})$. In the initial proposal of the occupancy grid map approach by A. Elfes et al. [23] [24], this method was implemented. The assumption of independence between the cells in the grid is required with this approach, which allows for a simplification of the sensor model and, therefore, an increase in the time performance of occupancy mapping.

For certain sensors with high accuracy, an ideal inverse sensor model which only considers the cell hit with the sensor beam as occupied and the cells along the beam as free, as shown in equation 2.21, may be used:

$$p(m_{ij}|z_k) = \begin{cases} p_{occ}, & \text{if beam hit cell } m_{ij} \\ p_{free}, & \text{if beam traversed cell } m_{ij} \end{cases} \tag{2.21}$$

where z_k is the latest sensor measurement and p_{occ} and p_{free} are the probabilities of a cell being occupied considering it was hit by the sensor beam, and considering it was traversed by the sensor beam, respectively. Furthermore, in case the sensor being modeled has really high accuracy, these values can be set to $p_{occ} = 1$ and $p_{free} = 0$, as it was done by C. Fernández [49] regarding a Velodyne Lidar.

However, most real sensors are noisy. Therefore, to obtain more accurate results, the sensor noise may also be modeled, which is achieved by distributing the occupancy probability over the grid for each sensor scan, as shown in the example of figure 2.6. In [60], a method to obtain an exact inverse sensor model is proposed to build a Bayesian occupancy grid map, which does not require the assumption of independence of separate measurements. The results showed an accuracy improvement compared to the classical use of the *log-odds* ratio representation. However, most approaches focus on obtaining an approximate inverse sensor model by means of a certain heuristic. A commonly used method to consists of using a Probability Density Function (PDF) to express the distribution of occupancy probability over the grid. This distribution is modeled to a specific sensor according to its characteristics and is often determined experimentally [52]. The most common approach in the literature for modeling this PDF and representing the sensor uncertainties consists of using a Gaussian distribution with mean μ and standard deviation σ, given by the following equation:

$$\mathcal{N}(x;\mu,\sigma) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right) \tag{2.22}$$

However, a PDF function of random variable X, such as the Gaussian, does not compute probability values. Rather, the output of this type of function for a given sample x corresponds to the relative likelihood that a random value would be near the given sample [61]. Therefore, a Cumulative Distribution Function (CDF) is used to obtain the distribution of occupancy probabilities, as it allows to compute the probability of the random variable X being less than or equal to a certain value x [62], as follows:

$$CDF(x) = P(X \leq x) \tag{2.23}$$

In case the random variable X follows a Gaussian distribution, this CDF probability corresponds to the area below the Gaussian curve, which can be calculated by computing the integral of the Gaussian function, as shown in the following equation:

$$CDF(x;\mu,\sigma) = \frac{1}{\sigma\sqrt{2\pi}} \int_{-\infty}^{x} \exp\left(-\frac{(t-\mu)^2}{2\sigma^2}\right) dt \tag{2.24}$$

As it can be observed in equation 2.24, the CDF of a sample x corresponds to the area below the Gaussian curve for the interval $]-\infty, x[$. To compute the CDF for a specific interval $[x_1, x_2]$, where $x_2 > x_1$, the two CDF values are calculated and then subtracted, as follows:

$$CDF([x_1, x_2]) = CDF(x_2) - CDF(x_1) \tag{2.25}$$

Furthermore, for any normally distributed random variable X, the corresponding *z-score* can be used to always obtain a Gaussian distribution with zero mean and variance equal to one

$\mathcal{N}(0,1)$ [63], which is given by the *z-formula*, as follows:

$$Z(x;\mu,\sigma) = \frac{x-\mu}{\sigma} \tag{2.26}$$

In [49], a Gaussian distribution is applied to model the uncertainty of a radar sensor, using a mean equal to the detection and a variance equal to the accuracy of the sensor. To obtain actual probability values from the Gaussian distribution, a normalization factor is used.

Figure 2.6 shows an example of an inverse sensor model applied to a single radar beam in two dimensions: range and azimuth angle, where a Gaussian kernel is applied to model the distribution of occupancy probability $p(\rho,\theta|z)$ over the grid. The mean in each dimension corresponds to the obtained sensor measurement: $\mu_r = 15m$ and $\mu_\theta = 0°$. The applied variance is equal to the sensor accuracy in the corresponding dimension: $\sigma_r = 0.2m$ and $\sigma_\theta = 1°$.

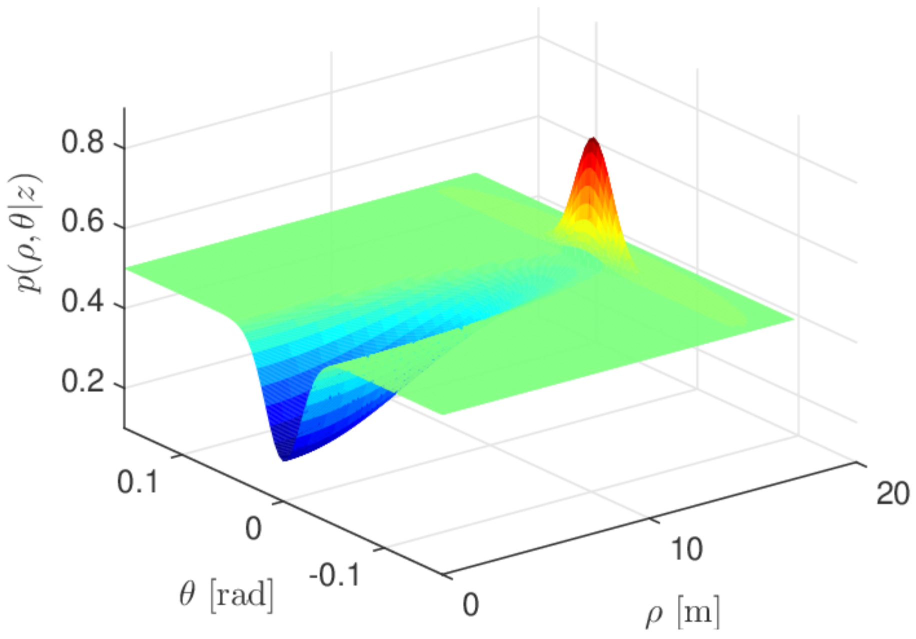


Figure 2.6: Inverse sensor model for a single radar beam, in 2D polar coordinates: range and azimuth angle, where a Gaussian kernel is applied to model the distribution of occupancy probability $p(\rho,\theta|z)$ over the grid. Image from [49].

J. Carvalho et al. [64] compare the performance of forward and inverse sensor models in the context of occupancy grid mapping using sonar sensors. The preliminary results show that using a forward sensor model allows for the creation of an occupancy map with fewer artifacts and, therefore, is more accurate in comparison to when an inverse sensor model was used. However, the increased computational load of the forward sensor model results in a less time-efficient occupancy map.

Some approaches in the literature which implement an inverse sensor model for occupancy mapping, also apply weighting to the sensor beams. C. Fernández [49] applies a beam weighting approach based on the received RCS in the radar measurement to mitigate the effect of multi-echo signals.

2.6 Occupancy Map Evaluation

The evaluation of an occupancy grid map is usually performed on memory usage, accuracy, and the time it takes to update the map or query a certain position on whether it is occupied or not. Both the time and memory usage metrics can be easily calculated in most programming languages. However, evaluating the accuracy of an occupancy map is a complex task. This section reviews some approaches used in the literature to perform an evaluation of the accuracy of an occupancy map.

The accuracy metric is usually evaluated in percentage units in relation to some reference with which the results are compared to. In the context of occupancy mapping, this reference is often described as a ground truth map [65]. In most occupancy mapping applications in the literature, a separate construction process of the ground truth map is performed. In certain cases, the goal is not to obtain absolute accuracy values, rather an evaluation relative to other state-of-the-art occupancy mapping methods using the same sensor data is sufficient. For example, in [33], the accuracy of the proposed mapping approach is calculated relative to the results of the *OctoMap* [3] method. Often in other applications where the main focus is on the accuracy of the sensor, rather than on the mapping approach, the ground truth map is constructed in a similar process to the proposed mapping method using data from other sensors with higher resolution. In [49], an obstacle detection application is proposed using an occupancy grid map based on radar and lidar sensors. To evaluate the accuracy of the map, the creation of a ground truth map is proposed using measurements from a Velodyne lidar, which has a very high resolution. The building of the ground truth map required a completely separate construction process, including the creation of an inverse sensor model for this specific sensor. Furthermore, a ground-removal algorithm was required as this sensor's measurements contained detections corresponding to the ground. The proposed model performed ground removal by first determining the equation of the plane which approximates the ground. After that, a Random Sample Consensus (RANSAC) algorithm was applied for plane extraction. After ground-removal, an ideal inverse sensor model, reviewed in section 2.5, was applied for this sensor using ideal probability values: $p_{occ} = 1$ and $p_{free} = 0$. Regarding the calculation of the accuracy values, a comparison between the occupancy map and the ground truth is proposed to obtain an accuracy metric, consisting of calculating the distance between matching occupied and free cells. However, it is concluded in the paper that a perfect alignment between the two maps would be required, which was not the case in the different implemented test scenarios due to errors in sensor calibration. Therefore, it is concluded that the results of this comparison metric are not useful for evaluation. Furthermore, the high price of the Velodyne sensor and the time-consuming creation of a separate map-building process for this specific sensor imposes limitations on this approach.

In [66], a comparison method for SLAM applications is proposed which uses a ground truth map. Data from a two-dimensional lidar was used to construct a map for the three most common SLAM libraries and the ground truth map was created based on data from a FARO laser tracker in an indoor static environment. To obtain an accuracy metric, a comparison process between each SLAM map and the ground truth was performed, which required narrowing the wall lines and then aligning the maps using the Iterative Closest Point (ICP) algorithm [40], which computes a minimum error based on the sum of distances between each point in the occupancy map and the corresponding nearest neighbor on the ground-truth map. Furthermore, the Average Distance to the Nearest Neighbor (ADNN) was used as the distance metric, by summing all distances and then dividing by the number of occupied cells, as shown in equation 2.27:

$$\textbf{ADNN} = \frac{\sum_{i=1}^{N} Nearest\_Neighbor(occupied\_grid\_cell(i))}{N} \tag{2.27}$$

Considering the presented difficulties of performing an objective accuracy evaluation, namely in obtaining a ground truth map and then comparing the two maps, an alternative approach can be used with the downside of obtaining more subjective results. It consists of creating a visualization module for the environment model and then comparing the rendered scene with the expected results, which may be obtained based on knowledge of the detected real-world structures or through a visual analysis of the camera's footage recorded alongside the sensor measurements.

2.7 Voxel Rendering

As mentioned in section 2.6, the evaluation of the accuracy of an occupancy map can be performed in a subjective way through visualization. Furthermore, considering the real-time performance requirement, it is important that the rendering also functions in real-time, since a slow visualization would limit the evaluation process.

A common approach to visualize three-dimensional structures in the field of computer graphics is to render a high number of cubic volumes (voxels) to create 3D solid models. As a literature example, C. Crassin et al. [67] propose an approach for real-time rendering of several billion voxels based on an adaptive data representation that takes into account the occlusion of objects from the camera's view associated with an efficient ray-casting algorithm.

One of the most popular tools for efficient three-dimensional visualization is *OpenGL* [68]. It consists of a software library for accessing features in graphics hardware. This section presents a brief overview of existent rendering approaches available in this tool, without going into much detail on each one, as it is not the focus of this dissertation.

OpenGL does not offer methods capable of directly rendering three-dimensional figures. Instead, the rendering process consists of sending the figure data to the Graphics Processing Unit (GPU), where the actual rendering occurs. These data must contain the set vertices which compose the figure and the corresponding primitive which details how *OpenGL* should interpret the specified set of vertices. The list of available primitives is the following:

- Point: each vertex is interpreted as a single point.
- Line: every two consecutive vertices are interpreted as the starting and end point of a line, respectively.
- Triangle: every three consecutive vertices are interpreted as the vertices of a triangle.
- Quad: every four consecutive vertices are interpreted as the vertices of a quad.

Nowadays triangles are preferred over quads to render three-dimensional geometry, as it allows for more diversity regarding the shape of the rendering object. Therefore, most modern GPUs work only with triangles, which can increase the CPU overhead in case the quad primitive is used in *OpenGL*, as the driver needs to transform each quad into two triangles before rendering.

The most simple way to perform rendering in *OpenGL* is called **immediate** mode. The geometric data is stored in RAM on the client-side and sent to the GPU on each render call, where rendering occurs immediately after the call, blocking the client's program until rendering is finished. This method offers flexibility as it allows for easier control over the geometric data of each shape. However, it is inefficient as it requires a high number of calls to the GPU and therefore, introduces CPU overhead [69].

Vertex arrays is another method for rendering in *OpenGL* which reduces the CPU overhead of the immediate mode, by storing the rendering data in a client array, which allows for only one function call to be required to communicate with the GPU and render the geometry [70].

However, even with the use of vertex arrays, the constant communication with the GPU for each new frame limits the rendering performance. Considering this, **display lists** can be used for certain rendering applications where the geometric data does not change often [70]. With this method, the rendering commands are compiled and stored in the GPU for later execution, allowing it to perform rendering in the most optimal way. This may result in a very fast rendering when used with certain GPUs. However, due to its immutability, it is not a suitable method for certain three-dimensional rendering applications which change geometric data at each time-step, as it receives new sensor measurements.

The most popular approach for efficient three-dimensional rendering is called **retained** mode. With this method, a **vertex buffer object** is used, which is filled in the client-side and then sent to the GPU once, without blocking [71]. The advantage of this method is that *OpenGL* offers methods to directly update the data contents of these buffers with limited CPU overhead.

Instanced arrays is another approach to efficiently render many three-dimensional figures which have an identical geometric structure. The idea of this method is to render many instances of the same shape in different positions, by sending the shape geometry data and the list of positions to the GPU, which then translates and colors each vertex at render time, using shaders [72]. These are small programs written in *OpenGL* Shading Language (GLSL) which are first compiled and then stored on GPU memory, where they are executed for each vertex sent to the GPU. Two types of shader programs exist:

- **Vertex shader**: calculates the pixel position on the screen for the received vertex.
- **Fragment shader**: outputs the RGBA color encoding for the corresponding vertex.

The use of these shaders allows to further reduce CPU overhead by moving certain vertex calculations to the GPU, where they can be performed more efficiently, and by further reducing the size of the data which needs to be sent to it. However, this approach has the downside of limiting the geometry diversity of the rendered three-dimensional objects, as it requires that all rendered objects have exactly the same shape.

Chapter 3

System Overview

This chapter presents a general overview of the work of this dissertation, by first describing the problem at hand in section 3.1, followed by the assumptions which will be made for this work in section 3.2, and then the requirements which must be fulfilled by the implemented system are described in section 3.3. The proposed solution is then presented in section 3.4.

3.1 Problem Statement

The Perception Development Kit (PDK) is a multi-sensor prototype system that provides sensor gateways for radars and cameras, and an Application Programming Interface (API) for easy integration into the customer's development environment, with the goal of facilitating and accelerating sensor prototyping, developed by Continental Engineering Services (CES) [2]. It is able to take sensory inputs from radar sensors and cameras and then provides building blocks for environmental perception.

As represented in the diagram of figure 3.1, showcasing the PDK's architecture, a series of optional software modules exist on top of the PDK that provide different environmental perception applications, such as the Dynamic Object Tracking (DOT) and the Occupancy Grid Processing (OGP), for example.

The OGP software module implements a two-dimensional static environmental model. It is able to build an occupancy grip by fusing and integrating sensor data from multiple radars into a grid map. This software module is currently mainly used alongside Continental radars with a two-dimensional point cloud output. However, recent advancements in radar hardware technology have led to the development of the Continental ARS540 radar sensor, which is capable of not only detecting the range, Doppler velocity, and the azimuth angle but also the elevation angle of detections caused by targets [73]. The compatibility of the PDK with this new radar sensor provides a suitable basis for the implementation of a three-dimensional environment model. Although the PDK is also compatible with cameras, this work focuses only on the use of radar sensors, specifically on the Continental ARS540 radar sensor, for three-dimensional environmental perception.

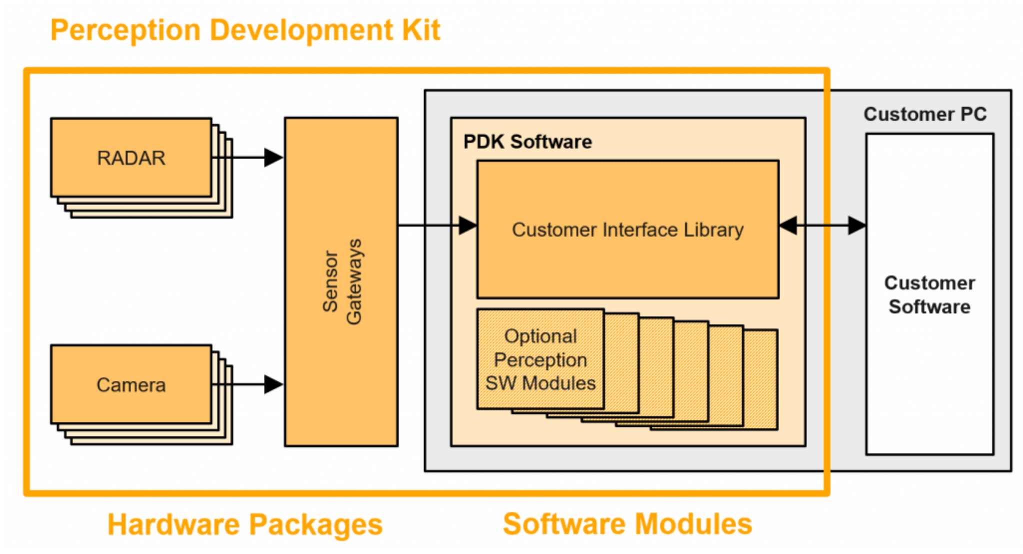


Figure 3.1: Perception Development Kit's Architecture. Image from [2].

Although a two-dimensional environment model is sufficient for many ADS applications, it brings some serious limitations to the system which uses it, as for example, the misrepresentation of overhanging objects, such as bridges. In a scenario in which an autonomous vehicle is facing a bridge, the two-dimensional grid may represent this object as an obstacle in front of the vehicle. This error in the perception module of the ADS system may cause a series of decisions in the planning module, such as an abrupt stop of the vehicle, creating a hazardous situation. By modeling the environment in a three-dimensional way, this behavior can be mitigated. The additional level of information about the structure of the surroundings allows for correctly representing over-hanging objects, including bridges. However, simply applying a 3D Occupancy Grid is not a reliable approach, since, as mentioned in section 2.2, it would require the beforehand knowledge of the dimensions of the area to be mapped so that the required memory is properly allocated, which in cases of high-resolution mapping of large areas requires huge memory consumption. This model is also unable to correctly represent unmapped areas, which may be important for some autonomous driving systems. Hence, in order for the perception system to be appropriate for ADS systems, it must fulfill a series of requirements, as listed in section 3.3, which include high accuracy, memory compactness, and real-time performance.

3.2 Assumptions

Considering the proposed solution is an adaptation of the traditional occupancy grid approach to three dimensions and the Bayesian filter is also applied, all of this method's assumptions, mentioned in section 2.4, will be followed and adapted to three dimensions, as follows:

- The environment is static, so all radar detections corresponding to dynamic objects are ignored.

- Each map state update depends only on its previous state and not on the previous sensor measurements or vehicle poses.
- The current vehicle pose x_t depends only on the pose at the previous time step x_{t-1}.
- There is conditional independence between measurements at different time steps:

$$p(z_t, z_{t-1}|m) = p(z_t|m).p(z_{t-1}|m) \tag{3.1}$$

- The occupancy probability of each voxel in the grid is independent of all other voxels.

Furthermore, to simplify the coordinates frames transformations from the sensor frame to the world frame, explained in section 3.4.1, the assumption that the vehicle moves on a flat surface is also made. In addition, the initial pose of the vehicle in world frame coordinates X_0, is assumed as follows, to allow vehicle pose estimation:

$$X_0 = [x, y, z, \theta] = [0, 0, 0, 0] \tag{3.2}$$

Also, considering the characteristics of the ARS540 radar sensor, the different dimensional detections, including range, azimuth, and elevation angles, are assumed to be independent of each other.

3.3 Requirements

For the three-dimensional environment model to be suitable for an autonomous driving application, it must follow a series of system requirements, namely:

- **Real-time performance**: the time it takes for a new sensor measurement to be integrated into the map needs to be below a certain threshold given by the cycle time, which is defined as the time between consecutive point cloud processing operations. During the evaluation experiments of this work, this value will be configured to 100 milliseconds. This means that the insertion of each point cloud into the occupancy map needs to be performed under 100 milliseconds so that all measurements are properly processed. This real-time performance is absolutely required for any autonomous driving system, since any delay in the response time of the perception module could have hazardous effects.
- **Reduced memory usage**: so that the proposed solution is suitable for ADS applications, which have a limited available memory size, the final three-dimensional environment model needs to be compact. Specifically for the evaluation of this work, this requirement is analyzed based on the required size to store the whole occupancy map in memory, which needs to be lower than the consumption of the three-dimensional grid which would store the same mapped area. The memory size of this model can be calculated as

follows:

$$mem_{grid} = \frac{x \times y \times z}{r^3} \times S, \tag{3.3}$$

where x, y, and z are the dimensions of the minimal bounding box of the mapped area, r is the map resolution and S is the memory consumption of each voxel. To calculate the memory size of the 3D grid, the total number of voxels in it, given by the fractional term of the equation, is multiplied by S at the end, which is equal to four, in case the sensor fusion is performed with the Bayesian theory, or is equal to eight, if, otherwise, the Dempster-Shafer method is used. It is important to note that these values correspond to a 32-bit architecture.

- **High modeling accuracy**: a set of detections corresponding to a three-dimensional figure should be identifiable as belonging to the same object.

Although all three system requirements mentioned above are quite relevant for the development of the proposed solution, the main requirement of this work is on obtaining real-time performance, while minimizing memory usage and maximizing modeling accuracy.

3.4 Proposed Solution

The proposed solution consists of adapting the two-dimensional grid map of the OGP software module to receive data from the Continental ARS540 radar sensor and create a three-dimensional environment model with this data, by taking advantage of this sensor's capability of measuring the elevation angle of detections caused by targets. Furthermore, the performance requirements concerning time and space efficiency and accuracy are taken into account in the decision of which environment modeling approach to use, which is explained in section 3.4.2.

A two-dimensional occupancy grid involves a data pipeline that processes each new radar sensor measurement taking into account the received mounting parameters and vehicle dynamics, and, in the end, integrates it into the map, as shown in the diagram of figure 3.2.

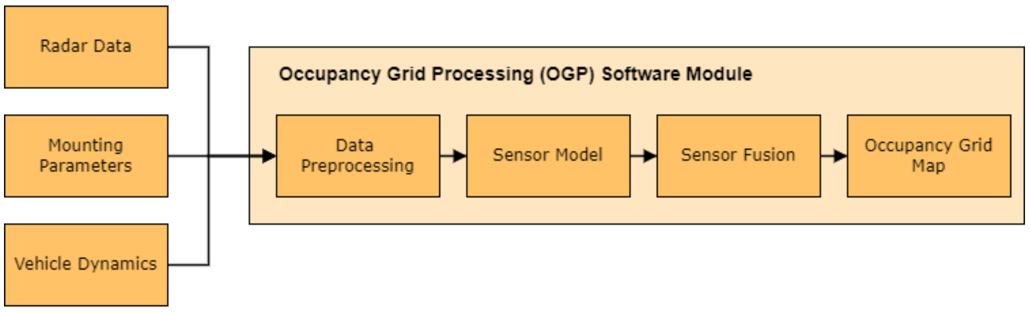


Figure 3.2: Occupancy Grid Processing (OGP) software module's architecture.

The first module of the pipeline is the data preprocessing, which filters out certain sensory inputs, namely:

- Invalid sensor measurements, such as those outside the radar maximum range or its angular Field of View (FOV), which are specified in appendix B.
- Radar detections corresponding to dynamic objects based on the Doppler velocity obtained in the sensor measurement and the current vehicle velocity obtained from the vehicle dynamics, similarly to the method review in section 2.3.

The transformation required to make this module work properly in the three-dimensional space consists only of adding the FOV of the elevation angle to the filtering process of each measurement. The remaining modules in the data pipeline require an adaptation to be used for the three-dimensional environment model. These modifications, which compose the proposed solution, are introduced in section 3.4.2, regarding the chosen three-dimensional environment modeling approach, in section 3.4.3, for the sensor modeling method, and in section 3.4.4, regarding the approach for probabilistic sensor fusion. Considering visualization is important to perform the evaluation of the proposed solution, section 3.4.6 presents an overview of the proposed methods to render the three-dimensional environment model.

3.4.1 Data

The Occupancy Grid Processing (OGP) software module receives, at each time step, a set of radar detections, where each contains the following data:

- Radial distance, azimuth, and elevation angular distance between the radar and the detection.
- Doppler velocity in the radial direction.
- Compensated Radar Cross-Section (RCS).
- Variance for each dimension: radial distance, Doppler velocity, azimuth angle, and elevation angle.

Furthermore, this module also receives other essential data which is not obtained from the sensors, such as:

- Mounting parameters: position and orientation of the radars in vehicle frame coordinates, which is explained further below in this section.
- Vehicle dynamics: absolute velocity in world frame coordinates, the longitudinal and lateral acceleration, and the rotational velocity.
- Video footage from two cameras installed on the vehicle, one facing forward and the other facing backward, recorded alongside the radar measurements.

The occupancy map keeps the position of sensor detections in the world coordinate system or frame. However, as shown above, measurements from the radar are in polar coordinates in the sensor frame. Therefore, a transformation between these two systems is required, which depends on knowledge of the sensors' positions, which are defined in the vehicle frame. The definition of these three coordinate systems/frames is as follows:

- World frame: centered on the initial vehicle pose, with Y-axis pointing in the direction to which the vehicle is facing and Z-axis pointing up.
- Vehicle frame: X and Y-axis origins in the center of the vehicle, and Z-axis pointing up with its origin on the ground.
- Sensor frame: Usually in polar coordinates, has its origin on the radar's receiving antenna. It is important to note that the elevation angle begins at the $Z = 0$ plane.

3.4.2 Environment Modeling

As it was reviewed in section 2.2, there are many approaches that have been used to model the environment in three dimensions. However, the context of this dissertation, including the use of the ARS540 radar sensor and the system requirements, listed in section 3.3, must be taken into account in order to choose the most appropriate method.

The point cloud approach, although it is commonly used alongside the lidar sensor, it is not the most appropriate method for modeling radar data, considering the lower resolution of the radar sensor leads to less dense point clouds. Furthermore, this method requires a huge memory space, which does not fulfill one of the system requirements. The three-dimensional occupancy grid offers some important benefits. Since it is a direct transformation of the traditional two-dimensional occupancy grid map to three dimensions, it facilitates the adaption of this method's modules, such as sensor modeling and sensor fusion. However, the memory space it requires is still too much to be considered appropriate for this dissertation. The Octree modeling method, on the other hand, is able to provide the same benefits as the three-dimensional occupancy grid, while significantly decreasing the required memory space, with the downside of requiring dynamic memory allocation, which is not possible in embedded systems. The Elevation Map approach is not suitable for the proposed solution considering it is unable to model overhanging objects, which means it would not solve the presented problem. Although the MLS Map method extends the Elevation Map to fix this limitation, its inability to represent unmapped areas makes it an inappropriate method for many ADS applications, such as path planning, therefore, it is not the approach used in the proposed solution.

Considering all the points presented above, the Octree is the modeling method chosen for the proposed solution. Furthermore, two libraries were reviewed in section 2.2 which implement an Octree for occupancy mapping: *OctoMap* and *UFOMap*. Although the latter tool presents advantages compared to the former, it requires the use of C++17. Since at the time of this work, the current available PDK version was implemented in C++14, this library can not be used in the proposed solution. Therefore, *OctoMap* is chosen.

3.4.3 Radar Sensor Model

Regarding the different approaches for sensor modeling, mentioned in section 2.5, an inverse sensor model method will be used and adapted to three dimensions in the proposed solution which considers the Continental ARS540 radar sensor characteristics. Although the forward sensor model approach may offer more accurate results due to its more complex model, the expensive computational time makes it inappropriate for the proposed solution. Furthermore, most environment modeling applications, such as obstacle detection, do not require an extremely accurate method. Rather a map over which the position of objects can be estimated is sufficient. In addition, the Dempster-Shafer method for sensor fusion, which will also be implemented in the solution, as explained in section 3.4.4, is only compatible with an inverse sensor model.

The *OctoMap* library currently implements an ideal inverse sensor model, explained in section 2.5, since a high-resolution lidar was used for its evaluation. However, considering the lower accuracy of the radar, this approach is not appropriate for the proposed solution as it does not take into account the uncertainty associated with the sensor accuracy. Therefore, it is necessary to implement an inverse sensor model which also updates the neighboring voxels with an occupancy probability according to a Gaussian distribution with variance according to the radar's accuracy.

3.4.4 Probabilistic Sensor Fusion

Although most sensor fusion applications intend to combine data from different types of sensors, with the goal of achieving redundancy, leading to an increase in robustness and reliability, as it allows sensors to complement each other's weaknesses, the proposed solution will focus only on performing the fusion of data from the multiple radars installed on the vehicle and from consecutive sensor scans that retrieve measurements from the same surrounding area.

As presented in section 2.4, sensor fusion can be performed at high-level or at low-level. Considering the proposed solution consists of an adaptation to three dimensions of the occupancy grid mapping approach, which implements low-level sensor fusion, this is the chosen method for sensor fusion.

Furthermore, two probabilistic inference methods were reviewed in section 2.4, where each approach offers advantages over the other one. Whereas using a Bayesian filter benefits time efficiency, using a Dempster-Shafer approach allows for the explicit representation of the unknown state and, therefore, may offer more accurate results. Considering this, both methods will be implemented in the proposed solution and their results compared based on accuracy and time and memory consumption efficiency.

3.4.5 Occupancy Map Evaluation

As reviewed in section 2.6, the evaluation of the accuracy of an occupancy map requires a ground-truth map with which the obtained results are compared to. One way to construct such a map is to add a high-resolution sensor, such as a Velodyne lidar, to the set of sensors installed on the vehicle, so that its measurements, recorded alongside those captured by the radar, are used to build a separate occupancy map, defining the ground-truth.

However, the datasets available for evaluation of the proposed solution do not include measurements from a high-resolution sensor, which impedes the creation of a ground-truth map. Nonetheless, in case such a dataset was made available, a possible methodology for an objective measurement of the three-dimensional occupancy map's accuracy is described in section 4.4.

Considering the aforementioned impossibility of performing an objective evaluation of the proposed solution's accuracy, the subjective alternative approach mentioned in section 2.6 which uses a visualization module will be implemented.

3.4.6 Visualization

As mentioned in the previous section, visualization is important to perform the evaluation of the proposed solution by comparing its rendering results with the real structures around the vehicle which are observed in the camera footage, recorded alongside the radar measurements. Furthermore, considering the PDK is a prototyping tool for the evaluation of Continental sensors, direct visualization of their detections enhances its flexibility.

As reviewed in section 2.7, *OpenGL* is one of the most popular tools for efficient visualization of three-dimensional structures. Furthermore, it is readily available as a C++ external library. Considering this, the visualization of the proposed solution will be implemented using *OpenGL*. However, this tool does not provide methods for window handling and reading of user inputs. Therefore, the tool *wxWidgets* will be used for this effect, which allows the use of *OpenGL* on top of it to render three-dimensional scenes.

In addition, five different methods for shape rendering in *OpenGL* were also reviewed in section 2.7:

- Immediate mode.
- Display lists.
- Vertex arrays.
- Vertex buffer objects.
- Instanced arrays.

Considering the unsuitability of the display lists method for applications that update rendering data frequently and since our solution is constantly being updated every time it receives a new radar measurement and integrates it into the map, this method is not appropriate for the presented solution. However, all other methods mentioned are suitable and, therefore, they will all be implemented and compared based on their time performance.

Chapter 4

3D Occupancy Mapping

This chapter details the implemented approach regarding the adaptation of the traditional occupancy grid mapping approach to the three-dimensional space, by first describing some important details of the *OctoMap* tool, which was used to model the environment, in section 4.1. However, the sensor model used in *OctoMap* is not appropriate for the ARS540 radar sensor due to its lower accuracy compared to the lidar range sensor, originally used to evaluate this tool. Therefore, a more complex sensor model which takes into account the uncertainty associated with each detection is implemented, as further explained in section 4.2. Furthermore, this occupancy mapping tool updates the probabilities of each voxel using a Bayesian Theory approach, by saving a *log-odds* ratio on each voxel. However, considering the points presented in chapter 3 in favor of the Dempster-Shafer Evidence Theory method to update the occupancy probabilities, the *OctoMap* tool is adapted in the proposed solution to also implement this method, as further explained in section 4.3.

4.1 3D Environment Model

As mentioned in chapter 3, the three-dimensional environment modeling method chosen for the solution was the Octree, using the *OctoMap's* library implementation, explained in more detail in section 2.2. This tool allows for the customization of some parameters which directly affect its performance, such as:

- **Resolution**, which defines the minimum size of a voxel. All leaf nodes of the Octree correspond to voxels with a size equal to this parameter.
- **Tree depth**, which determines the maximum number of hierarchical levels of the Octree.

The maximum map size of each Cartesian dimension is directly limited by these two parameters, as previously shown in equation 2.2. For example, with a resolution of 0.2 meters and a maximum tree depth set to its default value of 16 levels, each dimension is limited to a total size of $0.2 \times 2^{16} = 13107.2$ meters. Considering the assumption made in section 3.2 regarding the initial pose of the vehicle, this limitation means that detections of an object located outside of the bounding box with an edge length of 13107.2 meters and centered on the initial vehicle position are discarded and not integrated into the Octree. However, any detections located inside this bounding box, within the radar sensor's angular FOV, and closer than its maximum range are integrated into the Octree. Furthermore, regarding memory management, the *OctoMap* tool allocates the memory of each node dynamically so that space is more efficiently managed by only creating the necessary nodes to store the voxels of the mapped area. However, as explained in section 2.2, this type of memory allocation is not appropriate for embedded systems due to the reduced memory size available. This problem was not handled in this implementation, limiting the implemented solution, but could be solved with different approaches, as further explained in section 6.2, which lists the possibilities of improvements for the presented work. Furthermore, to update the occupancy of each voxel during a map update, the implementation requires traversing the Octree from its root to the desired voxel at the last hierarchical level. Therefore, an increase in time performance is expected when the tree depth parameter is reduced, with the downside of further limiting the maximum size of the total mapped area. Regarding the resolution, lower values result in a higher discretization of the space and may improve the level of detail for each detection and, therefore, obtain more accurate results. On the other hand, a higher resolution allows for a reduction in the number of updated voxels for each map update, as some detections in the point cloud become part of the same voxel, which may increase time performance.

Furthermore, with the goal of analyzing the performance impact of performing ray casting to compute the free space voxels, the *OctoMap*'s Octree implementation was adapted so that **free space** calculations could be disabled. When it is enabled and in case a voxel is determined as both occupied and free during a map update due to discretization errors, the voxel is updated as occupied, ignoring the free classification.

4.2 Radar Sensor Model

A probabilistic sensor model describes the characteristics of the sensor in a statistical form so that uncertainty regarding each sensor measurement can be represented over the occupancy grid. Furthermore, as mentioned in chapter 3, an inverse sensor model is implemented in the proposed solution, which is concerned with finding the occupancy probability for each voxel m_{ijk} based on the received sensor measurement z_t and current vehicle pose x_t: $p(m_{ijk}|z_t, x_t)$. As explained in section 2.1, the point cloud of the ARS540 radar measurements is not as dense as the one produced by a lidar sensor. Furthermore, in the context of occupancy grid mapping, the lower accuracy of the radar creates uncertainty regarding the determination of which cell or voxel was hit by the sensor beam. To deal with these two characteristics of the radar sensor, the implemented inverse sensor model intents to blur the point cloud, by assigning occupancy probability values not only to the voxels corresponding to each measurement but also to their neighboring voxels, as further explained in section 4.2.1. In addition, every detection of the radar point cloud is different, either due to different RCS values or because detections at higher ranges have increased associated uncertainty. Therefore, a more accurate inverse sensor model should also take into account these different characteristics of each detection to influence the final occupancy probabilities assigned to their corresponding voxels, as further detailed in section 4.2.2.

4.2.1 Occupancy Probability Distribution

As mentioned in section 2.5, some approaches in the literature distribute the probabilities of occupancy over the grid for the whole FoV and the range of the sensor for each scan. However, applying this approach to a three-dimensional environment model, such as an Octree, would consume a huge amount of time when updating the map, as the number of affected voxels would be significantly high. Therefore, the implemented solution only distributes the occupancy probabilities around each point in the point cloud during a map update.

This distribution of probabilities is performed using a Gaussian distribution so that a higher occupancy probability value is assigned to the voxel corresponding to the sensor detection, and its neighboring voxels receive lower occupancy values, which decrease proportionally to their distance to the detection point. Furthermore, considering the independence between each dimension of the sensor (range, azimuth, and elevation) made in section 3.2, the Gaussian is first applied to each one separately, by setting its mean μ to the value of the detection and its standard deviation σ to the resolution of the sensor in the corresponding dimension. Then, they are merged into one through multiplication. However, as mentioned in section 2.5, a Gaussian distribution is just a PDF function that gives a relative likelihood of the random variable X being close to the given sample x. Therefore, the occupancy probabilities of each voxel around the detection are obtained by calculating the CDF of the Gaussian, which outputs the probability of the random variable X being less than or equal to the sample x. In this implementation, the CDF is calculated for specific intervals by performing a space discretization around each radar detection for each one of its dimensions, resulting in a grid in polar coordinates around the detection point, where then the occupancy of each cell is calculated by computing the difference between the CDF values for the starting point and the ending point of the cell. Furthermore, to improve the time efficiency of the radar sensor model, the CDF values of the zero-mean Gaussian with a variance of one $\mathcal{N}(0,1)$ are pre-computed and stored into a table in memory. In addition, considering the symmetry of the Gaussian distribution, only one-half of the corresponding CDF values are calculated. Then, during an Octree map update, to obtain the CDF of each voxel around the radar detection, the *z-score* for each dimension is calculated and then used to consult the corresponding CDF value in the previously stored table, considering the *z-score* always follows the Gaussian distribution $\mathcal{N}(0,1)$.

In addition, it is important to define a sensor model which does not allow for the map to reach a high confident estimation about the state of a voxel with just one sensor measurement. Rather, multiple detections of the same object must be required for the occupancy probabilities of the corresponding voxels to reach a high value. To achieve this, a re-scaling of the computed CDF table is performed so that its values are in a pre-determined interval $[P_{min}, P_{max}]$, based on the minimum CDF_{min} and maximum CDF_{max} values of the table, as follows:

$$P(CDF_{ijk}) = P_{min} + (CDF_{ijk} - CDF_{min}) * (P_{max} - P_{min}) / (CDF_{max} - CDF_{min}) \tag{4.1}$$

This approach allows for direct control over the minimum and maximum occupancy probabilities used to update each voxel so that many detections of the same object are necessary for the occupancy map to achieve high confidence in the state of the corresponding voxels. To select the most correct values for the interval of the CDF table values $[P_{min}, P_{max}]$, multiple experiments were performed while varying them, with some points in mind:

- It is acceptable that the minimum value P_{min} is below 0.5 since only those voxels in the extremes of the Gaussian distribution are updated with it. However, this value cannot be too much lower than 0.5, such as 0.3, as it would create a voxel highly confident that it is free when it is actually part of a detection distribution.

- So that the occupancy probability of a voxel only achieves a high value when it is updated multiple times, the maximum value P_{max} cannot be too high, such as 0.8 and above. However, considering the weight factor, which is explained in section 4.2.2, lowers the final probability value obtained from the CDF table, the P_{max} value cannot be too low since it could make the center voxel of some detections have an occupancy value below 0.5, which is undesirable.

- Changing these values by just a few units does not have a noticeable impact on the results.

Finally, after some experiments, while taking into account the aforementioned points, the values of $P_{min} = 0.4$ and $P_{max} = 0.75$ were selected.

Considering sensors are not fully reliable, an overconfident occupancy map should be avoided, as the probability of occupancy for each voxel should never reach zero or one, which would mean the map is fully confident about its state. Therefore, the implemented inverse sensor model performs a clamping operation after each voxel update, so that its occupancy probability does not fall under or surpass the pre-determined threshold parameters, as follows:

$$P_{min} \leq p(m_{ijk}) \leq P_{max} \tag{4.2}$$

Furthermore, as explained in section 2.2, the *OctoMap*'s Octree implementation allows one to set clamping threshold values which are used to determine the stability of a voxel, where it is considered stable if its occupancy probability reached the minimum or the maximum thresholds. This is then used to prune nodes that have all eight children in the same stable state, removing them from memory and, therefore, further increasing the space efficiency of the model. In the implemented solution, the values $P_{min} = 0.10$ and $P_{max} = 0.95$ are used as occupancy probability thresholds since with these values the map is already highly confident about the occupancy state of the voxel and any further increment above these values does not offer much more confidence. Furthermore, it already requires four or more detections for the probability of a voxel to reach these values, in which case it is very unlikely that its occupancy state changes.

4.2.2 Weight Factor

As mentioned previously in this chapter, not all radar detections have the same associated uncertainty. Therefore, an accurate modeling of the radar should not be equal for all its measurements. Considering this, the proposed implementation applies a weight factor to the probability obtained from the radar sensor model, so that the absolute occupancy values assigned to each detection are according to their unique characteristics. The implemented approach computes two weight factors which are then multiplied by the occupancy probability obtained from the inverse sensor model so that the final probability value used to update each voxel is reduced according to the RCS value and the range of the corresponding detection.

As explained in section 2.1, the RCS value of each measurement is correlated with the detectability of the object. Therefore, detections with lower RCS values should update the corresponding voxels of the Octree with lower occupancy probabilities. Considering this, an RCS weight factor W_{rcs} is implemented, which reduces the computed probability based on the RCS of the corresponding detections, creating a noticeable difference in the results between detections with different RCS values. Since the RCS is not scaled between zero and one, it can not be directly used as a weight factor. Therefore, a normalization is performed using a minimum RCS_{min} and a maximum RCS_{max} values, as follows:

$$RCS_{norm}(RCS) = \begin{cases} RCS_{min}, & RCS < RCS_{min} \\ RCS \,/\, (RCS_{max} - RCS_{min}), & RCS \geq RCS_{min},\ RCS \leq RCS_{max} \\ RCS_{max}, & RCS > RCS_{max} \end{cases} \tag{4.3}$$

After an analysis of the RCS values of a set of ARS540 radar measurements, it was observed that they varied between −25 dB and 45 dB. However, these values were not uniformly distributed in this interval, as most detections had a negative value, and only a few reached high positive numbers. Therefore, the minimum and maximum RCS values were set to $RCS_{min} = -20$ dB and $RCS_{max} = 30$ dB. However, having the RCS factor scaled from zero to one, would make it so detections with low RCS values would have low occupancy probabilities, which would mean that a voxel hit with the sensor beam, for example, could be incorrectly classified as free. Therefore, the normalized RCS is re-scaled so that its values are in the interval $[0.75, 1]$, as shown in the following equation:

$$W_{rcs}(RCS_{norm}) = 0.75 + RCS_{norm} \,.\, 0.25 \tag{4.4}$$

The values for this interval were chosen so that, at most, the RCS factor is only able to lower the occupancy probability of a voxel by 25%, which was selected so that the RCS factor has a noticeable impact on the occupancy probabilities computed by the radar sensor model, while not being able to make the probability of the voxel centered in the Gaussian distribution of detection below 0.5, which is undesirable.

As the uncertainty of each measurement increases linearly with its distance to the sensor, a radial distance weight factor W_r was also implemented so that the update occupancy probabilities of voxels further away from the sensor are slightly lower. This factor is computed based on the range of received detection r and a maximum range parameter R_{max}, as follows:

$$W_r(r) = 1.0 - r/R_{max} \tag{4.5}$$

However, this formulation of the radial weight factor has the same problem presented above regarding low probability values for measurements far away from the sensor. Therefore, a re-scaling is also applied to this factor so that its values are in the interval $[0.95, 1]$, as follows:

$$W_r = 0.95 + W_r(r).0.05 \tag{4.6}$$

With this interval, the radial weight is only able to lower the occupancy probability by, at most, 5%, so that it has a reduced influence on the final probability, which is desirable since it is expected that the radial distance does not have a significant impact in the measurement uncertainty.

This approach using re-scaled weight factors with intervals of different sizes for the two weights allows for control over the influence each factor has on the final occupancy probability. Specifically, the selected intervals make it so the RCS factor has a significantly higher influence than the radial distance weight.

4.3 Probabilistic Sensor Fusion

After computing the occupancy probability to update each voxel affected by the received sensor scan, using the radar sensor model, as explained in the previous section, the original implementation of the *OctoMap* library is used to traverse the Octree from its root to the desired voxel. However, the *OctoMap*'s implementation to combine the old occupancy probability of the voxel and the new update value, which uses a Bayesian theory approach, was altered in the implemented solution so that the Dempster-Shafer Inference Theory could also be applied.

The *OctoMap* tool allows customization of each node of the Octree so that any type of data can be stored on each voxel, not restricting it to just one value. Therefore, the implementation of the two aforementioned sensor fusion methods consisted of creating two separate node classes that store, update, and calculate the occupancy probability differently, as further explained in Section 4.3.1 regarding the Bayesian Theory Octree node, and in Section 4.3.2 for the implementation of the Dempster-Shafer Theory.

4.3.1 Bayesian Probability Theory

With this approach, the node stores just one value corresponding to the *log-odds* ratio representation of the occupancy probability explained in section 2.4.

When a node is first created, the state of the corresponding voxel is assumed to be unknown. With the Bayesian Theory approach, this is represented by assigning an initial occupancy value as follows:

$$p(m_{ijk}) = 0.5 \;\; \forall i, j, k \tag{4.7}$$

To update the stored value, the occupancy probability value $p(m_{ij})$ obtained from the radar sensor model is transformed into its *log-odds* ratio representation $l(m_{ij})$, as shown in equation 4.8, and then added to the value previously stored on the node.

$$l(m_{ij}) = \log \frac{p(m_{ij})}{1 - p(m_{ij})} \tag{4.8}$$

To obtain the occupancy probability from the stored *log-odds* ratio, the inverse operation is performed, as follows:

$$p(m_{ij}) = 1 - \frac{1}{1 + \exp l(m_{ij})} \tag{4.9}$$

In the implemented solution, the operations to transform the *log-odds* ratio into a probability value, and vice-versa, were performed using the available functions in the *OctoMap* tool.

4.3.2 Dempster-Shafer Evidence Theory

As explained in section 2.4, the Frame of Discernment (FOD) Θ, which defines the list of possible states for each voxel of the occupancy grid, and its power set 2^Θ are defined as follows:

$$\Theta = \{E, O\} \quad \text{and} \quad 2^\Theta = \{\emptyset, E, O, \{E, O\}\} \tag{4.10}$$

Although there are four different subsets in total, each node of the Octree only needs to store two values corresponding to the masses of the occupied and free states: $m_{ij}(O)$ and $m_{ij}(E)$, respectively, since the mass function of the empty set $\emptyset$ is always zero and the mass function of the unknown state $m_{ij}(\{O, E\})$ is implicitly represented since it can be obtained from the occupied and free masses, as follows:

$$m_{ijk}(\{O, E\}) = 1 - m_{ijk}(O) - m_{ijk}(E) \tag{4.11}$$

The initial state of the node, corresponding to the unknown state, is modeled by setting both values stored on the node to zero. This way, the mass function of the unknown state is equal to one, according to equation 4.11.

To update the occupancy value stored on a node, this approach uses the combination rule shown in equation 2.15. Although this formulation allows for the independent modeling of the free and occupied probabilities, the implemented approach always assigns a value of zero to the free probability $m^s(E)$, while the occupied probability $m^s(O)$ is obtained from the inverse sensor model.

Obtaining the occupancy probability value from the two values of the mass functions stored on the node is done by using the pignistic transformation formula, as shown in equation 2.20 adapted to three dimensions:

$$P_{ijk}(O) = m_{ijk}(O) + \frac{m_{ijk}(\{E,O\})}{2} \tag{4.12}$$

4.4 Evaluation

The evaluation of the implemented solution was performed using two datasets provided by Continental Engineering Services (CES) which were recorded in real-world scenarios using a vehicle installed with two ARS540 radar sensors and two cameras. Both radar sensors were positioned right next to each other in the front of the vehicle and directed forward. Although there were two cameras installed on the vehicle, one in the front and another in the back, only the one facing forward was analyzed for evaluation, considering the aforementioned forward poses of the radar sensors.

One of the datasets, referred to in this chapter as Dataset One, was mostly recorded on two-lane streets, with multi-story buildings and, sometimes, parked cars on each side. The vehicle also goes through some road crossings, having to wait at a red traffic light a few times. The other dataset, referred to as Dataset Two, was mostly recorded on a highway with two lanes in each direction and a divider between them, where the vehicle goes under a few bridges and passes through some traffic lights. Furthermore, each side of the highway consisted mostly of trees. Table 4.1 presents additional information on each evaluation dataset, where the mapped area is given by the volume of the minimal bounding box which contains all radar measurements.

Table 4.1: Additional information on each evaluation dataset, such as total record time and mapped area, given by the volume of the minimal bounding box which contains all measurements.

Dataset	**Recording time**	**Mapped area (m^3)**
Dataset One	11 min 12 sec	1,043×1,058×23
Dataset Two	12 min 35 sec	5,136×5,337×23

To have a better idea of the scenery of each evaluation dataset, appendix C presents additional images captured by the front camera during the recording of both datasets, alongside the visualization results of the corresponding Octree.

The evaluation focused on three different performance metrics: **time**, **memory usage** and **accuracy**. The evaluation of the **time** performance is based on the time it takes to insert a point cloud, corresponding to a single scan of one radar sensor, into the Octree. This metric is obtained by calculating the mean insertion time of all point clouds of each evaluation dataset. Furthermore, the insertion time of each radar scan into the Octree is obtained by computing the timestamp difference between the moment the point cloud processing starts and when it finishes. As stated in section 3.3, for the implemented solution to be appropriated for occupancy mapping, this performance metric needs to be below 100 milliseconds. The time results presented in section 4.5, were obtained by running the implemented solution on a laptop with a CPU Intel i5-6300U @ 2.40 GHz (2.50 GHz boost) and 8 GB RAM.

Regarding the **memory usage** performance, this metric is computed the same way as it was done in the *OctoMap*'s proposal paper, as previously shown in equation 2.3. This calculation is performed after the entire evaluation dataset has been processed. Furthermore, as stated in section 3.3, the memory consumption of the Octree should be significantly lower than the corresponding three-dimensional grid when mapping the same area, which is given by equation 3.3.

As explained in section 3.4.5, an objective evaluation of the occupancy map's **accuracy** is not possible as a dataset that would enable the construction of a ground-truth map is not available. Nevertheless, an explanation of how this type of accuracy evaluation would be done, in case such a dataset was provided, is here explained. As reviewed in section 2.6, a possible solution for obtaining a ground-truth map would be to install a high-resolution sensor, such as the Velodyne lidar, on the vehicle, so that it would scan the environment simultaneously to the radars and obtain more accurate measurements in relation to the corresponding ones from the radar. In the context of this work, the ground-truth map would be an Octree model which incorporates the detections from this high-resolution sensor. However, to insert measurements from the Velodyne lidar into the model, a custom inverse sensor model would have to be implemented specifically for this sensor. Furthermore, as reviewed in section 2.6, a ground extraction process would probably also have to be implemented, similar to the presented literature approach. As also explained in that section, an ideal inverse sensor model could be used to model the detections of this sensor, due to its high accuracy. Regarding the comparison process between the ground-truth Octree and the one obtained from the radars' measurements, the approach reviewed in section 2.6 which uses the ICP algorithm to align the maps and then computes the error using the ADNN distance metric, would be appropriate in this context, as it could be applied to three dimensions.

As stated in section 3.4.5, a visualization module was implemented to perform the accuracy evaluation of the Octree, which is further explained in chapter 5. A relevant feature of this visualization module, which is important to note for a better understanding of the results presented in the next section, is the correspondence between the color of each voxel and its occupancy probability, as shown in the next chapter, in figure 5.1. Another important detail is that not all voxels of the Octree are rendered in the visualization module. To limit the time it takes to traverse the Octree and obtain the desired voxels for their rendering, the visualization module uses a bounding box of 60 meters in width and length, and 15 meters in height is used around the vehicle so that only the nodes corresponding to positions inside it are retrieved from the Octree and rendered as cubes. The vehicle is visually represented with two cubes of unit edge length, using different colors: the front cube has a light yellow color and the back is colored with a light red color. Furthermore, to better visualize the ground limit and have a better notion of sizes, a grid with gray lines is rendered on the ground, where each cell has a size of 10 meters, as shown in figure 4.1.

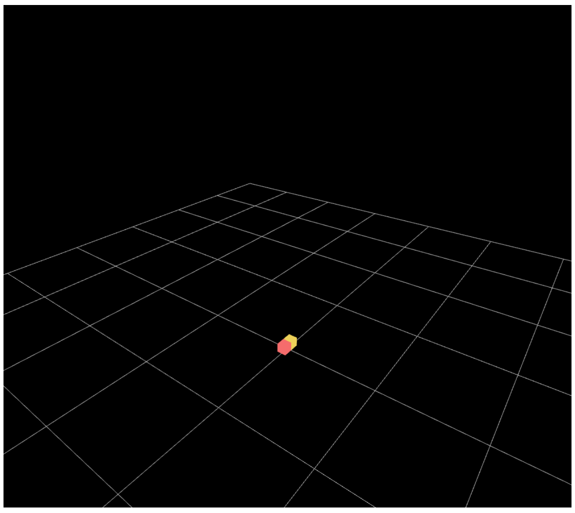
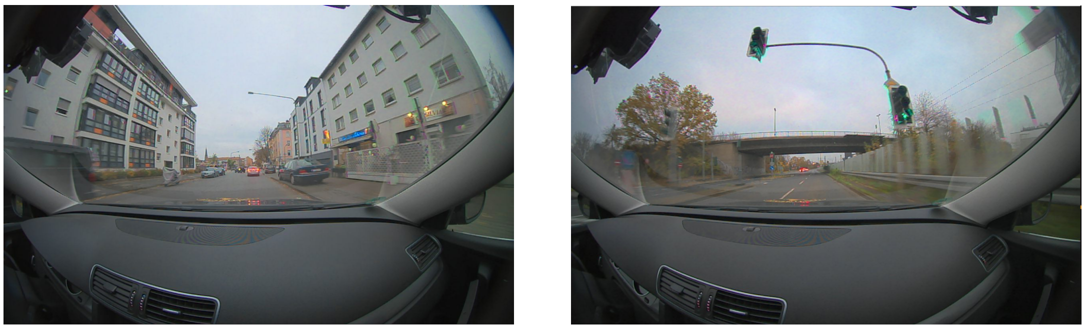


Figure 4.1: Visualization of the vehicle representation and the grid on the ground.

To better evaluate the obtained results in terms of accuracy, its analysis focused on two main scenes in each dataset which contain clear three-dimensional structures. The first scene, which is present on Dataset One, consists of a two-lane street, with multi-story buildings and parked cars on each side. The second scene focuses on a bridge and an elevated traffic light, recorded for Dataset Two. Figure 4.2 shows two images of these scenes from Dataset One and Dataset Two, respectively.


(a) Two-lane street scene

(b) Highway bridge scene

Figure 4.2: Special scenes from Dataset One and Dataset Two, respectively, used for accuracy evaluation.

4.5 Results

To better comprehend the overall results of the implemented occupancy map, and, in particular, how it models real-world three-dimensional structures, such as other vehicles, buildings, traffic lights, and bridges, this section starts off by performing an analysis of the visualization results of the Octree when modeling the particular scenes mentioned in the previous section, which were presented in figure 4.2. For this first analysis, a resolution of 0.2 meters and a tree depth of 16 levels were used. Furthermore, the calculations for the free space were disabled.

Figure 4.3 shows two images of the visualization of the obtained Octree corresponding to the two-lane street and the bridge scenes, respectively.

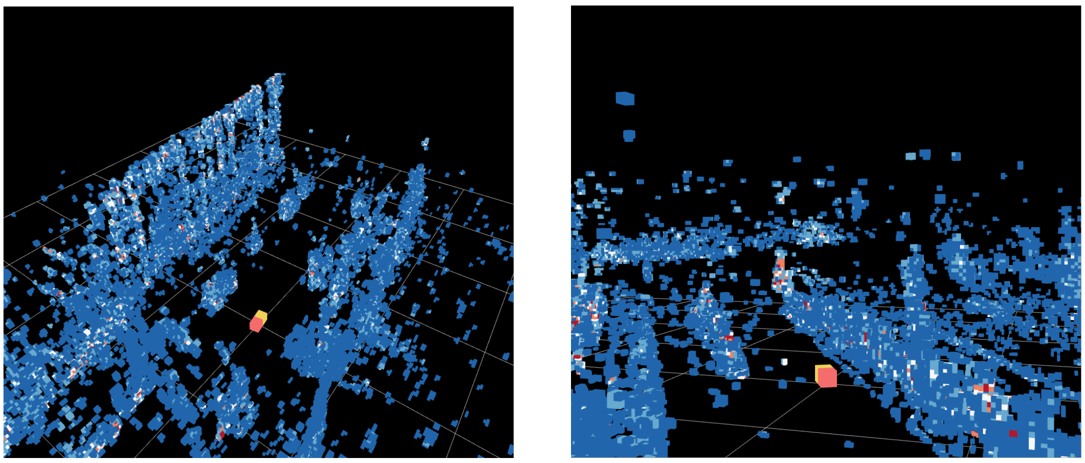

(a) Two-lane street scene (b) Highway bridge scene

Figure 4.3: Visualization of Octree for both evaluation scenes, with a resolution of 0.2 meters, a tree depth of 16 levels, free space calculations disabled, and using the Bayesian approach for sensor fusion.

In general, most visible voxels have low occupancy probabilities, represented by the dark blue color, as further explained in chapter 5. This is caused by the implemented radar sensor model, which, as previously explained in section 4.2, distributes the occupancy probabilities to voxels neighboring each detection, so that it decreases as the distance to the detection point increases. Therefore, in the visualization, it is expected that for most detections, the corresponding center voxels with high occupancy probabilities are occluded by the neighboring voxels with lower values. In particular, although not easily noticeable in the image, there are a few voxels with high occupancy in the detections corresponding to the closest parked car on the left of the vehicle, represented by the orange and red colors. Furthermore, there are quite a few misdetections scattered around the environment which do not correspond to any real object. These are likely caused by the following effects:

- Multipath effect. The same radio wave may traverse different paths in the environment. For example, it may be reflected from one object onto another and then back to the radar, leading to an incorrect calculation of the radial distance for that detection.

- Clutter. Unwanted reflections from the ground, rain, and animals or insects, for example, are received by the radar.
- Electromagnetic Interference (EMI). Unwanted electromagnetic waves from other noise sources affect radar performance.

In particular, by analyzing the visualization results presented in the two-lane street scene, and comparing them with the corresponding scene shown in figure 4.2, it is possible to identify the detections of the parked vehicles and the multi-story buildings on both sides of the street. Regarding the bridge scene, its visualization results clearly show the detections corresponding to the elevated traffic light in front of the vehicle, where some high occupancy voxels are visible, the road divisor on the left, with a few red voxels corresponding to the traffic light on it, the road limits on each of its sides and, finally, the bridge in the far front of the vehicle.

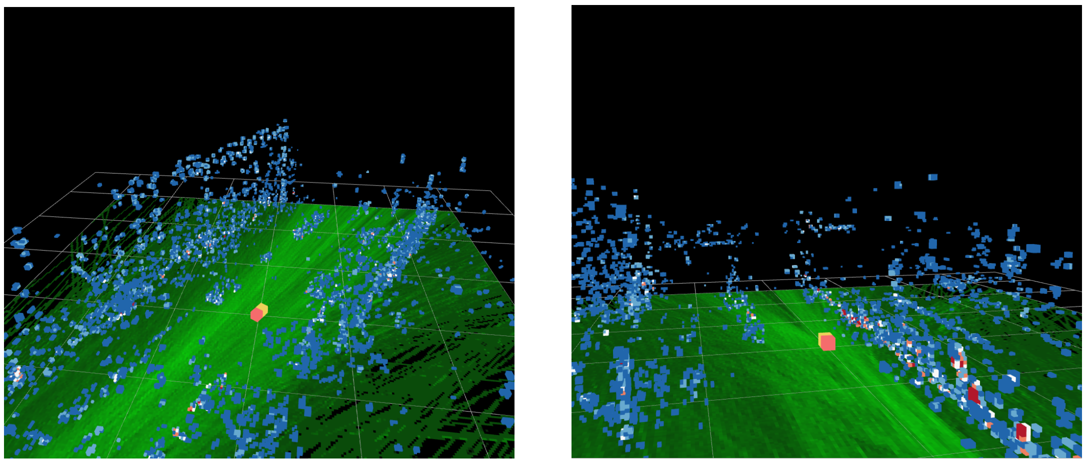
In the remaining part of this section, the results of the performance metrics mentioned in the previous section are presented and analyzed for each implemented occupancy mapping module explained in the previous sections, namely regarding the results while varying the Octree parameters in section 4.5.1, the evaluation of the implemented radar sensor model in section 4.5.2 and the comparison between the two implemented probability sensor fusion methods in section 4.5.3.

4.5.1 Octree Parameters

As previously mentioned in section 4.1, the Octree implementation allows customizing two parameters that directly affect the performance metrics mentioned in the previous section: the **resolution** and the **tree depth**. Furthermore, as also mentioned in that section, the computation of the **free space** voxels can be disabled, which creates another customizable parameter. In this section, these three parameters are evaluated by computing the performance metrics mentioned in the previous section over the entirety of each evaluation dataset, while changing their values and analyzing the obtained results. It is important to note that during this evaluation, the Bayesian approach was used as the probabilistic sensor fusion method, as it is expected to be the most time and space-efficient method.

The first parameter to be evaluated is the enabling of the **free space** computations since it is expected to have the most influence on the results. While obtaining the performance metrics with free space calculations enabled, the resolution parameter was set to 0.2 meters and the tree depth to its default value of 16 levels. Furthermore, to better comprehend the visualization results obtained with free space calculations enabled, it is important to understand the implementation for the rendering of the voxels corresponding to the free space, which is further explained in the next chapter, in section 5.2.

Figure 4.4 shows two images of the visualization of the obtained Octree with free space computations enabled, corresponding to the two-lane scene and the bridge scene, respectively.



(a) Two-lane street scene (b) Highway bridge scene

Figure 4.4: Visualization of Octree for both evaluation scenes, with a resolution of 0.2 meters, a tree depth of 16 levels, free space calculations enabled, and using the Bayesian approach for sensor fusion.

By comparing the visualization results with free space calculations enabled, presented in figure 4.4 and with those obtained previously with this parameter disabled, as shown in figure 4.3, a significant reduction in the number of occupied voxels is immediately noticeable. This is explained by the huge increase in computation time, as presented in table 4.2, which causes the application to stop functioning in real-time. This leads to the buffer which stores the received radar measurements to get completely full, and, consequently, some point clouds are not processed. However, it is still possible to make a correspondence between some sets of detections and the objects shown in the scene images, such as the road limits in both scenes, the parked car on the left of the vehicle in the two-lane scene and the road divisor in the bridge scene of figure 4.4. Furthermore, the rays from each radar to the detection points are easily noticeable since there are brighter green lines on the ground slightly oriented to the left and to the right, corresponding to the scans of the left and right radar, respectively.

Table 4.2 presents the mean insertion time, in milliseconds, of all radar scans, and the memory usage, in MB, of the whole Octree after inserting all radar scans, for each evaluation dataset, while varying the free space calculations parameter.

Table 4.2: Mean insertion time, in milliseconds, of all radar scans of, and the memory usage, in MB, of the whole Octree after inserting all radar scans, for each evaluation dataset, while varying the free space calculations parameter.

Free Space Setting	**Disabled**		**Enabled**	
Performance Metric	**Insertion Time (ms)**	**Memory Usage (MB)**	**Insertion Time (ms)**	**Memory Usage (MB)**
Dataset One	15.236	698	822.346	13,875
Dataset Two	16.303	722	831.872	18,320

The results presented in table 4.2 clearly show a significant increase in insertion time and memory usage when free space calculations are enabled. The decrease in time performance is caused by the computationally expensive three-dimensional ray-casting process, which, when associated with the long-range detections from the radars, causes a huge increment in processing time. Furthermore, these results show that without free space calculations, the implemented solution is able to fulfill the real-time requirement, previously defined in section 3.3. On the other hand, when the computations of the free voxels are enabled, the time results are significantly higher than the real-time threshold. Regarding the huge increase in memory usage, it is explained by the significant number of additionally affected voxels for each radar scan, when the free space calculations parameter is enabled since, for each detection in the received point cloud, all voxels between the sensor and this point are incorporated into the Octree. The memory usage requirement defined in section 3.3, defines the threshold for this metric according to equation 3.3, which, with the resolution used for this results of 0.2 meters and using the Bayesian theory for sensor fusion, sets its value for Dataset One mem_{one} to:

$$mem_{one} = \frac{1{,}043 \times 1{,}058 \times 23}{0.2^3} \times 4B = 12{,}690MB \tag{4.13}$$

The corresponding value for Dataset Two mem_{two} is equal to:

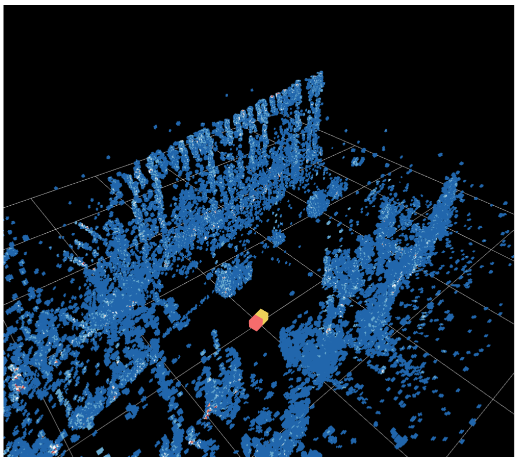
$$mem_{two} = \frac{5{,}136 \times 5{,}337 \times 23}{0.2^3} \times 4B = 315{,}225MB \tag{4.14}$$

By comparing the memory usage results presented in the table and the calculated required thresholds of each dataset, it's observable that the enabling of the free space calculations makes the defined threshold unachievable when the implementation is performed for Dataset One. This occurs due to the low value for the height of the mapped area using the three-dimensional grid, whereas, with an Octree, the size of each dimension of the mapped area must be equal, therefore, the height value always has significantly bigger values than required with this model. However, it is important to note that using the three-dimensional grid model with the dimensions used in the equations would require the before-hand knowledge of the size of the area to be mapped, which is not necessary with the Octree model, as the memory is allocated dynamically as new measurements are received.

Overall, the performance results regarding the free space calculations parameter clearly demonstrate that the implemented approach for determining the free voxels is not efficient enough to be considered appropriate for three-dimensional occupancy mapping, as it impedes the application to fulfill the real-time and memory usage requirements previously explained in section 3.3. As such, further evaluations of the other Octree parameters are performed with this parameter disabled.

To evaluate the performance impact of the **resolution** parameter, the implemented occupancy mapping solution was run on both datasets while varying this parameter between the values 0.1, 0.2, and 0.5, maintaining the tree depth at 16 levels and having the free space computations disabled.

Figures 4.5 and 4.6 show the visualization of the obtained Octree using a resolution of 0.1 and 0.5 meters, respectively, for both evaluation scenes. The results for a resolution of 0.2 meters were presented previously in figure 4.3.


(a) Two-lane street scene

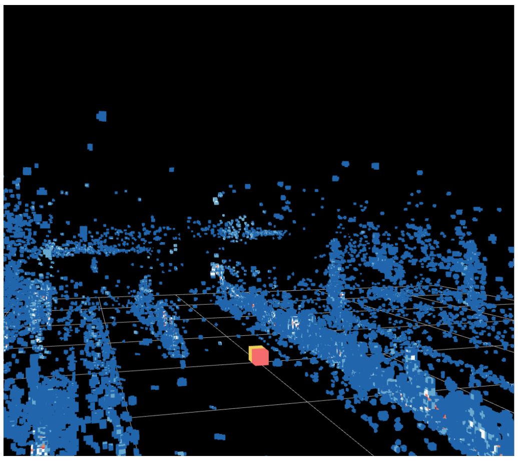
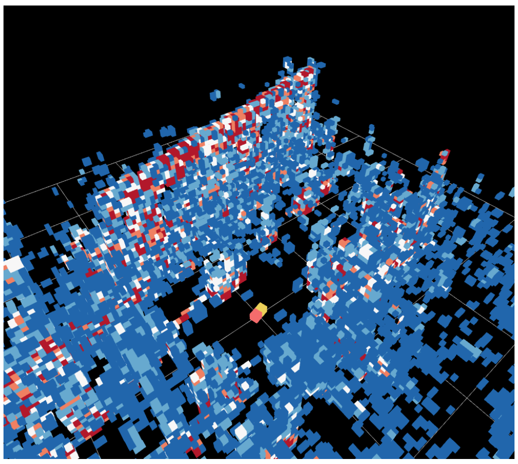

(b) Highway bridge scene

Figure 4.5: Visualization of Octree for both evaluation scenes, with a resolution of 0.1 meters, a tree depth of 16 levels, free space calculations disabled, and using the Bayesian approach for sensor fusion.

The most noticeable difference in the presented visualization results while varying the resolution parameter is in the occupancy probabilities of the voxels. With a resolution of 0.5 meters, each voxel is hit with more sensor beams, due to their increased size, which means it gets more occupancy updates, resulting in higher probability values. On the other hand, with a lower resolution value, such as 0.1 meters, each radar scan affects a higher quantity of voxels, which reduces the likelihood of each voxel being hit several times. Regarding the level of detail provided by the different values for this parameter, the visualization results indicate that with a resolution of 0.5 meters, there is an increased difficulty in identifying an object from a set of


(a) Two-lane street scene

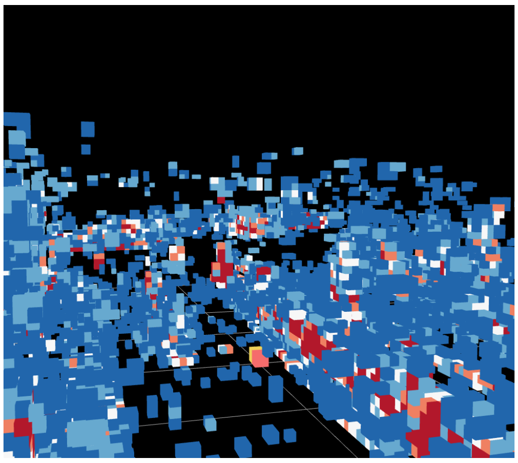

(b) Highway bridge scene

Figure 4.6: Visualization of Octree for both evaluation scenes, with a resolution of 0.5 meters, a tree depth of 16 levels, free space calculations disabled, and using the Bayesian approach for sensor fusion.

detections, as the bigger size of the voxels form a less detailed figure. On the other hand, the lower resolution of 0.1 meters offers the most level of detail, facilitating the identification of objects from a set of detections.

The time and memory usage results for the evaluation of the resolution parameter are presented in table 4.3.

Table 4.3: Mean insertion time, in milliseconds, of all radar scans, and the memory usage, in MB, of the whole Octree after inserting all radar scans, for each evaluation dataset, while varying the resolution parameter.

Resolution (m)	**0.1**		**0.2**		**0.5**	
Metric	**Insertion Time (ms)**	**Memory Usage (MB)**	**Insertion Time (ms)**	**Memory Usage (MB)**	**Insertion Time (ms)**	**Memory Usage (MB)**
Dataset One	32.665	2,460	15.236	698	5.663	150
Dataset Two	33.324	2,270	16.303	722	6.284	175

The results presented in the table show a linear decrease in insertion time and an exponential increase in memory usage with the increase of the resolution parameter. The time results are according to the expectations, as it is linearly dependent on the number of voxels affected by a radar sensor and, as explained before for the visualization results of this parameter, lower resolution values result in a higher number of voxels being hit by the sensor beam. Regarding memory usage, the presented results are similar to the ones presented in the *OctoMap* proposal paper and are explained by the increase in the size of each dimension.

Overall, the results of the performance metrics obtained for the evaluation of the resolution parameter confirm the expectations mentioned in section 4.1, regarding the relationship between this parameter and the accuracy of the model, where a lower value for this parameter increases the accuracy of the results. However, increasing this parameter brings benefits regarding the time and memory usage metrics. However, it is important to note that lower resolution values are required for certain ADS applications. For example, in the scenario of a tall truck going under a bridge, a higher resolution of 0.2 meters, would not be sufficient to correctly model the gap between the truck's roof and the bottom of the bridge. Therefore, in such situations, the compromise of obtaining worse time and memory consumption performance results needs to be made.

Considering these results, further evaluations are performed using a resolution of 0.2 meters.

The evaluation of the **tree depth** parameter does not require an analysis of the accuracy results, as it is not affected by this parameter. Therefore, only the time and memory usage performance metrics are computed and analyzed. As previously explained in section 4.1, it is expected that by decreasing the value of the tree depth parameter, the point cloud insertion time is reduced. However, as also stated in that section, this parameter limits the size of each dimension of the mapped area. Therefore, while varying this parameter to perform its evaluation, it cannot be set to a value that is too low for the mapped area of the evaluation datasets, which were previously shown in table 4.1. The minimum tree depth value which allows the Octree to model the mapped area of Dataset Two is 15, which limits the maximum size for each dimension to:

$$lim = 2^{15} \times 0.2 = 6,553.6 \text{ meters} \tag{4.15}$$

Considering this limitation, the evaluation of the tree depth parameter is performed by varying its value between 15, 16, and 17 levels and then analyzing and comparing the obtained time and memory usage results, which are presented in table 4.3.

Table 4.4: Mean insertion time, in milliseconds, of all radar scans, and the memory usage, in MB, of the whole Octree after inserting all radar scans, for each evaluation dataset, while varying the tree depth parameter.

Tree Depth (levels)	**15**		**16**		**17**	
Metric	**Insertion Time (ms)**	**Memory Usage (MB)**	**Insertion Time (ms)**	**Memory Usage (MB)**	**Insertion Time (ms)**	**Memory Usage (MB)**
Dataset One	14.890	687	15.236	698	15.873	737
Dataset Two	15.893	693	16.303	722	16.783	783

The results presented in the table show a minor increase in insertion time and memory usage with an increase in the value of the tree depth parameter. However, considering that an increase of just one unit in the value of this parameter doubles the size of each dimension in the mapped area, the small decrease in performance is insignificant. Therefore, for certain scenarios where it is expected that the area to be mapped is quite large, increasing this parameter by a few units may be an appropriate approach as it would not have a significant negative impact on the occupancy mapping performance.

4.5.2 Radar Sensor Model

To evaluate the implemented radar sensor model, an artificial point cloud was created, allowing for easier visualization of the distribution of the occupancy probability for each voxel corresponding to one radar detection. Regarding the evaluation of the time and memory usage metrics, the previous section's results already included the implemented sensor model so they are not analyzed again in this section. The artificial point cloud was obtained by creating a list of points in polar coordinates while varying each dimension as follows:

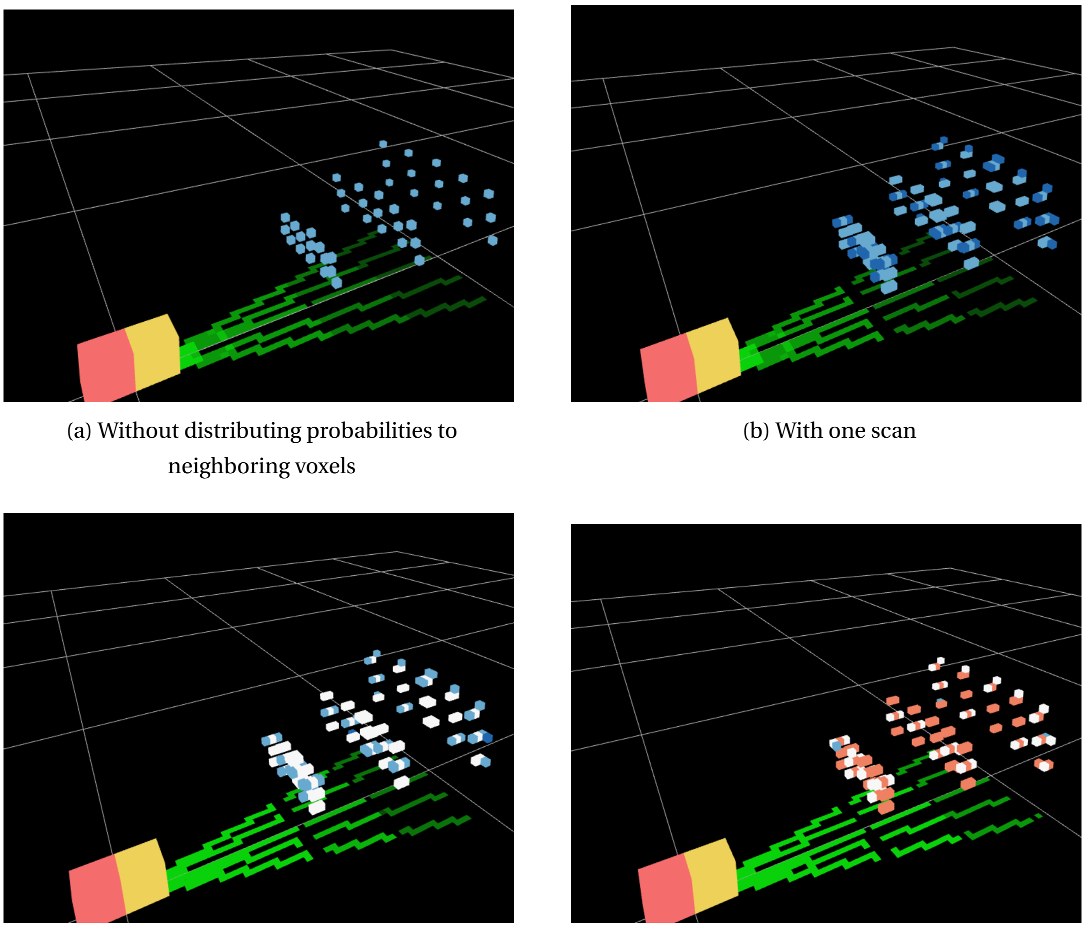
- Radial distance from 5m to 10m, with an interval of 2.5m,
- Azimuth angle from -15° to 15°, with an interval of 7.5°.
- Elevation angle from 5° to 15°, with an interval of 5°.

Furthermore, the RCS of each detection in the artificial point cloud was set to zero.

Based on the overall results for the evaluation of the Octree parameters presented in section 4.5.1, the evaluation of the radar model was performed using a resolution of 0.2 meters, and a tree depth of 16 levels. The free space calculations were enabled so that the rays from the vehicle to each detection of the artificial point cloud could be visualized. Furthermore, the Bayesian Probability Theory was used as the sensor fusion approach.

To better understand the results regarding the distribution of probabilities, it is important to note the correspondence between the color of each voxel and its occupancy probability, as explained in chapter 5.

Figure 4.7 shows a visualization of the application of the radar sensor model to the artificial point cloud, by first presenting it with the distribution of the probabilities to neighboring voxels disabled, and then enabling it and showcasing the results of occupancy increments when voxels are updated through multiple radar scans.



(a) Without distributing probabilities to neighboring voxels

(b) With one scan

(c) With two scans

(d) With three scans

Figure 4.7: Visualization of the application of the radar sensor model to an artificial point cloud, by first presenting it with the distribution of the probabilities to neighboring voxels disabled, and then enabling it and showcasing the results of occupancy increments when voxels are updated through multiple radar scans.

By comparing the first two images of figure 4.7, the distribution of probabilities by the radar sensor model is observable, as new voxels appear in the second image around each point. Furthermore, the effect of the Gaussian distribution implemented by the radar model is noticeable, as these new voxels have lower occupancy probabilities, represented by their darker blue color. The following two images show how the occupancies of the voxels increase with each new scan which updates their probabilities. The green lines on the ground representing the free space voxels show the effect of the approach explained in section 4.2 in which there is a preference for updating a voxel as occupied instead of free, in case it is identified as both in a single scan since there is a visible gap in the rays below the first set of voxels in front of the vehicle. Furthermore, these green rays are brighter closer to the vehicle and darker the further away they are from it, which results from the higher quantity of updates to the free space voxels closer to the vehicle.

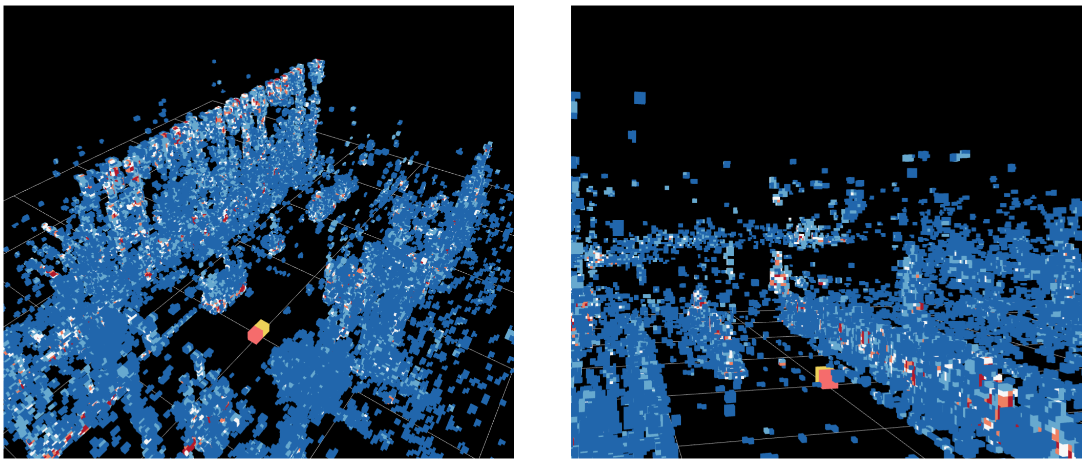
4.5.3 Probabilistic Sensor Fusion

As previously explained in section 4.3, two probabilistic sensor fusion approaches were implemented: **Bayesian Probability Theory** and **Dempster-Shafer Evidence Theory**, which differ in the way the occupancy probability of each voxel is updated, stored, and calculated.

Based on the overall results for the evaluation of the Octree parameters presented in section 4.5.1, the evaluation of the two probabilistic sensor fusion methods is performed using a resolution of 0.2 meters, a tree depth of 16 levels, and with free space calculations disabled.

The performance metrics for the evaluation of the Bayesian Probability Theory, using the aforementioned Octree parameters, have already been presented in figure 4.3, regarding the visualization results, and in all previously presented tables, such as table 4.2.

The visualization results of the obtained Octree, for the two evaluation scenes, using the Dempster-Shafer Evidence Theory approach for sensor fusion, are shown in figure 4.8.



(a) Two-lane street scene

(b) Highway bridge scene

Figure 4.8: Visualization of Octree for both evaluation scenes, with a resolution of 0.2 meters, a tree depth of 16 levels, free space calculations disabled, and using the Dempster-Shafer approach for sensor fusion.

When comparing the visualization results obtained with the Bayesian Theory method for sensor fusion, shown in figure 4.3, with the results of the Dempster-Shafer approach, there is not a clear difference between them. However, the occupancy values of the voxels are slightly higher with the Dempster-Shafer method, as can be observed in the set of detections corresponding to the parked car on the left side of the vehicle in the two-lane street scene. Furthermore, these higher probabilities cause an overall increase in the number of voxels corresponding to misdetections, as can be seen in the bridge scene, where the quantity of dark blue voxels in the middle of the street is noticeably higher compared to the results of the same scene when using the Bayesian method.

Table 4.5 shows the time and memory usage results for the two probabilistic sensor fusion methods, for each evaluation dataset.

Table 4.5: Mean insertion time, in milliseconds, of all radar scans, and the memory usage, in MB, of the whole Octree after inserting all radar scans, for each evaluation dataset, while varying the probabilistic sensor fusion approach.

Probabilistic Sensor Fusion Approach	**Bayesian Probability Theory**		**Dempster-Shafer Evidence Theory**	
Metric	**Insertion Time (ms)**	**Memory Usage (MB)**	**Insertion Time (ms)**	**Memory Usage (MB)**
Dataset One	15.236	698	10.528	911
Dataset Two	16.303	722	11.073	952

The results presented in the table show that the Dempster-Shafer approach consumes more memory but takes less time to update the map compared to the Bayesian method. The difference between the two approaches, regarding the time results, is explained by the need of having to constantly transform the *log-odds* ratio stored in the node to a probability value, and vice-versa, in the Bayesian Theory method, whereas, in the Dempster-Shafer approach, the combination rule uses the probability value directly to update the masses stored on the node. However, as explained in section 4.3, the Dempster-Shafer method requires the storage of another value on each voxel. In a 32-bit architecture, this equates to an additional four bytes for each node of the Octree, which leads to the higher memory consumption observed in the table compared to the Bayesian Theory method.

Overall, the performance results for the two probabilistic sensor fusion methods implemented do not show a clear advantage of one approach over the other. Regarding the accuracy metric, using the Dempster-Shafer method increases the number of misdetections, which may create a disadvantage for this approach. In relation to time and memory usage performances, both methods are equivalent, as the Bayesian method consumes less memory but takes more time to update the Octree. However, the higher insertion time of this approach could be mitigated, since, as explained before, it is related to the frequent operation of transforming the *log-odds* ratio to probability, and vice-versa, which could be avoided by adapting the implemented radar sensor model to work with and output *log-odds* ratio values instead of probabilities.

Chapter 5

Visualization

As reviewed in section 3.4.5, an objective evaluation of the accuracy of the occupancy map was not possible. Therefore, a visualization module was implemented to perform a subjective evaluation approach, by comparing the rendered model with the corresponding expected results. Furthermore, considering the goal of sensor prototyping of the implemented system, direct visualization of the sensors' scans increases the ease of evaluation of their performance.

This chapter focuses on the implemented solution for the visualization of the Octree environment model.

As stated in section 3.4.6, visualization of the solution is performed using the *wxWidgets* tool, to create the Graphical User Interface (GUI) window and handle user input, alongside the *OpenGL* library, to render the three-dimensional environment model in the GUI window.

The Octree environment model is composed of nodes that correspond to three-dimensional volumes (voxels) positioned in the world and which store the probability of being occupied, either with a log-odds representation, if the Bayesian Theory is used to perform sensor fusion, or with mass functions, in case the Dempster-Shafer Theory of Evidence is used, as explained in the previous chapter. Furthermore, a voxel is considered occupied if its occupancy probability is greater than a predetermined threshold, or as free, otherwise. Visualizing both occupied and free voxels in an easy-to-interpret way is a challenging task, considering that if both were to be rendered in the same scene, the occlusion of the occupied voxels by the free ones, would not allow for an easy interpretation of the environment model and the real-world three-dimensional structure it represents. Therefore, the implemented visualization only renders occupied voxels in their respective world positions, whereas free voxels are rendered as a two-dimensional grid at ground level, as further explained in section 5.2.

So that the visualization module is able to accurately represent the Octree environment model obtained during the three-dimensional occupancy mapping process and, specifically, the uncertainty associated with each radar measurement, it is important to have a visual representation of the occupancy probability of each voxel. A possible approach would be to compute the RGB encoding of each voxel in a continuous way, using a color gradient based on its occupancy probability. For example, a red gradient could be used so that voxels with low occupancy values would have a light red color, and, on the other hand, voxels with high occupancy probabilities would be colored with a dark red. However, with this approach, it would be difficult to estimate the actual occupancy value of each voxel based only on the visualization, since it would favor a relative comparison between the various voxels. Therefore, in the implemented solution, the interval of possible probabilities for the occupied voxels, $[0.5, 1.0]$, is divided into five different levels and a divergent color map is used to facilitate the visual differentiation between low and high occupancy voxels. The *RdBu / 5* color sequence was used so that low occupancy voxels have a blue color, and therefore, are easily distinguishable from the red color of the high occupancy voxels. Figure 5.1 shows the correspondence between the occupancy probability levels and the colors of the *RdBu / 5* sequence.

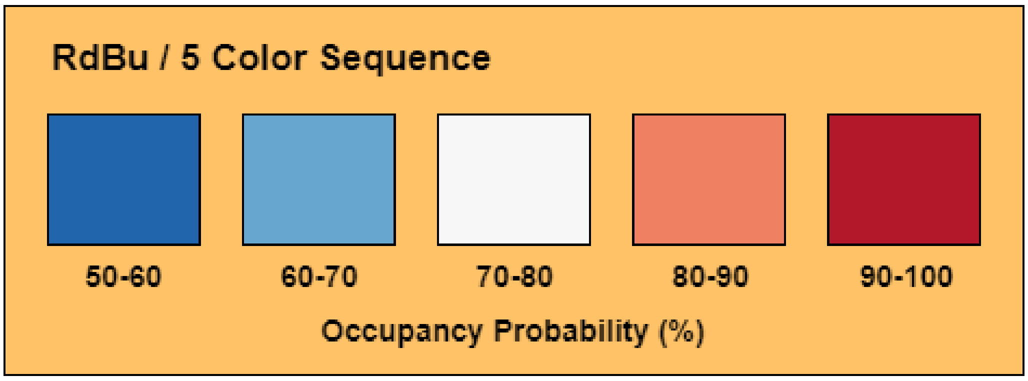


Figure 5.1: Correspondence between *RdBu / 5* color sequence and voxel occupancy probability.

To efficiently render the list of voxels that compose the environment model, different methods were implemented, which are detailed in section 5.1. The methodology for their evaluation is detailed in section 5.3, followed by the corresponding results in section 5.4.

5.1 Rendering Methods

The implemented visualization module receives a list of voxels, where each one is defined by the three-dimensional coordinate position of its center, its size, and its color in RGB encoding.

In the implemented approach, each voxel is rendered as a cube with an edge length equal to the size of the corresponding voxel. However, as reviewed in section 2.7, *OpenGL* does not offer methods capable of directly rendering three-dimensional figures, such as cubes. Therefore, the implemented approach consists of rendering each cube face independently. Furthermore, as also mentioned in section 2.7, using triangles over quads is preferred. Therefore, all implemented methods described further in this chapter, use two triangles to draw each face of the cube.

5.1.1 Immediate Mode

As reviewed in section 2.7, the **immediate** mode is the most simple *OpenGL* rendering method. In this method's implementation, the voxels are rendered sequentially and a render call is made for each corresponding cube. Furthermore, the vertices of each cube are not defined in their correct position. Instead, a cube with an edge of unit length is rendered for each voxel, with its center placed on the coordinate system's origin. To obtain the correct positioning and size of the cube on the screen, the view matrix is transformed according to the corresponding voxel's position and size, before rendering each cube. In addition, since each cube face is defined with two triangles, a total of six vertex specifications are necessary, as two vertices are repeated on both triangles.

5.1.2 Vertex Arrays

Using **vertex arrays** to store the vertices data allows for an improvement upon the previous approach by reducing the number of render calls to only one per list of voxels, as reviewed in section 2.7, instead of one per voxel as it is done in the immediate mode. This reduces CPU overhead and, therefore, improves time performance. Furthermore, this method eliminates the repetition of some vertices when defining the triangles of each cube face, by using an index array, as explained further in this section.

To create the vertex array, the voxels list is processed, where for each voxel the positions of the corresponding set of eight vertices are calculated to form a cube with unit edge length. To scale the cube to its correct size, each vertex is then multiplied by the voxel size. After these calculations, the set of vertices is added to the vertex array. This results in a single one-dimensional array containing the vertices of all cubes, where every three consecutive values in the array correspond to the three-dimensional position of a vertex, and every eight consecutive vertices correspond to a cube, as shown in the diagram of figure 5.2.

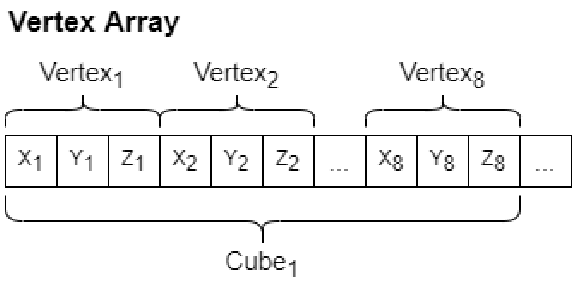


Figure 5.2: Vertex Array composition diagram.

To specify the color of each cube, the same principle is applied to a color array. Since the coloring in *OpenGL* cannot be directly applied to a primitive but, instead, must be specified for each vertex, each RGB encoding value in the color array is repeated eight times, one for each of the vertices of the cube. This way, every three consecutive values of the color array correspond to the RGB encoding for the color of the corresponding vertex in the vertex array.

Regarding the previously mentioned index array, every three consecutive indices define the positions in the vertex array of the vertices which compose a triangle. Furthermore, every 36 consecutive values of the index array define one cube, as 12 triangles are required to draw a cube (8 faces $*$ 2 triangles per face) and each triangle has three corresponding indices. By using this approach with indices, the required size of the data which needs to be sent to the GPU is reduced. In case no indices were used, each cube would require the definition of 36 vertices, where each one is defined by three values (X, Y, Z), resulting in a total of $36 * 3 = 108$ values per cube. On the other hand, by using indices, each cube requires only 36 index values and eight vertices, resulting in a total of $36 + 8 * 3 = 60$ values.

With all three arrays calculated (vertex, color, and indices), the rendering is performed by sending them to the GPU while specifying the triangle primitive and the sizes of each array.

5.1.3 Vertex Buffer Objects

This method consists of using *OpenGL* **vertex buffer objects** to store the vertex, color, and index array, defined in the previous section for the vertex arrays approach. The main advantage of this method is that it allows for the separation between the sending of data to the GPU and the actual rendering. In applications in which the data does not change frequently, this advantage may lead to a significant increase in time performance, since the data does not need to be repeatedly sent to the GPU on each render call, as reviewed in section 2.7. However, in the implemented solution, the voxels list is constantly changing due to the receiving of new radar detections at each time step. Therefore, this approach may not offer a major time performance increase in comparison with the previous one. However, an improvement can still be achieved by filling up the indices buffer on the first render call, as it is always composed of the same values independently of the vertex specifications, so that the following render calls will only need to send the vertex and the color array to the GPU, reducing the total transferred data size and, consequently, improving time performance.

5.1.4 Instanced Arrays

This approach intends to further improve time performance on top of the previous method, by further reducing the size of the data sent to the GPU and by reducing the CPU overhead associated with the calculations of the vertices positions for each cube, moving them to the GPU. This is achieved by performing **instancing**, which consists of rendering the same figure multiple times while varying its position and its color using *OpenGL* shaders, as reviewed in section 2.7. With this method, the vertex buffer only needs to be filled with the positions of eight vertices corresponding to a cube template. However, to perform the correct transformation on this template, the vertex shader needs to receive the position of each cube. Therefore, a new buffer is created to contain the center position of each voxel, which is used by the vertex shader to translate each vertex of the cube template. Regarding the colors buffer, its size is significantly reduced compared to the previous method, since it only needs to contain one RGB encoding per cube, as the fragment shader attributes the received color to each vertex of the cube.

5.2 Free Space Visualization

As previously mentioned in this chapter, visualizing the free space, defined by the Octree nodes which store an occupancy probability below the predefined threshold, is a challenging task. In case the voxels corresponding to these nodes were rendered the same way as the occupied voxels, most of these would be occluded as the number of free voxels is significantly higher, which would limit the quality of the evaluation. Therefore, the implemented approach to visualize free space consists of rendering a two-dimensional grid on the ground, where each cell is colored according to the average of occupancy probabilities for all free voxels in the Octree in the corresponding (X, Y) world position. To do this, a hash map is first created to calculate and store the average occupancy probabilities of each grid cell with a corresponding free voxel in the Octree, using an iterative mean formula, as follows:

$$p_{t+1}(x) = p_t(x) + \frac{1}{t+1}(x - p_t(x)) \tag{5.1}$$

After all Octree voxels are processed and the hash map contains all average occupancy probabilities of the corresponding cells, the rendering of the grid is performed using one of the rendering approaches previously mentioned in this chapter, which were all extended with a new method to render quads instead of cubes. To visually represent the average occupancy probability of each cell, a gradient of green is used to color the corresponding quads on the ground. The following is the implemented formula to calculate the RGB encoding of a quad at grid position ij, based on the corresponding voxel's occupancy probability p and the predefined occupancy threshold OT:

$$RGB_{ij}(p) = \begin{cases} R = 10 \\ G = 30 + 225 * (1 - (p/\text{OT})) \\ B = 10 \end{cases} \tag{5.2}$$

This formulation allows cells that have a lower occupancy value to be colored with brighter green and, on the other hand, cells with higher occupancy probabilities have a darker green color. Since free voxels are guaranteed to have occupancy values below the occupancy threshold OT, the term p/OT scales it up, allowing a higher gradient of colors.

5.3 Evaluation

The aforementioned implemented rendering methods have been evaluated based on their time performance when drawing a huge number of cubes, organized as a three-dimensional grid.

Furthermore, to showcase the coloring capability of each individual cube, a color gradient based on their position was implemented. Figure 5.3 shows a rendering example of this grid with a total of 1000 cubes with an edge length of one unit.

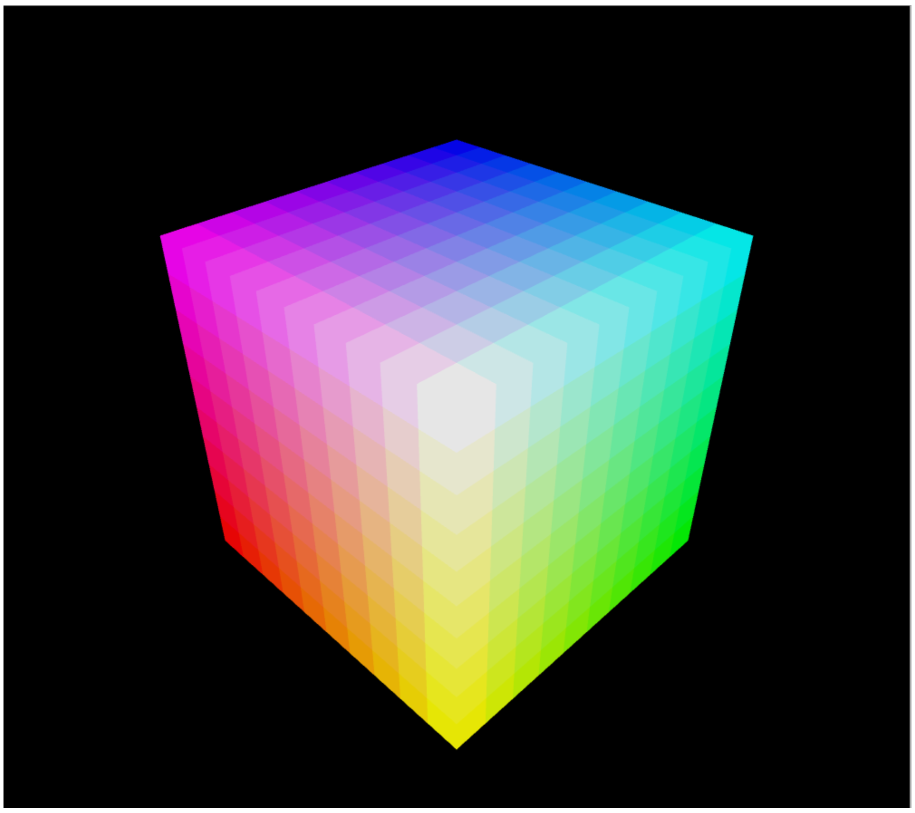


Figure 5.3: Grid of 1000 cubes with unit edge length used for evaluation.

The time performance of each rendering method is measured based on the mean drawing time of the grid over multiple consecutive frames, which is calculated in milliseconds. Furthermore, the rendering time of each frame is obtained by measuring the timestamp differences between the moment the list of voxels starts being processed and the final method call which sends the vertex data to the GPU. In addition, to evaluate the scaling capabilities of each rendering method, increasing sizes for the grid of cubes were applied.

It is important to note that the time performance of the implemented rendering methods is only relative to the processing which occurs on the CPU, as the computation time of the GPU calculations is not taken into account to evaluate the implemented approaches.

During the evaluation, a frame rate of 10Hz was used and a total of 500 frames were analyzed for each implemented method. To obtain real-time performance, the rendering time of each mode needs to be below a certain value T_{min}, which, according to the frame rate used of 10Hz, is equal to:

$$T_{min} = \frac{1}{10\text{Hz}} = 0.1s = 100ms. \tag{5.3}$$

The time results presented in section 5.4, were obtained by running the visualization solution, while varying the four implemented rendering approaches, on a laptop with a CPU Intel i5-6300U @ 2.40 GHz (2.50 GHz boost) and 8 GB RAM.

5.4 Results

The full results of the rendering times for all four implemented methods when drawing different amounts of cubes, from $1,000$ to $1,000,000$ cubes, are shown in appendix D. In this section, a discussion and analysis of these results are presented.

Figure 5.4 presents a line graph visualization of the rendering time results of the four implemented modes when drawing multiple cubes, from $1,000$ to $10,000$ cubes.

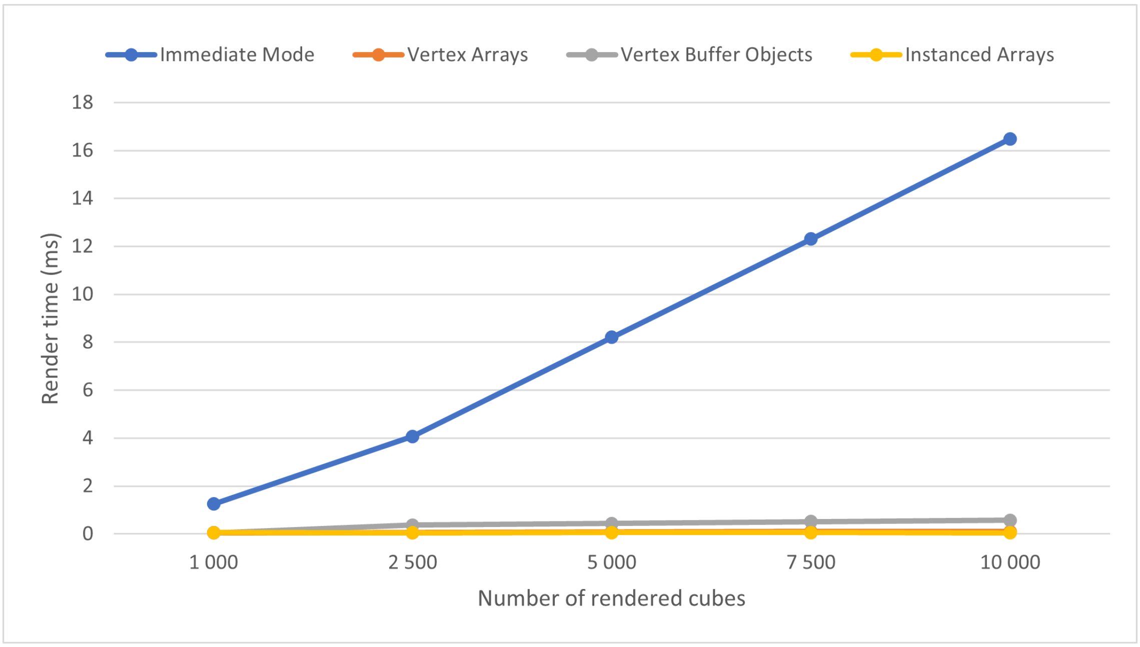


Figure 5.4: Line graph visualization of the rendering time results of the four implemented modes over an increasing number of cubes, from $1,000$ to $10,000$.

As can be observed in figure 5.4, there is a clear difference in time performance between the immediate mode and the other three methods. The immediate mode is much slower, as its rendering times are much higher for the same amount of rendered cubes compared to the other three modes. Furthermore, the time results of the immediate mode increase linearly and much faster alongside the increase in the number of rendered cubes. On the other hand, the rendering times of the other three methods for the presented number of cubes are very close to zero and barely have an increase in render time relative to the immediate mode. This huge difference in rendering time results can be explained by the overhead of performing a rendering call for each individual cube in the immediate mode, as previously explained in section 5.1.1. Since the other three modes perform grouping of all the data into arrays so that it can be sent to the GPU with only one call, the overhead observed in the immediate mode is avoided, resulting in much lower rendering times for these methods.

Based on these results, it is possible to affirm that the immediate mode is not an appropriate method for real-time rendering of numerous cubes, as its rendering time quickly rises with the increase of the number of rendered cubes. Furthermore, as it can be seen in the full results table in appendix D, this visualization mode is already barely able to achieve the real-time performance threshold, established in the previous section according to the set frame rate, when rendering only 50,000 cubes.

Considering that for the quantities of rendered cubes used for the results of figure 5.4 the rendering times of the vertex arrays, vertex buffer objects, and instanced arrays modes are practically zero, which impedes a fair time performance comparison between these three methods, a new set of results was calculated using higher numbers of rendered cubes, from 10,000 to 1,000,000, for these three approaches. Figure 5.5, shows a line graph visualization of the obtained rendering times.

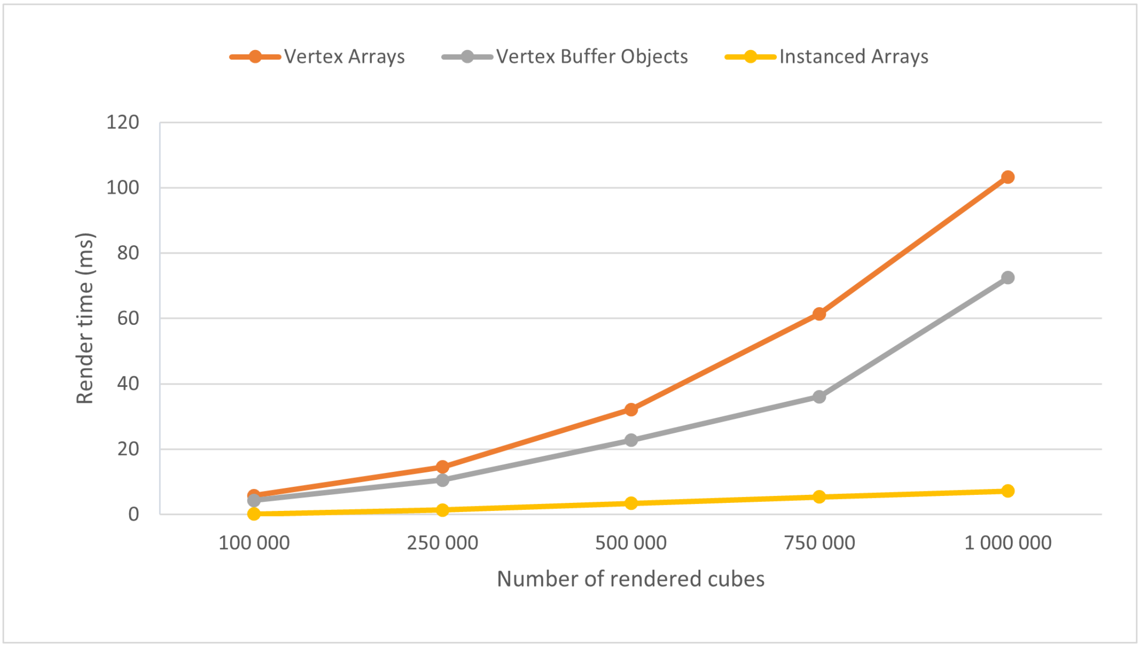


Figure 5.5: Line graph visualization of the rendering time results of the best three rendering methods over a higher number of cubes, from 100,000 to 1,000,000.

The results shown in figure 5.4 clearly demonstrate the time performance improvements, predicted in section 5.1, of the instanced arrays mode over the vertex buffer objects approach, and of this last method over the vertex arrays one. Since these modes only perform one rendering call per frame which sends the data to the GPU, their rendering time is directly related to the computation time of these data and the time it takes to send it to the GPU. Therefore, by reducing the memory size of these data, the time performance is proportionally increased, as explained in section 5.1. Considering this, the obtained results are justified:

- The major difference between the vertex arrays mode and the vertex buffer objects method is that the latter reduces the data size of each frame relative to the former approach, by filling the indices array and sending it to the GPU only once on the first rendered frame. This explains the difference in render time between these two methods, as the further increase in the number of rendered cubes attenuates their performance gap.

- Regarding the much better time performance of the instanced arrays mode, it can be justified by the decrease in CPU computation due to the use of shaders, which move the vertex calculations to the GPU, as explained in section 5.1.4, and the significant reduction in data size, as the vertex and color arrays have their sizes reduced by a factor of eight and the indices array by a factor equal to the number of rendered cubes, as also explained in section 5.1.4.

Regarding the real-time performance of these three methods, the vertex arrays and the vertex buffer objects modes are not able to achieve it when rendering $250,000$ or more cubes. On the other hand, the instanced arrays approach is still able to achieve render times below the real-time performance threshold even for $1,000,000$ cubes.

These results show a clear advantage of the instanced arrays mode over the other implemented approaches regarding time performance, as it is still able to achieve real-time results for huge amounts of cubes. However, as mentioned in section 2.7, this approach has the downside of limiting the geometry diversity of the rendered three-dimensional objects, as it requires that all rendered cubes have the same size. Furthermore, for certain rendering applications, the rendering performance may be improved through a smarter approach consisting of rendering one bigger cube instead of numerous smaller cubes. However, this approach would not be possible with the instanced arrays mode. Therefore, for certain situations, the vertex buffer objects may be preferred.

Chapter 6

Conclusion

To finish this report, this chapter presents a summary of the implemented solution, the obtained results, and the corresponding conclusions in section 6.1. In section 6.2, possible future improvements to this work are suggested.

6.1 Conclusions

The main goal of this work was to analyze the required adaptation of a traditional two-dimensional static occupancy grid mapping approach into the three-dimensional space, which was very successfully achieved since the implemented three-dimensional occupancy mapping application is able to achieve results that fulfill all of the presented system requirements. Furthermore, the compatibility of the implemented solution with the PDK further enhances this system's capability of facilitating and accelerating radar sensor prototyping.

The proposed solution involved first adapting the two-dimensional mapping approach to handle data from the ARS540 long-range radar sensor, which is capable of retrieving the elevation angle of each detection and, therefore, providing the required three-dimensional data. Secondly, each module of the two-dimensional grid mapping approach was transformed to function in the three-dimensional space, by first replacing the base grid model with *OctoMap*'s Octree implementation, which was then adapted to have a more complex sensor model, built specifically for the ARS540 radar sensor, and to implement two probabilistic low-level sensor fusion approaches, the Bayesian Theory and the Dempster-Shafer Evidence Theory, which were then compared based on their performance results.

The implemented solution was evaluated on two datasets recorded in real-world scenarios, using a vehicle installed with two radars positioned in its front. An objective evaluation of the accuracy of the obtained Octree map was not possible, as a dataset that would allow the construction of a ground-truth map with which the results could be compared was not available. Nevertheless, a subjective evaluation of the accuracy of the implemented solution was performed by implementing a visualization module that allows for a visual comparison of the obtained Octree and the corresponding real-world scenario observable through the camera footage recorded alongside the radar measurements. Furthermore, the evaluation also took into account the insertion time, consisting of the time it takes for a point cloud, obtained from a single radar scan, to be incorporated into the Octree. In addition, the memory consumption of the entire Octree model after insertion of all radar scans from the evaluation dataset was also analyzed.

The results with the default Octree parameters, consisting of a resolution of 0.2 meters and a tree depth of 16 levels, and free space calculations disabled were very positive, as the corresponding visualization allowed for the identification of the detected objects, real-time performance was achieved, and low memory consumption was required, showcasing the successful implementation of a three-dimensional occupancy mapping application. However, with free space calculations enabled, the desirable performance metrics were not obtained, as the expensive ray-casting operation makes each update take significantly more time than the real-time threshold and required a lot more memory consumption due to all of the added free voxels composing the free space. Therefore, for the implemented solution to be appropriate for real-world occupancy mapping applications, the compromise of not computing the free space needs to be made.

Furthermore, the other Octree parameters, including the resolution and tree depth, were also evaluated. The results indicate that better accuracy is achieved with a lower resolution value. However, the time performance increases linearly, and the memory usage performance increases exponentially with the resolution value. Regarding the tree depth, the results show that an increase of just a few units to this parameter worsens the performance metrics just slightly while having the significant benefit of doubling the size of the total mapped area.

Regarding the comparison made between the two probabilistic low-level sensor fusion methods: Bayesian Probability Theory and Dempster-Shafer Evidence Theory, the results indicate very little difference between the two methods in terms of accuracy. However, the insertion time was lower with the Dempster-Shafer approach, while the memory consumption was higher with this method.

Overall, the presented work was very successful since the implemented three-dimensional occupancy mapping process is able to solve the presented problem, as it achieves good accuracy, real-time performance, and low-memory consumption, while being able to create a three-dimensional environment model that is able to correctly represent three-dimensional and over-hanging objects, such as bridges, traffic signs, traffic lights, and trees, with the compromise of not computing free space and using a higher resolution value.

Regarding the implemented visualization module, the comparison results between the different implemented rendering methods show a clear advantage of the instanced arrays, which was demonstrated to be a highly-efficient method for rendering millions of voxels.

6.2 Future Work

The following is a list of possible future improvements to the presented work, which were not possible to be implemented, either due to time constraints or lack of resources:

- **Obstacle detection**: With the occupancy grid map fully constructed after the sensor fusion module, an obstacle detection process could be performed with the ultimate goal of obtaining their position and then extracting a high-level representation of them. In [49], a two-dimensional grid is built after fusing data from radar and lidar sensors to detect obstacles, using a connected components technique.

- **Clustering**: The most noticeable deficit in terms of accuracy was the presence of misdetections. To mitigate these, a clustering process could be explored, which would alter the point cloud before its insertion into the Octree, with the goal of removing those points that do not correspond to any obstacle. There are many approaches in the literature for this purpose, such as hierarchical clustering [74] [75]and k-means clustering [76] approaches.

- **Improved Octree**: As reviewed in section 2.2, there are proposals in the literature to improve the *OctoMap*'s implementation of the Octree. Therefore, as a way to improve the performance of the implemented solution, some of the reviewed proposals could be explored. Furthermore, as the results in section 4.5 showed, enabling the calculations of the free space imposes major limitations to the occupancy mapping application, as it makes the insertion time significantly higher than the real-time threshold. This is likely caused by the computationally expensive ray-casting process which needs to be performed to determine the voxels between the sensor and each detection of the received point cloud. The proposal of *UFOMap* [33] showcases an interesting approach to improve the performance of the ray-casting operation when computing the free space voxels of the Octree, consisting of using a higher voxel resolution while doing ray-casting. Another alternative for improving Octree's performance is to explore the use of GPUs, which are very efficient at performing ray-casting. One option for such an implementation is to use CUDA [77], which is a parallel computing platform and Application Programming Interface (API) that allows for general-purpose processing using GPUs. In [78], an Octree is implemented for environment modeling in the context of robotic collision detection. However, this tool is NVIDIA proprietary, limiting its use to only this kind of GPU. Considering this, a possible alternative is to use OpenCL [79], which is a framework that allows for the execution of programs across different computational units, such as Central Processing Units (CPU) and GPUs. Furthermore, the GPU's capability of massive parallel computing has been explored in the literature in the context of occupancy grid mapping[80] [81], taking advantage of the independence between each cell assumption which allows updating each cell independently on different cores.

- **Ground-truth map construction**: As explained in section 4.4, an objective evaluation of the accuracy of the implemented solution was not possible as there was no dataset available for the construction of a ground-truth map. Therefore, as a future work possibility, a new dataset could be recorded with an additional high-resolution sensor, so that its measurements could be used to build a ground-truth occupancy map.

- **Integration of more sensors**: The PDK system is ready for the easy integration of new sensor data into the occupancy map, as long as an accurate corresponding sensor model is added alongside it to the system. Therefore, the accuracy of the results could be improved by adding more sensors to the system, such as short-range radars.

- **Improved memory management**: As stated in section 2.2, the *OctoMap* tool performs dynamic memory allocation for the creation of new nodes when sensor measurements in unmapped areas are received. However, this is not an appropriate method for embedded systems due to the reduced available memory size. Furthermore, as also stated in section 2.2, the size of each Cartesian dimension of the mapped area is limited when using an Octree, according to equation 2.2, which limits the implemented solution as it would not be able to map the total driven area in some scenarios. These two problems could be handled with a better memory management process. For example, a possible approach could use a bounding box always centered on the vehicle so that older measurements outside of it are discarded and replaced in memory by newer measurements. Another method could consist of using various small grid maps, which are saved and loaded as the vehicle moves from one small local map to another.

Bibliography

[1] Sebastian Thrun. “Learning occupancy grid maps with forward sensor models”. In: *Autonomous robots* 15.2 (2003), pp. 111–127.

[2] Continental Engineering Services. *Perception Development Kit (PDK)*. Accessed: June 20, 2022. [Online]. URL: https://conti-engineering.com/highlights/new-mobility-ecosystem-development-kits/perception-development-kit-pdk/.

[3] Kai M Wurm et al. “OctoMap: A probabilistic, flexible, and compact 3D map representation for robotic systems”. In: *Proc. of the ICRA 2010 workshop on best practice in 3D perception and modeling for mobile manipulation*. Vol. 2. 2010.

[4] W.H.O. *Road traffic injuries*. Accessed: June 20, 2022. [Online]. URL: https://www.who.int/news-room/fact-sheets/detail/road-traffic-injuries.

[5] SAE. “Taxonomy and definitions for terms related to driving automation systems for on-road motor vehicles.” In: *SAE Tech* (2016).

[6] Alexandru Constantin Serban, Erik Poll, and Joost Visser. “A standard driven software architecture for fully autonomous vehicles”. In: *2018 IEEE International Conference on Software Architecture Companion (ICSA-C)*. IEEE. 2018, pp. 120–127.

[7] Ekim Yurtsever et al. “A survey of autonomous driving: Common practices and emerging technologies”. In: *IEEE access* 8 (2020), pp. 58443–58469.

[8] Travis J Crayton and Benjamin Mason Meier. “Autonomous vehicles: Developing a public health research agenda to frame the future of transportation policy”. In: *Journal of Transport & Health* 6 (2017), pp. 245–252.

[9] W David Montgomery et al. “America’s workforce and the self-driving future: Realizing productivity gains and spurring economic growth”. In: (2018).

[10] William J Fleming. “Overview of automotive sensors”. In: *IEEE sensors journal* 1.4 (2001), pp. 296–308.

[11] Wilbert G Aguilar et al. “Monocular depth perception on a micro-UAV using convolutional neuronal networks”. In: *International Symposium on Ubiquitous Networking*. Springer. 2018, pp. 392–397.

[12] Bing Pan, LiPing Yu, and QianBing Zhang. "Review of single-camera stereo-digital image correlation techniques for full-field 3D shape and deformation measurement". In: *Science China Technological Sciences* 61.1 (2018), pp. 2–20.

[13] Abdelmoghit Zaarane et al. "Distance measurement system for autonomous vehicles using stereo camera". In: *Array* 5 (2020), p. 100016.

[14] Kai Zhang et al. "Depth sensing beyond lidar range". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020, pp. 1692–1700.

[15] Yasir Dawood Salman, Ku Ruhana Ku-Mahamud, and Eiji Kamioka. "Distance measurement for self-driving cars using stereo camera". In: *International Conference on Computing and Informatics*. Vol. 1. 105. 2017, pp. 235–242.

[16] Gabriel da Silva Vieira et al. "A three-Layer architecture to support disparity map construction in stereo vision systems". In: *Intelligent Systems with Applications* 12 (2021), p. 200054.

[17] Paul Fritsche et al. "Radar and LiDAR Sensorfusion in Low Visibility Environments." In: *ICINCO (2)*. 2016, pp. 30–36.

[18] Malcolm Mielle, Martin Magnusson, and Achim J Lilienthal. "A comparative analysis of radar and lidar sensing for localization and mapping". In: *2019 European Conference on Mobile Robots (ECMR)*. IEEE. 2019, pp. 1–6.

[19] Timothy B. Lee. *How 10 leading companies are trying to make powerful, low-cost lidar*. Accessed: February 22, 2022. [Online]. URL: https://arstechnica.com/cars/2019/02/the-ars-technica-guide-to-the-lidar-industry/.

[20] Yunyi Jia, Longxiang Guo, and Xin Wang. "4 - Real-time control systems". In: *Transportation Cyber-Physical Systems*. Ed. by Lipika Deka and Mashrur Chowdhury. Elsevier, 2018, pp. 81–113. ISBN: 978-0-12-814295-0. DOI: https://doi.org/10.1016/B978-0-12-814295-0.00004-6. URL: https://www.sciencedirect.com/science/article/pii/B9780128142950000046.

[21] Hsueh-Jyh Li and Yean-Woei Kiang. "10 - Radar and Inverse Scattering". In: *The Electrical Engineering Handbook*. Ed. by WAI-KAI CHEN. Burlington: Academic Press, 2005, pp. 671–690. ISBN: 978-0-12-170960-0. DOI: https://doi.org/10.1016/B978-012170960-0/50047-5. URL: https://www.sciencedirect.com/science/article/pii/B9780121709600500475.

[22] Sampo Kuutti et al. "A survey of the state-of-the-art localization techniques and their potentials for autonomous vehicle applications". In: *IEEE Internet of Things Journal* 5.2 (2018), pp. 829–846.

[23] Hans Moravec and Alberto Elfes. "High resolution maps from wide angle sonar". In: *Proceedings. 1985 IEEE international conference on robotics and automation*. Vol. 2. IEEE. 1985, pp. 116–121.

[24] Alberto Elfes and Larry Matthies. "Sensor integration for robot navigation: Combining sonar and stereo range data in a grid-based representataion". In: *26th IEEE conference on decision and control*. Vol. 26. IEEE. 1987, pp. 1802–1807.

[25] Romain Neuville, Jordan Steven Bates, and François Jonard. "Estimating forest structure from UAV-mounted LiDAR point cloud using machine learning". In: *Remote sensing* 13.3 (2021), p. 352.

[26] Ole Schumann et al. "RadarScenes: A real-world radar point cloud data set for automotive applications". In: *2021 IEEE 24th International Conference on Information Fusion (FUSION)*. IEEE. 2021, pp. 1–8.

[27] Xiaowen Teng et al. "Three-dimensional reconstruction method of rapeseed plants in the whole growth period using RGB-D camera". In: *Sensors* 21.14 (2021), p. 4628.

[28] David M Cole and Paul M Newman. "Using laser range data for 3D SLAM in outdoor environments". In: *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006*. IEEE. 2006, pp. 1556–1563.

[29] Andreas Nüchter et al. "6D SLAM—3D mapping outdoor environments". In: *Journal of Field Robotics* 24.8-9 (2007), pp. 699–722.

[30] Donald Meagher. "Geometric modeling using octree encoding". In: *Computer graphics and image processing* 19.2 (1982), pp. 129–147.

[31] Y. Roth-Tabak and R. Jain. "Building an environment model using depth information". In: *Computer* 22.6 (1989), pp. 85–90. DOI: 10.1109/2.30724.

[32] John Amanatides, Andrew Woo, et al. "A fast voxel traversal algorithm for ray tracing." In: *Eurographics*. Vol. 87. 3. 1987, pp. 3–10.

[33] Daniel Duberg and Patric Jensfelt. "UFOMap: An efficient probabilistic 3D mapping framework that embraces the unknown". In: *IEEE Robotics and Automation Letters* 5.4 (2020), pp. 6411–6418.

[34] Liang Zhang et al. "Semantic SLAM based on object detection and improved octomap". In: *IEEE Access* 6 (2018), pp. 75545–75559.

[35] Niu Lian-Qiang and Shao Zhong. "A Fast Line Rasterization Algorithm Based on Pattern Decomposition [J]". In: *Journal of Computer Aided Design & Computer Graphics* 22.8 (2010), pp. 1286–1292.

[36] Jing Chen and Shaojie Shen. "Improving octree-based occupancy maps using environment sparsity with application to aerial robot navigation". In: *2017 IEEE international conference on robotics and automation (ICRA)*. IEEE. 2017, pp. 3656–3663.

[37] Anderson Souza and Luiz MG Gonçalves. "Occupancy-elevation grid: an alternative approach for robotic mapping and navigation". In: *Robotica* 34.11 (2016), pp. 2592–2609.

[38] John Bares et al. "Ambler: An autonomous rover for planetary exploration". In: *Computer* 22.6 (1989), pp. 18–26.

[39] Patrick Pfaff, Rudolph Triebel, and Wolfram Burgard. “An efficient extension to elevation maps for outdoor terrain mapping and loop closing”. In: *The International Journal of Robotics Research* 26.2 (2007), pp. 217–230.

[40] François Pomerleau, Francis Colas, and Roland Siegwart. “A review of point cloud registration algorithms for mobile robotics”. In: *Foundations and Trends in Robotics* 4.1 (2015), pp. 1–104.

[41] Rudolph Triebel, Patrick Pfaff, and Wolfram Burgard. “Multi-level surface maps for outdoor terrain mapping and loop closing”. In: *2006 IEEE/RSJ international conference on intelligent robots and systems*. IEEE. 2006, pp. 2276–2282.

[42] Kai Schueler et al. “360 degree multi sensor fusion for static and dynamic obstacles”. In: *2012 IEEE Intelligent Vehicles Symposium*. IEEE. 2012, pp. 692–697.

[43] Denis F Wolf and Gaurav S Sukhatme. “Mobile robot simultaneous localization and mapping in dynamic environments”. In: *Autonomous Robots* 19.1 (2005), pp. 53–65.

[44] Chieh-Chih Wang and Chuck Thorpe. “Simultaneous localization and mapping with detection and tracking of moving objects”. In: *Proceedings 2002 IEEE International conference on robotics and automation (Cat. No. 02CH37292)*. Vol. 3. IEEE. 2002, pp. 2918–2924.

[45] Jonathan Vincent et al. “Dynamic object tracking and masking for visual SLAM”. In: *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2020, pp. 4974–4979.

[46] Ángel Llamazares, Eduardo J Molinos, and Manuel Ocaña. “Detection and tracking of moving obstacles (DATMO): a review”. In: *Robotica* 38.5 (2020), pp. 761–774.

[47] Michael Slutsky and Daniel Dobkin. “Dual inverse sensor model for radar occupancy grids”. In: *2019 IEEE Intelligent Vehicles Symposium (IV)*. IEEE. 2019, pp. 1760–1767.

[48] Bahador Khaleghi et al. “Multisensor data fusion: A review of the state-of-the-art”. In: *Information fusion* 14.1 (2013), pp. 28–44.

[49] Carlos Galvez Del Postigo Fernandez. “Grid-based multi-sensor fusion for on-road obstacle detection: Application to autonomous driving”. PhD thesis. KTH, 2015.

[50] Mathias Perrollaz et al. “Long range obstacle detection using laser scanner and stereovision”. In: *2006 IEEE intelligent vehicles symposium*. IEEE. 2006, pp. 182–187.

[51] Xiao Wang et al. “Bionic vision inspired on-road obstacle detection and tracking using radar and visual information”. In: *17th International IEEE Conference on Intelligent Transportation Systems (ITSC)*. IEEE. 2014, pp. 39–44.

[52] Manish Kumar, Devendra P Garg, and Randy Zachery. “Multi-sensor fusion strategy to obtain 3-D occupancy profile”. In: *31st Annual Conference of IEEE Industrial Electronics Society, 2005. IECON 2005*. IEEE. 2005, 6–pp.

[53] Igor E Paromtchik, Mathias Perrollaz, and Christian Laugier. “Fusion of telemetric and visual data from road scenes with a lexus experimental platform”. In: *2011 IEEE Intelligent Vehicles Symposium (IV)*. IEEE. 2011, pp. 746–751.

[54] Arthur P Dempster. “A generalization of Bayesian inference”. In: *Journal of the Royal Statistical Society: Series B (Methodological)* 30.2 (1968), pp. 205–232.

[55] Glenn Shafer. “A mathematical theory of evidence”. In: *A mathematical theory of evidence*. Princeton university press, 1976.

[56] Sebastian Thrun. “Probabilistic robotics”. In: *Communications of the ACM* 45.3 (2002), pp. 52–57.

[57] Andreas Högger. *Dempster shafer sensor fusion for autonomously driving vehicles: association free tracking of dynamic objects*. 2016.

[58] Philippe Smets. “Decision making in the TBM: the necessity of the pignistic transformation”. In: *International journal of approximate reasoning* 38.2 (2005), pp. 133–147.

[59] Michael Darms, Paul Rybski, and Chris Urmson. “Classification and tracking of dynamic objects with multiple sensors for autonomous driving in urban environments”. In: *2008 IEEE Intelligent Vehicles Symposium*. IEEE. 2008, pp. 1197–1202.

[60] Evan Kaufman et al. “Bayesian occupancy grid mapping via an exact inverse sensor model”. In: *2016 American Control Conference (ACC)*. IEEE. 2016, pp. 5709–5715.

[61] Charles M Grinstead and J Laurie Snell. “Conditional probability–discrete conditional”. In: *Grinstead & Snell’s introduction to probability Orange Grove Texts* (2009).

[62] Marc Peter Deisenroth, A Aldo Faisal, and Cheng Soon Ong. *Mathematics for machine learning*. Cambridge University Press, 2020.

[63] Deborah J Rumsey. *Statistics II for dummies*. John Wiley & Sons, 2021.

[64] Joao Carvalho and Rodrigo Ventura. “Comparative evaluation of occupancy grid mapping methods using sonar sensors”. In: *Iberian Conference on Pattern Recognition and Image Analysis*. Springer. 2013, pp. 889–896.

[65] Ralph Grewe et al. “Evaluation method and results for the accuracy of an automotive occupancy grid”. In: *2012 IEEE International Conference on Vehicular Electronics and Safety (ICVES 2012)*. IEEE. 2012, pp. 19–24.

[66] Rauf Yagfarov, Mikhail Ivanou, and Ilya Afanasyev. “Map comparison of lidar-based 2d slam algorithms using precise ground truth”. In: *2018 15th International Conference on Control, Automation, Robotics and Vision (ICARCV)*. IEEE. 2018, pp. 1979–1983.

[67] Cyril Crassin et al. “Gigavoxels: Ray-guided streaming for efficient and detailed voxel rendering”. In: *Proceedings of the 2009 symposium on Interactive 3D graphics and games*. 2009, pp. 15–22.

[68] Mason Woo et al. *OpenGL programming guide: the official guide to learning OpenGL, version 1.2*. Addison-Wesley Longman Publishing Co., Inc., 1999.

[69] John Spitzer. “Opengl performance”. In: *Game Developer Conference 2001 Proceedings*. 2001.

[70] Dave Shreiner et al. “Performance opengl: Platform independent techniques”. In: *ACM SIGGRAPH*. 2001.

[71] Dave Shreiner, Bill The Khronos OpenGL ARB Working Group, et al. *OpenGL programming guide: the official guide to learning OpenGL, versions 3.0 and 3.1*. Pearson Education, 2009.

[72] Michael Burger and Christian Bischof. “Using Instancing to Efficiently Render Carbon Nanotubes”. en. In: *3rd International Workshop on Computational Engineering : Book of Abstracts*. Event Title: 3rd International Workshop on Computational Engineering. Stuttgart, Germany, 2022, pp. 206–210. DOI: https://doi.org/10.26083/tuprints-00021263. URL: http://tuprints.ulb.tu-darmstadt.de/21263/.

[73] Continental Engineering Services. *ARS540 Radar Sensor*. Accessed: June 20, 2022. [Online]. URL: https://www.continental-automotive.com/en-gl/Passenger-Cars/Autonomous-Mobility/Enablers/Radars/Long-Range-Radar/ARS540.

[74] Chen Feng, Yuichi Taguchi, and Vineet R Kamat. “Fast plane extraction in organized point clouds using agglomerative hierarchical clustering”. In: *2014 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2014, pp. 6218–6225.

[75] Yuxue Fan, Yan Huang, and Jingliang Peng. “Point cloud compression based on hierarchical point clustering”. In: *2013 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference*. IEEE. 2013, pp. 1–7.

[76] Bao-Quan Shi, Jin Liang, and Qing Liu. “Adaptive simplification of point cloud using k-means clustering”. In: *Computer-Aided Design* 43.8 (2011), pp. 910–922.

[77] Design Guide. “Cuda c programming guide”. In: *NVIDIA, July* 29 (2013), p. 31.

[78] Andreas Hermann et al. “GPU-based real-time collision detection for motion execution in mobile manipulation planning”. In: *2013 16th International Conference on Advanced Robotics (ICAR)*. IEEE. 2013, pp. 1–7.

[79] Aaftab Munshi. “The opencl specification”. In: *2009 IEEE Hot Chips 21 Symposium (HCS)*. IEEE. 2009, pp. 1–314.

[80] Florian Homm et al. “Efficient occupancy grid computation on the GPU with lidar and radar for road boundary detection”. In: *2010 IEEE intelligent vehicles symposium*. IEEE. 2010, pp. 1006–1013.

[81] Manuel Yguel, Olivier Aycard, and Christian Laugier. “Efficient gpu-based construction of occupancy girds using several laser range-finders”. In: *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2006, pp. 105–110.

Appendix A

Bayes Filter: Log-Odds Derivation

As mentioned in Section 2.4, the Binary Bayes Filter algorithm solves the calculation of the occupancy probability of each cell m_{ij} in the grid based on the previous sensor measurements $z_{1:t}$ and vehicle poses $x_{1:t}$, given by:

$$p(m_{ij}|z_{1:t}, x_{1:t}) \tag{A.1}$$

The following is the derivation of this probability into the previously mentioned *log-odds* representation.

First, the Bayes' rule is applied to filter out the latest sensor measurement z_t:

$$p(m_{ij}|z_{1:t}, x_{1:t}) = \frac{p(z_t|m_{ij}, x_t, z_{1:t-1}, x_{1:t-1}).p(m_{ij}|z_{1:t-1}, x_{1:t-1}, x_t)}{p(z_t|z_{1:t-1}, x_{1:t})} \tag{A.2}$$

Considering the assumptions made in Section 2.4, the independence between consecutive sensor measurements allows the removal of some terms in Equation A.2, resulting in the following simpler equation:

$$p(m_{ij}|z_{1:t}, x_{1:t}) = \frac{p(z_t|m_{ij}, x_t).p(m_{ij}|z_{1:t-1}, x_{1:t-1})}{p(z_t|z_{1:t-1}, x_{1:t})} \tag{A.3}$$

As it was also mentioned before, the term $p(z_t|m_{ij}, x_t)$ is a forward sensor model, which can be expanded using again the Bayes' rule:

$$p(z_t|m_{ij}, x_t) = \frac{p(m_{ij}|z_t, x_t).p(z_t|x_t)}{p(m_{ij|x_t})} \tag{A.4}$$

where the term $p(m_{ij}|z_t, x_t)$ is the inverse sensor model.

By replacing the previous equation into Equation A.3, the following is obtained:

$$p(m_{ij}|z_{1:t}, x_{1:t}) = \frac{p(m_{ij}|z_t, x_t).p(z_t|x_t).p(m_{ij}|z_{1:t-1}, x_{1:t-1})}{p(z_t|z_{1:t-1}, x_{1:t}).p(m_{ij|x_t})} \tag{A.5}$$

The next step of the derivation consists of dividing the term $p(m_{ij}|z_{1:t}, x_{1:t})$ and its analogous term $p(\overline{m_{ij}}|z_{1:t}, x_{1:t})$:

$$\frac{p(m_{ij}|z_{1:t}, x_{1:t})}{p(\overline{m_{ij}}|z_{1:t}, x_{1:t})} = \frac{p(\overline{m_{ij}}|z_t, x_t)}{p(\overline{m_{ij}}|z_t, x_t)} \cdot \frac{p(m_{ij}|z_{1:t-1}, x_{1:t-1})}{p(\overline{m_{ij}}|z_{1:t-1}, x_{1:t-1})} \cdot \frac{p(m_{ij})}{p(\overline{m_{ij}})} \tag{A.6}$$

Finally, by using the logarithm of Equation A.6, the *log-odds* ratio $l(m_{ij})$ representation is obtained, which enables the recursive update formula of the Binary Bayes Filter:

$$l(m_{ij}) = \log \frac{p(m_{ij}|z_{1:t}, x_{1:t})}{1 - p(m_{ij}|z_{1:t}, x_{1:t})} = l_{t-1}(m_{ij}) + \log \frac{p(m_{ij}|z_t, x_t)}{1 - p(m_{ij}|z_t, x_t)} - \log \frac{p(m_{ij})}{1 - p(m_{ij})} \tag{A.7}$$

Appendix B

ARS540 Specifications

The ARS540 is a high performance long range radar sensor with direct and independent measurement of four dimensions: range, doppler, azimuth, and elevation. Table B.1 lists some of this sensor's specifications that are publicly available in [73]. The ranges in the value column are related with the configurability of the sensor. Some sensor characteristics, such as the elevation FOV or the range accuracy, for example, are not present in the table due to confidentiality of this information by Continental Engineering Services (CES).

Table B.1: ARS540 Specifications

Characteristic	**Value**
Dimension (WxDxH)	137 x 90 x 39 mm (w/o con.)
Mass	500 g
Range	300 m
Azimuth Field of View (FOV)	$\pm$ 60º
Angular Accuracy (in azimuth and elevation)	$\pm$ 0.1º
Update Rate	60 ms
Temp. Range	-40º to +85º
Power Dissipation	23 W
Supply Voltage	12 V (nominal)
Operation Frequency	76 to 77 GHz

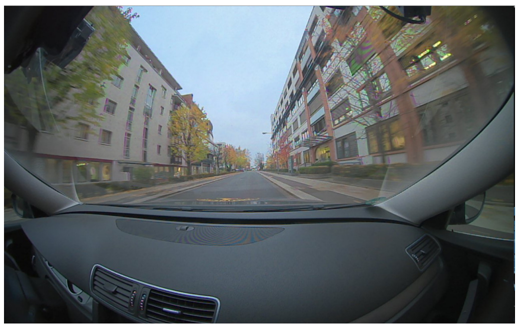
Figure B.1: Picture of ARS540 radar sensor. Image from [73].

Appendix C

Additional Visualization Results for 3D Occupancy Mapping

This appendix presents images captured by the front camera during the recording of each evaluation dataset alongside the visualization results of the corresponding Octree, namely section C.1 shows images from Dataset One and section C.2 from Dataset Two. The results were obtaining using the default Octree parameters, consisting of a resolution of 0.2 meters, the tree depth set to 16 levels and the free space calculations disabled. Furthermore, the Dempster-Shafer Evidence Theory method was used as the low-level sensor fusion approach.

C.1 Dataset One


(a) Front camera image

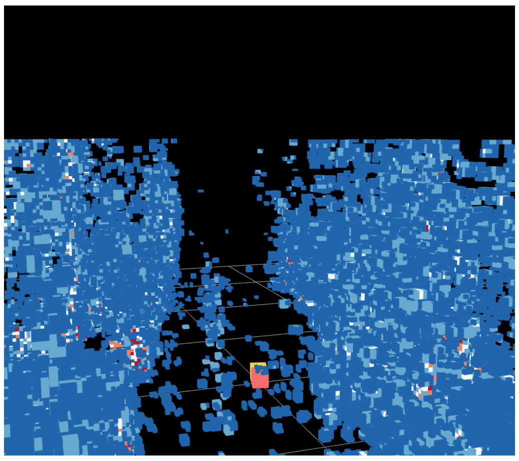

(b) Octree visualization

Figure C.1: Two-lane street, with multi-storey buildings, from Dataset One, and the corresponding Octree, with a resolution of 0.2 meters, a tree depth of 16 levels, free space calculations disabled, and using the Dempster-Shafer approach for sensor fusion.

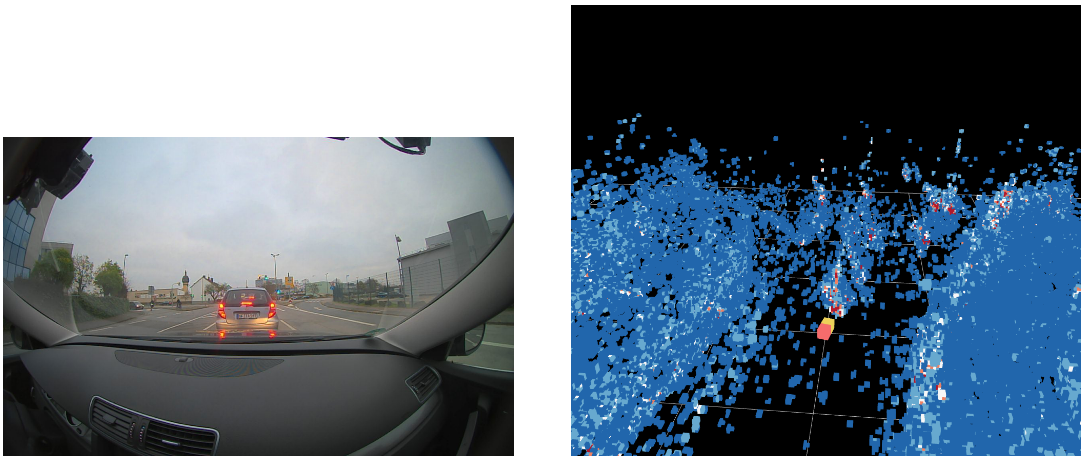

(a) Front camera image (b) Octree visualization

Figure C.2: Vehicle stopped at a traffic light from Dataset One, and the corresponding Octree, with a resolution of 0.2 meters, a tree depth of 16 levels, free space calculations disabled, and using the Dempster-Shafer approach for sensor fusion.

C.2 Dataset Two

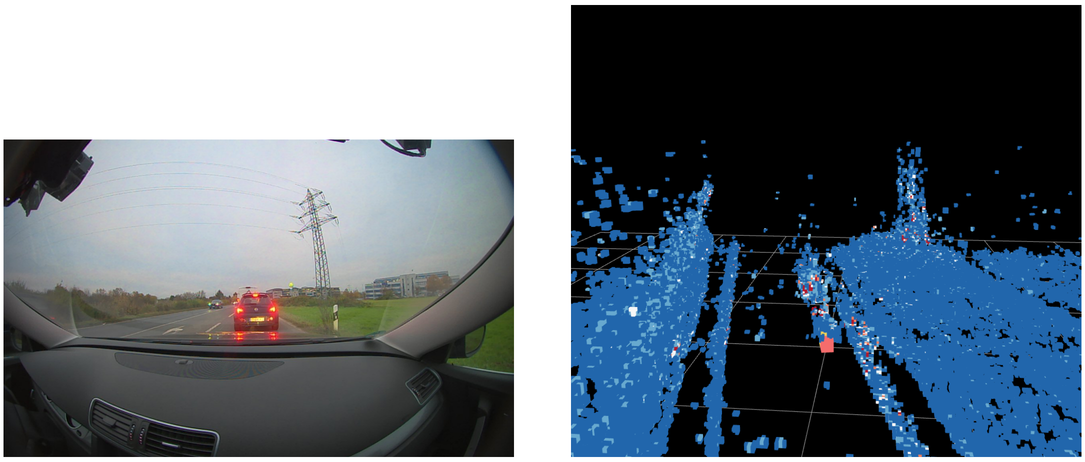
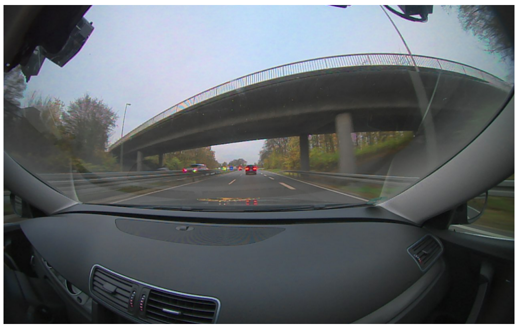

(a) Front camera image (b) Octree visualization

Figure C.3: Vehicle stopped at traffic light from Dataset Two, and the corresponding Octree, with a resolution of 0.2 meters, a tree depth of 16 levels, free space calculations disabled, and using the Dempster-Shafer approach for sensor fusion.


(a) Front camera image

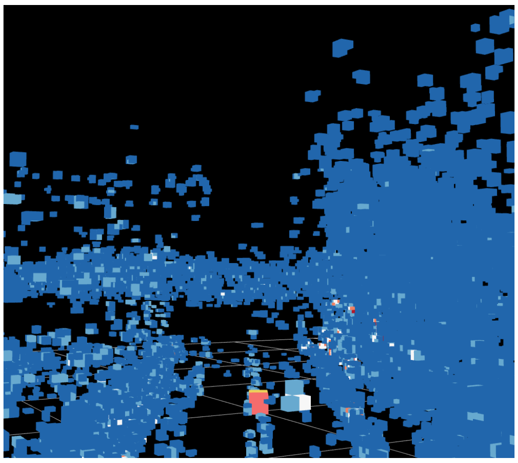

(b) Octree visualization

Figure C.4: Highway street with a bridge from Dataset Two, and the corresponding Octree, with a resolution of 0.2 meters, a tree depth of 16 levels, free space calculations disabled, and using the Dempster-Shafer approach for sensor fusion.

Appendix D

Visualization Methods' Results

Figure D.1 presents the rendering time results for the implemented visualization approaches: immediate mode, vertex arrays, vertex buffer objects, instanced arrays. The rendering times on the table are in milliseconds with three decimal cases. Some values were not calculated for the immediate mode as the obtained results for this method were already significantly higher than the real-time threshold.

Table D.1: Rendering time results for the implemented visualization approaches.

Number of rendered cubes	**Immediate Mode**	**Vertex Arrays**	**Vertex Buffer Objects**	**Instanced Arrays**
1,000	1.246	0.044	0.049	0.052
2,500	4.074	0.063	0.369	0.044
5,000	8.210	0.088	0.438	0.056
7,500	12.308	0.100	0.512	0.056
10,000	16.484	0.108	0.566	0.050
50,000	99.442	2.718	2.488	0.070
10,0000	188.668	5.778	4.328	0.140
250,000	-	14.526	10.500	1.398
500,000	-	32.124	22.760	3.420
750,000	-	61.444	36.080	5.358
1,000,000	-	103.312	72.464	7.180