Integer Occupancy Grids : a probabilistic multi-sensor fusion framework for embedded perception

Tiana Rakotovao Andriamahefa

Communauté
UNIVERSITÉ Grenoble Alpes

THESIS

for the degree of

DOCTOR OF THE COMMUNITY UNIVERSITY GRENOBLE ALPES

Option: **Computer Science**

Ministerial Order: August 7th, 2006

Presented by

Tiana RAKOTOVAO ANDRIAMAHEFA

Thesis supervised by **Christian LAUGIER**

Prepared at the laboratories
CEA LETI/LIALP, INRIA Rhône-Alpes/Chroma
in **the Doctoral School MSTII (Mathématiques, Sciences et Technologies de l'Information et de l'Informatique)**

Integer Occupancy Grids: a probabilistic multi-sensor fusion framework for embedded perception

Defended on **February 21st, 2017**
Jury:

Prof. Dr.-Ing. Christoph STILLER
Professor, Karlsruher Institut für Technologie, Reviewer
Prof. Philippe BONNIFAIT
Professor, Université de Technologie de Compiègne, Reviewer
Dr. Christian LAUGIER
Research Director, Inria Grenoble Rhône-Alpes, Advisor
Prof. James L. CROWLEY
Professor, Institut Polytechnique de Grenoble, President
Dr.-Ing. Jochen LANGHEIM
Vice President Automotive Systems R&D Projects, STMicroelectronics SA Paris, Examiner
Dr. Diego PUSCHINI
Research Engineer CEATech Grenoble, Examiner

Integer Occupancy Grids: a probabilistic multi-sensor fusion framework for embedded perception

Abstract: Perception is a primary task for an autonomous car where safety is of utmost importance. A perception system builds a model of the driving environment by fusing measurements from multiple perceptual sensors including LIDARs, radars, vision sensors, *etc.* The fusion based on occupancy grids builds a probabilistic environment model by taking into account sensor uncertainties. This thesis aims to integrate the computation of occupancy grids into embedded low-cost and low-power platforms. Occupancy Grids perform though intensive probability calculus that can be hardly processed in real-time on embedded hardware.

As a solution, this thesis introduces the Integer Occupancy Grid framework. Integer Occupancy Grids rely on a proven mathematical foundation that enables to process probabilistic fusion through simple addition of integers. The hardware/software integration of integer occupancy grids is safe and reliable. The involved numerical errors are bounded and is parametrized by the user. Integer Occupancy Grids enable a real-time computation of multi-sensor fusion on embedded low-cost and low-power processing platforms dedicated for automotive applications.

Keywords: probabilistic fusion, sensor fusion, perception, occupancy grids, embedded integration

Grille d'occupation entière: une méthode probabiliste de fusion multi-capteurs pour la perception embarquée

Resumé: Pour les voitures autonomes, la perception est une fonction principale où la sécurité est de la plus haute importance. Un système de perception construit un modèle de l'environnement de conduite en fusionnant plusieurs capteurs de perception incluant les LIDARs, les radars, les capteurs de vision, *etc.* La fusion basée sur les grilles d'occupation construit un modèle probabiliste de l'environnement en prenant en compte l'incertitude des capteurs. Cette thèse vise á intégrer le calcul des grilles d'occupation dans des systèmes embarqués à bas-coût et à basse-consommation. Cependant, les grilles d'occupation effectuent des calculs de probabilité intenses et difficilement calculables en temps-réel par les plateformes matérielles embarquées.

Comme solution, cette thèse introduit une nouvelle méthode de fusion probabiliste appelée Grille d'Occupation Entière. Les Grilles d'Occupation Entières se reposent sur des principes mathématiques qui permettent de calculer la fusion de capteurs grâce á des simple addition de nombre entiers. L'intégration matérielle et logicielle des Grilles d'Occupation Entière est sûre et fiable. Les erreurs numériques engendrées par les calculs sont connues, majorées et paramétrées par l'utilisateur. Les Grilles d'Occupation Entière permettent de calculer en temps-réel la fusion de multiple capteurs sur un système embarqué bas-coût et â faible consommation dédié pour les applications pour l'automobile.

Mots clés: fusion probabiliste, fusion de capteurs, perception, grille d'occupation, intégration embarquée

Acknowledgments

First and foremost, I would like to express my acknowledgments to my parents, my sister and my brother for supporting me despite the distance between us. This thesis is a part of my long journey abroad, and my family was always my first source of inspiration and strength to continue hard working whatever is the situation.

Second, my gratitude to my thesis supervisor Christian Laugier for his precious guidance and support. I also address my very special thanks to Diego Puschini and Julien Mottin who were my first interlocutors and mentors during this thesis. They always knew how to motivate me and pushed me to surpass myself. Diego, Julien and other colleagues at the LIALP laboratory have done great works for highlighting the results of this thesis by developing demonstrators that have been exhibited in international events and conferences. With Christian Laugier and Suzanne Lesecq, they have also organized a workshop on automated cars the day of the PhD defense. I appreciate and I am thankful.

I also want to thank the members of jury for accepting to review and to examine the present thesis, and for dedicating time for assisting to the PhD defense. I would like to address my thanks to all my colleagues at LIALP. Special thanks to Vincent Olive, the director of the laboratory. I also thank researchers, engineers, PhD candidates and interns that work at LIALP. Working with you was a great pleasure. Thanks also for the various, interesting and even particular discussions, moments and events that we shared together. Besides, I would like to express my gratitude to my colleagues at INRIA/e-motion and INRIA/Chroma. Your support was very important for me. I particularly remember the way you have supported me at the very first moment of my thesis.

Last but not the least, I would like to thank all my friends in France, particularly those in Toulouse and Grenoble. I especially mention Patrice and Ony Randriamampianina. My thanks also to the Malagasy community of Grenoble, particularly to the catholic community. Finally, I want to thank Felana Andriatsitoaina for her absolute support especially in the toughest moments.

This thesis was supported by the Commissariat à l'Énergie Atomique et aux Ènergies Alternatives (CEA). This theis has been carried out mainly in the Laboratoire Intégration et Atelier Logiciel pour Puces (LIALP), a laboratory of the institute CEA LETI, and within CHROMA, a team that is part of the Institut National de Recherche en Informatique et en Automatique (INRIA) of Rhône-Alpes. In one side, the CEA LETI is one of the world's largest institute for applied research in microelectronics. On the other side, the INRIA Rhône-Ales is globally-known for researches in computer science, namely in the field of robotics and automated cars. This thesis consists in designing a mathematical framework that enables to safely and efficiently process on an embedded platform the fusion of perceptual sensors mounted on automated cars.

Contents

List of Figures

List of Tables

Chapter 1

INTRODUCTION

The utmost challenge that modern road transportation must face is the road safety. Traffic injuries cost lives and affect the economy of a whole country. Energy consumption and the respect of environment also become major challenges in the era of global warming. Cars are at the center of these challenges. Intelligent cars are proposed as one of the scientific and technological step ahead to improve safety, to save energy and to improve the respect of environmental standards.

Intelligent cars are upgraded with various kind of sensors for developing driving assistance systems and autonomous navigation. Perception is the task of gathering information about the driving environment through perceptual sensors. A perception system builds an environment model by fusing measurements from multiple sensors.

The fusion of multiple sensors provides several advantages over using a single sensor. First, sensors are subject to physical limitations and noises which introduce uncertainty in measurements. Using multiple sensors improves the robustness and the reliability of the perception system. It provides a level of redundancy of information that allows to improve safety and overcome the risk of sensor failure. It also allows to extend the coverage of the driving environment by range sensors.

Multi-sensor fusion is the main task of a perception system. The fusion must handle sensor uncertainties. The environment model must be able to represent any kind of obstacles whatever their nature is (eg. cars, pedestrians, animals, cyclists, buildings, vegetation, road infrastructures, *etc*).

This thesis aims to process multi-sensor fusion on a computing platform embedded on-board the car. The computing hardware is subject to a constraint of low cost and low power budget. The environment model produced by the HW/SW integration must be numerically reliable in order to ensure safety.

Occupancy Grids are a probabilistic framework that are able to fuse multiple sensors by taking into account uncertainties. They produce a probabilistic model of the environment that can cope with the diversity of obstacles. Occupancy grids require though an intensive probability calculation that embedded resource-constrained computing platforms can hardly process in real-time.

As a solution, this thesis introduces the Integer Occupancy Grid framework. This framework processes probabilistic multi-sensor fusion through simple integer arithmetic. Its HW/SW integration guaranties a bounded numerical error that is parametrized by the user. Integer Occupancy Grids enable the integration of multi-sensor fusion on low-cost and low-power processing platforms. The latter becomes even able to process multi-sensor fusion in real-time.

The following sections will provide introduction and motivation on the topics treated in this thesis. The addressed problem and the proposed approach will be presented thereafter.

1.1 Motivation

1.1.1 Societal and environmental challenges

Cars are a widely accepted mean of transportation of people and goods. They contribute significantly to the economic development of cities, countries and even at worldwide scale. Transport, industry, trades, services, defense, health, environment protection, *etc* always rely on cars at some levels in order to ensure mobility and to perform tasks on time, efficiently and safely. Since their mass production at the beginning of the 20th century, cars have considerably shaped cities. Major part of the infrastructure in cities – such as streets, parkings, highways, traffic signs, and bridges – are dedicated for cars. Moreover, cars have also brought significant social impacts. They favor the connection between persons, families and friends. They are at the same time a sign of prestige, social class, image and personality.

Nevertheless, cars are also at the origin of the major challenges that modern societies have to face: the road safety, the energy consumption and the environmental challenges. In [WHO 2015], the World Health Organization (WHO) reports that road accidents cause over 1.2 million of deaths and 50 millions of non-fatal injuries worldwide each year. If these statistics persist, road accidents would cause 36 millions of deaths within 30 years, which is equivalent to the population of Canada in 2015 ([Le Quotidient 2015]). The lost of an active family member or the handicap due to injuries lead households into deep poverty. They also constitute a significant burden for health, insurance and legal systems and cause globally a loss of 3% of Gross Domestic Product (GDP). Concerning the energy and environmental challenges, according to a study about energy efficiency in France ([RAC-F 2014]), transportation is the second leading sector in term of energy consumption (32 % of national consumption). Furthermore, this sector is the first leading in term of emission of greenhouse gas (37 % of CO_2 emissions). About 95 % of the emissions due to the sector of transportation are caused by road transportation. Moreover, personal vehicles are responsible of 58 % of the emissions due to road transportation.

To address these challenges, governments, national and international organizations promote road legislations, road awareness campaigns and environmental standards for improving road safety, energy efficiency, and environment protection ([IEA 2010, WHO 2015]). In addition, researchers, engineers and the automotive

industry also contribute actively by providing technological solutions and innovations. During the 20th century, mechanical, electrical and electronic technologies – such as automatic transmission, fuel injection and Anti-lock Braking System (ABS) – have been continuously integrated into cars to enhance safety, reliability, comfort and efficiency. Nevertheless, these solutions are not sufficient to address the above challenges. The safety and the efficiency of cars are mainly under the responsibility of drivers. However, 93 % of traffic injuries are caused by human errors ([Yeomans 2014]). At the end of the 20st century, thanks to the advancement of sensors, digital technologies and advanced algorithms, new and more *intelligent* functions have begun to be developed for helping drivers in the task of driving.

1.1.2 Towards autonomous cars

At the beginning of the 21st century, in the era of digital technologies, the task of driving is more and more left out of the hands of human drivers. Driving is rather assigned to computerized systems composed of sensors, actuators, microprocessors, communicating devices, algorithms and software. An *intelligent vehicle* is a vehicle equipped with computer systems capable of handling certain aspects of driving ([Eskandarian 2012]). Systems which purpose is to assist the human driver for performing the task of driving are called *Advanced Driver Assistance Systems (ADAS)*. These systems help the human driver to achieve a driving performance that is safer, more efficient and more respectful of environmental standards. When such systems can handle totally the task of driving without any human intervention, the vehicle is called *autonomous, self-driving or driverless*.

The idea of autonomous cars is not recent. In 1925, the American firm Houdina Radio Control Co. unveiled a radio-controlled driverless car running in the streets of New York. In the 1950s until 1970s, driverless cars were experienced on highways where they were guided by wires and electronics buried in the pavements ([Press-Courier 1960]). In the 1980s, Eureka PROMETHEUS was the first European project launched in the field of autonomous cars. Various universities and car manufacturers participated in this project. In the 1990s, research teams have begun to equip prototype cars with different range sensors including cameras, radars, LIDARs and ultrasonic ([Catling 1991, Ulmer 1994, Thorpe 1997]). Special functions such as automated parking and lane detection have begun to be developed ([Pomerleau 1989, Broggi 1995, Paromtchik 1996]).

In the 2000s, the Defense Advanced Research Projects Agency (DARPA) in the United States, organized challenges where autonomous cars compete for completing predefined courses without human driver ([Behringer 2005, Thrun 2006, Urmson 2008]). In parallel, the European Union funded projects such as PREVENT [Pre], HAVEit [Hav] or SARTRE [Sar] for developing and testing ADAS. These projects brought together different partners from automotive industry and research. Collaborative projects have continued in 2010s ([Broggi 2013, Ziegler 2014]). Most recent trends show various demonstrations of autonomous cars fruit of research and developments mainly lead by the automotive industry and tech companies. Google is

the initiator of these trends by creating its well-known autonomous car The Google Car ([Birdsall 2014]). Then the traditional car companies such as Daimler, BMW, Renault, Ford, *etc* followed ([Hohm 2014, Aeberhard 2015]).

1.1.3 Scientific and technical challenges

Software become more and more predominant and increasingly complex in cars. Complex Electric/Electronic systems powered by software are integrated into cars. Software is present in major systems that vary from powertrain and braking to driver assistance and infotainment. These systems may affect the *safety* and/or the *security* of the car and the passengers. They raise an enormous safety and quality challenges ahead ([Paliotta 2016]). Any software component must guaranty a level of safety. This includes the firmware, the operating system, the device driver, the network, the user applications, *etc*. The need for software certification becomes compulsory in order to guaranty safety. Standards such as the ISO 26262 ([ISO 2011]) are developed for this purpose.

Despite the scientific and technological progress, certified fully autonomous cars do not still exist nowadays. Autonomous vehicles are still in the phase of research and development. Recent cars on the market are however already equipped with some certified ADAS functions. For clarifying the differences between autonomous driving and ADAS, the SAE International – a worldwide association of automotive engineers – established a system of classification for identifying the level of automation of a car ([SAE 2014]).

The SAE classification is composed of six levels of driving automation as shown on fig. 1.1. Starting from level 0 to level 2, the human driver has to monitor the driving environment. While there is no automation at level 0, systems at level 1 assist the driver in the driving task. At level 2, the system can steer, brake or accelerate the car in predefined situations but the environment monitoring remains a job of the driver. From level 3 to level 5, the system is capable of monitoring the driving environment and even replaces the human driver. While at level 3, the driver should always be ready to intervene if the system reaches its limits, at level 4, such human intervention is no more required. The car is able to drive itself autonomously in predefined situations at level 4, and in every situations at level 5.

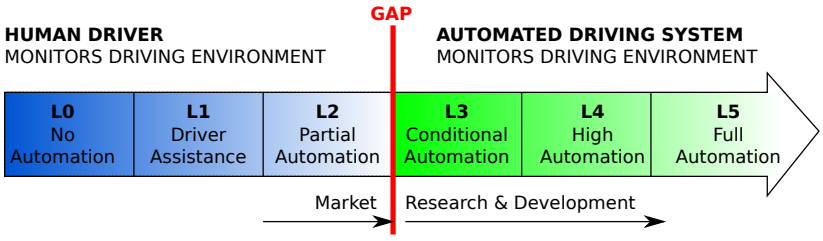


Figure 1.1 – Levels of driving automation (inspired from [SAE 2014])

Nowadays, no autonomous car can be classified at level 5. However, both

academy and industry already develop cars with automation level 4. For instance, the driverless cars from Google have been tested in urban environment in California but they cannot yet cope with all weather conditions ([Harris 2014]). The project CityMobil2 tested driverless buses running in the downtown of Trikala (Greece) in 2015 [Cit]. The buses shared pavements with other cars, pedestrians and cyclists but run at a maximum speed of 20 km/h.

Concerning the automotive market, ADAS systems like Adaptive Cruise Control (ACC) and the Lane Departure Warning (LDW) have been integrated since 2000s. The ACC keeps a vehicle at a desired speed and at a desired distance to the vehicle ahead set by the driver. The LDW warns the driver in case the car moves to close to the edge of the lane. Lane Keeping Assist systems (LKA) are able to actively steer the vehicle to keep it in the lane.

Cars in the automotive market are still limited to automation level 0,1 or 2. The gap that makes difficult to get over the level 2 is the miss of a safe electric/electronic system that would enable cars to monitor safely the driving environment. Environment monitoring is a safety critical task. If it fails, the damage can cost human life. Environment monitoring constitutes a scientific and technological challenge. Moreover, the requirements of robustness, reliability, certification and costs – familiar to the domain of automotive – makes this challenge difficult.

Environment monitoring is a task of a special system called **perception system**. The need for a safe, robust and reliable perception system constitutes the starting point for the present thesis.

1.2 Perception system

An autonomous car is driven by a computer system instead of a human driver. As a human driver, the driving system needs to continuously monitor the driving environment in order to make a driving decision like accelerate, brake or steer. The question arises: how the system can monitor the driving environment? The proposed approaches find their origins in the field of robotics. A mobile robot needs to be aware of its environment in order to operate within it. For instance, to navigate in a cluttered environment, a mobile robot needs to know the location of obstacles around and where are the navigable spaces. For this purpose, the robot rely on multiple sensors that observe the environment. The sensors provide measurements according to the spatial disposition and the physical properties of the sensed objects.

Range sensors such as laser-based sensors, stereo-camera, radars and ultrasonic sensors are commonly used in robotics and also for autonomous vehicles. These sensors are able to provide range measurements without a physical contact with the sensed objects. Laser sensors are based on Light Detection and Ranging (LIDAR) technology. They estimate the distance to the sensed object by measuring the time-of-flight (ToF) of laser pulses sent toward the object. The principle of ToF is also used in ultrasonic sensors, ToF cameras and radars. Ultrasonic sensors use acoustic waves instead of laser pulses while infrared light is used in ToF cameras. Radars

are able to estimate both distance and velocity of moving objects. Stereo-cameras also allow to estimate the distance to objects by computing depth information from pair of images.

Sensors provide raw measurements that do not contain directly the information required by the car for making decision. For instance, depth images from a stereo-camera are not enough for deciding whether the car should brake or steer. Consequently, measurements are processed by a **perception algorithm** for building a kind of a map of the environment called **environment model**. As shown on fig. 1.2, the system composed of range sensors and perception algorithm is called **perception system**. The role of the perception system consists in constructing a model of the environment. The model constitutes a dynamic computational representation of the environment that the autonomous car can interpret in order to make decisions and to perform actions. For instance, for performing self-navigation, the environment model can be used to compute a path or to avoid collision to obstacles. Once the path is chosen, the car can accelerate, throttle back, brake or steer by commanding the actuators.

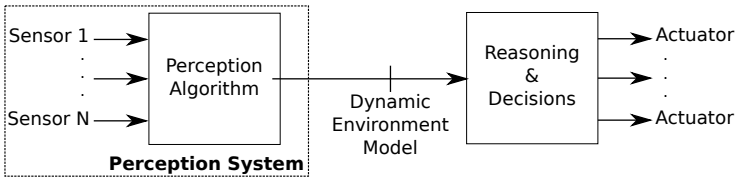


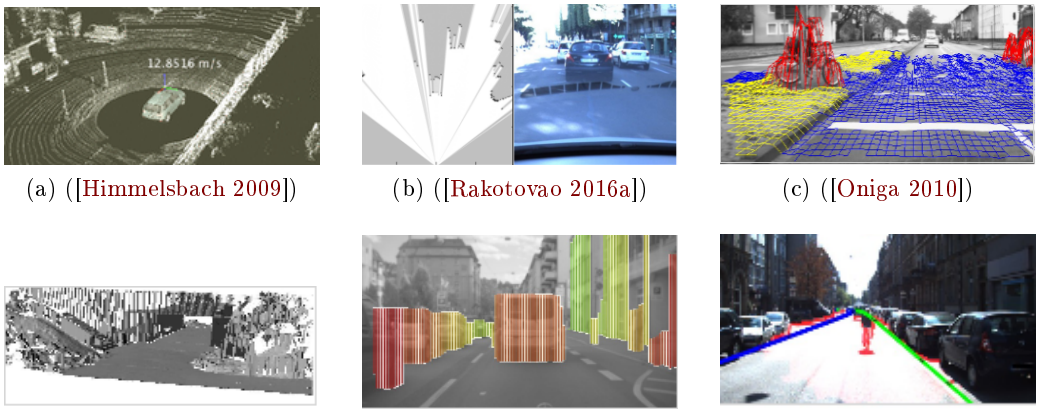
Figure 1.2 – Perception system

Remark Vision sensors provide additional data such as colors and textures. They can be used for various functions such as obstacles recognition, lane detection, traffic light, traffic sign recognition, detection and tracking, *etc*([Stiller 1997, Ernst 1999, Lorei 1999, Sun 2006, Tao 2013]). Vision-based sensors provide rich information about the environment. However, this thesis focuses only on range measurements and do not exploit the other information provided by vision sensors.

1.2.1 Environment model

There are different ways to model computationally the driving environment of a car. Models differ on the level of spatial details they provide, on the representation of free spaces, the way to model objects, and on the compactness of the environment representation. Figure 1.3 shows the main examples of environment model used for automotive perception. In the followings, we will refer as obstacle any object or alive being or infrastructure that may be present on a driving environment. Obstacles include cars, pedestrian, cyclists, vegetation, buildings, *etc*.

First, raw sensor models (fig. 1.3a) represent the environment with point clouds. They are commonly built from sensors providing dense point data such as LIDARs



(a) ([Himmelsbach 2009]) (b) ([Rakotovao 2016a]) (c) ([Oniga 2010])

(d) ([Triebel 2006]) (e) ([Pfeiffer 2011]) (f) ([Beyeler 2014])

Figure 1.3 – Examples of environment representations used for automotive perception: point cloud (1.3a), occupancy Grid (1.3b), elevation map (1.3c), multi-level surface map (1.3d), stixel world (1.3e), and geometric feature (1.3f)

and stereo camera ([Christensen 2008]). Point clouds model obstacles sensed by the sensor. They do not represent neither free spaces nor regions that are not yet explored by the sensors. The number of points explodes rapidly as long as sensors provide data points. This penalizes the compactness of the environment model. Next, two dimensional (2D) occupancy grids (1.3b) model the environment as a lattice composed of a finite number of cells ([Elfes 1989b]). Occupancy grids compute the probability that each cell is occupied by an obstacle by using sensor measurements. Obstacles are represented by likely occupied cells while likely empty cells represent free spaces.

Two-and-half dimensional (2.5D) representations propose an extension of 2D occupancy grids by providing more spatial details. Elevation maps (fig. 1.3c) store at a cell level the height of the object occupying the cell ([Oniga 2010]). The cell height is extracted from measurements from LIDAR or stereo-camera ([Christensen 2008, Nguyen 2009, Vatavu 2012]). Therefore, a cell is assimilated to a solid vertical structure with a given height. This makes elevation maps fail to represent overhanging structures such as bridges and tunnels. Multi-level surface maps (fig. 1.3d) propose an alternative that stores a list of multiple height values at a cell level ([Triebel 2006]). This allows the cell to represent more than one vertical structure. Multi-level maps are able to represent correctly overhanging structures. After that, full 3D representations can be modeled by 3D occupancy grids [Schmid 2010, Wurm 2010, Hornung 2013]. The environment is subdivided into adjacent cubic cells called voxels. The probability that each voxel is occupied by an obstacle is computed from sensor measurements.

The compactness of grid-based models depends on the number of cells: the higher is the number of cells, the less compact is the model. Apart from grid models, the

stixel worlds (fig. 1.3e) model the environment with adjacent rectangular sticks of a given width and height ([Pfeiffer 2011]). The sticks limit the free space in the front of the car. The stixel worlds are more compact than grid-based representations but they represent less spatial details. For instance, free spaces are represented implicitly. Finally, feature-based maps (fig. 1.3f) are the most compact environment models since obstacles are represented by simple geometric shapes. The environment is represented by a set of landmarks ([Thrun 2003]), or by geometric patterns such as points, lines and polygons ([Thrun 2003, Burgard 2008]). Lines are used for detecting and delimiting lanes ([Beyeler 2014]). Bounding boxes are frequently used for tracking cars or pedestrians ([Barth 2009, Enzweiler 2009]).

Despite their differences, the environment models are rather complementary. For instance, sensors producing point clouds can be used for building occupancy grids. Then, the latter can be used in turn to build stixel worlds [Pfeiffer 2011]. Choosing an environment model depends actually on the function that will be implemented upon the model. For instance, for the fusion of measurements from heterogeneous sensors, occupancy grids are well-adapted [Thrun 2005]. For a parking assistant system, bounding boxes are enough to warn the driver whether the car is approaching an obstacle too closely.

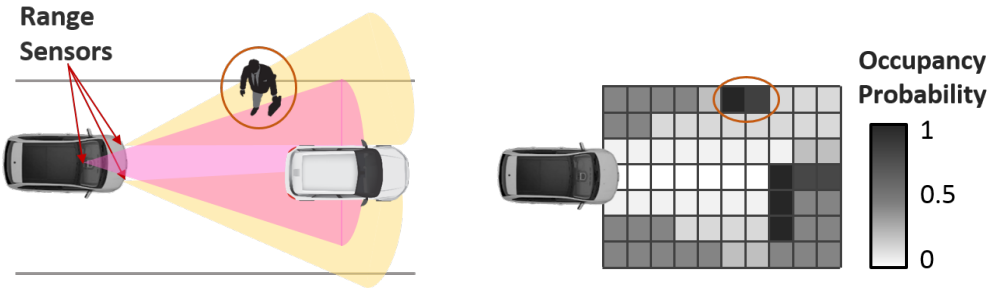
Another challenge for environment models is the support of dynamics. The detection and tracking of moving objects require sophisticated techniques that have to take time into account. The objective is to estimate at each time step both position and speed of moving objects. For instance, filtering techniques have been intensively applied for developing dynamic grid based environment models ([Coué 2006, Gindele 2009, Danescu 2011, Vatavu 2014]). The dynamics allow to predict the future state of moving objects, to assess risk of collisions and to plan further actions ([Laugier 2011]).

1.2.2 Occupancy grids

The present thesis focus on occupancy grids for a historical reason. The e-Motion project-team common to INRIA Rhône-Alpes and the LIG (Laboratory of Informatics of Grenoble) has developed a family of perception algorithms called Bayesian Occupancy Filter (BOF) ([Coué 2006, Nègre 2014, Rummelhard 2015]). The hardware integration of BOF was included in the long-term plan of the team. However, BOF takes multi-sensor occupancy grids as input. Then, a hardware integration of BOF requires first a successful hardware integration of occupancy grids.

Occupancy grids have been initially developed by Moravec and Elfes in the mid-80s ([Moravec 1985, Elfes 1987, Moravec 1988, Elfes 1989b, Elfes 1989a]). They have been widely used for mapping the environment of indoor and outdoor robots ([Thrun 2003, Stepan 2005, Burgard 2008, Jessup 2014]). They have been also adopted as a base of automotive perception algorithms ([Coué 2006, Weiss 2007, Schmid 2010, Laugier 2011, Nuss 2015]). In the automotive context, an occupancy grid represent the driving environment with a collection of cells paired with their occupancy probabilities. The location of empty spaces and obstacles is estimated

from the spatial disposition and the occupancy probabilities of cells.



(a) The driving environment (b) The corresponding occupancy grid

Figure 1.4 – An example of a driving environment in the front of a car and the corresponding occupancy grid

Figure 1.4 shows an example of a driving environment modeled by occupancy grids. Sensors mounted on board an intelligent vehicle observe the environment and provide measurements (fig. 1.4a). The occupancy grid on fig. 1.4b represents the environment as a 2D flat grid composed of squared cells. The occupancy grid framework computes the probability that each cell is occupied. Cells with occupancy probabilities close to zero are likely empty. Those with occupancy probabilities close to unity are likely occupied. Finally, those non discovered by sensors have unknown state: neither occupied nor empty. Their occupancy probabilities are equal to $1/2$.

The occupancy grid framework is based on probabilistic principles. This provides the following advantages. Measurements from sensors are noisy due to sensor imperfections and external conditions like the temperature or the lighting condition. The probabilistic approaches allow occupancy grids to handle appropriately the uncertainties of sensor measurements by modeling them with probabilistic distributions. By this way, measurement uncertainties are taken into account in the process of multi-sensor fusion. This enables the fusion of heterogeneous sensors with different characteristics mounted at different locations on the car.

Occupancy grids support the common range sensors used in robotics such as LIDARs, radars, ultrasonic and stereo-vision. The fusion of multiple and heterogeneous sensors yields two advantages. First, the fusion allows to add more sensors in order to get more information about the environment. Using multiple sensors increases the sensor coverage of the environment and enables a redundancy of information that can be useful whenever a sensor fails. Second, the fusion improves robustness. It allows to overcome the physical limitations of a sensor alone. For instance, in dark luminosity, cameras are less efficient but can be reinforced or replaced by LIDARs ([Laugier 2011]).

Finally, the subdivision of the environment into cells allows to represent at the same time occupied regions, free regions and unknown regions. The notion of cells present a low level abstraction that is enough for performing basic taks for mobile robots like free space searching, path planning, obstacle avoidance and navigation.

Objects are modeled by occupied cells, regardless of their nature. This makes occupancy grids suitable for representing unstructured environments like in urban area.

1.3 Objective: multi-sensor fusion module

The present thesis aims to integrate the computation of multi-sensor occupancy grids on a hardware embedded on-board intelligent cars. This is equivalent to build the **Multi-sensor Fusion (MSF)** module depicted on fig. 1.5. The role of the module is to fuse the measurements periodically produced by range sensors. Within a period, the module takes as inputs range measurements and fuses them into a unique occupancy grid.

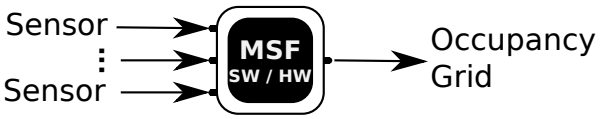


Figure 1.5 – The Multi-sensor Fusion (MSF) module composed of sensors and a software (SW) and hardware (HW) integration of MSF

The occupancy grids produced by the MSF module include only the occupancy probability of each cell. No information about dynamics is included. The MSF module is intended to be paired with the INRIA Rhône-Alpes's perception algorithm BOF ([Coué 2006]). The latter identifies, through a Bayesian filter, which cells are likely dynamics and estimates their velocities.

The timeline of the execution of MSF module coupled with the BOF algorithm is illustrated on fig. 1.6. At each iteration, a process called **Multi-sensor Fusion (MSF)** builds an Occupancy Grid (OG) by fusing all measurements produced by on-board sensors. After that, the occupancy grid is processed by the BOF in order to extract the information about the dynamics of cells and their velocities. At an iteration t, the BOF takes as inputs both the occupancy grid produced by MSF, and the dynamics and the velocities of cells computed at iteration $t-1$. It subsequently produces estimations about the velocities of cells.

The long-term objective is to embed the MSF module on cars commercialized on the automotive market. This intends to enable cars to monitor their driving environment by using multiple sensors mounted on-board. Environment monitoring is required to move cars from driving automation level 2 toward automation level 3. Targeting the automotive market, and the automotive domain in general, implies that the MSF module is subjected to the following constraints:

- First, computations have to be performed in real-time. The rate of production of occupancy grids must be at least equal to the rate of sensor measurements. By respecting this, the MSF module follows the evolution of the environment as it is captured by sensors.

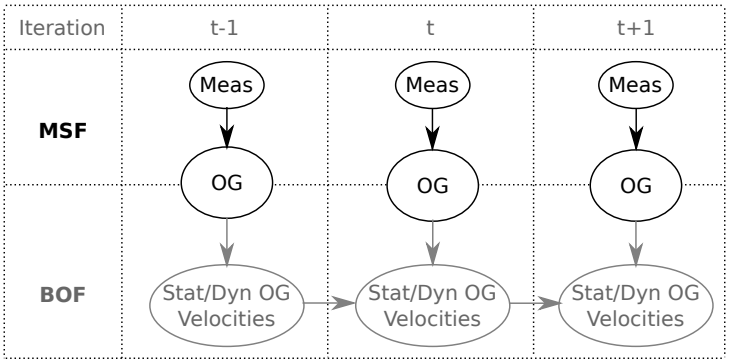


Figure 1.6 – How BOF exploits the occupancy grids built by the MSF module

- Second, a mass production cannot afford expensive hardware. Thus, the integration must be realized on low-cost computing platforms.
- Third, the computing platform must have a low electrical power-consumption to fit within the limited source of electrical energy on cars. A low energy consumption is also required to face environmental challenges.
- Finally, the HW/SW integration of occupancy grids must be safe. Safety includes handling sensor uncertainties, knowing numerical errors during computations, and guarantying determinism.

1.4 Addressed problem

To show the challenges imposed by the above constraints, let us detail the computing demands required by occupancy grids, the computing performance available on low-cost and low-power processing platforms, and the safety challenges involved by the HW/SW integration of occupancy grids.

1.4.1 Computing requirements of occupancy grids

The computing requirements of occupancy grids are influenced by the number of cells, the number of sensor measurements and the real-time requirement. To model the driving environment of a car, an occupancy grid with thousands to millions of cells is required. For instance, an occupancy grid of 100 m-by-100 m with a cell size of 10 cm-by-10 cm contains 1 Million of cells. When extended in three dimensions with a height of 2 m, the occupancy grid will contain 20 Millions of cells.

Concerning the number of measurements, common range sensors such as LIDARs or stereo camera provide thousands to million of measurements per second. When these numbers are multiplied by the number of cells, the real-time calculation of occupancy grids requires to execute a dozen of Billions of operations per second in order to guaranty a minimum of latency.

For implementing occupancy grids, a digital model of real-number operations is required in order to process probabilities. Thanks to the availability of processors with hardware floating-point units and to their wide support by different programming languages, occupancy grids are traditionally implemented and tested with floating-points ([Coué 2006, Wurm 2010, Hornung 2013, Nègre 2014]). However, a real-time processing of occupancy grids require in practice a hardware having a computing performance with a dozen of Giga Floating Points Per Second (GFLOPS).

1.4.2 Hardware constraints

Those Billions of Floating Points Per Second can be found on workstations. Hence, occupancy grids have been integrated and experienced into workstations composed of a high-end CPUs and/or a Graphical Processing Unit (GPU) ([Yguel 2006, Homm 2010, Adarve 2012]). These platforms provide flexibility for testing algorithms and proof of concepts. GPUs allow to apply parallel computing techniques for calculating simultaneously the occupancy probabilities of several cells ([Nègre 2014]).

These platforms are not however certified for being used for automotive applications. In addition, they are expensive to purchase and have high power consumption. The power consumed by a high-end CPU is about 50 W ([Dargie 2015, Abou-Of 2016]). A GPU consumes more than hundreds of watts ([Stroia 2015]). To asses these numbers, the engine of a mild hybrid-electric vehicle is equipped with an electric energy storage of up to 150 Wh ([Burke 2007]). If the same quantity of energy powered a high-end CPU and GPU, the energy storage would drain in less than two hours.

The AUTOSAR – a joint initiative of industrial players for managing the complexity of automotive electric and electronic architectures – reported that the processing platforms considered as safe and reliable for automotive applications are mainly based on microcontrollers ([Heinecke 2004]). In recent years, due to the increasing number and the diversity of automotive applications, automotive manufacturers and suppliers have gradually increased the computing power by utilizing dedicated embedded CPUs ([Monot 2010]).

Microcontrollers and embedded CPUs designed for automotive applications respect the constraints of low-cost and low-power. They are also designed upstream to answer to the requirements of safety and reliability of automotive applications. However, microcontrollers and embedded CPUs have low computing performance compared to workstations. They are not able to process the dozen of Billions of operations per second required for processing occupancy grids in real-time. Furthermore, microcontrollers and embedded CPUs have limited floating-point performance. The advent of embedded parallel processing platforms such as embedded GPUs may be seen as an alternative ([NVIDIA 2015]).

We have explored the use of embedded parallel architecture for computing occupancy grids ([Rakotovao 2015b, Rakotovao 2015a]. In a previous work, the occupancy grid framework has been integrated on an embedded platform based on

many-cores. The platform is described in [Melpignano 2012]. It is composed of 64 independent computing cores. Programming paradigms based on parallel computing techniques have been experienced to meet the real-time objective. The integration achieved real-time performance while consuming less than 1 W of power. However, such platform is still at an experimental phase. Parallel processing platforms are not yet ready for critical automotive tasks such as perception where certification is primordial in order to guaranty safety.

1.4.3 Safety challenges involved by the HW/SW integration

Sensor uncertainties Occupancy grids constitute a computational model of the driving environment. Information about the environment are derived from sensors mounted on-board the car. Thus, occupancy grids must reflect what the sensors have sensed. Since sensor measurements are uncertain, the process of building occupancy grids must take into account measurement uncertainties when computing occupancy probabilities.

Numerical errors Occupancy grids are a well established probabilistic framework for performing multi-sensor fusion. Probabilities are real-numbers. A computer is finite. On a numerical point of view, processing probability calculations on finite resource introduces forcibly a numerical error. Calculations cannot be processed with arbitrary precision. The difference between the mathematics and the numerics must be known in order to ensure safety and robustness.

For instance, assume that a high-level application has to decide whether a cell is occupied or empty. For this purpose, the occupancy probability of the cell is compared to a threshold. To make a reliable decision, the comparison must take into account the numerical error involved by the SW/HW integration of probability calculations.

Determinism Given a set of sensor measurements, the occupancy probabilities computed by the SW/HW integration must be known deterministically. Determinism means here that numerical values of occupancy probabilities must be the same regardless of technical details such as programming language, compilers, compiling options or processor architecture. In fact, if the values of occupancy probabilities change if one of the above parameters is modified, then the SW/HW integration is not robust with respect to technical details. That reduces the safety and the reliability of the HW/SW integration.

1.5 Research contribution

To deal with the computing requirements, the hardware constraints and the safety challenges, both theoretical and algorithmic improvements of occupancy grids are required for a successful HW/SW integration of the MSF module. For this purpose, this thesis introduces the **Integer Occupancy Grid** framework. Integer

Occupancy Grids express the occupancy state of a cell through an integer called **occupancy index**. Integer Occupancy Grids support probabilistic fusion of multiple sensor. Their originality consists in the process of multi-sensor fusion as being an arithmetic addition of occupancy indexes.

The main contributions of this thesis are listed below:

- The formulation of the theoretical foundation of the paradigm of **Integer Occupancy Grids** and the multi-sensor fusion through integer arithmetic.
- The exploration of algorithmic data structure for storing Integer Occupancy Grids.
- The design of algorithms for computing Integer Occupancy Grids. The computation of Integer Occupancy Grids takes into account sensor uncertainties.
- The theoretical and experimental study of the numerical errors involved by Integer Occupancy Grids in order to verify their accuracy.
- The application of Integer Occupancy Grids for the fusion of four automotive LIDARs mounted on a prototype car.
- The experimental validation of the HW/SW integration of Integer Occupancy Grids on an embedded CPU. The integration is able to fuse LIDARs into an Integer Occupancy Grid whit a real-time performance on a low-cost and low-power embedded CPU.

1.6 Thesis overview

To conclude this chapter, this manuscript presents the theoretical development of Integer Occupancy Grids and their experimentation on an intelligent car equipped with multiple range sensors. The remainder of the manuscript is composed by the following chapters.

Chapter 2 reviews the paradigm of standard occupancy grids. It presents some basics of probability calculus followed by the definition of occupancy grids with probabilistic terms. A review of the literature about the computation of occupancy grids is provided thereafter.

Chapter 3 introduces the Integer Occupancy Grid framework. It explains how to represent occupancy probabilities with integers and how to perform probabilistic multi-sensor fusion based on integer arithmetic.

Chapter 4 describes the application of Integer Occupancy Grids for fusing multiple range sensors mounted on an experimental intelligent car. It presents the integration of Integer Occupancy Grids on an embedded CPU designed for automotive applications. Experimental results, performance analyses and discussions are presented in this chapter.

Finally, chapter 5 concludes with an evaluation of the work performed and presents perspectives for future research.

Chapter 2

STATE-OF-THE-ART ON OCCUPANCY GRIDS

The objective of this thesis is to develop the Multi-sensor fusion (MSF) module. The role of the module is to fuse periodically the measurements from multiple range sensors mounted on a car. The module implements the Occupancy Grid framework on a processing platform. To master both theoretical and algorithmic foundations of the framework, this chapter reviews occupancy grids: their formal definition, their computation, and the algorithmic data structures for storing them for an HW/SW integration.

The occupancy grid framework was initially developed by Moravec and Elfes in the mid-80s ([Moravec 1985, Elfes 1987, Moravec 1988, Elfes 1989a, Elfes 1989b]). It uses the Theory of Probability for handling sensor uncertainties [1]. This chapter begins with a brief review of the basics of probability in Section 2.1. The concepts of probability will be used to give a formal definition of occupancy grids in the next section. Section 2.3 reviews the computation of occupancy grids from a single sensor measurement. The case of multi-sensor measurements is reviewed in Section 2.4. Finally, the algorithmic data structures used for storing occupancy grids are presented and discussed in Section 2.5.

2.1 Basics of Probability

The mathematical study of probability begun between the 16th and the 17th centuries. Precursor works were realized by mathematicians Girolamo Cardano, Blaise Pascal and Pierre de Fermat who studied the mathematical treatment games of chance. In 1718, Abraham Moivre published the first textbook on probability theory entitled "*The Doctrine of Chances or a Method of Calculating the Probability of Events in Play*" ([Moivre 1718]). In 1763, a first version of the theorem of Bayes was

[1] The Dempster-Shafer Theory of Evidence, the fuzzy sets, and other formalisms have been also applied in the literature for handling sensor uncertainties and for building grid-based environment models ([Ribo 2001, HoseinNezhad 2002, Noykov 2007, Moras 2014, Yu 2015]).

first introduced in a posthumous paper attributed to Thomas Bayes ([Bayes 1763]). After that, a more precise formulation of the theorem was established by Pierre Simon de Laplace in the beginning of the 18th century ([Laplace 1812]).

At the beginning of the 20th century, probabilities are applied in various scientific disciplines. However, a mathematical foundation proper to theory of probability and application-independent was still missing. Kolmogorov filled this gap by giving an axiomatic foundation of the theory of probability ([Kolmogorov 1956]). Probabilities are no more limited to game of chance and different sementical interpretations of probability have appeared as reported by [Jaynes 2003]. Despites the differences of interpretation, the mathematics rely on the same theory.

The probability theory can be explained through the notion of random variables. Probabilities are computed over the possible values of such variables. The probability of a variable can be conditioned by the value of another variable. This gives the concept of conditional probabilities. Besides, two random variables can be though independent. The above concepts constitute the basics of probability required for understanding the present manuscript. They are part of the mathematical theory of probability, independent of any application context. In the scope of this thesis, probabilities can be interpreted in various ways. Such interpretations are anticipated and discussed at the end this section.

2.1.1 Random variables

A **random variable** X is a variable subject to modification and that can take on multiple values. A random variable can designate the result of an experiment, the state of a system, the value of a measurement, *etc.* For instance, the result of the toss of a coin is a random variable X that can take on two values: head or tail.

Let X denotes a random variable and S the set of all possible values of X. The random variable X is **discrete** if S is finite or is countably infinite. The toss of a coin or the roll of a die are for instance two examples of discrete random variables. The random variable X is called **real-valued** if S is a subset of $\mathbb{R}$. For instance, the roll of a die is a real-valued random variable that has six possible values: $1, 2, 3, 4, 5, 6$. Finally, the random variable X is **continuous** if X is real-valued and S is uncountably infinite. For instance, the temperature measured by a thermometer is a continuous random variable since the measurement can be any real number between a minimal temperature and a maximal temperature.

2.1.2 Probability

Definition 2.1.1. *Let X be a discrete random variable, S the set of all possible values of X, and x an element of S. The **probability distribution** of X is a function P that assigns a non-negative real number to each value x of X such that:*

$$P(x) \geq 0 \quad \text{and} \quad \sum_{x \in S} P(x) = 1 \tag{2.1}$$

The quantity $P(x)$ denotes the **probability** that X takes a value x. From eq. (2.1) follows that $P(x)$ is a real-number between 0 and 1.

Definition 2.1.2. *Let X be a discrete random variable having N possible values $x_1, \ldots, x_N$. The values $x_i, i \in \{1, \ldots, N\}$ are **equiprobable** if:*

$$P(x_1) = \ldots = P(x_N) = 1/N \tag{2.2}$$

If X is continuous, it has infinite number of possible values, then intuitively the probability $P(x)$ is null. A function that allows to capture the probability distribution a continuous variable is the cumulative distribution function.

Definition 2.1.3. *Let X be a continuous random variable. The **cumulative distribution function** F_X denotes the function that assigns to a real number a the probability that the value of X is less than or equal to a: $F_X(a) = P(X \leq a)$.*

The cumulative distribution function is a monotonically increasing function. Notice that the definition of cumulative distribution function is also valid for real-valued discrete random variable.

Definition 2.1.4. *Let X be a continuous random variable and F_X its cumulative distribution function. If the cumulative distribution function of X is differentiable, its derivative p_X is called the **Probability Density Function (PDF)**:*

$$p_X(a) = \frac{dF_X(a)}{da} \tag{2.3}$$

In order to simplify the notations, we will denote $p_X(x)$ by $p(x)$. The PDF $p(x)$ of a continuous random variable X respects the following properties. First, since $F_X(x)$ is a monotonically increasing function, then $p(x)$ is a positive or null function. Second, unlike probabilities, the values of the PDF of X are in $\mathbb{R}$ and is not always limited to the interval $[0, 1]$. Third, like probabilities, the integral of a PDF is equal to 1:

$$\int_{-\infty}^{+\infty} p(x)dx = 1 \tag{2.4}$$

A common example of PDF is the *Gaussian* function with parameters σ and μ:

$$p(x) = \frac{1}{\sqrt{2\pi\sigma^2}} exp\left\{-\frac{(x-\mu)^2}{\sigma^2}\right\} \tag{2.5}$$

The parameters σ and μ denote respectively the standard deviation and the mean of the distribution.

2.1.3 Conditional probability and the Theorem of Bayes

Let X and Y be two random variables, and S_X and S_Y the sets of all possible values of X and Y. Consider the variable Z that takes as a value the pair (x, y) where

$x \in S_X$ and $y \in S_y$. The variable Z forms a random variable. If X and Y are discrete, the probability distribution of Z is called **joint distribution of X and Y**, and is noted by $P(x \wedge y)$. It denotes the probability that X takes value x **and** Y takes value y. If X and Y are continuous, the PDF of Z is a multivariate function denoted by $p(x \wedge y)$ and is called **joint probability density function of X and Y**.

Definition 2.1.5. *Let X and Y be two random variables. Let y be the value of Y. If X and Y are discrete and $P(y) > 0$, then the **conditional probability of X given** Y denotes the quotient:*

$$P(x|y) = \frac{P(x \wedge y)}{P(y)} \tag{2.6}$$

*If X and Y are continuous and if $p(y) > 0$, the **conditional probability density function of X given** Y is defined by the quotient:*

$$p(x|y) = \frac{p(x \wedge y)}{p(y)} \tag{2.7}$$

The conditional probability $P(x|y)$ and the conditional PDF $p(x|y)$ satisfy:

$$\sum_{x \in S_X} P(x|y) = 1 \quad \text{and} \quad \int_{x \in \mathbb{R}} p(x|y)dx = 1 \tag{2.8}$$

Two main theorems follow from the definition of conditional probabilities: the *Theorem of Total Probability* and the *Theorem of Bayes*.

Theorem 1. Theorem of Total Probability

. Let X and Y be two random variables, then

$$P(x) = \sum_y P(x|y)P(y) \quad \text{if } X \text{ and } Y \text{ are discrete} \tag{2.9}$$

$$p(x) = \int_{y \in \mathbb{R}} p(x|y)p(y)dx \quad \text{if } X \text{ and } Y \text{ are continuous} \tag{2.10}$$

The strength of the above theorem is that it allows to compute the probability of x as a function of the conditional probability of x given all possible values of Y.

Theorem 2. Theorem of Bayes

. Let X and Y be two random variables, then

$$P(x|y) = \frac{P(y|x)P(x)}{P(y)} = \frac{P(y|x)P(x)}{\sum_{x'} P(y|x')P(x')} \quad \text{(discrete)} \tag{2.11}$$

$$p(x|y) = \frac{p(y|x)p(x)}{p(y)} = \frac{p(y|x)p(x)}{\int_{x'} p(y|x')p(x')dx'} \quad \text{(continuous)} \tag{2.12}$$

The Theorem of Bayes can be extended in the case where X is discrete and Y is continuous:

$$P(x|y) = \frac{p(y|x)P(x)}{p(y)} \tag{2.13}$$

Similarly, if X is continuous and Y is discrete, the extension of the Theorem of Bayes gives:

$$p(x|y) = \frac{P(y|x)p(x)}{P(y)} \tag{2.14}$$

2.1.4 Independence

This section gives a mathematical definition of independence. No semantical interpretation of the concept of independence is emitted.

Definition 2.1.6. *Two random variables X and Y are* ***mutually independent*** *if*

$$P(x \wedge y) = P(x)P(y) \qquad \textit{(discrete)} \tag{2.15}$$
$$p(x \wedge y) = p(x)p(y) \qquad \textit{(continuous)} \tag{2.16}$$

A necessary and sufficient condition of the mutual independence of two random variables X and Y is:

$$P(x|y) = P(x) \qquad \text{(discrete)} \tag{2.17}$$
$$p(x|y) = p(x) \qquad \text{(continuous)} \tag{2.18}$$

Let Z be a random variable. The variables X and Y are **conditionally independent** if:

$$P(x|y \wedge z) = P(x|z) \qquad \text{(discrete)} \tag{2.19}$$
$$p(x|y \wedge z) = p(x|z) \qquad \text{(continuous)} \tag{2.20}$$

The independence of X and Y is conditioned by the knowledge that the value of Z is z. Notice that X and Y are conditionally independent does not mean that they are mutually independent.

2.1.5 Interpretation of probability

The subsections 2.1.1 to 2.1.4 gave the mathematical basics of probabilities, regardless of their semantical interpretation. In the present thesis, the meaning of a probability can be interpreted in two ways: in the frequentist view or in the bayesian view.

The frequentist interpretation of a probability is associated to random variables that represent a physical observation. The process (or the experiment) leading to the observation can be repeated for a large number of time in order to affect a probability to a random variable. For instance, consider a human operator that rolls a die. Let X be the discrete random variable that represents the result of a

roll. For computing the probability that a roll gives $X = 3$, the human operator rolls the die m number of times, checks the result, and counts the number n of times where the result is 3. When m is a large number, [Kolmogorov 1956] states that the ratio $\frac{n}{m}$ becomes very closed to the real value of $P(X = 3)$. Therefore, $P(X = x)$ is interpreted as the probability or the chance of the event "X takes the value x" to occur after a long run of trials.

Besides, on a bayesian point of view, a probability represents the degree of certainty associated to a statement. The probability $P(x)$ measures how certain is the statement "X takes the value x" [Jaynes 2003]. For instance, in the example of the roll of a die, assume that the human operator is blindfolded. He rolls the die and one asks to him to guess the result. Let X denotes the result guessed by the human operator. It is a discrete random variable with values in $\{1, 2, 3, 4, 5, 6\}$. Then since the human operator is blindfolded, he cannot state absolutely that the result of the roll is x. However, he can answer with a degree of confidence $P(x)$ that the result of the roll is $X = x$.

Unlike the frequentist interpretation, the bayesian interpretation does not require a repetition. Moreover, it allows to apply probabilities for reasoning under *uncertainties*, where information about a random variable are only partial and *incomplete*. The Theorem of Bayes is essential for reasoning under uncertainty. For understanding its importance, let us consider the following example. Peter is on the way back to his home. By far, he sees through a window that the light is on. Therefore, he thinks that someone is inside his house.

Let us now analyze this example with random variables. Let X be the random variable that represents the presence of a person within Peter's home. Two values are possible: *yes* or *no*. The statement $X = yes$ means that "someone is within the room" while $X = no$ means "no one is within the room". Additionally, let D represent the status of the light in Peter's home. The light can be *on* or off. By observing that the light is on ($D = on$), Peter thinks that someone is inside ($X = yes$). However, since Peter is still far from his house, he cannot be certain that there is actually a person inside.

The Theorem of Bayes allows to computes the degree of certainty of Peter about the presence of a person in his house:

$$P(x|d) = \frac{P(d|x)P(x)}{P(d)} \tag{2.21}$$

where, $x \in \{yes, no\}$ and $d \in \{on, off\}$. The probability $P(x|d)$ is called the **posterior distribution of** X while $P(x)$ is called **prior distribution of** X. The probability $P(d|x)$ is called the **generative model**. It describes probabilistically a causal relation about how the variable X influences the value of the variable D. For instance, knowing that there is someone inside the house, the probability that the light is on is 70%. Therefore, when no observation is not available, the degree of certainty about the value of a variable is described by the prior distribution. Once an observation is available, the Theorem of Bayes of Bayes allows to update the degree of certainty by using the generative model.

2.2 The Occupancy Grid Framework

The occupancy grid framework aims to model the environment of a mobile robot by using measurement from range sensors ([Elfes 1989a]). The framework follows the following principles. An occupancy grid is a grid-based representation of the environment. It subdivides the environment into a collection cells. A cell is a portion of the environment that can be occupied by an obstacle or not. The occupancy grid framework infers the state of a cell from sensor measurements. Measurement uncertainties are modeled by probability distributions called sensor models. The latter serve for computing the occupancy probability of each cell, that is the probability that a cell is occupied. The occupancy probabilities can be derived from a single sensor measurement or from multiple measurements.

In the following sub-sections, let us give successively an overview of the problem of environment modeling, followed by definitions of the concepts of sensor model, grids, and cells. A formal definition of occupancy grids is given thereafter.

2.2.1 Problem overview

An occupancy grid serves as an environment model of the world in which a mobile robot operates. The world is composed of various physical obstacles of different natures, with different sizes and physical properties. In the context of an autonomous car, the term "obstacle" designates any living beings or objects that can be found in a traffic scene. Example of obstacles are human beings, animals, vegetation, cars, bicycle, buildings, road infrastructure, traffic signs, *etc.*

Modeling the physical world means estimating where obstacles are located and where are the empty regions. As illustrated on fig. 2.1a, such information should be captured by the environment model which is an occupancy grid in the this thesis. As shown on fig. 2.1b, modeling the surrounding world with an occupancy grid requires two major steps. First, sensors sense the physical world and provide outputs called **measurements**. Second, sensor measurements are processed for building an occupancy grid. Hence, the resulting occupancy grid models the physical world as "seen" by the sensors.

2.2.2 Sensor model

For retrieving information about the world surrounding an autonomous vehicle or a mobile robot in general, range sensors are commonly used [Elfes 1989b, Thrun 2005, Siciliano 2008, Eskandarian 2012]. A **range sensor** is a device that senses the world by exploiting the properties of a physical support such as light, radio waves, or acoustic waves. The process of sensing outputs a measurement that reflects the world as sensed by the sensor.

Range sensors commonly used in robotics include laser scanners, time-of-flight (ToF) sensors, vision sensors, radars, *etc.* Thus, the measurement can be a single point in the space, a collection of points, a distance, a full image with depth infor-

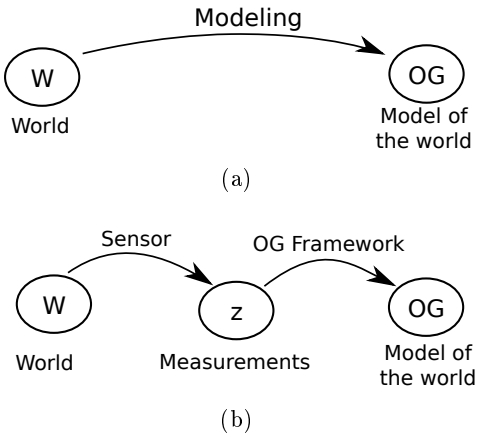


Figure 2.1 – Modeling the world into an occupancy grid

mation, *etc.* The mathematical nature of the measurement depends on the type of the sensor. It can be a scalar, a vector, a set of vectors, a matrix, and so forth.

A range sensor has a limited field-of-view (FOV). The FOV is the extent of the world observable by the sensor. Only obstacles located within the FOV can influence the value of the measurement of a sensor. For instance, fig 2.2 shows an example of a range sensor having a FOV represented by a gray sector. The environment is composed of two obstacles: one within the sensor's FOV and another outside. The sensor senses the obstacle located inside it's FOV and returns a measurement z accordingly.

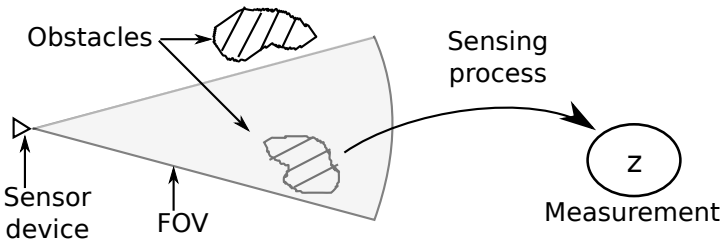


Figure 2.2 – A range sensor senses the world within its FOV and returns a measurement z

Let us now look for a mathematical model that describes the process of sensing. Such model consists to a mathematical relation between the measurement and the cause of the measurement. For this purpose, let us consider the example of a sensor which measurement depends on the distance to the nearest obstacle.

Let z denote a measurement and d the distance to the nearest obstacle. Thus, the mathematical model of the process of sensing must be a mathematical relation between z and d. Assume that the sensor is perfect. Since, the measurement z

depends on the distance d, when the value of d is known, it should be possible to determine the value of z in a **deterministic** manner. Consequently, for a perfect sensor, there would exist a function f such that $z = f(d)$.

However, in reality, if a sensor observes twice the same obstacle located at the same distance, it may output two different values of measurement. That means, sensors are actually imperfect and measurements are *uncertain*. In practice, measurements do not depend only on the distance to the sensed obstacle. External conditions such as the temperature of the environment, the lighting condition, or the materials of the sensed obstacle may influence on the measurement. Uncertainties may be also due to imperfections of the sensor device, its mechanical parts, or its electric/electronic components.

Thus, the mathematical relation between z and d has to take into account the uncertainties of measurements. In order to capture sensor uncertainties, let us model the relation between the measurement z and the distance d not with a deterministic function, but with a PDF. A distance is a continuous value. Let us assume that the measurement is also continuous. Then, the relation between z and d is modeled as follows.

Definition 2.2.1. *A **sensor model** designates a conditional density function that describes the relation between a sensor measurement and the cause of that measurement. It models the measurement uncertainties.*
When a measurement z depends on the distance d to the nearest obstacle, the sensor model is represented by the PDF $p(z|d)$.

Herein, d is considered as the *cause* of z by abstracting the other parameters that may influence the value of z. For a perfect sensor, by knowing the distance d to the nearest obstacle, the corresponding measurement cannot be different to $f(d)$. Hence, the sensor model is derived from the Kronecker's delta distribution:

$$p(z|d) = \begin{cases} 1 & \text{if } z = f(d) \\ 0 & \text{otherwise} \end{cases} \tag{2.22}$$

For building the sensor model of a non-perfect sensor, we can employ the frequentist technique that consists in repeating the measurement process several times in order to derive a probability distribution. A measurement is caused by the presence of obstacles within the sensor's FOV. In order to model the sensor's behavior, multiple observation scenarios can be set up. A scenario consists to put obstacles at a known location within the sensor's FOV. The sensor will react by giving measurements. After repeating several scenarios, the sensor model can be built by using the data about the obstacles, their locations relative to the sensor and the corresponding measurements. Advanced statistical methods like those proposed in [Bishop 1995] can be used for getting the PDF of the sensor model.

Finally, the sensor model describes the process of sensing with a PDF. It keeps track of the relation between a measurement and the physical world that has caused the measurement. Let us see in the followings how to utilize the measurement for building a computational model the physical world.

2.2.3 Grid and cells

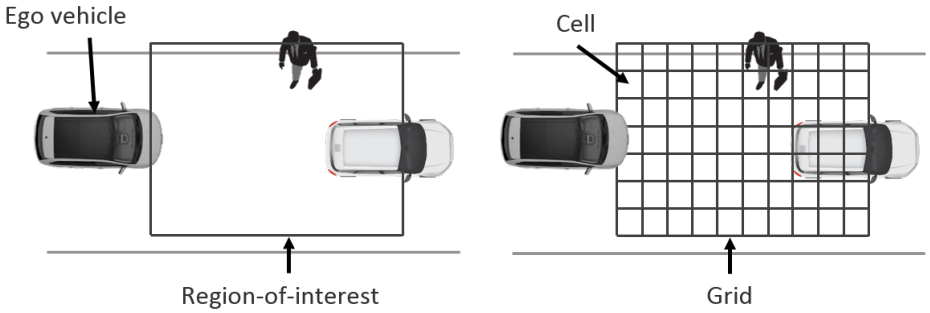
The present thesis uses occupancy grids for modeling the driving environment of autonomous cars. As indicated in their name, occupancy grids model a physical world with a grid subdivided into cells. For understanding the principle of occupancy grids, let us give a formal definition of the concept of grids and cells.

Definition 2.2.2. *Consider a bounded region-of-interest within a spatial world. A* ***grid*** *is a subdivision of the region-of-interest into a finite number of adjacent-but-disjoint subregions. The subregions are called* ***cells****.*

Let R denote the region-of-interest, $\mathcal{G}$ the grid, N the number of cells, and c_i the i-th cell. Therefore,

$$\mathcal{G} = \{c_i\}, i = 1, \dots N$$
$$\forall i \neq j : c_i \cap c_j = \emptyset$$
$$R = \bigcup_{i=1}^{N} c_i$$

Depending on the application, a grid can be one-dimensional (1D), two-dimensional (2D) or three-dimensional (3D). On the example on fig. 2.3a, the region-of-interest is a rectangular region in the front of an ego vehicle. A 2D grid subdivides the region into multiple cells on fig. 2.3b. Cells cover squared subregions having the same size.



(a) (b)

Figure 2.3 – Subdivision of a region-of-interest into a grid

There exists different forms of grid: cartesian, polar, spherical, *etc.* The grid is called **uniform** if cells have uniform size. This thesis focus on uniform cartesian grids. A grid has two parameters: a **size** and a **resolution**. The size measures

the length of a 1D grid, the surface of a 2D grid or the volume of a 3D grid. The resolution designates the density of cells within a grid. For instance, for a 3D grid, the resolution designates the number of cells per 1 m^3 of volume. For the same grid size, a high resolution grid has then more number of cells than a lower one.

2.2.4 Occupancy state of cells

Once the concept of grid is formalized, let us see how it can be used for building a computational model of a physical environment. The idea behind occupancy grids is to capture the spatial structure of a physical environment through a grid. This can be achieved by identifying which cells are occupied by obstacles, and which ones are empty.

A cell is considered as "occupied" if an obstacle is located within the region covered by the cell. The nature of the obstacle is irrelevant. In the context of an autonomous car, an obstacle can be an object, a human, a vegetation, a building and so forth. On a mathematical point of view, an **obstacle** is a bounded region within the physical environment.

Hence, an obstacle A "occupies" a cell c_i if it intersects partially or totally the region covered by the cell: $A \cap c_i \neq \emptyset$. Consequently, a cell c_i is considered as **occupied** if there exists at least one obstacle A that intersects with it:

$$\exists obstacle(A) : A \cap c_i \neq \emptyset \tag{2.23}$$

At the opposite, a cell is **empty** if no obstacle intersects with it:

$$\forall obstacle(A) : A \cap c_i = \emptyset \tag{2.24}$$

Equations (2.23) and (2.24) guaranty that a cell is either occupied or empty. Then, occupiedness and emptiness are actually mutually exclusive and exhaustive. Thus, the occupancy state of a cell can be defined as a random variable as follows.

Definition 2.2.3. *Let $\mathcal{G}$ be a grid and c_i a cell of $\mathcal{G}$. The **occupancy state** of cell c_i is a binary random variable s_i which value is o_i if c_i is occupied and e_i otherwise.*

Since the occupancy state s_i is a discrete random variable, the sum of the probability of its values is equal to 1:

$$P(o_i) + P(e_i) = 1 \tag{2.25}$$

2.2.5 Occupancy Grids

The objective of the occupancy grid framework is to estimate the occupancy state of all cells of a grid. Assume that at a given instant, sensors observe the surrounding obstacles and return measurements. The framework utilizes these measurement to estimate whether a cell is occupied or empty. We suppose that the position of sensors relative to the grid is known. In the literature, this problem is commonly called *occupancy grid mapping at known position* [Thrun 2005].

Definition 2.2.4. *Let $z_1, \ldots, z_K$ denote measurements from K number of sensors and $\mathcal{G}$ a grid. An* ***Occupancy Probability*** *is a function $P_{z_1,\ldots,z_K}$ that maps a cell to the probability that the latter is occupied given sensor measurements:*

$$\begin{array}{rcccl} P_{z_1,\ldots,z_K} & : & \mathcal{G} & \mapsto &]0,1[\\ & & c_i & \mapsto & P(o_i|z_1 \wedge \ldots \wedge z_K) \end{array} \tag{2.26}$$

The term ***inverse sensor model (ISM)*** *designates a particular case of occupancy probability computed from a single measurement z:*

$$\begin{array}{rcccl} P_z & : & \mathcal{G} & \mapsto &]0,1[\\ & & c_i & \mapsto & P(o_i|z) \end{array} \tag{2.27}$$

The probability $P(o_i|z)$ is called ISM to differ it from the sensor model $p(z|d)$. In one hand, an ISM $P(o_i|z)$ estimates the effect of a sensor measurement on the occupancy state a cell. On the other hand, the sensor model $p(z|d)$ estimates the effect of physical location of obstacles on the sensor measurement.

Besides, an occupancy probability is interpreted in the bayesian signification of probabilities. It measures how certain is the statement "$s_i = o_i$" regarding sensor measurements. That means, if the statements "$s_i = o_i$" and "$s_i = e_i$" are equiprobable, a cell c_i can be occupied, but it can be also empty. Then, the occupancy state of c_i is actually **unknown**. Since s_i is a binary random variable, the equiprobability occurs if and only if the occupancy probability is equal to $1/2$. Consequently, if the occupancy probability is less than $1/2$, the cell is likely empty. If it is more than $1/2$, the cell is likely occupied.

Definition 2.2.5. *Let $\mathcal{G}$ be a grid and $z_1, \ldots, z_K$ be the measurements from K number of sensors. An* ***Occupancy Grid (OG)*** *is a function that maps a collection of measurements $z_1, \ldots, z_K$ to the set of the occupancy probabilities of all cells of $\mathcal{G}$:*

$$OG(z_1, \ldots, z_K) = \{P(o_i|z_1 \wedge \ldots \wedge z_K), \forall c_i \in \mathcal{G}\} \tag{2.28}$$

Particularly, an occupancy grid $OG(z)$ built from a single measurement z is called ***mono-sensor occupancy grid****.*

Figure 2.4 shows an example of a scenario with the corresponding occupancy grids. The scenario is on the top-left of the figure. It shows two sensors observing the environment in the front of a car. The first sensor has sensed an obstacle which corresponds to a pedestrian. It returned a measurement z_1. The latter is used for computing the occupancy grid $OG(z_1)$ on the top-right of the figure. The second

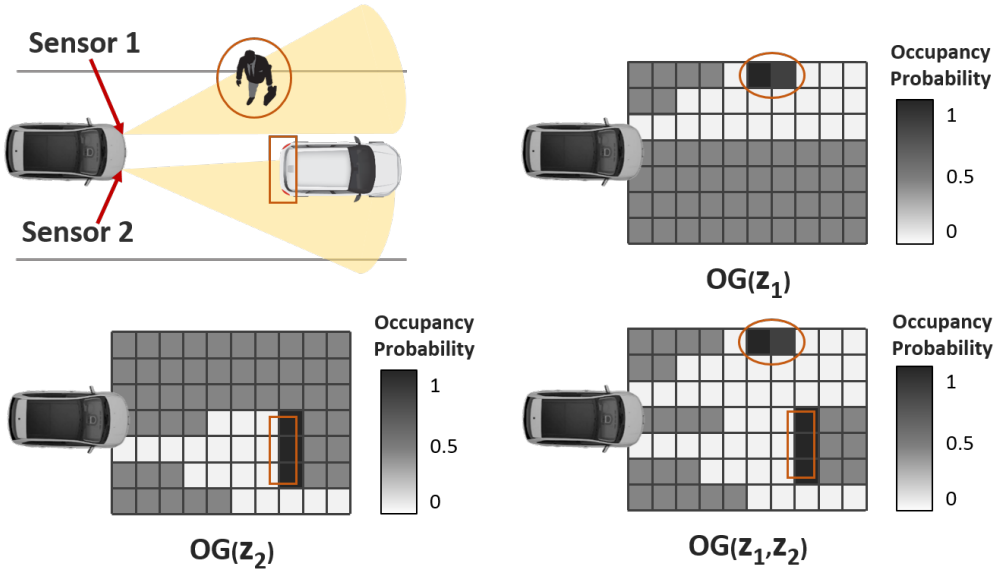


Figure 2.4 – Example of a scenario with the corresponding occupancy grids

sensor has also sensed another obstacle which corresponds to a car in the front. It returned a measurement z_2 that is subsequently used for generating the occupancy grid $OG(z_2)$ on the bottom-left of the figure.

When both sensors are taken into account, they are fused into a unique occupancy grid $OG(z_1, z_2)$ depicted on the bottom-right of the figure. Both the pedestrian and the car in the front are represented on $OG(z_1, z_2)$. Notice that on $OG(z_1), OG(z_2)$ and $OG(z_1, z_2)$ represent the environment as seen by the sensors. Cells corresponding to obstacles sensed by sensors have occupancy probabilities greater than $1/2$. Cells that sensors see as empty have occupancy probabilities less than $1/2$. Finally, the occupancy probability of the cells outside the field-of-view of sensors are equal to $1/2$.

According to Definition 2.2.5 (page 26), "building an occupancy grid" is equivalent to computing the occupancy probabilities of all cells of a grid ([Moravec 1988, Elfes 1989b]). In the followings, let us see the approaches proposed in the literature for building occupancy grids.

2.3 Mono-sensor occupancy grid

The problem of building a mono-sensor occupancy grid is described as follows. A sensor observes a physical world and provides a measurement z. The process of sensing is described by the PDF of the sensor model $p(z|d)$. By knowing z and $p(z|d)$, how to build the occupancy grid $OG(z)$?

This section reviews the different approaches proposed in the literature for tackling the above problem. Approaches differ from 1D occupancy grids to high-

dimensional ones (in 2D or 3D). This section focuses on 1D linear grids and 2D cartesian grids. It also considers only kind of sensors called **single-target sensor** and defined as follows.

Definition 2.3.1. *A **single-target sensor** is a range sensor which field-of-view is composed of a unique line-of-sight. A line-of-sight is modeled by a ray starting from the sensor, emitted towards a given direction and having a maximal range. The sensor outputs a scalar measurement z given the distance d to the nearest obstacle along the line-of-sight. The sensor model is in the form of $p(z|d)$.*

Range sensors that output point clouds can be modeled as a collection of multiple single-target sensors. For instance, each single point provided by LIDARs or RGB-D cameras can be modeled as the output of a unique single-target sensor. For a LIDAR, a single-target sensor is constituted by a single beam. For a depth image from vision-sensors, a pixel is modeled as a single-target sensor.

The output of a sensor device can contain then multiple measurements. For instance, an individual point among the point clouds returned by a LIDAR is considered as an individual measurement. The depth of a pixel within an image from a stereo camera is also an individual measurement.

In the following, let us see how to build a 1D occupancy grid from an individual measurement. After that, let us review the approaches for building 2D mono-sensor occupancy grids based on cartesian grids.

2.3.1 One-dimensional mono-sensor occupancy grid

This section reviews the main approaches proposed in the literature for building a 1D mono-sensor occupancy grid. Such occupancy grid model the physical world along an individual ray of a single-target sensor.

Figure 2.5 illustrates the scenario of the measurement. The sensor observes the world along a ray and returns a measurement z. The value of the later depends on the distance d to the nearest obstacle. The sensor model $p(z|d)$ is assumed to be available. For modeling the world as it is viewed by the sensor along the ray, the latter is subdivided into multiple cells forming a 1D grid. Let N denote the number of cells, c_1 the cell nearest to the sensor and c_N the farthest cell. The symbol d_i designates the distance of cell c_i from the sensor.

According to the definition of occupancy grids (Definition 2.2.5 (page 26)), building a 1D mono-sensor occupancy grid regarding a measurement z is equivalent to computing the ISM $P(o_i|z)$ of all cells c_i, $i = 1, \ldots, N$. The state s_i of cell c_i is a binary random variable. The value of s_i is either empty (e_i) or occupied (o_i). To compute the ISM, applying the Theorem of Bayes described in eq. (2.13) gives:

$$P(o_i|z) = \frac{p(z|o_i)P(o_i)}{p(z|o_i)P(o_i) + p(z|e_i)P(e_i)} \tag{2.29}$$

Hypothesis 2.3.1. *Hypothesis of **non-informative** prior:* $P(o_i) = P(e_i) = 1/2$.

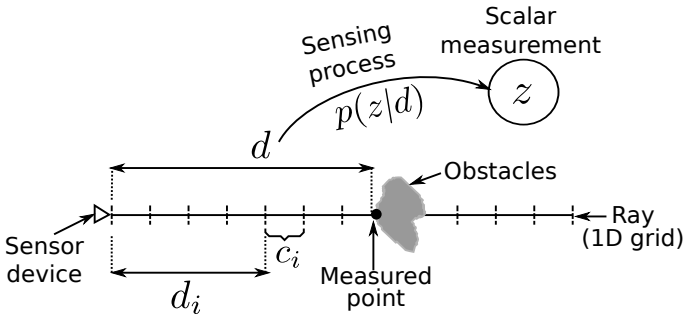


Figure 2.5 – Modeling the world from the sensor's viewpoint along a ray

Based on eq. (2.29), the probabilities $P(o_i)$ and $P(e_i)$, along with the PDFs $p(z|o_i)$ and $p(z|e_i)$ are required for computing the ISM of a cell. In the absence of any a priori information, the hypothesis of non-informative priors (Hypothesis 2.3.1 (page 28)) can be adopted to set the values of both $P(o_i)$ and $P(e_i)$.

The literature propose different techniques for computing the PDFs $p(z|s_i), s_i \in \{o_i, e_i\}$. They can be classified into three groups: the bayesian approach, the analytics approach and the methods based on neural networks. Let us review these approaches one-by-one.

2.3.1.1 Bayesian approach

The Bayesian approach computes the ISM by utilizing the sensor model. It was proposed by Elfes in [Elfes 1989b]. This approach introduces the notion of grid configurations for computing $p(z|s_i), s_i \in \{o_i, e_i\}$.

Definition 2.3.2. *Let $\mathcal{G}$ be a grid with N number of cells. A **grid configuration** g designates a conjunction of the states of all cells of $\mathcal{G}$:*

$$g \triangleq x_1 \wedge \ldots \wedge x_N \qquad \textit{where } x_j \in \{o_j, e_j\} \tag{2.30}$$

By applying the Theorem of Total Probability over all possible grid configurations, $p(z|s_i)$ becomes:

$$p(z|s_i) = \sum_g p(z|g \wedge s_i) P(g|s_i) \tag{2.31}$$

Let us separate the set $\{g\}$ of all possible grid configurations into two disjoint subsets $\{g^{s_i}\}$ and $\{g^{\overline{s_i}}\}$. The state of cell c_i is set to s_i within a grid configuration g^{s_i} and $\overline{s_i}$ within a grid configuration $g^{\overline{s_i}}$. Equation (2.31) gives:

$$p(z|s_i) = \sum_{g^{s_i}} p(z|g^{s_i} \wedge s_i) P(g^{s_i}|s_i) + \sum_{g^{\overline{s_i}}} p(z|g^{\overline{s_i}} \wedge s_i) P(g^{\overline{s_i}}|s_i) \tag{2.32}$$

Since for a grid configuration g^{s_i}, the state of cell c_i is s_i, then $p(z|g^{s_i} \wedge s_i) = p(z|g^{s_i})$ and $P(g^{s_i}|s_i) = P(g^{s_i})$. For a grid configuration $g^{\overline{s_i}}$, the state of cell c_i is

$\overline{s_i}$, therefore $P(g^{\overline{s_i}}|s_i) = 0$. Equation (2.32) gives:

$$p(z|s_i) = \sum_{g^{s_i}} p(z|g^{s_i})P(g^{s_i}) \tag{2.33}$$

For computing $p(z|g^{s_i})$, [Elfes 1989b] proposes to utilize the sensor model $p(z|d)$ where the distance d is replaced by the distance d_h of the first occupied cell within the configuration g^{s_i}:

$$p(z|g^{s_i}) = p(z|d_h) \text{ where } (s_j = e_j \forall j < h) \text{ and } (s_h = o_h) \tag{2.34}$$

Figure 2.6 presents the typical profile of a 1D occupancy grid. The ISMs are plotted as a function of the cell indexes. The sensor measurement z lays in cell c_k. Before c_k, the occupancy probabilities are almost null. Cells before c_k are likely empty according to the sensor. In the vicinity of c_k, occupancy probabilities increase until reaching a value greater than 1/2 on c_k. Cell c_k is likely occupied according to the sensor. After that, occupancy probabilities drop progressively to 1/2. The occupancy states of cells beyond c_k are unknown to the sensor.

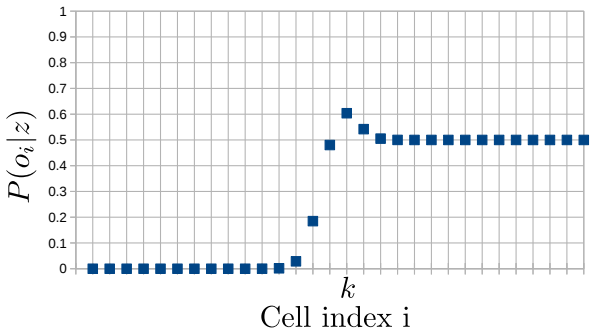


Figure 2.6 – Typical profile of a Bayesian ISM of a single-target sensor

Benefits The approach of [Elfes 1989b] has the following properties. First, the ISM is computed from the sensor model. The sensor model expresses measurement uncertainties given the physical location of the sensed obstacle. By considering all possible grid configurations, the Bayesian approach evaluates all possible location of the obstacle by taking into account the measurement and its uncertainty.

Second, the value of ISMs take into account the way in which the physical world along the ray is subdivided into a linear grid. The grid subdivision is characterized by the distances d_i, $i \in \{1, \ldots, N\}$, of cells. These distances are taken into account in eq. (2.34). Hence, the Bayesian approach establishes a relation between the physical world, the sensor uncertainties, the grid subdivision and the value of the ISMs. Third, the Bayesian approach can be applied for any single-target sensor as long as the sensor model $p(z|d)$ is available.

Limitations The main drawback of the Bayesian approach is its exponential complexity. In fact, the number of grid configurations where the state of a single cell is known is equal to 2^{N-1}. Thus, the number of elements in the sum of eq. (2.33) grows exponentially with the number of cells. This hinders practical implementation of this approach, especially when real-time performance is required.

[Pathak 2007] proposed an extension of the Bayesian approach where an ISM is computed in a linear complexity. To improve complexity, cell states are assumed to be conditionally independent given sensor measurements. Such assumption is however incorrect along a ray. For instance, on fig. 2.6, the occupancy states of cells behind c_k tend to be unknown since the obstacle within c_k hides them from the sensor. Then, the occupancy states of these cells are influenced by the occupancy state of c_k.

2.3.1.2 Analytic approach

The analytic approaches for computing $P(o_i|z)$ have been motivated by the exponential complexity of the Bayesian method in [Elfes 1989b]. The analytic approaches propose to approximate $p(z|s_i), s_i \in \{o_i, e_i\}$, or $P(o_i|z)$ by a continuous function.

Analytic model of $P(z|s_i)$. In order to avoid the enumeration of all possible grid configurations as in the Bayesian approach, [Konolige 1997] and [Yguel 2006] propose to approximate $p(z|s_i)$ with an analytic form:

$$p(z|s_i) \approx p_z^{dist}(d_i) \tag{2.35}$$

where $p_z^{dist}(x)$ is a continuous function defined over the distance x from the sensor. This function is based on a Gaussian distribution in [Konolige 1997] and on power function in [Yguel 2006].

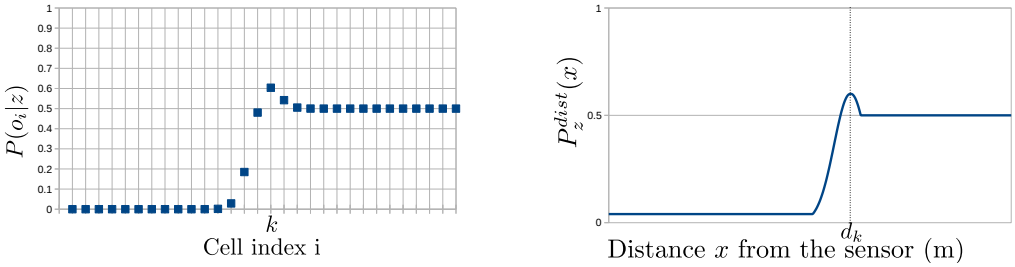
Analytic model of $P(o_i|z)$. Instead of an analytic form of $p(z|s_i)$, other authors in the literature, especially in recent works, propose to model directly the ISM $P(o_i|z)$ by a continuous function as follows:

$$P(o_i|z) \approx P_z^{dist}(d_i) \tag{2.36}$$

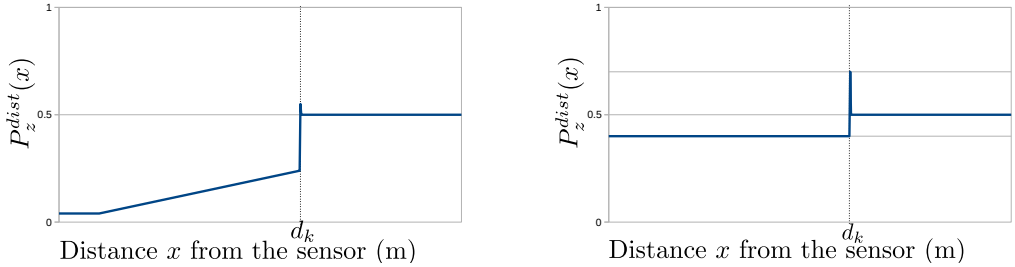
The function $P_z^{dist}(x)$ is a continuous function defined over the distance x from the sensor. Various forms of this function are proposed in the literature, especially for LIDARs and stereo-cameras.

Figure 2.7 shows some examples of the approximation of ISMs by a continuous function. The symbol k denotes the index of the cell c_k where the obstacle is located regarding the sensor measurement. Such cell is located at distance d_k from the sensor. For the purpose of comparison, the original non-approximated ISM computed by the bayesian approach is plotted on fig. 2.7a.

When the sensor model is Gaussian, the ISM is approximated by a continuous function based on a Gaussian distribution. This technique is applied for LIDARs in



(a) Bayesian ISM [Elfes 1989b] (b) Gaussian approximation [Homm 2010]



(c) Linear approximation [Weiss 2007] (d) Three valued-ISM [Wurm 2010]

Figure 2.7 – Examples of approximations of the ISM by continuous functions

[Homm 2010, Einhorn 2011, Adarve 2012, Rakotovao 2015a]. The approximation of the ISM is depicted on fig. 2.7b. Similar technique is applied for a 3D-point of a stereo camera in [Payeur 1998, Gartshore 2002, Nguyen 2012]. Besides, [Weiss 2007] proposes the linear approximative ISM depicted on fig.2.7c. Furthermore, an ISM having only three possible values is adopted in [Wurm 2010, Hornung 2013]. The profile of such ISM is plotted on fig. 2.7d.

Benefits Analytic models enable a fast computation of ISMs. The value the ISM $P(o_i|z)$ is obtained by evaluating continuous functions at distance d_i. The complexity of ISM does not depend anymore on the number of cells. It becomes $O(1)$.

Limitations Analytic approaches have drawbacks. First, occupancy probabilities become continuous functions defined over distances with analytic approaches. They are no more a function defined over a grid as specified in Definition 2.2.4 (page 26). Consequently, analytic approaches ignore the subdivision of the physical world into a grid. The sizes of cells are not taken into account by the ISM. Second, by approximating directly the ISM or the PDF $p(z|s_i)$, the analytic approaches bypass the sensor model $p(z|d)$. By doing so, they lose the relation between the measurement uncertainty, the size of cells and the value of ISMs.

2.3.1.3 Neural networks-based approach

Neural networks can be applied for computing ISMs ([Thrun 1993, Burgard 1999, Thrun 1998, Kortenkamp 1998, Thrun 2001b]). The idea is to train a neural network in a controlled environment where the occupancy state of cells and the measurements are known. Based on these training data sets, the neural network learns to estimate the occupancy state of cells regarding measurements. After the training, the neural network acts as a function approximator of the ISM: it takes as input a measurement and provides an estimation of the occupancy probability. The sensor model can be taken into account during the training.

Benefits The advantage of this approach is that once the neural network is trained, it produces an approximate of the ISM within a computational complexity independent to the number of cells. Once the network is trained, the complexity of computing an ISM becomes $O(1)$.

Limitations As an approximation, the accuracy of the neural network depends on the training process and on the training data sets. A huge volume of training data is required to ensure safety. The training data must include huge number of traffic scenarios to be sure that the neural network can react safely in any traffic situation. However, collecting and processing such a data is a challenging task, even for big tech companies. As a solution, [Thrun 1993] trained a neural network within a simulated environment. Training on real traffic scenarios is though mandatory to ensure safety.

2.3.1.4 Summary

Table 2.1 summarizes the properties of the discussed approaches for computing ISMs. The following properties are highlighted. Is the approach based on sensor model? Does it takes into account the grid subdivision? What is the complexity of the computation of the ISM of a single cell? The symbol N indicates the number of cells in the grid. Finally, is the approach safe to be applied in the domain of automotive?

	Bayesian	**Analytic approach**	**Neural networks**
Sensor model	Yes	No	Yes
Grid subdivision	Yes	No	No
Safe	Yes	No	No
Complexity	$O(2^{N-1})$	$O(1)$	$O(1)$

Table 2.1 – Comparison of the approaches for computing the ISM

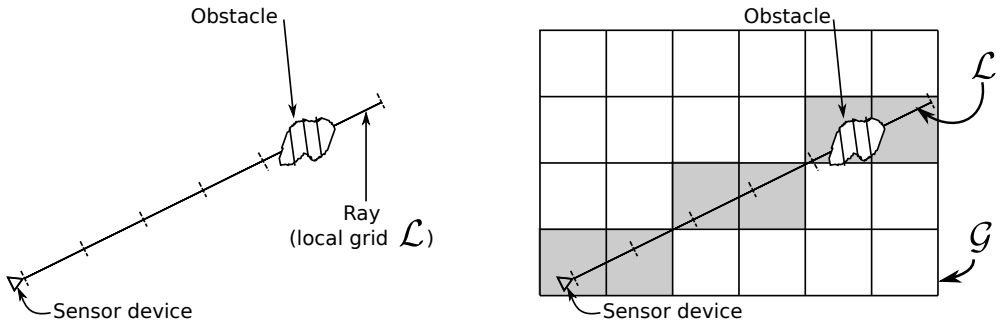
The Bayesian approach computes ISM from sensor model and takes into account the grid subdivision. By doing so, it keeps a mathematical relation between uncer-

tainty of measurement, the size of cells, and the value of the ISM. The above relation makes the Bayesian approach safe. On the opposite, both analytic approach and neural networks lose the above relation. The Bayesian approach suffers however from a computational complexity which explodes exponentially with the number of cells. This motivates the development of another approach based on the Bayesian one but having a lower complexity.

2.3.2 Two-dimensional cartesian mono-sensor occupancy grid

The previous section presented various techniques for building a 1D occupancy grid for a single-target sensor. The occupancy grid covers a linear FOV along a ray. However, two dimensional or three dimensional grids are required for capturing the spatial structure of the physical world surrounding an autonomous car. The present section focuses on 2D cartesian grids.

As in the previous section, let us consider a ray of a single-target sensor. How to build a 2D cartesian occupancy grid given a measurement from that ray? The 2D grid is illustrated on fig. 2.8b. The physical world is subdivided into squared cells forming a 2D cartesian grid denoted by $\mathcal{G}$. A single-target sensor observes the environment along the ray and returns a measurement. Building an occupancy grid defined over the grid $\mathcal{G}$ is equivalent to computing the occupancy probabilities of all squared cells given the measurement from the ray.



(a) First step: local 1D grid $\mathcal{L}$ (b) Second step: global 2D grid $\mathcal{G}$

Figure 2.8 – Two steps for building a mono-sensor 2D cartesian occupancy grid

The cells of the grid $\mathcal{G}$ can be classified into two groups: those which intersect with the ray, and those which do not. In the first group, cells that do not intersect with the ray are not affected by the measurement. That means, the measurement does not provide any information about the occupancy state of these cells ([Moravec 1988]). Consequently, their occupancy states regarding the measurement are unknown. Their occupancy probabilities given the measurement are set to $1/2$.

In the second group, the occupancy probabilities of cells that intersect with the ray are computed in two steps:

- First, the ray is subdivided into a local 1D grid $\mathcal{L}$. The occupancy grid based on the grid $\mathcal{L}$ is computed with respect to the measurement (see fig. 2.8a).
- Second, the 1D occupancy grid is sampled by the global 2D cartesian grid $\mathcal{G}$ in order to get a 2D cartesian occupancy grid (see fig. 2.8b). A 2D cell takes the occupancy probability of the 1D cell that hits it.

The 2D grid is called global since it is not attached to a specific sensor device. This approach composed two steps is widely used in the literature ([Elfes 1989b, Payeur 1997, Fairfield 2007, Homm 2010, Nguyen 2012]). The first step can be achieved by applying one of the previous methods for computing an ISM over 1D grid (see Section 2.3 (page 27)). The second step requires an algorithm called **range mapping algorithm** that perform the following task.

Definition 2.3.3. *The task of **range mapping** consists in finding out all cells of an occupancy grid that intersect with the FOV of a sensor.*

For single-target sensors, the FOV is modeled by rays. Hence the range mapping consists in finding out all 2D cells that are intersected by rays. Fig. 2.8b shows range-mapping along a ray. The cells intersected by the ray are colored in gray.

In the literature, two families of algorithms are mainly used for performing range mapping for a single-target sensor: line rasterization algorithms and traversal algorithms. Let us make a brief comparison between both algorithms.

2.3.2.1 Line rasterization algorithms

Line rasterization consists in rendering a visually acceptable line on a 2D image. The line can be defined by two end points or by a couple of an origin and a direction vector. The most well known algorithm for line rasterization is the Bresenham's algorithm [Bresenham 1965] and its variants [Pitteway 1967, Van Aken 1984, Foley 1990]. These algorithms draw a straight line between two end points localized at two pixels.

Line rasterization algorithms are designed to be executed fast on a computing hardware for optimizing the rendering time of a line on a image. Time constraints are preponderant for intensive graphics applications such a video games and Computer-Aided Design (CAD) software. To reduce the execution time, line rasterization algorithms manipulate mainly integer arithmetic. They work within a discrete 2D frame that allows to identify a pixel by a couple of integers.

An analogous is be made between an image and a 2D occupancy grid where a pixel would correspond to a cell. With this analogous, the range-mapping can be performed by applying line rasterization algorithms. This takes profit of the efficiency of rasterization algorithms for finding out cells traversed by rays of a single-target sensor ([Miller 2005, Fairfield 2007, Nguyen 2012, Souza 2015]). The algorithm may however miss cells that are actually traversed by a ray. In fact, the objective of the rasterization is only to produce an acceptable visual aspect of a line on an image, instead of finding out all pixels traversed by a line.

2.3.2.2 Traversal algorithm

If line rasterization algorithms are specifically designed for working within an image, traversal algorithms are not restricted to images, they can be applied to any 2D grid. The objective of a traversal algorithm is to find out exactly all cells of a grid traversed by a line. Unlike line rasterization algorithms, no traversed cell is missed by a traversal algorithm.

The algorithm widely used for occupancy grid traversal is the Amanatides'algorithm ([Amanatides 1987, Cleary 1988]). This algorithm was applied in [Fournier 2007, Einhorn 2010, Wurm 2010, Shade 2011, Hornung 2013] for performing range mapping on occupancy grids. This algorithm works within a continuous frame of reference. It consequently manipulates real-numbers. Operations on real-numbers are implemented in floating-point in practice ([Amanatides 1987]). This hinders the utilization of the algorithm on computing platforms that do not have floating-point units such as some microcontrollers. On embedded CPUs that support floating-points, integer arithmetic are faster than floating-point operations. Since a line rasterization algorithm requires only integer arithmetic, its execution time is faster that that of a traversal algorithm.

2.3.2.3 Summary

To summarize, the algorithms used for performing range-mapping for occupancy grids are based on either line rasterization algorithms or traversal algorithms. Table 2.2 (page 36) compares both algorithms. Traversal algorithms find out exactly all cells traversed by a ray. They cannot miss traversed cells. However, they process real-numbers. In opposite, line rasterization algorithms process only integers. Nevertheless, they may miss finding out some cells even if the latter are actually traversed by the ray.

	Line rasterization algo.	**Traversal algo.**
Pros	Process only integers	No missed cell
Cons	Can miss cells	Process real-numbers

Table 2.2 – Comparison of the application of the Bresenham's algorithm and the Amanatides' algorithm for range mapping

Which of line rasterization algorithms and traversal algorithms are suited for occupancy grids? Both algorithms have been used in the literature. The absence of miss guaranties that the occupancy probability of every cell traversed by a ray is updated with respect to the measurement corresponding to the ray. Hence, we consider in this thesis traversal algorithms for performing range-mapping. Traversal algorithms do not have misses. They process however real-numbers.

With high number of cells, executing a range mapping algorithm along an individual ray is time consuming. In fact, a ray hits a large number of cells. When

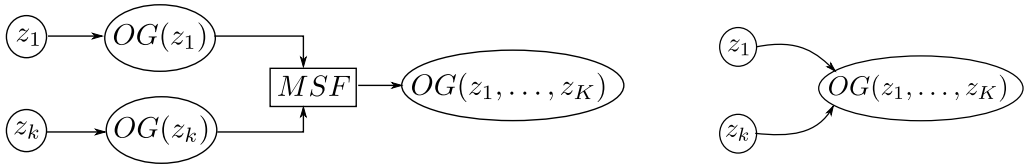
processing range mapping on multiple rays, the number of hit cells increases exponentially. Hence, when both real-time and low-power constraints have to be considered, a little improvement on the execution time of range mapping along an individual ray becomes a considerable energy saving and time saving when processing range mapping for multiple rays.

For embedded hardware, operations on integers are faster and more power efficient than the simulation of operations on real-numbers (like floating-points). Therefore, a traversal algorithm that works with integers can be a solution to improve both execution time and the power efficiency of the HW/SW integration of occupancy grids into embedded platforms.

2.4 Multi-sensor occupancy grid

In this section, let us review the different methods used in the literature for building multi-sensor occupancy grid. The problem statement is the following. Given measurements $z_1, \ldots, z_K$ from K number of sensors ($K > 1$), how to build the occupancy grid $OG(z_1, \ldots, z_K)$? Various approaches are proposed in the literature. They that can be classified into two groups. The first group assumes that sensor measurements are conditionally independent while the second one refutes this hypothesis. Unlike the previous section, these approaches are no more limited to single-target sensors. Unless noticed, they can be applied to any kind of range sensors.

In the first group, the hypothesis about the independence of sensors may be conditional or not. This hypothesis allows to build multi-sensor occupancy grids incrementally as shown on fig. 2.9a. First, mono-sensor occupancy grids $OG(z_k)$ are built independently from each measurement z_k. After that, a process called **Multi-Sensor Fusion (MSF)** combines them into a unique multi-sensor occupancy grid $OG(z_1, \ldots, z_K)$. Hence, the MSF is central for the fusion of multiple independent sensors.



(a) Independent sensor measurements

(b) Forward sensor model

Figure 2.9 – Approaches for building multi-sensor occupancy grids

Besides, the second group is composed by a unique paradigm called the **forward sensor model** ([Thrun 2001a]). Unlike the previous group, this paradigm refutes the independence hypothesis of sensor measurements. As depicted on fig. 2.9b, sensor measurements $z_1, \ldots, z_K$ are taken into account simultaneously (instead of independently) for computing the multi-sensor occupancy grid $OG(z_1, \ldots, z_K)$. Forward sensor model does not compute intermediary mono-sensor occupancy grids.

In the remainder of this section, let us review the main techniques that adopt the independence hypothesis. These techniques include the Bayesian fusion, the Independent Opinion Pool, the Linear Opinion Pool, and the maximum policy. After that, let us see the paradigm of forward sensor model.

2.4.1 Bayesian fusion

The Bayesian fusion is initially introduced by Moravec in [Moravec 1988]. It establishes a probabilistic formulation of the MSF by using the Theorem of Bayes and the hypothesis on sensors independence.

Suppose that our objective is to build an occupancy grid from two sensor measurements z_1 and z_2. According to Definition 2.2.5 (page 26), building the occupancy grid $OG(z_1, z_2)$ is equivalent to the calculate the occupancy probability $P(o_i|z_1 \wedge z_2)$ for any cell c_i. But instead of estimating $P(o_i|z_1 \wedge z_2)$, Moravec proposed to compute the so-called odds ratio:

$$\frac{P(o_i|z_1 \wedge z_2)}{1 - P(o_i|z_1 \wedge z_2)} = \frac{P(o_i|z_1 \wedge z_2)}{P(e_i|z_1 \wedge z_2)} \tag{2.37}$$

By applying the Theorem of Bayes on $P(s_i|z_1 \wedge z_2), s_i \in \{o_i, e_i\}$, we obtain:

$$P(s_i|z_1 \wedge z_2) = \frac{p(z_2|s_i \wedge z_1)P(s_i|z_1)}{p(z_2|z_1)} \tag{2.38}$$

Consequently, eq. (2.37) becomes:

$$\frac{P(o_i|z_1 \wedge z_2)}{1 - P(o_i|z_1 \wedge z_2)} = \frac{P(o_i|z_1)}{P(e_i|z_1)} \frac{p(z_2|o_i \wedge z_1)}{p(z_2|e_i \wedge z_1)} \tag{2.39}$$

Hereafter, the following hypothesis is introduced.

Hypothesis 2.4.1. *Sensor measurements are conditionally independent given the occupancy state of cells.*

This hypothesis means that $p(z_2|s_i \wedge z_1) = p(z_2|s_i), s_i \in \{o_i, e_i\}$. This hypothesis is too strong for some sensors where measurements may be correlated in some way ([Moravec 1988]). However, the hypothesis is valid for measurements from recent sensor devices like LIDARs. By considering the above hypothesis (2.39) gives:

$$\frac{P(o_i|z_1 \wedge z_2)}{1 - P(o_i|z_1 \wedge z_2)} = \frac{P(o_i|z_1)}{P(e_i|z_1)} \frac{p(z_2|o_i)}{p(z_2|e_i)} \tag{2.40}$$

The terms $p(z_2|s_i), s_i \in \{o_i, e_i\}$ can be derived from the Theorem of Bayes:

$$p(z_2|s_i) = \frac{P(s_i|z_2)p(z_2)}{P(s_i)} \tag{2.41}$$

Hence, by inserting eq. (2.41) into eq. (2.40), and by considering that $P(e_i|z_1) = 1 - P(o_i|z_1)$ and $P(e_i) = 1 - P(o_i)$, we obtain:

$$\frac{P(o_i|z_1 \wedge z_2)}{1 - P(o_i|z_1 \wedge z_2)} = \frac{P(o_i|z_1)}{1 - P(o_i|z_1)} \cdot \frac{P(o_i|z_2)}{1 - P(o_i|z_2)} \cdot \frac{1 - P(o_i)}{P(o_i)} \tag{2.42}$$

Finally, we have:

$$P(o_i|z_1 \wedge z_2) = \frac{P(o_i|z_1) \cdot P(o_i|z_2) \cdot [P(o_i) - 1)]}{P(o_i) \cdot [P(o_i|z_1) + P(o_i|z_2) - 1] - P(o_i|z_1) \cdot P(o_i|z_2)} \tag{2.43}$$

Equation (2.43) is the central equation of the Bayesian fusion. It allows to compute $P(o_i|z_1 \wedge z_2)$ from the ISMs $P(o_i|z_1)$ and $P(o_i|z_2)$ generated independently from two different measurements. Thus, the Bayesian fusion allows to build multi-sensor occupancy grids in an incremental way.

2.4.2 Independent Opinion Pool

The formula of **Independent Opinion Pool** combines evidences x and y from two independent sources ([Berger 1985]):

$$F(x, y) = \frac{xy}{xy + (1 - x)(1 - y)} \tag{2.44}$$

In the above equation, the quantities x and y do not designate occupancy probabilities. However, under the assumption of non-informative priors, $P(o_i|z_1 \wedge z_2)$ is mathematically equivalent to $F(P(o_i|z_1), P(o_i|z_2))$:

$$P(o_i|z_1 \wedge z_2) = F(P(o_i|z_1), P(o_i|z_2)) \qquad \text{if } P(o_i) = P(e_i) = 1/2 \tag{2.45}$$

Proof ■ Assume that $P(o_i) = 1/2$ and insert that into eq. (2.43). ■

In other words, the Independent Opinion Pool is equivalent to the Bayesian fusion under the non-informative priors hypothesis. It also allows to compute multi-sensor occupancy grids in an incremental way.

Benefits Two important characteristics of the Independent Opinion Pool formula are noticed in [Elfes 1989b]: the property of **mitigation** and the property of **reinforcement**. When measurements are conflicting, they result into two contradictory opinions about the occupancy state of a cell. The property of mitigation increases the uncertainty about the state of a cell. For instance, if a sensor estimates that a cell is likely occupied while another one says that the same cell is likely empty, their fusion will conclude that the occupancy state of the cell is actually unknown. The property of mitigation makes the occupancy probability tends to 1/2.

On the opposite, the property of reinforcement means that the fusion of similar opinions result into a more certain opinion. This occurs when measurements are non-conflicting. For instance, if two sensors estimate that a cell seems to be occupied, the fusion will say, with a higher certainty, that the cell is actually occupied. Similarly, if both sensors estimate that a cell seems to be empty, the fusion will state, with a higher certainty, that the cell is actually empty.

Limitations In most of the cases, measurements are not conflicting (if it was not the case, the used sensors would not be reliable at all). The property of reinforcement makes occupancy probabilities tend to 0 or to 1 after fusing only few number of measurements. However, in practice, the number of measurements can reach thousands to million. Then, occupancy probabilities reaches practically 0 or 1 when implemented on a hardware.

This leads though to a problem of numerical instability on the implementation viewpoint. When the occupancy probability of a cell reaches 0 or 1, the fusion becomes non-reactive to new measurement. The fusion of a probability x with 1 returns 1. The fusion of x with 0 returns 0. New measurements cannot modify the values of occupancy probabilities anymore once 0 or 1 is reached.

As a solution, the log-odds form of (2.45) are proposed in [Elfes 1989b]:

$$l(o_i|z_1 \wedge z_2) = l(o_i|z_1) + l(o_i|z_2) \text{ where } l(x) = log\frac{P(x)}{1-P(x)} \tag{2.46}$$

The advantage of the above equation is that, unlike probabilities, log-odds are not restricted between 0 and 1. The addition of log-odds is more numerically stable than the addition, multiplication and division in eq. (2.45). The drawback is that recovering occupancy probabilities from log-odds requires additional operations:

$$P(o_i|z_1 \wedge z_2) = 1 - \frac{1}{1 + exp(l(o_i|z_1 \wedge z_2))} \tag{2.47}$$

2.4.3 Linear Opinion Pool

The Linear Opinion Pool is an aggregation method for combining multiple opinions from different sources ([Berger 1985]). Each opinion has a relative weight ω that quantifies the confidence in the corresponding information source. When applied to occupancy probabilities, the Linear Opinion Pool gives ([Elfes 1989b]):

$$P(o_i|z_1 \wedge \ldots \wedge z_K) \triangleq \sum_{k=1}^{K} \omega_k P(o_i|z_k) \qquad \text{where } \sum_{k=1}^{K} \omega_k = 1 \tag{2.48}$$

In eq. (2.48), the fusion of ISMs is equivalent to a weighted average. The weights allow to favor some sensors that may be more precise and more accurate than others. This approach is not based on probabilistic principle. Different methods have been proposed for determining the weights. In [Thrun 1993], the weights are computed by a neural network. In [Adarve 2012], weights are computed as a function of the height of the points where laser beams of a LIDAR hit obstacles.

Benefits and limitations The weighted average enables mitigation in the case of conflicting measurements. It does not reinforce however non-conflicting ones. Hence, Linear Opinion Pool does not feature the property of reinforcement.

2.4.4 Maximum policy

The maximum policy [Payeur 1997, Thrun 2005] fuses occupancy probabilities with the following formula:

$$P(o_i|z_1 \wedge z_2) \triangleq max(P(o_i|z_1), P(o_i|z_2)) \tag{2.49}$$

Benefits Eq. (2.49) has the advantage of being conservative [Yguel 2008] and simple to calculate. Moreover, it have no problem of numerical instability since it does not perform any computations apart from a single comparison.

Limitations The maximum policy favors false positive. Assume that a sensor has estimated that a cell is occupied. The occupancy probability is then greater than $1/2$. Assume now that all the other sensors estimate that the cell is likely empty. These sensors lead to occupancy probabilities less than $1/2$. However, since the multi-sensor fusion returns the highest occupancy probability, the cell will remain likely occupied despite th measurements of the other sensors.

2.4.5 Forward sensor model

In the previous approaches, multi-sensor occupancy grids are built by combining multiple mono-sensor occupancy grids computed independently. This requires the hypothesis that sensor measurements are independent. On the opposite, the paradigm of forward sensor model rejects the hypothesis of sensor independence. This paradigm was introduced in [Thrun 2001a].

A forward sensor model allows to find out the grid configuration that explains at the best the causes of a set of measurements. All measurements are considered simultaneously (instead of independently) for estimating the grid configuration. The occupancy probabilities of cells are computed subsequently.

Given a set of measurements $\mathcal{Z}$, the following function presents the intuitive form of a forward sensor model:

$$h(g) = p(\mathcal{Z}|g) \tag{2.50}$$

The symbol g denotes a grid configuration (see Definition 2.3.2 (page 29)). The higher is $h(g)$, the more likely is the configuration g. Hence, the most likely grid configuration is the one that maximizes $h(g)$. Consequently, finding the most likely grid configuration becomes a problem of optimization that is resolved in [Thrun 2001a] by applying the Expectation Maximization (EM) algorithm ([Dempster 1977]).

Benefits This approach has the following advantage. For sensors with low angular resolution like sonars, the paradigm of forward sensor model produces occupancy grids that are more consistent than those computed by the previous techniques. The consistency means here that the occupancy grid reflects better the spatial structure and the spatial disposition of obstacles in the modeled physical world.

Limitations The paradigm of forward sensor model is adapted for sensors with low angular resolution. It is less applicable for single-target sensors like LIDARs or stereo camera ([Thrun 2001a]). Besides, unlike the previous techniques, this approach is not incremental. Sensor measurements cannot be integrated separately. All measurements should be available before computing occupancy grid. Moreover, the convergence of the EM algorithm can take a long computation time which makes this approach inappropriate for real-time constraints.

2.4.6 Summary

To summarize, Table 2.3 (page 42) resumes the main properties of the approaches for computing multi-sensor occupancy grids. The properties are the followings. Is the fusion incremental? Does it support the property of reinforcement and the property of mitigation? Is the implementation numerically stable? Does it process operations on real-numbers? What are the arithmetic operators required by the fusion approach? These properties influence on the quality of an approach, on its efficiency and on its computing requirements on an implementation viewpoint. Finally, is the fusion safe?

	Bayes./IOP	**LOP**	**Max Policy**	**Forward SM**
Incremental	Yes	Yes	Yes	No
Reinforcement	Yes	No	No	Yes
Mitigation	Yes	Yes	No	Yes
Real-numbers	Yes	Yes	Yes	Yes
Arithmetic operator	$\times, \div, +, -$	$\times, \div, +, -$	$<, >$	$\times, \div, +, -$
Numerically stable	No	Yes	Yes	Yes
Safe	Yes	No	No	No

Table 2.3 – Comparison of approaches for computing multi-sensor occupancy grids

The forward sensor model is not incremental. It requires a long execution time which makes it not adapted for safety applications where real-time constraints are primary. The Bayesian fusion performs sensor fusion based on Bayesian principles. Unlike the other approaches, the Bayesian fusion is incremental. It reinforces non-conflicting measurements and mitigates the conflicting ones. The reinforcement and the mitigation make the Bayesian fusion robust to the convergence or divergence of sensor measurements.

However, the Bayesian fusion suffers from a problem of numerical instability. This makes the Bayesian fusion unsafe. The log-odds form of the Bayesian fusion may be a solution. It requires though a HW/SW integration that is able to simulate operations on real-numbers to perform the fusion. When the fusion is implemented in floating-points, dozens of GFLOPs is required in practice for computing multi-sensor occupancy grids in real-time.

2.5 Data structure for occupancy grids

In the context of autonomous vehicle, the purpose of occupancy grids is to model the physical driving environment by using range sensors. The occupancy grid framework is implemented as a software components of the autonomous driving system. Its role is to build occupancy grids by fusing sensor measurements. Figure 2.10 presents a software overview of an autonomous driving system based on occupancy grids. The system is divided into three parts: occupancy grid building, occupancy grid storage, and occupancy grid exploitation.

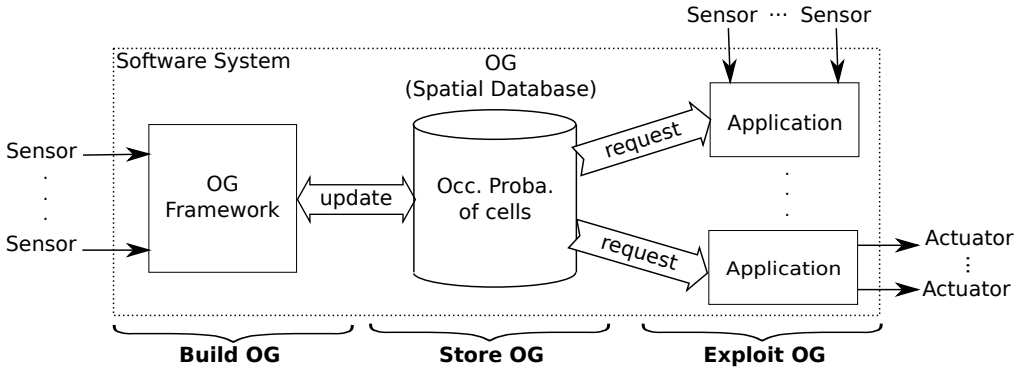


Figure 2.10 – Software overview of an autonomous driving system based on occupancy grids

The occupancy grid storage can be viewed as a database that stores the occupancy probabilities of all cells of the grid. Any application component that performs computations on occupancy probabilities has to interact with the database. Occupancy grid building is performed by a software implementation of the occupancy grid framework. The later takes measurements from range sensors and updates the values of occupancy probabilities stored within the database. To update the occupancy probability of a cell, the old value of the probability is first fetched from the database. Then, it is combined with the ISM computed from new sensor measurements. Finally, the newly computed probability is stored back into the database.

Occupancy grids are exploited by various applications for reasoning based on the environment model and for making driving decisions. Applications can be an advanced perception, an obstacle tracking, a free path searching, a path planning, a collision avoidance, *etc.* Some applications may need additional sensors such as odometers, Inertial Measurement Units (IMUs), Global Positioning System (GPS) or others. For instance, the perception algorithm BOF and its variants ([Coué 2006, Nègre 2014, Rummelhard 2015]) requires IMUs to track the ego motion of a car. Other applications make decisions and send commands to actuators. For instance, the navigation application needs to steer or brake the car according to a planned path and the evolution of the driving situation.

Regardless of the objective of the application, the latter need to request oc-

cupancy probabilities from the database. The simplest request consists in getting the occupancy probability of a single cell. Applications may though perform more complex requests. For instance, navigation requires to find out the first occupied cells within determined directions ([Fairfield 2007]). The database will be requested several times for this purpose. Another complex request consists to fetch the occupancy probabilities of a block of adjacent cells having similar occupancy states. Due to the spatial disposition of obstacles within a driving environment, adjacent cells potentially have similar occupancy states. Free spaces are represented by blocks of likely empty cells while an obstacle can occupy a block of more than one cell ([Mekhnacha 2008]).

Both occupancy grid building and occupancy grid exploitation have to be executed in real-time to ensure safety. Both tasks require interactions with the database. The performance of both tasks depends directly on the efficiency of the database for supporting these interactions. For instance, if the update of the occupancy probability of a single cell takes too long time, a multi-sensor occupancy grid will be hardly built in real-time. If the database is able to answer fast to a request about blocks of cells, this will accelerate considerably the access to occupancy probabilities ([Soucy 2004]).

The efficiency of the database depends on the data structure that actually stores occupancy probabilities. In the literature, two types of data structures are mainly used: arrays and trees. Let us review them in the following paragraphs.

2.5.1 Array-based data structure

An array is an algorithmic data structure for storing a collection of a fixed number of elements in an ordered manner. The location of an element is identified by an integer index as shown on fig. 2.11. Arrays can be considered as the basic data structure for storing occupancy grids ([Elfes 1987, Moravec 1988, Borenstein 1991, Moravec 1996, Coué 2006, Rakotovao 2015a]). They can store 1D, 2D or 3D occupancy grids. The occupancy probability of cell c_i is stored at the element of index i of the array.

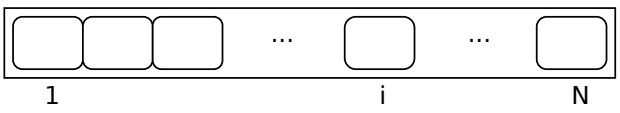


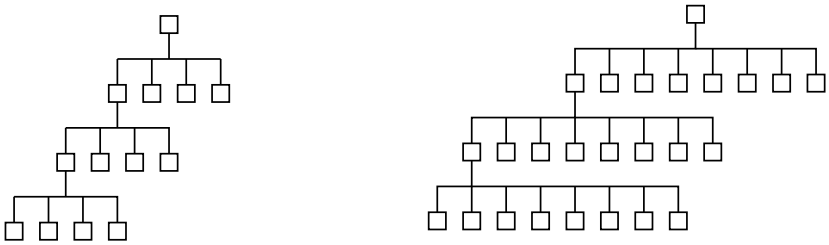
Figure 2.11 – An array with a capacity of N elements

Benefits Arrays enable fast build-time of occupancy grids. The algorithms of range-mapping reviewed in Section 2.3.2 (page 34) are designed and optimized for arrays. Updating the occupancy probability of a cell is straight forward once the index of cell is known.

Limitations Arrays store a fixed number of occupancy probabilities as many as cells. For a large and high resolution grid, the number of cells explodes. This has two consequences. First, working on arrays is time consuming for navigation tasks ([Kambhampati 1986, Soucy 2004]). The latter need to find the location of obstacles and free spaces. Obstacles (resp. free spaces) are modeled by blocks of adjacent occupied (resp. empty) cells. Searching blocks of adjacent empty cells – or blocks of adjacent occupied cells – within an array is however time consuming. Second, arrays are memory consuming for grids with high number of cells. Memory consumption is limiting especially for 3D grids ([Hornung 2013]).

2.5.2 Tree-based data structure

The tree-based data structures mainly used for storing occupancy grids are the 2^d-trees. A 2^d-tree [2] is a data structure composed of nodes organized into parent-children relations. Fgure 2.12 shows two examples of 2^d-trees. The tree is called **quadtree** if d is equal to 2. It is called **octree** when d is equal to 3.



(a) Quadtree (b) Octree

Figure 2.12 – The tree structure of 2^d-trees

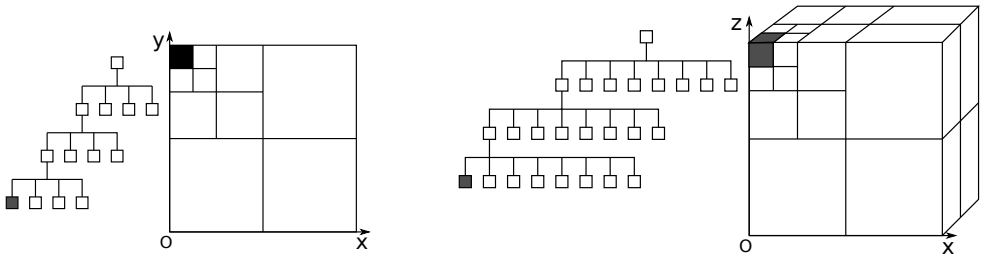
The node at the top of the tree is called **root**. Every node except the root has a unique parent. A node has exactly either zero or 2^d number of children nodes. A node that has no child is called **leaf**. Leaves are situated at the extremities of the tree. The application of octrees for storing 3D occupancy grids was proposed by [Payeur 1997]. Besides, quadtrees can be applied for the storage of 2D occupancy grids ([Kraetzschmar 2004]).

[2] The 2^d-trees are originated from the domain of computer graphics for storing spatial data ([Finkel 1974, Samet 1984, Samet 1988]). The symbol d actually means the dimension of the data to be stored. Quadtrees were initially designed for storing 2D points scattered on a plan ([Finkel 1974]). They can be other 2D geometric patterns such as lines, polygons, circles, *etc* ([Samet 1990]). In the beginning of 80s, octrees have been used for modeling 3D objects ([Meagher 1982]).

2.5.2.1 Split

A grid covers a bounded physical region. When 2^d-trees are applied to occupancy grids, a correspondence exists between a node and a subregion. The root corresponds to the whole region covered by the grid. To determine the region that corresponds to an arbitrary node, the tree is traversed from the top to bottom in order to search for the node.

The tree traversal starts from the root and goes down to the lower levels towards the searched node. At each level, the region corresponding to the visited node is split into 2^d number of sub-regions if the visited node has children. The region is split into four quadrants for a node of a quadtree and into eight octants for an octree. Each sub-region corresponds to a children of the visited node. The split is applied recursively until the search node is visited.



(a) Quadtree (b) Octree

Figure 2.13 – The tree structure of 2^d-trees with the corresponding spatial subdivision

Figure 2.13a shows an example of the application of the recursive subdivision. Its objective is to determine the region that corresponds to the black node of the quadtree on the left of the figure. The corresponding region is colored in black on the right of the figure. The application of the recursive subdivision on an octree is illustrated on fig. 2.13b.

2.5.2.2 Condition of merge

For 2^d-trees, occupancy probabilities of cells are stored on leaves. Since a leaf corresponds to a region, a single leaf can represents all cells included within its region. Nevertheless, the cells must satisfy a precise condition, to be represented by a single leaf. We will refer to the condition as **condition of merge**. Several conditions of merge are proposed in the literature. They will be reviewed later.

When the cells included within the region of a leaf respect the condition of merge, the leaf stores a unique value representative of the occupancy probabilities of the cells. The value becomes then the occupancy probability of the cells.

If the cells do not respect the condition of merge, the leaf is extended downward. It gets new children. The extension is applied recursively to each new children or

grand children until the condition of merge is satisfied.

2.5.2.3 Merge

Consider 2^d sibling leaves (that are, leaves sharing the same parent). If the values stored on sibling leaves are equal, the operation of merge is applied. The sibling leaves are pruned out of the tree. The value is then transfered to their common parent.

The operation of merge shrinks the tree upward. It is applied recursively as long as there exist sibling leaves that stores equal value representative of occupancy probabilities. The operation reduces the number of nodes within the tree whenever some sibling nodes are pruned out.

2.5.2.4 Discussion

Benefits A leaf can represent all the cells included within its corresponding region. This approach makes 2^d-trees more compact than arrays. Since a leaf can represent several cell at once, the number of leaves becomes less than the number of cells. In practice, the number of nodes is even an order less than the number of cells ([Kraetzschmar 2004, Hornung 2013]). Besides, thanks to their compactness, 2^d-trees consumes up to 2.5× less memory than arrays when mapping large environments such as a university campus or malls ([Kraetzschmar 2004, Hornung 2013]).

Thanks to the condition of merge, cells represented by a leaf share about the same occupancy state. For instance, they all are likely occupied or likely empty. This accelerates the exploitation of occupancy grids for applications that make decisions based on the occupancy probabilities of groups of cells. Example of such applications are free space search, path planning, navigation, *etc*([Soucy 2004, Fournier 2007, Fairfield 2007, Wurm 2010, Hornung 2013]).

Limitations The compactness of 2^d-trees is lossy. The lossy compaction means, given the same set of measurements, it is possible that the occupancy probability of a cell is equal to a first value if the data storage was an array; and the occupancy probability is equal to another value different from the first if the data storage was a 2^d-tree.

The lossy compaction is due to the condition of merge. Several conditions of merge have been proposed in the literature. They can be grouped in three: condition based on intervals, condition based on one threshold and condition based on two thresholds.

- The condition based on intervals subdivides the interval $[0, 1]$ into multiple equally-sized sub-intervals ([Kraetzschmar 2004]). Adjacent cells covered by the region of a leaf satisfy the condition if their occupancy probabilities belong to the same sub-interval. If it is the case, the leaf stores a value representative of the sub-interval.

- The condition based on one threshold uses one threshold between 0 and 1. A cell is labelled as *Occupied* if its occupancy probability is greater than the threshold. Otherwise, the cell is labelled as *Empty*. Cells covered by the region of a leaf satisfy the condition is they all have the same label ([Fairfield 2007]). If it is the case, the leaf stores a value representative of the label.

- For the condition based on two thresholds, a cell is labelled as *Occupied* if its occupancy probability is greater than a threshold near 1. On the opposite, a cell is labelled as *Empty* if its occupancy probability is less than a threshold near 0. As in the above condition, cells covered by the region of a leaf satisfy the condition is they all have the same label ([Yguel 2008, Wurm 2010, Einhorn 2011, Hornung 2013, Li 2013]). If it is the case, the leaf stores a value representative of the label.

Thus, a leaf does not store the occupancy probabilities of individual cells even when it represents several adjacent cells at once. The value stored by a leaf replaces the real values of the occupancy probabilities of cells included within the region of the leaf. When this is not equal to the occupancy probabilities of individual cells, the leaf alters the occupancy probabilities. This makes the compaction lossy and impacts negatively on the safety and on the numerical confidence about the computation of occupancy probabilities.

2.5.3 Summary

2^d-trees are more compact than arrays. The compaction accelerates the process of decision making based on occupancy grids. It also reduces the memory consumption especially when dealing with large and high-resolution grids. The compaction is however lossy due to the conditions of merge proposed in the literature. This leaves room for establishing new condition of merge that actually enables a lossless compaction of occupancy grids.

In this chapter, we have seen the basics of probability and the formal definition of occupancy grids and occupancy probability. We have also reviewed the approaches and algorithms for building both mono-sensor and multi-sensor occupancy grids. The data structures for storing occupancy grids have been also reviewed. Let us proceed now over Chapter 3 to introduce the Integer Occupancy Framework and present its impact over the issues discussed previously.

Chapter 3

THE INTEGER OCCUPANCY GRID FRAMEWORK

The emphasis of the research in this thesis is the HW/SW integration of multi-sensor occupancy grids on low-cost and low-power embedded platform. The integration is subject to real-time constraint and must take into account numerical errors, sensor uncertainties and determinism of computations. The above requirements ensure the safety of the HW/SW integration.

The previous chapter reviewed occupancy grids and presented their limitations. The limitations concern the computation of mono-sensor occupancy grids, the computation of multi-sensor ones, and their compaction into 2^d-trees. These limitations constitute challenges for getting a safe HW/SW integration.

This chapter tackles the limitations concerning the computation of multi-sensor occupancy grids by using Bayesian fusion, and the compaction occupancy grids. The Bayesian fusion enables incremental computation of multi-sensor occupancy grids. It also supports reinforcement and mitigation of sensor measurements. It suffers though from a numerical instability and also requires to process Billions of operations on real-numbers to meet the real-time constraint. Besides, the compaction of occupancy grids with 2^d-trees is beneficial for high-level applications that exploit occupancy grids. The compaction proposed in the literature is however lossy.

To tackle these limitations, this chapter introduces a new paradigm called **Integer Occupancy Grids**. This paradigm pairs the values of occupancy probabilities with integers. This property enables to process the Bayesian fusion through simple arithmetic addition of integers. Integer arithmetic have significant advantages in term of SW/HW integration: their execution on processors is fast, they consume less power, they are not erroneous and they are supported by most of the modern computing platforms. Besides, the paradigm of Integer Occupancy Grids enables a lossless compaction of occupancy grids.

This chapter begins by presenting formally the properties of the Bayesian fusion. After that, sections 3.2 to 3.4 will introduce the mathematical foundation of the

Integer Occupancy Grid paradigm. The definition of Integer Occupancy Grids and the fusion based on the paradigm will be presented in Section 3.4. Then, Section 3.5 will study the compaction of Integer Occupancy Grids.

3.1 Properties of the Bayesian Fusion

The Bayesian fusion performs multi-sensor fusion by combining occupancy probabilities computed from different measurements into a unique probability. It assumes that sensor measurements are independent (see Section 2.4.1 (page 38)) Under the non-informative prior, the Bayesian fusion becomes equivalent to the formula of Independent Opinion Pool (see Section 2.4.2 (page 39)) When no measurement is available, the non-informative prior is **safe** since it stipulates that the occupancy state of a cell is unknown, neither occupied nor empty. The Bayesian fusion is then expressed by the following proposition.

Proposition 3.1.1. *Let z_1, z_2 be two sensor measurements. The occupancy probability of a cell c_i given measurements z_1 and z_2 is:*

$$P(o_i|z_1 \wedge z_2) = F(P(o_i|z_1), P(o_i|z_2)) \tag{3.1}$$

*where F designates the **fusion function** defined as follows:*

$$F(p,q) = \frac{pq}{(1-p)(1-q)+pq} \tag{3.2}$$

To simplify the notation, the fusion operator is introduced in the subsequent definition.

Definition 3.1.1. *The **fusion operator** designates the operator $\odot$ that combines two occupancy probabilities p and q as follows:*

$$\begin{array}{rccl} \odot & : &]0,1[\times]0,1[& \mapsto &]0,1[\\ & & (p,q) & \mapsto & p \odot q = F(p,q) \end{array} \tag{3.3}$$

The Bayesian fusion enables an incremental approach for computing multi-sensor occupancy grids. It also features reinforcement and mitigation. The next paragraphs will present these properties in a formal way.

3.1.1 Incremental fusion

The Bayesian fusion is incremental. Put mathematically, the incremental property allows to compute a multi-sensor occupancy probability through multiple combinations of mono-sensor occupancy probabilities. Proposition 3.1.1 (page 50) shows that the occupancy probability given two sensor measurements is computed through a combination of the occupancy probabilities computed independently from both

measurements. This equation can be generalized to any number of measurements as follows.

Property 3.1.1. *Let $z_1, \ldots, z_k$ denote sensor measurements. Then*

$$P(c_i|z_1 \wedge \ldots \wedge z_k) = P(c_i|z_1 \wedge \ldots \wedge z_{k-1}) \odot P(c_i|z_k) \tag{3.4}$$

Proof ■ The above property can be proved by replacing z_1 with $z_1 \wedge \ldots \wedge z_{k-1}$ and z_2 by z_k in eq. (3.1). ■

The Inverse Sensor Model (ISM) $P(c_i|z_k)$ appears in the above property. The recursive application of the incremental property from z_1 to z_k gives:

$$P(c_i|z_1 \wedge \ldots \wedge z_k) = P(c_i|z_1) \odot \ldots \odot P(c_i|z_k) \tag{3.5}$$

The incremental property shows how a multi-sensor occupancy probability is derived from the fusion of different ISMs given multiple measurements.

Applied at the level of a grid, the incremental property enables the fusion of two occupancy grids defined over the same grid as follows:

$$OG(z_1, \ldots, z_k) = OG(z_1) \odot^* \ldots \odot^* OG(z_k) \tag{3.6}$$

The operator $\odot^*$ applies eq. (3.4) at each cell of the grid. Both eq. (3.4) and eq. (3.6) express the Bayesian method for performing multi-sensor fusion based on occupancy grids. Both equations require the independence hypothesis between measurements and the non-informative prior hypothesis.

3.1.2 Reinforcement

Consider two measurements z_1 and z_2 and a cell c_i. Assume that both measurements reflects the same opinion about the occupancy state of the cell. Assume for instance that the cell considered to be occupied for both measurements. In term of occupancy probability, the above assumption means that both $P(o_i|z_1)$ and $P(o_i|z_2)$ are greater than $1/2$. How about the result of their fusion $P(o_i|z_1) \odot P(o_i|z_2)$?

Assume that the fusion results into an occupancy probability less than $1/2$. Then, the fusion would estimate that the cell is likely empty. That conclusion is contradictory to the measurements. Then, the fusion has to result into an occupancy probability greater than $1/2$. Even better, the fusion produces an occupancy probability that is greater than both $P(o_i|z_1)$ and $P(o_i|z_2)$. This is the property of reinforcement.

The property of reinforcement occurs when both measurements estimate that the same cell is empty. In general, the fusion reinforces the estimation of the occupancy state of a cell given **non-conflicting** measurements. The property of reinforcement is summarized in the following property.

Property 3.1.2. *Let p and q be two occupancy probabilities of the same cell. Then*

$$\text{if } p, q > \tfrac{1}{2}, \text{ then } p \odot q > Max(p, q) \tag{3.7}$$

$$\text{if } p, q < \tfrac{1}{2}, \text{ then } p \odot q < Min(p, q) \tag{3.8}$$

The functions Min and Max returns respectively the minimum and the maximum between the input arguments. Equation (3.7) is illustrated by fig. 3.1a. When both probabilities p and q estimate that a cell is occupied, the fusion increases $p \odot q$ towards 1. Figure 3.1b depicts eq. (3.8). When both probabilities p and q estimate that a cell is now likely empty, the fusion reinforces the both estimations and decreases $p \odot q$ towards 0.

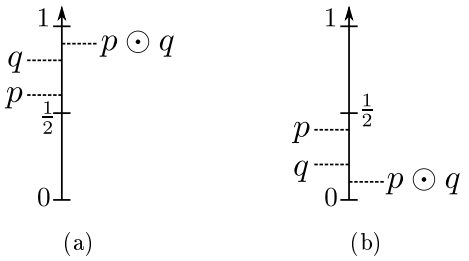


Figure 3.1 – The property of reinforcement

3.1.3 Mitigation

Consider two measurements z_1 and z_2 and a cell c_i. Assume now that the first measurements estimates that an obstacle may occupy the cell. In opposite, the second estimates that the cell is likely empty. Hence, both measurements are **conflicting**. In term of occupancy probabilities, the above assumptions mean $P(o_i|z_1)$ is greater than $1/2$ while $P(o_i|z_2)$ is lesser. Both probabilities are either side of $1/2$. How about the result of their fusion $P(o_i|z_1) \odot P(o_i|z_2)$?

If the result of the fusion was greater than $P(o_i|z_1)$, that would mean the fusion has reinforced the opinion of the first sensor and has not taken into account the opinion of the second sensor. Similarly, if the fused probability was less than $P(o_i|z_2)$, the opinion of the first sensor would not be taken into account. In conclusion, $P(o_i|z_1) \odot P(o_i|z_2)$ should reside between $P(o_i|z_1)$ and $P(o_i|z_2)$. That means, the result of the fusion becomes closer to $1/2$.

When measurements are conflicting, the fusion makes the occupancy state of the cell more uncertain: neither likely occupied nor likely empty. The occupancy probability tends to $1/2$. This is the property of mitigation, summarized by the following property.

Property 3.1.3. *Let p and q be two occupancy probabilities of the same cell. Then*

$$\text{if } p < \tfrac{1}{2}, q > \tfrac{1}{2}, |q - \tfrac{1}{2}| < |p - \tfrac{1}{2}|, \text{ then } p < p \odot q < \tfrac{1}{2} \tag{3.9}$$

$$\text{if } p < \tfrac{1}{2}, q > \tfrac{1}{2}, |q - \tfrac{1}{2}| > |p - \tfrac{1}{2}|, \text{ then } q > p \odot q > \tfrac{1}{2} \tag{3.10}$$

Figure 3.2 illustrates the above property. Consider a cell and two measurements. The first measurement estimates that the cell is likely empty and produces the occupancy probability $p < 1/2$. For the second measurement, the cell is likely occupied

with an occupancy probability $q > 1/2$. Both measurements are conflicting. Figure 3.2a shows that if q is closer to $1/2$ than p, then $p \odot q$ will be between p and $1/2$. In opposite, if it is now p which is closer to $1/2$ than q, then $p \odot q$ will be between $1/2$ and q. In both cases, $p \odot q$ approaches $1/2$. The occupancy state of the cell becomes more and more unknown after the fusion of conflicting measurements.

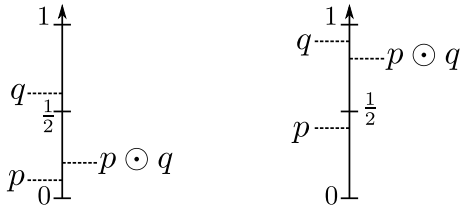


Figure 3.2 – The property of mitigation

3.1.4 Group of probability under the fusion operator

In addition to the incremental fusion, the property of reinforcement and the support of mitigation, the fusion operator also features the following properties.

Property 3.1.4. *For all $p, q, r \in]0,1[$, the fusion operator features the following properties:*

$$p \odot q \in]0,1[\quad \textit{(Closure)} \tag{3.11a}$$
$$p \odot (q \odot r) = (p \odot q) \odot r \quad \textit{(Associativity)} \tag{3.11b}$$
$$p \odot \tfrac{1}{2} = \tfrac{1}{2} \odot p = p \quad \textit{(Identity element)} \tag{3.11c}$$
$$p \odot (1-p) = (1-p) \odot p = \tfrac{1}{2} \quad \textit{(Inverse element)} \tag{3.11d}$$
$$p \odot q = p \odot q \quad \textit{(Commutativity)} \tag{3.11e}$$

Proof ■ Consider $p, q, r \in]0,1[$. According to the definition of the operator $\odot$ (Definition 3.1.1 (page 50)):

$$p \odot q = F(p,q) \in]0,1[\quad \text{(Closure)}$$
$$p \odot q = F(p,q) = F(q,p) = q \odot p \quad \text{(Commutativity)}$$
$$p \odot (q \odot r) = F(p, F(q,r)) = F(F(p,q), r)) = (p \odot q) \odot r \quad \text{(Associativity)}$$
$$p \odot 1/2 = 1/2 \odot p = F(p, 1/2) = p \quad \text{(Identity element)}$$
$$p \odot (1-p) = (1-p) \odot p = F(p, 1-p) = 1/2 \quad \text{(Inverse element)}$$

■

Equation (3.11a) to (3.11d) satisfy the four axioms that allow to qualify the interval $]0,1[$, together with the fusion operator $\odot$, as a **group**. The closure ensures

that the fusion of two occupancy probabilities remains between 0 and 1. The fusion operator is associative. Its identity element is 1/2. The inverse element of an occupancy probability p is $1 - p$.

Equation (3.11e) shows that the fusion operator is commutative. Hence, the algebraic structure $(]0,1[, \odot)$ is an **abelian group**. The commutativity ensures that the order in which occupancy probabilities are fused does not matter. When multiple measurements are available, the order of the integration of measurements into an occupancy grid does not change the value of the final occupancy probabilities.

Remark. The fusion function combines probabilities within $]0,1[$. Zero and one are deliberately excluded since an occupancy probability equal to 0 would mean that the cell is certainly empty while 1 means certainly occupied. However, occupancy probabilities never reach 0 or 1 since sensors are not perfect and measurements are uncertain. Moreover, an occupancy probability equal to 0 or 1 introduces a problem of numerical stability. In fact, we have:

$$\forall p \in]0,1[: \quad 1 \odot p = 1 \quad \text{and} \quad 0 \odot p = 0 \tag{3.12}$$

The above equation means, once it is certain that a cell is occupied, its occupancy probability remains at 1 even if incoming measurements estimates that the cell is likely empty. Similarly, once a cell is certainly empty, it occupancy probability cannot be updated anymore and stays at 0.

Equation (3.12) leads into a problem of numerical instability for a HW/SW integration. After fusing few number of measurements that estimate that a cell is empty, the occupancy probability becomes rapidly equal to 0 on the computing platform, even if obtaining 0 is theoretically impossible (eq. (3.11a)). The same problem occurs during the fusion of few number of measurements that estimate that the cell is occupied. The occupancy probability reaches 1 and the estimation of the occupancy state of the cell cannot be updated anymore despite incoming measurements.

3.2 Set of probabilities

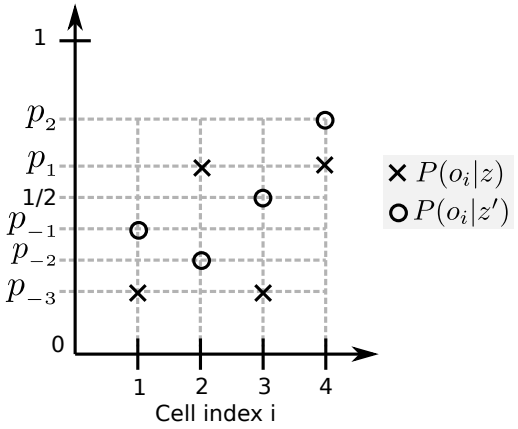
This section introduces the concept of set of probabilities. To gain an intuitive understanding of this concept and its usage, let us begin with an introductory example.

3.2.1 Introductory example

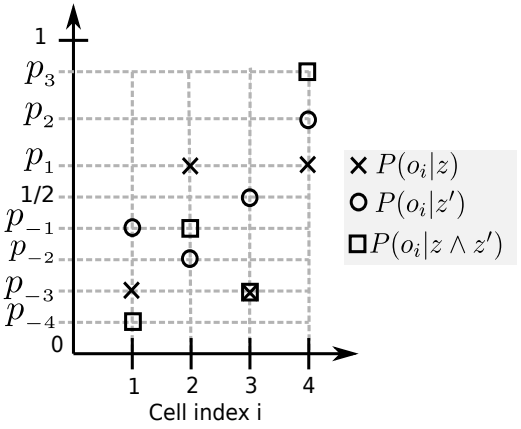
Consider a grid having four number of cells. The spatial dimension of the grid is irrelevant. Figure 3.3a shows the occupancy probabilities of cells of a grid given two measurements z and z'. On the figure, both $P(o_1|z)$ and $P(o_3|z)$ are equal to a number p_{-3}. Both $P(o_2|z)$ and $P(o_4|z)$ are equal to p_1.

Let us denote by $S(z)$ the set of the values taken by the occupancy probabilities of all cells of the grid. The set $S(z)$ has only two elements denoted by p_{-3} and p_1

on fig. 3.3a. Consider another measurement z'. The figure shows that $S(z')$ has four elements $p_{-2}, p_{-1}, 1/2$ and p_2. The values of the occupancy probabilities of all cells given measurement z' are different to one another.



(a)



(b)

Figure 3.3 – Discrete nature of occupancy probabilities given sensor measurements

By proceeding to the cell-by-cell fusion of the measurements z and z', the fusion returns new values of probability that lie between 0 and 1. Let us denote by $S(z, z')$ the set of the values of the occupancy probabilities given both z and z'. The result of the fusion is depicted on fig. 3.3b. The set $S(z, z')$ has four elements p_{-4}, p_{-2}, p_{-1} and p_3.

Hence, when measurements z and z' are taken into account independently or jointly, the value of the occupancy probability of any cell of the grid is included within $S(z) \cup S(z') \cup S(z, z')$. Let us denote by S the reunion of the three sets and let us call it **set of probabilities**. The elements of S are $\{p_{-4}, p_{-3}, p_{-2}, p_{-1}, 1/2, p_1, p_2, p_3\}$. Notice that the elements of $S(z, z')$ were obtained

by fusing the elements of $S(z)$ with the elements of $S(z')$ through the fusion operator $\odot$. That means the result of the fusion of two elements of the set of probabilities S is also included within S.

To understand deeper the relation between the fusion operator and the set of probabilities, let us consider again fig. 3.3b. Fusing p_{-1} with p_{-3} gives p_{-4} on cell c_1. On cell c_3, the fusion of p_{-3} with $1/2$ returns p_{-3} (in fact, $1/2$ is the identity element of the fusion operator). By assuming that p_0 is equal to $1/2$, the fusion of p_0 and p_{-3} gives p_{-3}. By considering these results, one can notice that $p_{-1} \odot p{-}3 = p_{-1-3}$ and $p_0 \odot p_{-3} = p_{-3}$. In other words, fusion of two elements of S can be computed by the addition of the indexes of the elements. That means, the fusion is now computed through an addition of integers.

3.2.2 Definition of set of probabilities

After the above introductory example, let us now give a formal definition of set of probabilities [1].

Definition 3.2.1. *A **set of probabilities** S is a set of real-numbers such that:*

$$S = \{p_n \in]0,1[, \forall n \in \mathbb{Z}\} \quad \textit{(Inclusion into }]0,1[\textit{)} \tag{3.13a}$$

$$\forall m, n \in \mathbb{Z} : p_m \neq p_n \Leftrightarrow m \neq n \quad \textit{(Countability)} \tag{3.13b}$$

$$\forall p_m, p_n \in S : p_m \odot p_n \in S \quad \textit{(Closure)} \tag{3.13c}$$

A set of probability is then a set of real-numbers between 0 and 1 such that the set is countable and is closed with respect to the fusion operator. Since an element of the set is between 0 and 1, it can be considered as a possible value of a probability. The formal definition of a probability was given in Section 2.1.2 (page 16).

Definition 3.2.2. *Let S be a set of probabilities, and p_n an element of S. The integer n is called **index** of the probability p_n.*

The countability of a set of probabilities maintains a bijection between S and $\mathbb{Z}$. Each element of S has a unique integer index. The closure stipulates that the fusion of two elements of S returns a probability that also belongs to S.

3.2.3 Index fusion operator

To formalize the computation of the fusion through integer arithmetic, the concept of index fusion operator is introduced as follows.

[1] The concept of set of probabilities has been published in [Rakotovao 2016a].

Definition 3.2.3. *Let S be a set of probabilities and p_m and p_n two elements of S. An **index fusion operator** designates the operator $\oplus$ that combines two integer indexes m, n such that:*

$$\forall m, n \in \mathbb{Z} : (m \oplus n) \in \mathbb{Z} \tag{3.14}$$

$$\forall m, n \in \mathbb{Z} : p_m \odot p_n = p_{m \oplus n} \tag{3.15}$$

In the previous introductory example, the index fusion operator is equivalent to an addition of indexes. Since $m \oplus n$ is an integer, $p_{m \oplus n}$ designates a unique element of the set S. The operator $\odot$ is then mirrored by the operator $\oplus$:

$$p_m \odot p_n = p_r \quad \Leftrightarrow \quad m \oplus n = r \tag{3.16}$$

While the operator $\odot$ combines probabilities, the operator $\oplus$ combines integer indexes. Let us call **probabilistic fusion** the fusion of probabilities with the operator $\odot$ and **index fusion** the fusion of indexes with the operator $\oplus$. When the elements of the set of probabilities are known, both fusions are equivalent. Performing the one is equivalent to calculating the other.

In addition, the operator $\oplus$ is associative and commutative like the operator $\odot$.

Property 3.2.1. *Let S be a set of probability with an index fusion operator $\oplus$. Then the index fusion operator is associative and commutative. $\forall m, n, r \in \mathbb{Z}$:*

$$\textit{(Associativity)} \qquad m \oplus n \oplus r = m \oplus (n \oplus r) = (m \oplus n) \oplus r \tag{3.17}$$

$$\textit{(Commutativity)} \qquad m \oplus n = n \oplus m \tag{3.18}$$

Proof ■ The associativity of $\odot$ gives:

$$\begin{aligned} m \oplus n \oplus r \Leftrightarrow & p_m \odot p_n \odot p_r \\ & = p_m \odot (p_n \odot p_r) \\ & = p_m \odot (p_{n \oplus r}) \\ & \Leftrightarrow m \oplus (n \oplus r) \end{aligned}$$

Then $m \oplus n \oplus r = m \oplus (n \oplus r)$. The same principle can prove that $m \oplus n \oplus r = (m \oplus n) \oplus r$.

The commutativity of $\odot$ gives:

$$\begin{aligned} m \oplus n \Leftrightarrow & p_m \odot p_n \\ & = p_n \odot p_m \\ & \Leftrightarrow n \oplus m \end{aligned}$$

Then $m \oplus n = n \oplus m$. ■

3.2.4 Existence of set of probabilities

Following the above definitions, the question arises: do sets of probabilities exist? If yes, is there a set of probability that has an index fusion operator? To answer to both questions, this section introduces three examples of sets of probabilities.

Proposition 3.2.1. *Set of probabilities exist.*

Proof ■ To prove the above proposition, let us present a trivial set of probabilities. Consider the singleton $S = \{p_0 = 1/2\}$. This sets is countable and its element is between 0 and 1. Besides, $p_0 \odot p_0$ is equal to $1/2$. The singleton is closed with respect to $\odot$. The set S satisfies the three assertions in Definition 3.2.1 (page 56). Therefore, S constitutes a set of probabilities. ■

Proposition 3.2.2. *There exist non-trivial set of probabilities.*

Proof ■ Let us give two examples of non-trivial set of probabilities.

Example 1: Since a set of probabilities is countable, a way to define the set is to use sequences. Let $a \geq 1$ be a positive and non-null integer. Consider the sequence $(p_n)_{n\in\mathbb{N}}$ such that:

$$p_n = \frac{a \cdot n + 1}{a \cdot n + 2}, n \in \mathbb{N} \tag{3.19}$$

Let us verify whether the set $S = \{p_n\}$ constitutes a set of probability.

- Inclusion into $]0, 1[$: since $a \geq 1$, then $a \cdot n + 1 < a \cdot n + 2$. Therefore, $p_n < 1 \forall n \in \mathbb{N}$. Besides, $\forall n \in \mathbb{N}, p_n \geq 1/2$ (see Property A.1.1 (page 119)). We subsequently get $\forall n \in \mathbb{N}, 1/2 \leq p_n < 1$.
- Countability: eq. (3.19) guaranties that $\forall m \neq n$, we get $p_n \neq p_m$.
- Closure: consider $m, n \in \mathbb{N}$. The application of the operator $\odot$ gives:

$$p_n \odot p_m = \frac{p_m p_n}{p_m p_n + (1 - p_m)(1 - p_n)} \tag{3.20}$$

$$= \frac{a \cdot (m + n + a \cdot m \cdot n) + 1}{a \cdot (m + n + a \cdot m \cdot n) + 2} \tag{3.21}$$

$$= p_{m+n+a \cdot m \cdot n} \tag{3.22}$$

The three conditions are verified, therefore S is actually a set of probabilities [2]. The set S has an index fusion operator $\oplus$ where:

$$m \oplus n = m + n + a \cdot m \cdot n \tag{3.23}$$

[2] This set of probabilities has been published in [Rakotovao 2016a].

Example 2: Let $a \geq 1$ be a positive and non-null integer and let $\mathbb{Z}^-$ designate the set of negative or null integers. Consider the sequence $(p_n)_{n\in\mathbb{Z}^-}$ such that:

$$p_n = \frac{1}{2 - a \cdot n}, n \in \mathbb{Z}^- \tag{3.24}$$

Let us prove that the set $S = \{p_n\}$ is a set of probabilities.

- Inclusion into $]0, 1[$: since $a \geq 1$ and $n \leq 0$, then $2 - a \cdot n \geq 2$. Therefore, $0 < p_n \leq 1/2$.
- Countability: eq. (3.24) guaranties that $\forall m \neq n$, we get $p_n \neq p_m$.
- Closure: consider $m, n \in \mathbb{N}$. Applying the operator $\odot$ gives:

$$p_n \odot p_m = \frac{p_m p_n}{p_m p_n + (1 - p_m)(1 - p_n)} \tag{3.25}$$

$$= \frac{1}{2 - a \cdot (m + n - a \cdot m \cdot n)} \tag{3.26}$$

$$= p_{m+n-a\cdot m\cdot n} \tag{3.27}$$

The set S verifies the three assertions, therefore it constitutes a set of probabilities[3]. In addition, it has an index fusion operator where:

$$m \odot n = m + n - a \cdot m \cdot n \tag{3.28}$$

■

3.3 The recursive set of probabilities

The previous subsection gave three examples of set of probabilities. The first set of probabilities is a singleton. The second set of probabilities is defined by a sequence of real-numbers that lie between $1/2$ and 1. The third set is also defined by a sequence of real-numbers, but its elements lie between 0 and $1/2$.

The set of probabilities that is able to capture the values of the occupancy probabilities within an occupancy grids must have elements both within $]0, 1/2]$ and within $[1/2, 1[$. In fact, the value of the occupancy probability of a likely empty cell lies within $]0, 1/2]$. For a likely occupied cell, the occupancy probability lies within $[1/2, 1[$.

In this section, let us design a set of probabilities that have elements within both $]0, 1/2]$ and $[1/2, 1[$. To ensure that such a set is countable and is included within $]0, 1[$, the set can also be defined through sequences as the previous examples. The difficulty resides in guarantying that the set is closed with respect to the fusion operator $\odot$.

[3] This set of probabilities has been published in [Rakotovao 2016a].

3.3.1 Definition of the recursive set of probabilities

To ensure the closure with respect to $\odot$, the following theorem presents a new set of probabilities where the fusion operator expressly intervenes in the formulation of the elements of the set.

Theorem 3. Recursive set.

Let ε be a real-number such that $\varepsilon \in]0, 1/2[$. Let $(a_n)_{n\in\mathbb{N}}$ and $(b_n)_{n\in\mathbb{N}}$ be infinite sequences of numbers defined as follows:

$$a_n = \begin{cases} 1/2 & \text{if } n = 0 \\ 1/2 + \varepsilon & \text{if } n = 1 \\ a_{n-1} \odot a_1 & \text{otherwise} \end{cases}$$

$$b_n = \begin{cases} 1/2 & \text{if } n = 0 \\ 1/2 - \varepsilon & \text{if } n = 1 \\ b_{n-1} \odot b_1 & \text{otherwise} \end{cases}$$

Consider the set $S_\varepsilon = \{p_n, n \in \mathbb{Z}\}$ such that:

$$p_n = \begin{cases} a_n & \text{if } n \geq 0 \\ b_{-n} & \text{otherwise,} \end{cases}$$

The set S_ε – called **recursive set** – constitutes a set of probabilities equipped such that:

$$\forall m, n \in \mathbb{Z} : p_m \odot p_n = p_{m+n} \tag{3.29}$$

Proof ■ For the sake of clarity, the proof is detailed in Section A.3 (page 120). ■

The set of probabilities S_ε is called recursive set since its elements are defined by recursion. The recursion is involved in the definition of the element p_n. It involves explicitly the fusion operator $\odot$. Notice that the set S_ε is parametrized by an ε which can be any positive real-number less than $1/2$.

3.3.2 Index fusion operator

The recursive set of probabilities features a lightweight integer fusion operator. Equation (C.7) in the above theorem shows that the fusion of two elements p_n and p_m gives p_{n+m}. Therefore, the recursive set is equipped by the following the index fusion operator.

Corollary 3.3.1. *The recursive set of probabilities has an index fusion operator $\oplus$ such that:*

$$\forall m, n \in \mathbb{Z} : m \oplus n = m + n$$

Proof ■ Theorem 3 (page 60) gives:

$$p_m \odot p_n = p_{m+n}$$

However, Definition 3.2.3 (page 57) defines the index fusion operator as follows:

$$p_m \odot p_n = p_{m \oplus n}$$

Therefore:

$$m \oplus n = m + n$$

■

The Fusion operator $\odot$ is computed through the Bayesian fusion function F. The later have to perform at the same time addition, subtraction, multiplication and division of real-numbers in order to fuse two probabilities (Proposition 3.1.1 (page 50)). The recursive set of probabilities enables to compute the fusion through an addition of integers.

3.3.3 Properties of the recursive set of probabilities

The recursive set of probabilities features the following properties.

3.3.3.1 Repartition of elements over the interval $]0, 1[$

The location of an element with respect to $1/2$ depends on the sign of its index. As illustrated on fig. 3.4, the elements of the recursive set that have negative index are less than $1/2$. Those with positive index are greater than $1/2$. The relation between the sign of the index and the value of an element is summarized by the property below.

Property 3.3.1. $\forall n \in \mathbb{N} : 0 < p_{-n} < 1/2 \quad and \quad 1/2 \leq p_n < 1$

Proof ■ See Property A.4.11 (page 126). ■

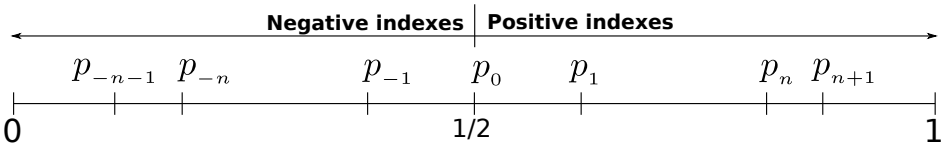


Figure 3.4 – Repartition of the elements of the recursive set over the interval $]0, 1[$ $(\forall n \in \mathbb{N})$

Moreover, elements greater than $1/2$ increase when their indexes also increase. Similarly, elements less than $1/2$ decrease with their index. These properties are illustrated on fig. 3.4 and presented formally on the following property.

Property 3.3.2. $\forall n \in \mathbb{N} : p_n < p_{n+1} \quad and \quad p_{-n} > p_{-n-1}$

Proof ■ Let n be a positive or null integer. (see Property A.4.3 (page 125)) shows that $a_n < a_{n+1}$. Since we have $p_n = a_n$, then we obtain $p_n < p_{n+1}$.

Besides, Property A.4.7 (page 125) shows that $b_n > b_{n+1}$. Since by definition $p_{-n} = b_{-(-n)}$, then $p_{-n} = b_n$. Subsequently, we obtain, $p_{-n} > p_{-n-1}$. ■

3.3.3.2 Inverse element

As expressed in the property below, the inverse of the element p_n with respect to the operator $\odot$ is p_{-n}.

Property 3.3.3. $\forall n \in \mathbb{Z} : p_n \odot p_{-n} = 1/2$

Proof ■ See Property A.4.12 (page 126). ■

3.3.3.3 Distance between successive elements

Let n be positive. If p_n, p_{n+1} and p_{n+2} are three successive elements of the recursive set, then the distance between p_n and p_{n+1} is greater than the distance between p_{n+1} and p_{n+2}. That means, the closer to 1 are two elements, the smaller is the distance between them.

Similarly, p_{-n}, p_{-n-1} and p_{-n-2} are successive elements less than $1/2$. The distance between p_{-n} and p_{-n-1} is greater than the distance between p_{-n-1} and p_{-n-2}. The closer to 0 are two elements, the smaller is the distance between them. Both properties are summarized as follows.

Property 3.3.4. $\forall n \in \mathbb{N} :$
$|p_{n+2} - p_{n+1}| < |p_{n+1} - p_n| \quad and \quad |p_{-n-2} - p_{-n-1}| < |p_{-n-1} - p_{-n}|$

Proof ■ See Property A.4.13 (page 127). ■

3.3.3.4 Maximal distance between successive elements

The maximal distance between successive elements of the recursive set is equal to the parameter ε. That means, the distance maximal is **user-adjustable**. The distance maximal is equivalent to the distance between $1/2$ and p_1 and to the distance between $1/2$ and p_{-1}.

Property 3.3.5. $\forall n \in \mathbb{Z} : |p_{n+1} - p_n| \leq \varepsilon$

Proof ■ See Property A.4.14 (page 127). ■

3.3.3.5 Influence of the parameter ε

The parameter ε expresses the distance between p_0 and p_1, as well as the distance between p_0 and p_{-1} (Theorem 3 (page 60)). As presented above, ε is also the upper bound of the distance between two successive elements of the recursive set of probabilities. The influence of ε on the distance between successive elements is illustrated on fig. 3.5.

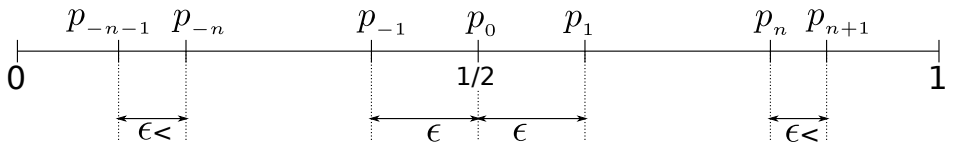


Figure 3.5 – Influence of ε on the distance between successive elements

Besides, fig. 3.6 shows three examples of recursive sets with three values of the parameter ε, namely $0.05, 0.01$ and 0.005. Due to Property 3.3.4 (page 62), only few elements of the recursive set lie around $1/2$ for an ε equal to 0.05. Elements close to 0 and 1 are though numerous. Elements with indexes greater than 20 (resp. less than -20) are very close to 1 (resp. very close to 0).

To increase the number of values around $1/2$, ε can be lowered. For instance, when ε is lowered to 0.01, the density of elements around $1/2$ increases. Lowering ε decreases also the maximum bound of the distances between successive elements (Property 3.3.5 (page 62)). Reaching values near 1 and 0 requires though a wide range of indexes with. For instance, indexes lie between -25 and 25 on fig. 3.6. With an ε equal to 0.05, the recursive set have elements that spread over $]0, 1[$. However with the same range of indexes, the elements of the recursive set are shrunk within $]0.2, 0.8[$ when ε is equal to 0.01 or 0.005.

3.4 Integer Occupancy Grids

After presenting the definition of the recursive set of probabilities, let us now introduce the Integer Occupancy Grid paradigm. To give an intuition about the principle of Integer Occupancy Grids, let us begin by an introductory example.

3.4.1 Introductory example

Consider a grid composed of four cells and two sensor measurements z and z'. The set of the occupancy probabilities given measurement z constitutes the mono-sensor occupancy grid denoted by $OG(z)$. Similarly, $OG(z')$ denotes the mono-sensor occupancy probability given measurement z'. Both $OG(z)$ and $OG(z')$ are depicted on fig. 3.7a.

For a cell c_i, the fusion of the ISM $P(o_i|z)$ with the ISM $P(o_i|z')$ gives the multi-sensor occupancy probability $P(o_i|z \wedge z')$. The fusion is performed by the

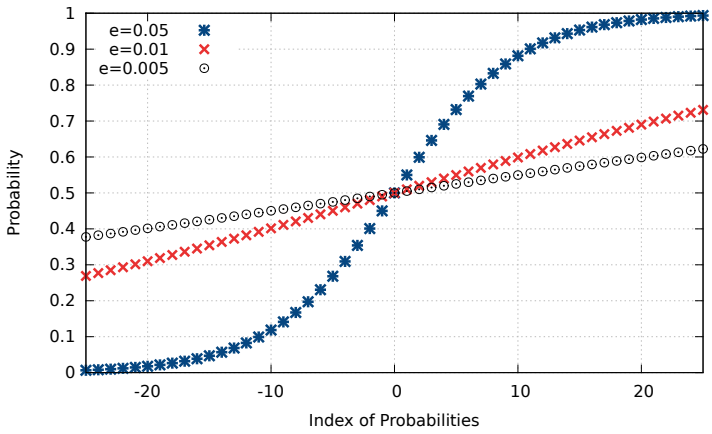


Figure 3.6 – Example of recursive set of probabilities with three values of the parameter ε

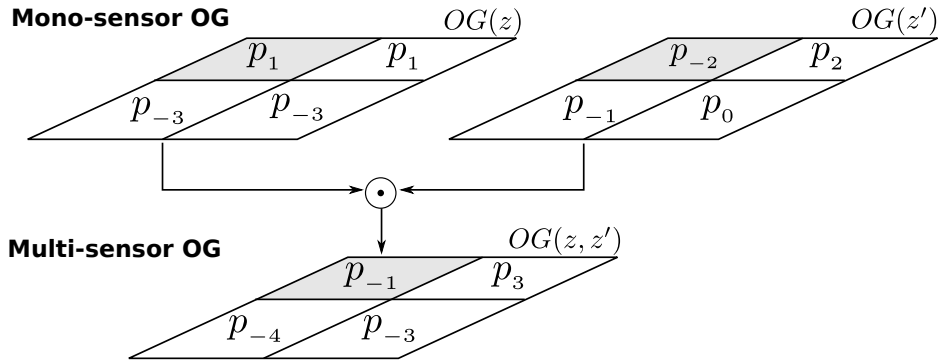
operator $\odot$. Let us assume that the values of the ISMs are equal to some elements of the recursive set. If $P(o_i|z)$ is equal to p_n while $P(o_i|z')$ is equal to p_m, therefore the value of $P(o_i|z \wedge z')$ becomes p_{m+n}. The occupancy probabilities $P(o_i|z \wedge z')$, $i \in \{1, \ldots, 4\}$ for the 4 cells, constitutes the multi-sensor occupancy grid $OG(z, z')$.

Consider a cell c_i. Assume that its occupancy probability $P(o_i|z \wedge z')$ is equal to p_n. In this case the index n is called **occupancy index** of cell c_i given measurements z and z'. It is denoted by $I(o_i|z \wedge z')$. The knowledge of the occupancy index of a cell is enough to determine the value of its occupancy probability. In fact, inversely, if $I(o_i|z \wedge z')$ is equal to n, therefore the occupancy probability $P(o_i|z \wedge z')$ is equal to p_n.

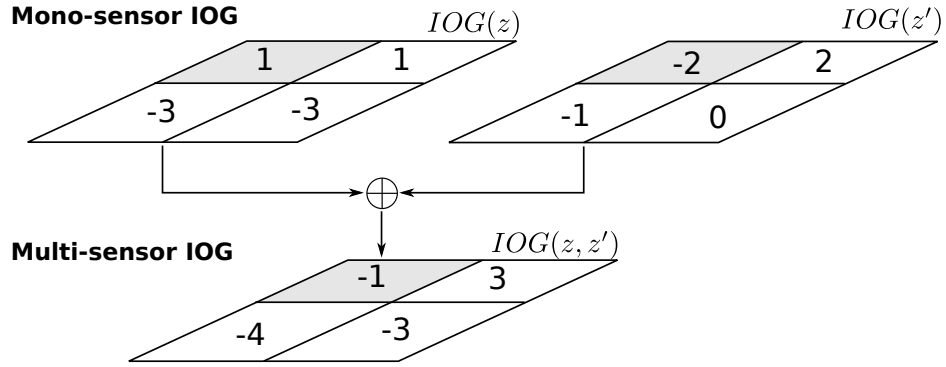
The set of the occupancy indexes of all cells of the grid constitutes the **Integer Occupancy Grid** given both measurements z and z'. The integer occupancy grid is denoted by $IOG(z, z')$. It is depicted on fig. 3.7b. Notice that integer occupancy grids can also be computed from a unique sensor. The integer occupancy grid given measurement z is denoted by $IOG(z)$. Similarly, $I(o_i|z)$ designates the occupancy index of a cell c_i given measurement z.

Like occupancy grids, integer occupancy grids can also be fused cell-by-cell in order to perform fusion. For a given cell, the fusion of two occupancy indexes given different measurements is computed by the index fusion operator $\oplus$. For the recursive set of probabilities, $\oplus$ is equivalent to a sum of indexes. Therefore, the occupancy index $I(o_i|z \wedge z')$ is equivalent to the sum of both $I(o_i|z)$ and $I(o_i|z')$.

For instance, let c_1 denote the gray cell on fig. 3.7. The occupancy probability of c_1 given z is equal to p_1 (fig. 3.7a). Therefore, its occupancy index $I(c_1|z)$ is equal to 1 (fig. 3.7b). Furthermore, the occupancy probability of c_1 given z' is equal to p_{-2}. Its occupancy index $I(c_1|z')$ is then equal to -2. By using occupancy indexes instead of occupancy probabilities, performing the fusion of both z and z' at the level of cell c_1 is equivalent to computing the sum of 1 and -2. The fusion returns



(a) Occupancy Grids



(b) Integer Occupancy Grids

Figure 3.7 – Occupancy Grid vs. Integer Occupancy Grid

$I(c_1|z \wedge z')$ which is equal to -1.

3.4.2 Definition of occupancy indexes and Integer Occupancy Grids

Notation Let z be a measurement and p_n ($n \in \mathbb{Z}$) an element of a set of probabilities. To indicate that the value of the occupancy probability $P(o_i|z)$ is equal to p_n, two equivalent notations are adopted:

$$P(o_i|z) \doteq p_n \quad \text{or} \quad p_n \doteq P(o_i|z) \tag{3.30}$$

Let S be a set of probabilities equipped with an index fusion operator $\oplus$. The set S can be the recursive set of probabilities or any other set of probabilities. Let $\mathcal{G}$ be a grid and $z_1, \ldots, z_K$ a collection of sensor measurements. The grid can be 1D, 2D or 3D.

Definition 3.4.1. *The* ***Occupancy Index*** *given measurements* $z_1, \ldots, z_k$ *designates the function* $I_{z_1,\ldots,z_k}$ *that maps a cell* c_i *to an integer* n*:*

$$\begin{array}{rccc} I_{z_1,\ldots,z_K} & : & \mathcal{G} & \mapsto & \mathbb{Z} \\ & & c_i & \mapsto & n \end{array} \tag{3.31}$$

such that $P(o_i|z_1 \wedge \ldots \wedge z_k) \doteq p_n \in S$.

Notation To adopt a notation similar to occupancy probabilities, let $I(o_i|z_1 \wedge \ldots \wedge z_k)$ designate the occupancy index of cell c_i given measurements $z_1, \ldots, z_k$. Therefore, the above definition gives:

$$\forall c_i \in \mathcal{G} : \left[I(o_i|z_1 \wedge \ldots \wedge z_k) \in \mathbb{Z} \ \wedge \ P(o_i|z_1 \wedge \ldots \wedge z_k) \doteq p_{I(o_i|z_1 \wedge \ldots \wedge z_k)}\right] \tag{3.32}$$

The term $p_{I(o_i|z_1 \wedge \ldots \wedge z_k)}$ designates the element of the set of probabilities S which index is equal to $I(o_i|z_1 \wedge \ldots \wedge z_k)$.

Equation (3.32) means that the knowledge of the occupancy index $I(o_i|z_1 \wedge \ldots \wedge z_k)$ of a cell is enough to determine its occupancy probability $P(o_i|z_1 \wedge \ldots \wedge z_k)$. In the previous chapter, an occupancy grid was defined as a set of the occupancy probabilities of all cells of a grid. Similarly, we introduce the definition of Integer Occupancy Grids as the set of the occupancy indexes of all cells of the same grid.

Definition 3.4.2. *Let* $\mathcal{G}$ *be a grid. The* ***Integer Occupancy Grid (IOG)*** *given measurements* $z_1, \ldots, z_k$ *designates the function that maps the collection of the same measurements, to the set of the occupancy indexes of all cells:*

$$IOG(z_1, \ldots, z_k) = \{I(o_i|z_1 \wedge \ldots \wedge z_k), \forall c_i \in \mathcal{G}\} \tag{3.33}$$

In particular, an integer occupancy grid $IOG(z)$ *built from a single measurement* z *is called* ***mono-sensor integer occupancy grid****.*

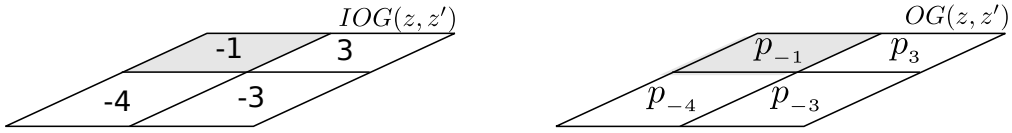
Example Figure 3.8 shows an example of an integer occupancy grid with the corresponding occupancy grid given two measurements z and z'. Let c_1 designates the gray cell. The occupancy index $I(o_1|z \wedge z')$ is equal to -1. Therefore, the value of the occupancy probability of c_1 given measurements z and z' is:

$$P(o_1|z \wedge z') \doteq p_{-1} \tag{3.34}$$

If the set of probabilities S is equivalent to the recursive set with the parameter ε, Theorem 3 (page 60) gives:

$$p_{-1} = 1/2 - \varepsilon \tag{3.35}$$

For an ε equal to 0.05, the numerical value of $P(o_1|z \wedge z')$ is equal to 0.45.



(a) Integer Occupancy Grid (b) Occupancy grid

Figure 3.8 – Example of integer occupancy grid with the corresponding standard occupancy grid

3.4.3 Multi-sensor integer occupancy grids

This section presents how to compute an integer occupancy grids given multiple sensor measurements. The Bayesian fusion allows to compute multi-sensor occupancy probabilities incrementally by combining occupancy probabilities cell-by-cell through the fusion operator $\odot$. The later enables to compute occupancy probabilities in an incremental way (Property 3.1.1 (page 51)).

Besides, consider a set of probabilities that has an index fusion operator $\oplus$. The Bayesian fusion of two elements of a set of probabilities is equivalent to combining the indexes of the elements by the index fusion operator (Definition 3.2.3 (page 57)). The following property shows that, the index fusion operator also enables incremental fusion of occupancy indexes.

Property 3.4.1. *Let $\mathcal{G}$ be a grid and S a set of probability that has an index fusion operator. Let c_i be a cell and $z_1, \ldots, z_k$ a collection of sensor measurements. Then*

$$I(o_i|z_1 \wedge \ldots \wedge z_k) = I(o_i|z_1 \wedge \ldots \wedge z_{k-1}) \oplus I(o_i|z_k) \tag{3.36}$$

Proof ■ By applying the equivalence between $\odot$ and $\oplus$ on eq. (3.14) (Definition 3.2.3 (page 57)), we obtain:

$$I(o_i|z_1 \wedge \ldots \wedge z_{k-1}) \oplus I(o_i|z_k) \Leftrightarrow p_{I(o_i|z_1 \wedge \ldots \wedge z_{k-1})} \odot p_{I(o_i|z_k)}$$

The definition of the occupancy index (Definition 3.4.1 (page 66)) gives

$$p_{I(o_i|z_1 \wedge \ldots \wedge z_{k-1})} \doteq P(o_i|z_1 \wedge \ldots \wedge z_{k-1}) \quad \text{and} \quad p_{I(o_i|z_k)} = P(o_i|z_k)$$

Then

$$\begin{aligned} I(o_i|z_1 \wedge \ldots \wedge z_{k-1}) \oplus I(o_i|z_k) \Leftrightarrow & P(o_i|z_1 \wedge \ldots \wedge z_{k-1}) \odot P(o_i|z_k) \\ & = P(o_i|z_1 \wedge \ldots \wedge z_k) \\ & \doteq p_{I(o_i|z_1 \wedge \ldots \wedge z_k)} \\ & \Leftrightarrow I(o_i|z_1 \wedge \ldots \wedge z_{k-1}) \end{aligned}$$

Finally, $I(o_i|z_1 \wedge \ldots \wedge z_{k-1}) \oplus I(o_i|z_k) = I(o_i|z_1 \wedge \ldots \wedge z_{k-1})$ ■

Property 3.4.1 (page 67) brings up the term $I(o_i|z_k)$ which is the occupancy index of a cell computed from a unique measurement. The recursive application of this property from z_1 to z_k gives the following theorem.

Theorem 4. Incremental index fusion

Let $\mathcal{G}$ be a grid and S a set of probability that has an index fusion operator. Let $z_1, \ldots, z_k$ denote sensor measurements. Then the multi-sensor occupancy index of c_i given the above collection of measurements is computed as follows:

$$I(o_i|z_1 \wedge \ldots \wedge z_k) = I(o_i|z_1) \oplus \ldots \oplus I(o_i|z_k) \tag{3.37}$$

Proof ■ Proof by induction.

- *Base case*: Property 3.4.1 (page 67) gives $I(o_i|z_1 \wedge z_2) = I(o_i|z_1) \oplus I(o_i|z_2)$.
- *Inductive step*: assume that $I(o_i|z_1 \wedge \ldots \wedge z_{k-1}) = I(o_i|z_1) \oplus \ldots \oplus I(o_i|z_{k-1})$. Let us prove that $I(o_i|z_1 \wedge \ldots \wedge z_k) = I(o_i|z_1) \oplus \ldots \oplus I(o_i|z_k)$.

  Property 3.4.1 (page 67) gives:

$$\begin{aligned} I(o_i|z_1 \wedge \ldots \wedge z_k) =& I(o_i|z_1 \wedge \ldots \wedge z_{k-1}) \oplus I(o_i|z_k) \\ =& [I(o_i|z_1) \oplus \ldots \oplus I(o_i|z_{k-1})] \oplus I(o_i|z_k) \end{aligned}$$

  Therefore, we obtain $I(o_i|z_1 \wedge \ldots \wedge z_k) = I(o_i|z_1) \oplus \ldots \oplus I(o_i|z_k)$.

■

Theorem 4 (page 68) combines occupancy indexes given different sensor measurements through the operator $\oplus$. It expresses how integer occupancy grids supports fusion. Theorem 4 (page 68) can be applied with the support of any set of probabilities provided that the set has an index fusion operator. In particular, the index fusion operator of the recursive set of probabilities is equivalent to a simple addition. Therefore, the multi-sensor fusion under the recursive set of probabilities is summarized by the following theorem.

Theorem 5. Multi-sensor fusion with the recursive set of probabilities

Let $\mathcal{G}$ be a grid and S_ε the recursive set of probability with a parameter ε. Let $z_1, \ldots, z_k$ denote sensor measurements. Then the multi-sensor occupancy index of cell c_i given the above collection of measurements is computed as follows:

$$I(o_i|z_1 \wedge \ldots \wedge z_k) = I(o_i|z_1) + \ldots + I(o_i|z_k) \tag{3.38}$$

Proof ■ Theorem 4 (page 68) shows that:

$$I(o_i|z_1 \wedge \ldots \wedge z_k) = I(o_i|z_1) \oplus \ldots \oplus I(o_i|z_k)$$

Besides, Corollary 3.3.1 (page 60) states that for the recursive set of probabilities:

$$\forall m, n \in \mathbb{Z} : m \odot n = m + n$$

Therefore, the fusion under the recursive set of probabilities becomes:

$$I(o_i|z_1 \wedge \ldots \wedge z_k) = I(o_i|z_1) + \ldots + I(o_i|z_k) \tag{3.39}$$

■

3.4.4 Mono-sensor integer occupancy grids

Theorem 5 (page 68) allows integer occupancy grids to support multi-sensor fusion. Based on the recursive set of probabilities, Theorem 5 (page 68) enables to compute multi-sensor occupancy index by summing multiple occupancy indexes computed from individual independent measurements. But how to compute the occupancy index of a cell given a unique measurement?

Consider a measurement z, a grid $\mathcal{G}$ and a cell c_i. If there exists an element p_n of the recursive set S_ε such that:

$$P(o_i|z) \doteq p_n \tag{3.40}$$

then, the occupancy index of c_i given the measurement z is equal to n:

$$I(o_i|z) = n \tag{3.41}$$

There is however no a priori reason that the numerical value of $P(o_i|z)$ would be exactly equal to an element of S_ε. In other words, there is no a priori reason that an element p_n exists such that eq. (3.40) holds. As a solution, we propose to quantize the ISM $P(o_i|z)$ by an element of S_ε.

3.4.4.1 Quantization of the inverse sensor model

Assume that the ISM $P(o_i|z)$ of cell c_i is available [4]. Quantizing $P(o_i|z)$ by an element of S_ε is equivalent to looking for an element p_n that can approximate the numerical value of $P(o_i|z)$. The elements of S_ε are spread over $]0, 1[$ in a symmetric way with respect to $1/2$. Moreover, the sequence $\{p_n\}_{n\in\mathbb{Z}}$ is monotonically increasing (see Property A.4.8 (page 125)). Consequently, whatever is the value of the ISM $P(o_i|z)$, there exist an element p_n of S_ε such that:

$$p_n \leq P(o_i|z_k) < p_{n+1} \tag{3.42}$$

The ISM can be then approximated by either p_n or p_{n+1}. To choose among these two possibilities, the following quantification policy is proposed.

[4] A new theorem for computing the ISM of single-target sensors is presented in Section 4.2 (page 89)

The nearest quantization policy

Definition 3.4.3. *Let z be a measurement, c_i a cell and ε a number between 0 and 1/2. Let p_n be an element of the recursive set S_ε such that $p_n \leq P(o_i|z) < p_{n+1}$. The* ***nearest quantization policy*** *approximates the ISM by the nearest element of S_ε:*

$$P(o_i|z) \approx \begin{cases} p_n & \text{if } |P(o_i|z_k) - p_n| \leq |P(o_i|z_k) - p_{n+1}| \\ p_{n+1} & \text{otherwise} \end{cases}$$

Therefore,

$$I(o_i|z) = \begin{cases} n & \text{if } |P(o_i|z_k) - p_n| \leq |P(o_i|z_k) - p_{n+1}| \\ n+1 & \text{otherwise} \end{cases}$$

The nearest quantization policy is an intuitive method for quantizing the ISM. Depending on its numerical the value, the ISM is approximated either by p_n or by p_{n+1}. That means, the approximation may result into an approximated value greater or lower than the real value of $P(o_i|z)$. Let us note $\hat{P}(o_i|z)$ the approximate of $P(o_i|z)$ and let us analyze the effect of the nearest quantization. Four cases are possible:

- If $P(o_i|z) > 1/2$ and $\hat{P}(o_i|z) > P(o_i|z)$: the sensor estimates that the cell is likely occupied. The approximate of the ISM tends towards 1 as shown on fig. 3.9a. Therefore, the quantization introduces an overestimation of the occupancy state of the cell estimated by the sensor.

- If $P(o_i|z) > 1/2$ and $\hat{P}(o_i|z) \leq P(o_i|z)$: the sensor estimates that the cell is likely occupied. However, the approximate of the ISM tends now towards $1/2$ as shown on fig. 3.9b. The occupancy state of the cell to tend to unknown (neither occupied, nor empty). No overestimation is introduced by the quantization. The latter blurs the estimation of the occupancy state of the cell.

- If $P(o_i|z) < 1/2$ and $\hat{P}(o_i|z) < P(o_i|z)$: the sensor estimates that the cell is likely empty. The approximate of the ISM tends towards 0 (see fig. 3.9c). That means, the quantization overestimates the confidence of the sensor about the emptiness of the cell.

- Finally, if $P(o_i|z) < 1/2$ and $\hat{P}(o_i|z) \geq P(o_i|z)$: the sensor estimates that the cell is likely empty. The approximate of the ISM tends however towards $1/2$ (see fig. 3.9d). Thus, the quantization blurs occupancy state of the cell. It does not overestimate the confidence of the sensor.

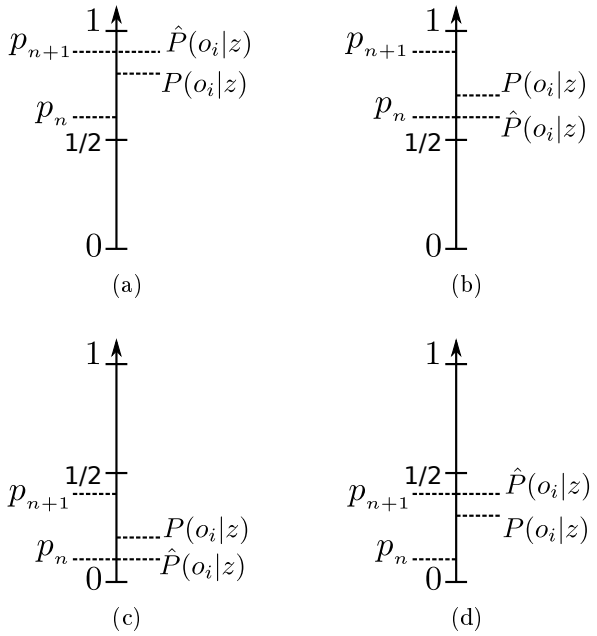


Figure 3.9 – Quantization of ISM

By considering the above discussions, a second quantization policy is proposed. This policy avoids to overestimate the confidence of sensor measurement about the occupancy state of the cell.

The blurring quantization policy

Definition 3.4.4. *Let z be a measurement, c_i a cell and ε a number between 0 and $1/2$. Let p_n be an element of the recursive set S_ε such that $p_n \leq P(o_i|z) < p_{n+1}$. The **blurring quantization policy** approximates the ISM by the element of S_ε nearest to $P(o_i|z)$ and closest to $1/2$:*

$$P(o_i|z) \approx \begin{cases} p_n & \text{if } P(o_i|z_k) \geq 1/2 \\ p_{n+1} & \text{otherwise} \end{cases}$$

Consequently,

$$I(o_i|z) = \begin{cases} n & \text{if } P(o_i|z_k) \geq 1/2 \\ n+1 & \text{otherwise} \end{cases}$$

The result of the blurring policy is illustrated on fig. 3.9b for the case where $P(o_i|z)$ is greater than $1/2$. Otherwise, the result of the blurring policy is depicted

on fig. 3.9d. In both cases, the approximate of the ISM tends towards 1/2. The quantization makes the occupancy state of cells to tends towards unknown.

3.4.5 Overview of multi-sensor fusion based on integer occupancy grids

Section 3.4.3 (page 67) has detailed the computation of multi-sensor integer occupancy grids by fusing mono-sensor ones. Section 3.4.4 (page 69) showed that mono-sensor integer occupancy grids are computed by quantizing ISMs. Finally, performing multi-sensor fusion based on integer occupancy grids require three steps. These steps are summarized on fig. 3.10 in the case of the fusion of two sensor measurements. They can be though generalized for any number of measurements.

The three steps for fusing two sensor measurements z and z' are described as follows.

1. ISMs are computed[5] from sensor measurements. This steps produces the mono-sensor occupancy grids $OG(z)$ and $OG(z')$.

2. The step of quantization quantizes the mono-sensor occupancy probabilities $P(o_i|z)$ and $P(o_i|z')$ for obtaining the mono-sensor occupancy indexes $I(o_i|z)$ and $I(o_i|z')$ for all cell of the grid. This steps outputs mono-sensor integer occupancy grids $IOG(z)$ and $IOG(z')$

3. Mono-sensor integer occupancy grids are fused cell-by-cell. The fusion consists in an addition of the mono-sensor occupancy indexes of each cell. The fusion produces the multi-sensor integer occupancy grid $IOG(z, z')$ that is the result of the fusion of both measurements z and z'.

3.4.6 Discussion

3.4.6.1 Bayesian properties of the index fusion

With the recursive set of probabilities, the index fusion is equivalent to a sum of occupancy indexes. Like the probabilistic fusion, the index fusion is also incremental, and supports both mitigation and reinforcement.

Incremental The incremental property of the index fusion was already demonstrated by Theorem 5 (page 68).

Reinforcement The index fusion reinforces the estimation of non-conflicting measurements. If two measurements are non-conflicting, both estimate either a cell is occupied or the cell is empty. The corresponding occupancy probabilities are then

[5] The methods for computing mono-sensor occupancy grids reviewed in Section 2.3 (page 27) can be applied. A new theorem for computing ISMs of single-target sensors will be also presented in Section 4.2 (page 89).

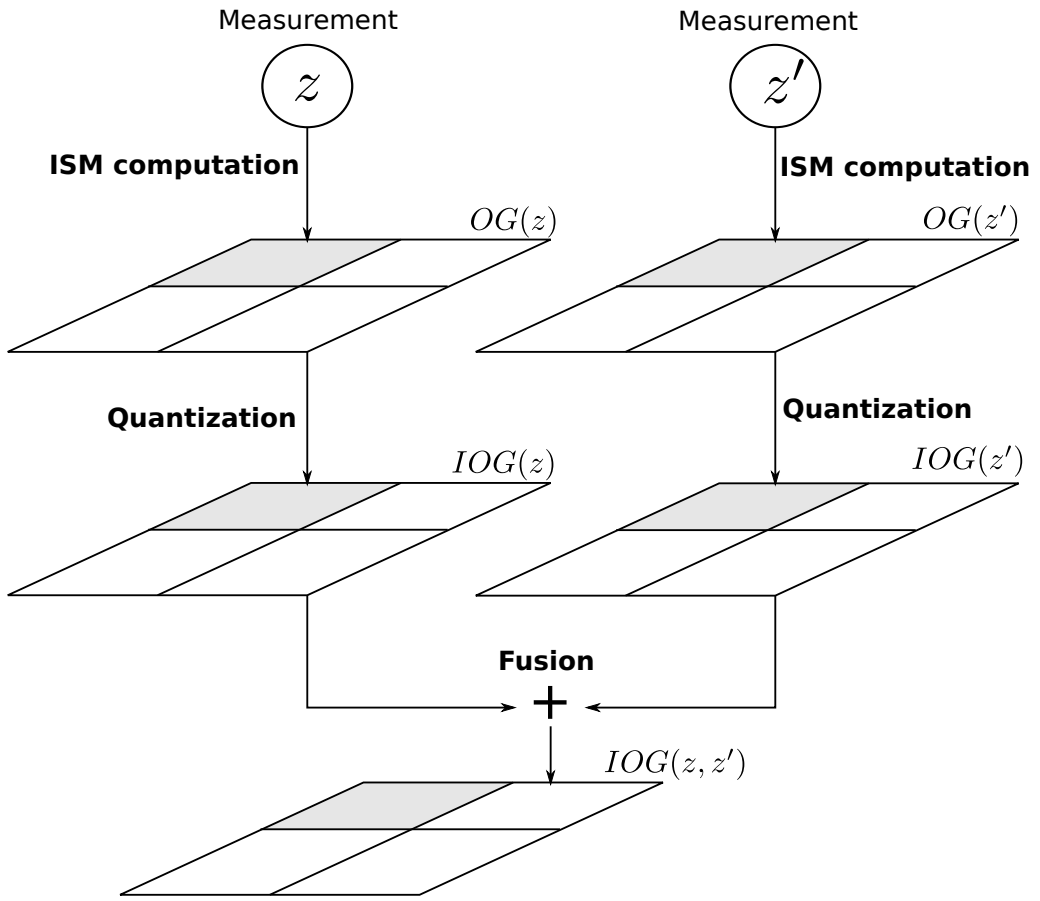


Figure 3.10 – Overview of the multi-sensor fusion based on integer occupancy grids

either greater than $1/2$ or less than $1/2$. The corresponding occupancy indexes have the same sign. The property of reinforcement is expressed as follows.

Property 3.4.2. *Let c_i be a cell and z and z' be two sensor measurements. Then, with the recursive set of probabilities, the index fusion gives:*

$$\begin{aligned} &\text{If } I(o_i|z), I(o_i|z') > 0, \text{ then } I(o_i|z \wedge z') > Max(I(o_i|z), I(o_i|z')) \\ &\text{If } I(o_i|z), I(o_i|z') < 0, \text{ then } I(o_i|z \wedge z') < Min(I(o_i|z), I(o_i|z')) \end{aligned} \tag{3.43}$$

Proof ■ With the recursive set of probabilities, we have $I(o_i|z \wedge z') = I(o_i|z) + I(o_i|z')$. Therefore, eq. (3.43) becomes trivial. ■

Mitigation The index fusion mitigates the estimation of conflicting measurements. If two measurements are conflicting, one estimates that a cell is occupied while the other estimates that the cell is occupied. The occupancy probability given the first measurement is less than $1/2$ while the occupancy probability given the

second measurement is greater than 1/2. Therefore, with the recursive set of probabilities, one occupancy index is negative while the other is positive. Subsequently, the sum of both occupancy indexes tends towards 0. The occupancy state of the cell tends to unknown. The property of mitigation is summarized as follows.

Property 3.4.3. *Let c_i be a cell and z and z' be two sensor measurements. Then, with the recursive set of probabilities, the index fusion gives:*

$$\textit{If } sign(I(o_i|z)) \neq sign(I(o_i|z')), \textit{ then } |I(o_i|z \wedge z')| < Min(|I(o_i|z), |I(o_i|z')|)$$

Proof ■ With the recursive set of probabilities, we have $I(o_i|z \wedge z') = I(o_i|z) + I(o_i|z')$. Therefore, the above equation becomes trivial. ■

3.4.6.2 Numerical error

The multi-sensor fusion based on integer occupancy grids relies on the step of quantization for computing mono-sensor occupancy indexes from ISMs. The quantization approximates a mono-sensor occupancy probability $P(o_i|z)$ by an element of the recursive set of probabilities. As seen in Section 3.4.4.1 (page 69), $P(o_i|z)$ is approximated either by p_n or p_{n+1} such that:

$$p_n \leq P(o_i|z) < p_{n+1} \tag{3.44}$$

Therefore, the quantization introduces a numerical error. The error is however bounded by the distance between successive elements p_n and p_{n+1} regardless of the quantization policy. According to Property 3.3.5 (page 62), the maximal distance between successive elements of the recursive set is bounded by ε. The latter can be any real-number between 0 and 1/2 that is defined by the user (Theorem 3 (page 60)). Therefore, the numerical error introduced by the quantization is **bounded** by ε, moreover, **the bound is defined by the user**.

Remark. Notice that due to Property 3.3.4 (page 62), the error is maximal only around 1/2. The closer to 0 or to 1 is an occupancy probability, the smaller is the quantization error.

3.4.6.3 Choosing the parameter ε

Once mono-sensor integer occupancy grids are computed, the fusion is performed through sums of occupancy indexes. In other words, the fusion requires only sums of integers. Such a sum is exact on an implementation viewpoint provided that no integer overflow [6] occurs.

Unlike the step of quantization, the fusion of multiple integer occupancy grids does not introduce additional numerical errors. Consequently, the overall error is

[6] For instance, on a 32-bit machine, an integer has to be between $-2^{31}+1$ and $2^{31}-1$. Beyond these limits, an integer cannot be encoded in 32-bit anymore.

parametrized by ε, the maximal error of the quantization step. The smaller is epsilon, the smaller is the numerical error. At the same time, Section 3.3.3.5 (page 63) has showed that for a small value of ε, a large range of index of probability is required to get elements of the recursive set close to 1 or 0.

Furthermore, the recursive set of probabilities maintain a one-to-one mapping between integers and real-numbers between 0 and 1. The elements of the recursive set are computed recursively. Computing the value of an element given its index is then expensive in term of complexity. As a solution, a lookup table can be used for storing the elements with indexes between $-M$ and M. For instance, if M is equal to 127, the lookup table stores the elements of the recursive set with indexes between -127 and 127.

Such a lookup table allows to retrieve fast the value of the element that corresponds to an index. It allows to convert an occupancy index into the corresponding occupancy probability. It can be computed once at the beginning of the program and then reused latter when fusing multiple sensors. This method has however a disadvantages. For a small value of ε, the lookup table has to be very large in order to contain elements close to 1 or 0.

To sum up, the value of ε should be chosen as a function of the maximum acceptable quantization error, the maximum value of occupancy probabilities (near 1), the minimum value of occupancy probability and the size of the lookup table. In practice, a consensus between these parameters can be found.

For instance, if ε is equal to 0.05, the maximum quantization error is 0.05. For the same value of ε, p_{-127} is less than 10^{-11} while p_{127} is greater than $1 - 10^{-11}$. Therefore, an index range between -127 and 127 is enough to have elements of the recursive set that are close to 0 and close to 1. Thus, a lookup table with 256 number of elements is enough.

3.4.6.4 Implementation of the index fusion

Integer fusion The index fusion performs only sums of occupancy indexes. It does not require a digital representation of real-numbers. Processing integer arithmetic on a computing hardware is exact. The implementation of the fusion does not involve additional numerical errors.

Notice however that the step computation of ISMs may still need operation on real-numbers. For instance, the approaches for computing ISMs reviewed in Section 2.3.1.2 (page 31) process probabilities as real-numbers. A practical solution for minimizing these operations will be proposed in Section 4.3 (page 100).

Number of operation For fusing two sensor measurements at the level of a cell, the Bayesian fusion function performs seven operations on real-numbers (Proposition 3.1.1 (page 50)). On the opposite, the integer fusion performs only a single addition of integers (Theorem 5 (page 68)). Consequently, the integer fusion has divided 7 the number of operations required by the fusion.

The log-odds form of the Bayesian fusion also performs only a single addition (see eq. (2.46) in page 40). However, it adds two real-numbers while the index fusion adds two integers. If the fusion based on log-odds is implemented with floating-points, the error introduced by floating-points is not user-bounded. The error is imposed by the floating-point standard. Second, the transformation of log-odds to probability requires a call to the exponential function. This becomes expensive with high-number of cells, especially on a resource-constrained embedded hardware.

Numerical stability The Bayesian fusion function suffers from numerical instability when computing probabilities near 0 and 1 (see Section 2.4.2 (page 39)). In opposite, the index fusion is stable since it is based on addition of integers. The implementation should only pay attention to integer overflow. In the example above where ε is equal to 0.05 and the range of indexes is limited between -127 and 127, occupancy indexes can be encoded in 8-bits. Therefore, the sum of occupancy indexes must be saturated so that the result fits within 8-bit to avoid overflow.

Deterministic fusion The implementation of the index fusion is deterministic. It gives the same result regardless of the hardware architecture, the code writing, the compiler, the compiling options or other technical details. In fact, integer addition behave the same on every processors that can handle integers with the same number of bits. For instance, on all 32-bit machines, the result of the addition of two integer is the same.

3.4.6.5 Summary of the discussion

Let us summarize the above discussion.

Benefits The advantages of the integer occupancy grid paradigm are:

- The index fusion is mathematically equivalent to the Bayesian fusion.
- The index fusion is incremental and supports both reinforcement and mitigation like the original Bayesian fusion.
- The index fusion is based on addition of integers and has 7× less operation than the original Bayesian fusion.
- The fusion is numerically stable and its implementation its SW/HW integration is deterministic.

By taking into account the above properties, the index fusion enables a safe SW/HW integration of probabilistic multi-sensor fusion.

Limitations Integer occupancy grids introduce explicitly an error in the step of quantization. The error is however **bounded** and is **parametrized by the user**. The error ε can be any real-number between 0 and $1/2$. A too small value of ε would however require a too large lookup table necessary for converting an occupancy index into an occupancy probability.

3.5 Compaction of Integer Occupancy Grids

The previous section has introduced the formal definition of integer occupancy grids and the steps required for computing them. Let us now consider the implementation viewpoint. An integer occupancy grid is by definition constituted by the set of the occupancy indexes of all cells within a grid. On an implementation viewpoint, the occupancy indexes are stored within a data structure. This data structure will be requested when the occupancy indexes are updated by new sensor measurements, or when decision-making applications need to exploit integer occupancy grids.

Section 2.5 (page 43) reviewed the two main data structures – namely the arrays and the 2^d-trees – for storing occupancy grids. An array stores in its elements the occupancy probability of each cell. For a high-number of cell, constituting blocks of cells that have similar occupancy state is time-consuming with arrays. Such task is though required for navigation applications that have to search for free spaces and obstacles. Arrays can be also memory consuming, especially for high-resolution and high-dimensional grid. As a solution, the 2^d-trees have been proposed for compacting occupancy grids. The compaction is though lossy.

Both arrays and 2^d-trees can be also used for storing integer occupancy grids. Arrays can be used to store the occupancy indexes of all cells. This approach would suffer from the same limitations of arrays as explained above. This chapter presents the use of 2^d-trees for compacting integer occupancy grids. The differences with the state-of-the-art is that the compaction of integer occupancy grids with 2^d-trees is lossless. A lossless compaction is required for safety.

To explain the lossless compaction, this chapter will begin by the traditional definition of 2^d-trees and presents their extension enabling the storage of integer occupancy grids. The update of elements stored by 2^d-trees requires two fundamental operations: the split and the merge. Both operations will be also detailed, followed by a discussion about the application of 2^d-trees on integer occupancy grids.

3.5.1 Definition of 2^d-trees

A 2^d-tree is composed of a set of nodes having a parent-children relation. Any node, except the root node, must have a parent. A node has either zero or 2^d number of children, d corresponds to the dimension of the grid. A node that has no child is called a leaf. A node has a depth. The latter is a positive integer that measures how height from the root is located the node. A 2^d-tree has a maximum depth. The depth of a node cannot be greater than the maximum depth of the tree. The above characteristics are formally summarized into the following definition.

Definition 3.5.1. *A 2^d-tree is a set $\mathcal{N}$ of node that holds the following assertions:*

1. $\exists! \eta \in \mathcal{N} : [Root(\eta) \Leftrightarrow Parent(\eta) = null]$
2. $\forall \eta \in \mathcal{N} : [Leaf(\eta) \Leftrightarrow Children(\eta) = \emptyset]$
3. $\forall \eta \in \mathcal{N} : [\neg Leaf(\eta) \Leftrightarrow Children(\eta) = \{\eta_j, j = 1, \ldots, 2^d\}]$
4. $\forall \eta \in \mathcal{N} : [\neg Root(\eta) \Rightarrow (\exists! \eta' \in \mathcal{N} : Parent(\eta) = \eta') \wedge \eta \in Children(\eta')]$
5. $\forall \eta \in \mathcal{N} : Depth(\eta) = \begin{cases} 0 & \text{if } Root(\eta) \\ 1 + Depth(Parent(\eta)) & \text{otherwise} \end{cases}$
6. $\forall \eta \in \mathcal{N} : Depth(\eta) \leq MaxDepth(\mathcal{N})$

The predicate $Root(\eta)$ holds if and only if the node η is the root. The function $Parent(\eta)$ returns the node parent of η. The function $Children(\eta)$ returns the set of the children of η. Assertion 1 means that a 2^d-tree has a unique root node. The latter has no parent. Assertion 2 imposes that a leaf node does not have any child. In opposite, if a node is not a leaf, it has exactly 2^d number of children according to Assertion 3. Assertion 4 states that any node different of the root must have a unique parent. Finally, the function $Depth(\eta)$ returns the depth of a given node. The root has a depth equal to 0. The depth of the other nodes is determined recursively by the Assertion 6. The depth has a maximum value returned by the function $MaxDepth(\mathcal{N})$.

3.5.2 Extension of 2^d-trees for storing integer occupancy grids

Definition 3.5.1 (page 78) presents a general definition of an 2^d-tree. For storing an integer occupancy grid into a 2^d-tree, Definition 3.5.1 (page 78) needs to be extended. Storing an integer occupancy grid means storing the occupancy indexes of cells. For this purpose, we introduce the following definition.

Definition 3.5.2. *Let $\mathcal{N}$ be a 2^d-tree, and η a node of $\mathcal{N}$. Let $\mathcal{G}$ be a grid of d dimension. The function $Region(\eta)$ returns a set of cells such that the following assertion holds:*

1. $\forall \eta \in \mathcal{N} : Region(\eta) \neq \emptyset$
2. $\forall \eta \in \mathcal{N} : [Root(\eta) \Leftrightarrow Region(\eta) = \mathcal{G}]$
3. $\forall \eta \in \mathcal{N} : [\neg(Leaf(\eta)) \Rightarrow Region(\eta) = \bigcup_{\eta' \in Children(\eta)} Region(\eta')]$
4. $\forall c_i \in \mathcal{G}, \exists \eta \in \mathcal{N} : c_i \in Region(\eta)$
5. $\forall c_i \in \mathcal{G}, \exists! \eta \in \mathcal{N} : [Leaf(\eta) \wedge c_i \in Region(\eta)]$
6. $\forall \eta \in \mathcal{N} : [cardinal(Region(\eta)) > 1 \Rightarrow \forall c_i, c_j \in Block(\eta), adjacent(c_i, c_j)]$

The function $Region(\eta)$ maintains a correspondence between a tree structure and a grid. A region is a set of cells. A region cannot be empty according to the first assertion. The second assertion states that the region of the root is equal to the whole grid. Assertion 3 stipulates that the region of a non-leaf node is equal to the reunion of the regions of its children. In Assertion 4, a cell must belong to the region of a node. Assertion 5 imposes though that a cell belongs to a unique leaf node. A consequence of this assertion is that the regions of leaf nodes are disjoints:

$$\forall \eta, \eta' \in \mathcal{N} : Region(\eta) \cap Region(\eta') = \emptyset \tag{3.45}$$

Finally, Assertion 6 requires that all cells belonging to the region of the same node must be adjacent.

To identify the cells that belong to the region of a node, the principle of recursive **split** is applied. Consider a grid of d dimensions and a 2^d-tree. If l denotes the maximum depth of the tree, the grid contains $2^l \times 2^l$ for 2D and $2^l \times 2^l \times 2^l$ for 3D. The region of the root contains all cells of the grid. To determine the region of the children of the root, the grid is split by 2 on each dimension. The subdivision results into 2^d adjacent but disjoint regions. These regions are affected to each child of the root. The same split operation can be applied recursively to each child of the root until reaching the maximum depth. The recursive split guaranties that eq. (3.45) holds. It is also conform to the assertions presented on Definition 3.5.2 (page 78).

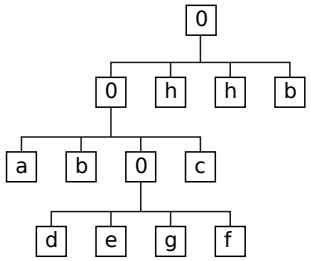
For allowing a 2^d-tree to store the occupancy indexes of cells, the following function is used.

Definition 3.5.3. *Let $\mathcal{N}$ be a 2^d-tree, and η a node of $\mathcal{N}$. Let $\mathcal{G}$ be a grid of d-dimension. Consider the recursive set S_ε and a collection of measurements $z_1, \ldots, z_N$. The function $Index(\eta)$ returns an integer such that:*

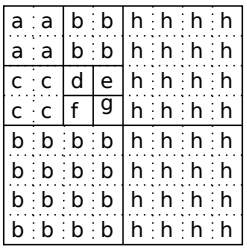
1. $\forall \eta \in \mathcal{N} : [\neg Leaf(\eta) \Rightarrow Index(\eta) = 0]$
2. $\forall \eta \in \mathcal{N} : [Leaf(\eta) \Rightarrow (\forall c_i \in Region(\eta), Index(\eta) = I(o_i | z_1 \wedge \ldots \wedge z_N))]$

The first assertion means that the index of a non-leaf node is null. With respect of the recursive set S_ε, this index correspond to the occupancy probability 1/2. The second assertion requires that the index of a leaf is equal to the occupancy indexes of the cells that belong to the region of the leaf. If the region of a leaf contains a single cell, the index of the leaf is equal to the occupancy index of that cell. Otherwise, if the region of a leaf contains multiple cells, these cells must have the same value of occupancy indexes.

Figure 3.11a shows an example of a quadtree that stores an integer occupancy grid. Leaves are at the extremity of the tree. Non-leaf nodes have an index equal to 0. The indexes of leaves are represented by letters where a letter is an integer value. The regions of leaves are shown on fig. 3.11b. The figure shows that cells



(a) A quadtree with the indexes of nodes



(b) Regions of the leaf nodes

Figure 3.11 – Example of an integer occupancy grid stored within a quadtree

belonging to the region of the same node have exactly the same occupancy indexes. The root is the node at the top of the quadtree. Its region corresponds to the whole grid, regrouping all the cells.

On fig. 3.11, the quadtree has in total 13 nodes while the grid has 64 cells. If the same integer occupancy grid was stored within an array, the latter would require 64 elements. This shows, through an example, how a 2^d-tree can compact the storage of an integer occupancy grid. The compaction is due to the requirement of Assertion 2 in Definition 3.5.3 (page 79). This assertion enables to store within a unique leaf the occupancy indexes of adjacent cells, provided that the occupancy indexes are mathematically equal.

3.5.3 Split

The index of a leaf stores the occupancy indexes of all cells contained within the region of the leaf. This technique of storage requires that the occupancy indexes of the cells within the region of the leaf must be equal. When a new sensor measurement is available, it updates the occupancy index of at least one cell. Consequently, the above equality does not holds anymore for the leaf which region contains the updated cell.

To take into account the new occupancy index of the updated cell, the operation of split is applied. The operation of split must be realized if the following condition is satisfied:

$$\begin{aligned} \forall \eta \in \mathcal{N} : \{ & Leaf(\eta) \\ & \wedge [\exists c_i \in Region(\eta) : I(o_i | z_1 \wedge \ldots \wedge z_K) \neq Index(\eta)] \\ & \Rightarrow Split(\eta, c_i, \mathcal{N})\} \end{aligned} \tag{3.46}$$

The above equation means that the operation of split must be realized on a leaf which region contains at least a cell c_i, such that the occupancy index of c_i is different to the index of the leaf.

The function *Split* is described on Algorithm 1 (page 81). It works as follows. Assume that a node η satisfies the condition on eq. (3.46). Let c_i denote the cell

Algorithm 1 The function of split

```
1:  function SPLIT(Node η, Tree 𝒩, Cell c_i)
2:      if Depth(η) < MaxDepth(𝒩) then
3:          𝒩 ← 𝒩 ∪ {η_j, j = 1, ..., 2^d}                ▷ Create 2^d new nodes.
4:          Children(η) ← {η_j, j = 1, ..., 2^d}          ▷ The new nodes are children of η.
5:          for all η_j ∈ Children(η) do
6:              Parent(η_j) ← η
7:              Depth(η_j) ← Depth(η) + 1
8:              Index(η_j) ← Index(η)                     ▷ A child receives the index of its parent.
9:          end for
10:         Index(η) ← 0                                  ▷ η is not a leaf anymore.
11:     else                                              ▷ η reaches the depth max.
12:         Index(η) ← I(o_i|z_1 ∧ ... ∧ z_K)             ▷ η stores the occupancy index of c_i.
13:     end if
14: end function
```

which occupancy index is different to the index of η. If the depth of η is less than the maximum depth, then the tree is extended by creating 2^d new nodes that become children of η. Otherwise, the index of η is set to the occupancy index of c_i.

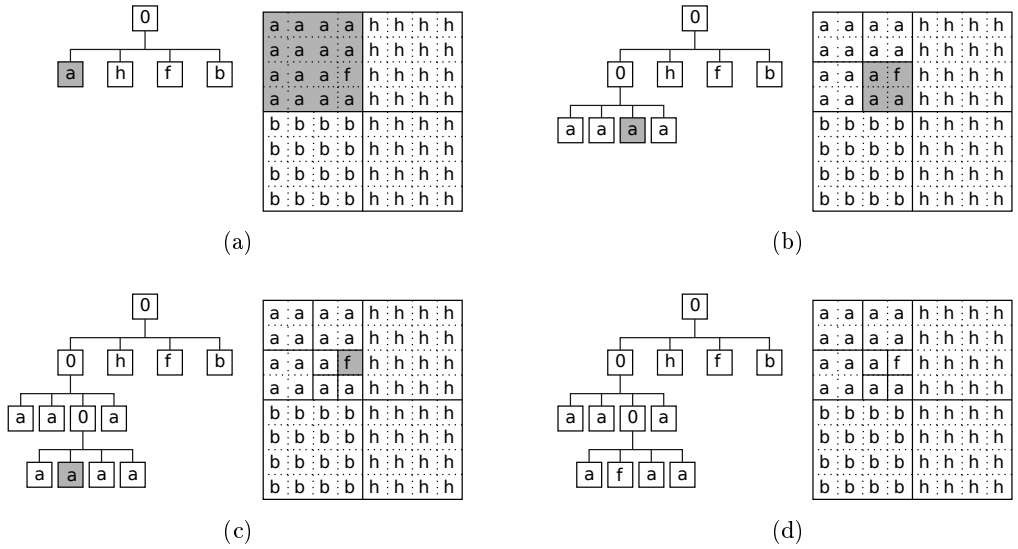


Figure 3.12 – Application of split

Figure 3.12 illustrates the application of the split function on a 2^d-tree. The tree is initially the one depicted at the left of fig. 3.12a. Consider the leaf colored in gray. Its region is also colored in gray on the grid at the right of the figure. According to the tree, the occupancy indexes of all gray cell are equal to a. However, according to the grid, there exists a gray cell that has an occupancy index equal to f. Consequently,

Algorithm 2 The function of merge

1: **function** MERGE(Node η,Tree $\mathcal{N}$)
2: $Index(\eta) \leftarrow Index(getAchildOf(\eta))$ ▷ η receives the index of its children.
3: $\mathcal{N} \leftarrow \mathcal{N} \setminus Children(\eta)$ ▷ The children of η are removed from the tree $\mathcal{N}$.
4: $Children(\eta) \leftarrow \emptyset$ ▷ η becomes a leaf.
5: **end function**

the gray leaf satisfies the split condition on eq. (3.46). The operation of split is applied on this leaf.

When the gray leaf is split, its children are created. Each child receives the index of its parent. This gives the tree depicted on fig. 3.12b. On this figure, another gray leaf satisfies the split condition. New children are also added at the level of this leaf. The resulting tree is illustrated on fig. 3.12c. Herein, the maximum depth is reached. However, there still exist a gray leaf that satisfies the split condition. The index of this leaf is finally set to the occupancy index of the cell included in the region of the leaf. The resulting tree is depicted on fig. 3.12d.

In practice, the operation of split is applied to any leaf which region is affected by a new sensor measurement. Figure 3.12 shows that the split operation extends the tree downwards until leaves affected by measurements reach the maximum depth. When several cells see their occupancy indexes updated by measurements, the number of newly created leaves increases considerably. This impacts negatively on the compactness of the tree structure. As a solution, the operation of **merge** is performed.

3.5.4 Merge

The operation of merge improves the compactness of a tree by removing some nodes from the tree. This enables to decrease the number of nodes within the tree. The operation of merge is applied only if the following equation holds.

$$\begin{aligned} \forall \eta \in \mathcal{N} : \{ & \neg Leaf(\eta) \\ & \wedge [\forall \eta_j \in Children(\eta) : Leaf(\eta_j)] \\ & \wedge [\forall \eta_i, \eta_j \in Children(\eta) : Index(\eta_i) = Index(\eta_j)] \\ & \Rightarrow Merge(\eta, \mathcal{N}) \} \end{aligned} \tag{3.47}$$

A node can be merged only if three conditions are satisfied. First, the node is not a leaf. Second, its children are all leaves. Third, the indexes of its children are equal.

The function $Merge$ is presented on Algorithm 2 (page 82). It takes two arguments: the node η to be merged and the 2^d-tree. First, the function affects to η the indexes of its children. Its children are then removed from the tree. This makes η to become a leaf.

Figure 3.13 shows an example of application of the operation of merge. The initial 2^d-tree is depicted on fig. 3.13a. The gray node satisfies the condition on eq. (3.47). It is merged and the result is shown on fig. 3.13b. There exists again

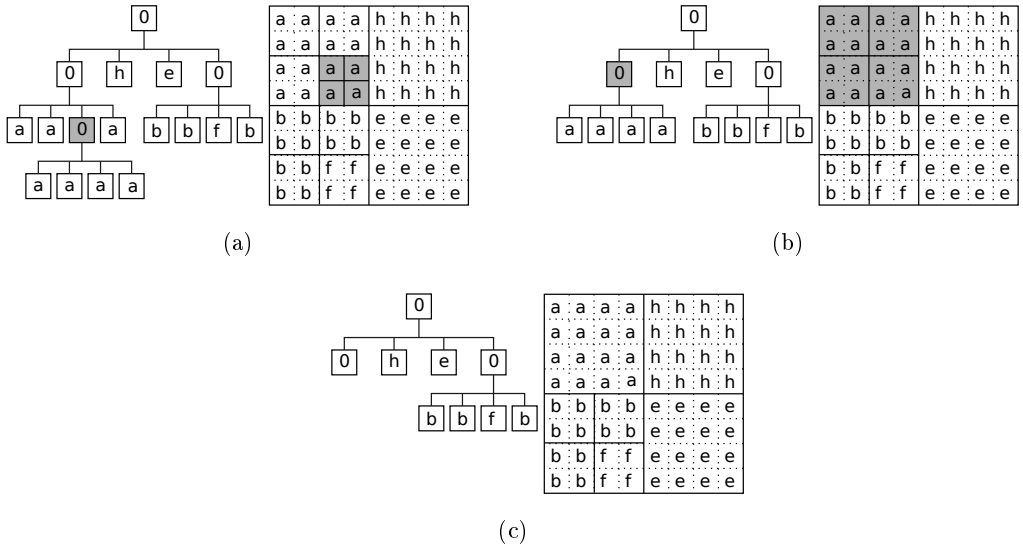


Figure 3.13 – Application of merge

another gray node that satisfies the merge condition on eq. (3.47). The merge is applied and gives the tree on fig. 3.13c. On this figure, no node satisfies eq. (3.47), no merge is then performed. Figure 3.13 shows that the operation of merge shortens the tree upwards by pruning nodes that satisfies eq. (3.47). This makes the tree more compact by decreasing the number of nodes.

3.5.5 Discussion

Benefits Integer occupancy grids can be stored within an array or a 2^d-tree. For a tree, occupancy indexes of adjacent cells are stored within a leaf if and only if they are equal. This condition has two advantages.

First, it makes 2^d-trees more compact than arrays for storing integer occupancy grids. While an array has to have the same number of array elements as cells, the number of nodes within a 2^d-tree is potentially less than the number of cells. In fact, due to the spatial disposition of obstacles on a physical world, likely occupied cells tends to be adjacent. The same for likely empty ones and also for the unknown cells. Consequently, adjacent cells potentially have similar, even equal, occupancy indexes. An experimental study about the compactness of 2^d-trees will be presented in the next chapter.

Second, the compaction offered by 2^d-trees is lossless. Lossless means that the data structure for storing integer occupancy grids does not alter the occupancy indexes per cells. The occupancy index of a cell is the same whether the integer occupancy grid is stored within a 2^d-tree or within an array. Such a result cannot be obtained by the condition of merge proposed in the literature (see Section 2.5.2 (page 45)).

Limitations While 2^d-trees are compact, maintaining the data structure is expensive in term of execution time. On an array, updating the occupancy index of a cell is straight forward after incoming measurements. On a 2^d-tree, the update must perform at least the operation of split. The later subdivides the leaves that cover the updated cell until the maximum depth is reach. Such operation is time consuming when it is repeated for a high number of cells. After the split, the merge becomes also too expensive in term of execution time when it is applied frequently. However, the operation of merge improves the compactness of the tree by pruning some nodes out of the tree.

After that, the tree structure also requires additional memory load. For instance, the pointer based implementation of 2^d-trees need at least two pointers: a pointer that points to the children and another one that points to the parent ([Samet 1990]). On the opposite, array structure does not require additional memory load. Consequently, depending on the size of the grid and on the environment, an array may consume less memory than a 2^d-tree. Nevertheless, the literature reports that 2^d-trees are less memory consuming when mapping a large environment with a high-resolution and high-dimensional grid ([Payeur 1997, Kraetzschmar 2004, Hornung 2013]).

3.6 Summary

This chapter presented the integer occupancy grid paradigm.

- The formal definition and properties of set of probabilities were introduced. Examples of sets of probabilities have been proposed and demonstrated.
- The recursive set of probabilities was highlighted since it has important properties that have enable to fuse multiple sensors through simple addition of integers.
- The definition and properties of Occupancy Indexes and Integer Occupancy Grids were formalized. The fusion of Occupancy Indexes is equivalent to a sum of integers.
- The numerical error involved by the computation of Integer Occupancy Grids was specified. The step of quantization introduces explicitly a known, bounded and user-adjustable numerical error. The latter can be any real-number between 0 and $1/2$.
- The lossless compaction of Integer Occupancy Grids by using 2^d-trees was studied.

Chapter 4

APPLICATION OF INTEGER OCCUPANCY GRIDS FOR AUTOMOTIVE MULTI-SENSOR FUSION

The previous chapter established the theoretical foundation of integer occupancy grids. The computation of integer occupancy grids is composed of three steps: the computation of ISMs, the quantization and the fusion. While the two last steps were presented in the previous chapter, the computation of ISMs was not tackled. Section 2.3.1 (page 28) presented the Bayesian approach for computing the ISM of a cell. This approach establishes a relation between measurement uncertainties, the size of cells and the value of ISM. It suffers however form an exponential complexity.

This chapter proposes new approaches for computing ISMs. Experimental analysis and discussion about these approaches will be presented. After that, this chapter will describe the SW/HW integration of the integer occupancy grid framework. The SW/HW integration constitutes a multi-sensor fusion (MSF) module that fuses measurements from range sensors mounted on a car and produces an environment model of the driving environment.

Sensor measurements are produced periodically. According to the initial constraints evoked in Section 1.3 (page 10), the fusion must be processed in real-time with respect to sensor period. Second, its HW/SW integration must be realized on a low-cost and low-power processing platform. Third, the HW/SW integration must be safe. It must consider sensor uncertainties, numerical errors and determinism of computation.

To develop such a module, the SW/HW integration of the integer occupancy grid framework is realized on a low-cost and low-power processing platform. The module fuses periodical measurements produced by four state-of-the-art LIDARs mounted on a prototype car. At each period of measurement, a 2D integer occupancy grid combines the measurement from the four LIDARs. The integer occupancy grid models the driving environment surrounding the car.

This chapter is organized as follows. Section 4.1 presents the experimental setup including the prototype vehicle, the LIDARs and the processing platform. After that, Section 4.2 proposes and discusses the new approaches for computing ISMs. Next, the HW/SW integration of the integer occupancy grid framework will be described in Section 4.3, where discussions about the experimental results will be reported.

4.1 Experimental setup

Experiments realized in this thesis were conducted on a prototype car: a ZOE from Renault (see fig. 4.1). The car belong to the Institut de Recherche Technologique (IRT) NanoElec ([IRT NanoElec]). The IRT NanoElec is a French technological institute. Its role is to diffuse innovations in Information and Communication technologies towards companies and enterprises. The prototype car is upgraded by sensors and additional computing hardware. It serves as a platform for development and test of ADAS and self-driving functions. The platform aims to accelerate technological transfer from research to industry.

(a) Three front LIDARs ibeo LUX (b) One rear LIDAR ibeo LUX

Figure 4.1 – The prototype vehicle with its four LIDARs on the bumpers

4.1.1 The prototype car

As shown on fig. 4.1, the prototype car is equipped by four ibeo LUX LIDARs: three on the front bumper (fig. 4.1a) and a forth on the rear bumper (fig. 4.1b). Other sensors such as cameras, a rotating Velodyne LIDAR, a GPS and an inertial measurement unit are also installed on the car. However, the experiments in the scope of this thesis focus only on the four ibeo LIDARs.

Additional computing facilities are installed in the trunk as shown in fig. 4.2. A workstation with an Intel CPU and GeForce GPU from NVIDIA processes sensor

measurements and execute applications for intelligent vehicle functions. Figure 4.2 shows that the workstation is powered by a dedicated battery. The trunk also contains networking boxes and weiring for transferring data between sensors and the workstation. A linux-based operating system runs on the workstation. It executes the middleware Robot Operating System (ROS). The latter provides a set of libraries and tools for supporting and accelerating the development of robotics applications.

Figure 4.2 – Computing facilities on the ZOE

Even if these computing facilities are ready to use, they do not correspond to the criteria of cost, power budget and reliability found in the world of automotive. The CPU and the GPU in the trunk are not designed for automotive use cases. Despite their computing performance, these hardware consume hundreds of watts of power and do not support extreme conditions in a car (e.g. high temperature, humidity, physical shocks, *etc*). Hence, experiments in this thesis are implemented on an embedded hardware that is designed for automotive applications.

4.1.2 The embedded computing hardware

Experiments presented in this chapter are implemented on a SABRE Lite development board [element14]. This hardware platform is based on the $i.MX6$, an application processor from Freescale based on a quad-core ARM Cortex-A9. The processor runs at 1 GHz. The platform has a RAM DDR3 of 1 GByte, having 64-bit wide and running at 532 MHz. It has an SD card interface and runs an operating system based on an Ubuntu distribution.

The $i.MX6Q$ processor is specifically designed for industrial and automotive applications [Freescale 2015]. It combines significant computing performance and low-power constraints, which makes it a good candidate for implementing the integer occupancy grid framework. A single-core of the $i.MX6Q$ processor consumes less than 1 W of power [Freescale 2015]. For a comparative purpose, the Intel CPU within the trunk of the prototype car consumes up to 40 W [Dargie 2015, Abou-Of 2016] and the NVIDIA GPU consumes more than 200 W [Stroia 2015].

Remark The SABRE Lite platform is designed to enable rapid development of automotive multimedia applications. It is not yet a hardware certified for safety critical applications. It allows though to test integer occupancy grids on an embedded CPU dedicated for automotive.

4.1.3 Experimental data

During this thesis, the SABRE Lite board was not yet installed in the prototype car. For processing data from the ibeo LIDARs, experiments were realized as follows.

The prototype car was driven in real traffic in down town of Grenoble, in France, and also on highways. In the meantime, measurements from the four ibeo LIDARs were saved into files by the ROS middleware. After that, the files are copied on a SD card. The latter is inserted on the SABRE Lite in order to computed integer occupancy grids from the saved LIDAR measurements.

4.1.4 The ibeo LUX LIDAR

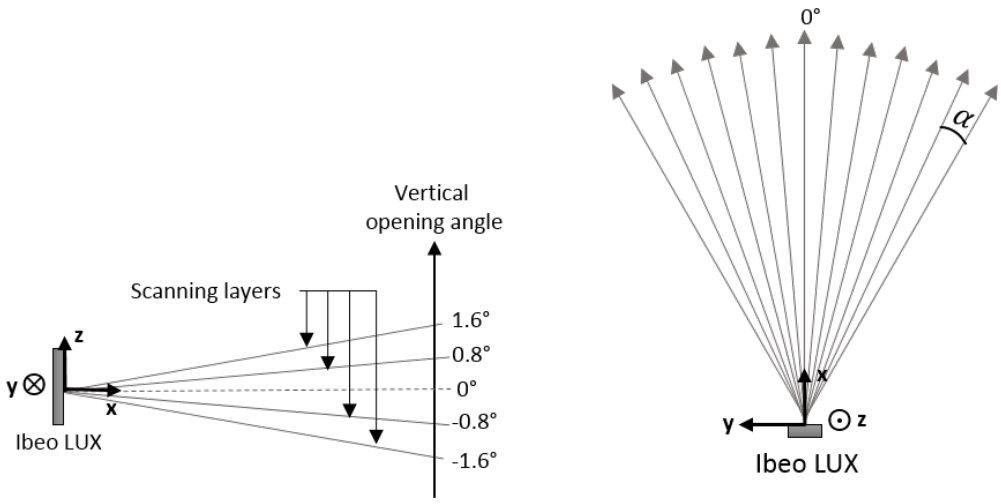
The ibeo LUX LIDAR is a laser scanner that serves for scanning a physical environment within a field-of-view [Ibeo 2010, Ibeo 2013]. It targets the automotive domain and is designed to meet the constraints of safety, robustness and power budget found in the domain. An ibeo LUX LIDAR measures ranges to obstacles, within several directions relative to the sensor. The measuring process works as follows.

The sensor emits laser beams towards several directions, receives echoes on a receiver, and then computes ranges based on the time-of-flight of the beams. A laser beam can measure a range up to 200 m. Under 50 m, at least 10% of the emitted energy is received back to the sensor. Under this range, a measurement has an accuracy of 10 cm, regardless of the distance of the observed obstacle [Ibeo 2013].

Direction of beams The direction of a laser beam is defined by two angles: the *elevation* and the *azimuth*. Laser beams are emitted within four degrees of elevation angles as shown on fig. 4.3a. The group of laser beams having the same elevation angle is called *scan layer*. An ibeo LUX device produces in total four scan layers, one per elevation angle. Within a scan layer, each laser beam is emitted within a known azimuth angle. The azimuths of two successive laser beams are separated by an angular step of $\alpha = 0.5°$ (fig. 4.3b).

Scan points The range measurement, the elevation of the scan layer and the azimuth angle of a laser beam form the spherical coordinates of a point within a local frame of reference attached to the LIDAR device. Such a point is called a *scan point*. It spatially estimates the location where the laser beam has hit an obstacle. A complete scan of the surrounding by an ibeo LUX provides up to 800 scan points. Complete scans are produced at a rate of 25 Hz.

Figure 4.4 presents an example of scan points the LIDAR on the center of the front bumper of the prototype car. The scan points are projected on a two-



(a) Vertical angle of the four scan layers

(b) Top view of the directions of beams within a scan layer

Figure 4.3 – The scanning layers of an ibeo LUX device

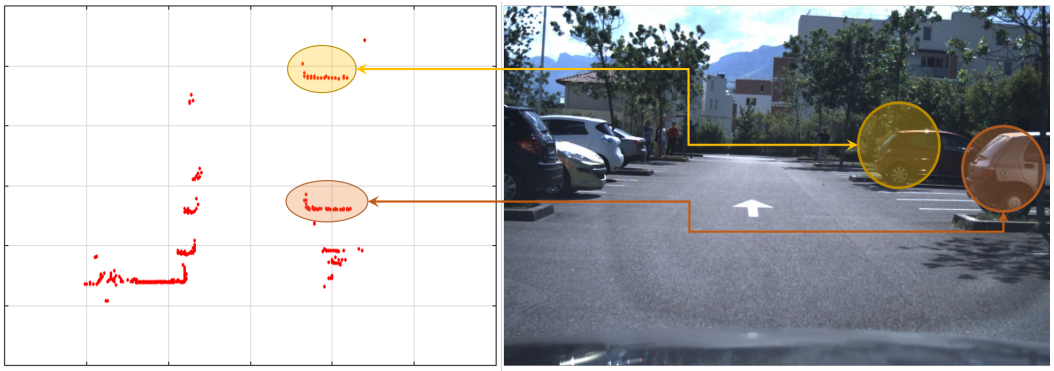


Figure 4.4 – Scan points of an ibeo LUX LIDAR mounted on the front bumper of the prototype car (image courtesy of INRIA Rône-Alpes)

dimensional plan parallel to the prototype car (see the image on the left). The physical scenario is shown on the right of the figure. The two encircled cars are situated in the front-right of the prototype vehicle. Both cars are hit by laser beams. The corresponding scan points are also encircled on the image on the left.

4.2 Computation of Inverse Sensor Model

A scan point of a LIDAR is considered as an independent measurement. It is generated by a beam directed within known azimuth and elevation. Two scan points are

considered as independent one another. Such assumption is generally admitted for LIDARs in the literature ([Thrun 2005, Homm 2010, Einhorn 2011, Adarve 2012]).

The objective of this chapter is to fuse the scan points produced by the four LIDARs into a 2D integer occupancy grid. As presented in the previous chapter, the computation of integer occupancy grids require a step of computation of ISMs from individual sensor measurements. Hence, this section will describe the computation of ISMs from a unique scan point.

For a 2D grid, the ISM of a cell given an individual measurement is computed in two steps (Section 2.3 (page 27)). First, a 1D occupancy grid local to the measurement is built. Second, a range-mapping algorithm is applied for retrieving the ISMs of 2D cells from the local occupancy grid.

Section 2.3.1 (page 28) highlighted the advantages of the Bayesian approach for building the local grid. This approach suffers though from an exponential complexity with respect to the number of cells. This section will present a new theorem that extends the Bayesian approach and that allows to build the local grid within a linear complexity.

Besides, Section 2.3.2 (page 34) explained the advantages of traversal algorithms for performing range-mapping. It showed the need for designing a traversal algorithm that works exclusively on integers. Such algorithm will be also presented and discussed in the present section.

4.2.1 Building the local occupancy grid

Like the approaches reviewed in Section 2.3 (page 27), the approach proposed here is applicable only for single-target sensor.

4.2.1.1 Inverse sensor model of a single-target

Consider the situation on fig. 4.5 where a single-target sensor observes a physical environment through a line-of-sight. The latter is subdivided into a 1D grid composed of cells $c_i, i = 1, \ldots, N$. A cell c_i is located at a distance d_i from the sensor.

The sensor provides a scalar measurement z. The value of the measurement depends on the distance d between the sensor and the sensed obstacle. The process of measurement is modeled by the sensor model $p(z|d)$. We propose the following theorem for computing the ISM $P(o_i|z)$ of a cell c_i of the local grid.

Theorem 6. Inverse Sensor Model of a single-target sensor

Under the non-informative prior, the ISM of the cell c_i given a measurement z from a single-target sensor is:

$$P(o_i|z) = \begin{cases} \alpha^{-1} \cdot p(z|d_1) & \text{if } i = 1 \\ \alpha^{-1} \cdot \left(\sum_{h=1}^{i-1} \left[\frac{p(z|d_h)}{2^h} \right] + \frac{p(z|d_i)}{2^{i-1}} \right) & \text{otherwise} \end{cases}$$

$$\text{where } \alpha = \sum_{h=1}^{N} \left[\frac{p(z|d_h)}{2^{h-1}} \right]$$

Proof ■ This theorem is derived from the Bayesian approach (Section 2.3.1.1 (page 29)). The mathematical derivation is detailed in Section B.1 (page 129). ■

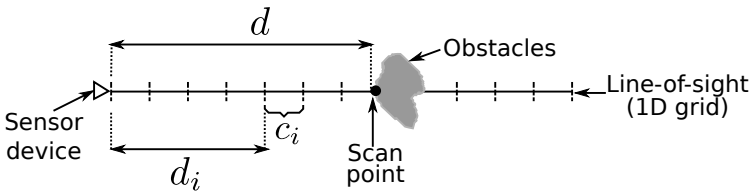


Figure 4.5 – The 1D grid along a line-of-sight of a single-target sensor

Computing $P(o_i|z)$ requires now a sum of at most N number of terms. Consequently, Theorem 6 (page 91) enables to calculate an ISM with a linear complexity $O(N)$ with respect to the number of cells N. Theorem 6 (page 91) is derived from the Bayesian computation of ISMs. In the followings, let us apply this theorem on a scan point from a LIDAR.

4.2.1.2 Application to a LIDAR scan point

For a LIDAR scan point, the direction of a beam is determined by both elevation and an azimuth. Only the range measured by the beam varies as a function of the distance between the sensor and the obstacle along the beam's direction. Hence, for a scan point, z denotes the range measurement. The range measurement has a precision of 10 cm under a maximal distance of 50 m (Section 4.1.4 (page 88)). The process of measurement is therefore modeled by a Gaussian distribution with a standard deviation $\sigma = 10cm$.

Let d denote the ground truth distance to the nearest obstacle. The sensor model becomes:

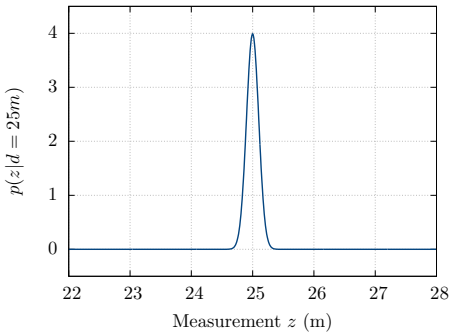
$$p(z|d) = \frac{1}{\sqrt{2\pi\sigma^2}} exp\left\{ -\frac{(z-d)^2}{\sigma^2} \right\} \tag{4.1}$$

Figure 4.6a plots the sensor model when the nearest obstacle is situated at $d = 25m$ from the sensor. The sensor model is quasi null everywhere except in the vicinity of

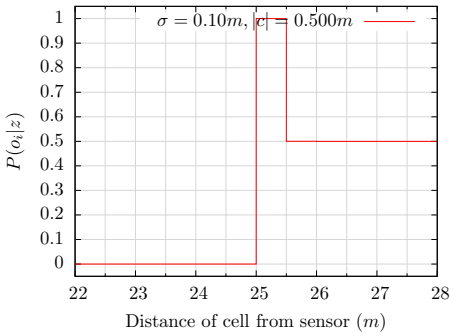
d. That means, there are a great chance that the measurement z returned by the sensor is situated in the vicinity of d.

Let us now build the local occupancy grid given a scan point. To be able to apply Theorem 6 (page 91), the distance of each cell from the sensor must be known. This distance is derived from the size of cells along the grid. Therefore, a question arises: how to chose the size of a cell of the local grid?

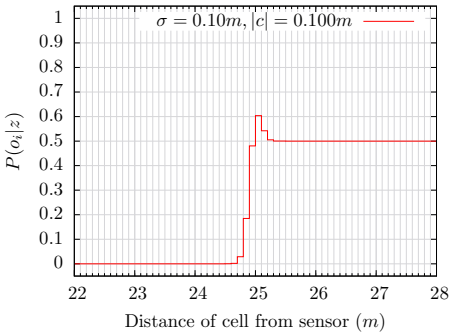
To answer to this question, several cell sizes have been tested. The profile of the ISMs for three different cell sizes are plotted on figures 4.6b to 4.6d.



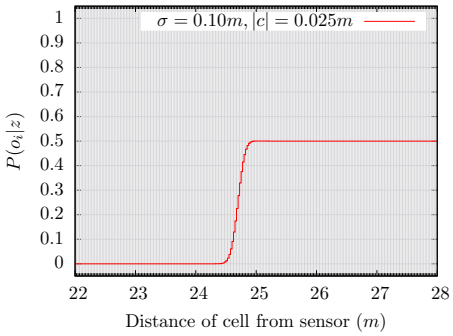
(a) Sensor model for $d = 25m$



(b) ISM for $z = 25m$ with cells of $0.5m$



(c) ISM for $z = 25m$ with cells of $0.1m$



(d) ISM for $z = 25m$ with cells of $0.025m$

Figure 4.6 – The sensor model and the profiles of ISM given a measurement $z = 25m$

Profile of ISMs Figures 4.6b to 4.6d shows a common profile among the three different cell sizes. ISMs are close to zero between the sensor and the measurement z. Cells in this region are likely empty. Around the measurement, ISMs increase until reaching a maximum value. An ISM greater than $1/2$ signifies that the corresponding cells are likely occupied. ISMs decrease beyond the measurement and get stable at $1/2$. The occupancy state of cells beyond the measurement are unknown.

Cell size vs. numerical value of ISMs Figure 4.6 shows that ISMs vary as a function of cell size. In fact, the sensor model and the distance of each cell from the sensor intervene within the formula of ISM in Theorem 6 (page 91). Consequently, this theorem maintains a relation between the measurement, the sensor model, the cell size and the numerical value of ISMs. This relation is actually a property of the Bayesian approach as explained in Section 2.3.1 (page 28).

Choosing the cell size On fig. 4.6b, the maximum of ISM is closed to 1. The cell size is 5× larger than the precision of the sensor. When the cell size is decreased, the maximum of ISM also decreases. On fig. 4.6d where the cell size is a quarter of the precision of the sensor, the maximum of ISM does not even overtake 1/2. That means, the cell size is too small for the sensor's precision. Beyond its precision, the sensor cannot estimate with a high confidence that a cell is occupied. Thus, to be able to estimate likely occupied cells, the cell size is chosen such that it does not overtake the sensor's precision.

Comparison with the Bayesian approach Table 4.1 (page 93) compares Theorem 6 (page 91) with the Bayesian approach. Since Theorem 6 (page 91) is derived from the Bayesian approach, both methods are mathematically equivalent. The difference resides in their respective complexities. Notice that being based on the Bayesian approach, Theorem 6 (page 91) can also be generalized to any single-target sensor, but not only for LIDARs. Both methods compute the ISM from the sensor model.

	Bayesian approach (state-of-the-art)	**Proposed approach** (Theorem 6 (page 91))
Sensor model	Yes	Yes
Grid subdivision	Yes	Yes
Safe	Yes	Yes
Complexity	$O(2^{N-1})$	$O(N)$

Table 4.1 – Comparison of the Bayesian approach and the proposed approach for computing the ISM

4.2.2 Traversal algorithm based on integer arithmetic

After building a local occupancy grid given a LIDAR scan point, the 1D grid is mapped on the 2D grid in order to compute the ISMs of 2D cells. Cells that are not traversed by the local grid are outside of the line-of-sight of the laser beam that has generated the scan point. Their occupancy state is then unknown for the laser beam. Their ISM is set to 1/2. The ISM of the traversed cells are however derived from the local occupancy grid. For finding out the cells traversed by the local grid, the following algorithm is proposed.

4.2.2.1 The continuous traversal algorithm

Consider the local occupancy grid on fig. 4.7. The grid has two endpoints: a start point S where the sensor is located, and an end point T located at the extremity of the grid. The line segment $\overline{ST}$ traverses multiple cells of the 2D grid. The latter is called **global grid** since it is not attached to a specific sensor. The objective consists to design a traversal algorithm that finds out all 2D cells traversed by the line segment $\overline{ST}$.

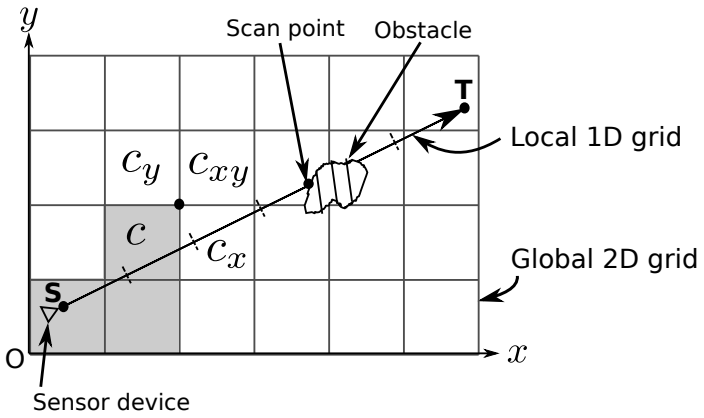


Figure 4.7 – The traversal algorithm

The *continuous traversal algorithm* presented on Algorithm 3 (page 95) is able to find out the traversed cells. This traversal algorithm has the advantage of being exact: no cells traversed by the line segment $\overline{ST}$ is missed by the algorithm. The algorithm works within a spatial frame of reference (O, x, y), as depicted on fig. 4.7. The algorithm takes as inputs the coordinates of point S and point T, the coordinates of the cells that contain both points, and the length of the sides of 2D cells. A cell is located by the coordinates of a point that belongs to the cell. For instance, the coordinates of the south-west corner can be chosen as the coordinates of a cell.

The algorithm finds out iteratively the traversed cells starting from point S and going toward point T. Figure 4.7 actually shows an iteration of the algorithm where the gray cells are known to be traversed. Cell c is the last cell found as traversed by the segment $\overline{ST}$. Let c' denote the next traversed cell after c. Cell c' can be either c_x or c_y or c_{xy}. To identify the right one, the sign of an error variable e is checked:

$$c' = \begin{cases} c_{xy} & \text{if } e = 0 \\ c_x & \text{if } e > 0 \\ c_y & \text{otherwise} \end{cases} \tag{4.2}$$

The mathematical derivation of Algorithm 3 (page 95) is detailed in Section B.2 (page 132). The demonstrations explain the mathematical principle behind the error variable and detail how the algorithm was designed.

Algorithm 3 Continuous Traversal Algorithm

```
1: function TRAVERSE(Point (x_S, y_S), Point (x_T, y_T), Point (x_0, y_0), Point
   (x_1, y_1), Real β)
2:     ▷ (x_S, y_S): coordinates of point S
3:     ▷ (x_T, y_T): coordinates of point T
4:     ▷ (x_0, y_0): location of the cell containing the point S
5:     ▷ (x_1, y_1): location of the cell containing the point T
6:     ▷ β: length of a side of a squared cell
7:     Point (x, y)                    ▷ Location of the currently traversed cell c
8:     Real e                                                  ▷ Error variable
9:     Real Δx ← x_T − x_S
10:    Real Δy ← y_T − y_S
11:    (x, y) ← (x_0, y_0)
12:    e ← (y + β − y_S) × Δx − (x + β − x_S) × Δy
13:    while (x, y) ≠ (x_1, y_1) do
14:        if e = 0 then
15:            (x, y) ← (x + β, y + β)                          ▷ c_xy is traversed
16:            e ← e + β × (Δx − Δy)
17:        else if e > 0 then
18:            x ← x + β
19:            e ← e − β × Δy                                   ▷ c_x is traversed
20:        else
21:            y ← y + β                                        ▷ c_y is traversed
22:            e ← e + β × Δx
23:        end if
24:        ▷ The cell traversed at the current iteration is located at (x, y)
25:        SampleISM(x, y)
26:    end while
27: end function
```

Algorithm 3 (page 95) works as follows. The variables x and y at line 7 store the coordinates of the last cell discovered by the algorithm. Both variables are initialized by the coordinates of the cell containing the point S at line 11. After that, the error variable is initialized at line 12. In the main loop, the sign of e is analyzed to determine the next traversed cell. The coordinate of the traversed cell is then stored within the variables x and y and the value of the error variable is also updated. Once the new traversed cell is discovered, the function $SampleISM()$ is called (line 25). The loop iterates until the algorithm reaches the cell that contains the point T.

The function $SampleISM()$ computes the ISM of the traversed cell. This function samples the local occupancy gird at the level of the traversed cell. The process of sampling is depicted on fig. 4.8. The figure assumes that cell c is the newly discovered cell. Let c^* denotes the 1D cell of the local grid that hits the cell c. Then,

the ISM of c^* is affected to cell c.

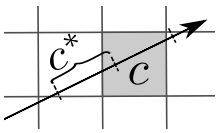


Figure 4.8 – Sampling the ISM of a 1D cell c^*

4.2.2.2 Discrete traversal algorithm

By working within the continuous frame of reference (O, x, y), the continuous traversal algorithm has to deal with real number operations. This requires then a HW/SW support of real numbers. The good news is that all operations involve only additions and multiplication of coordinates of points and cells.

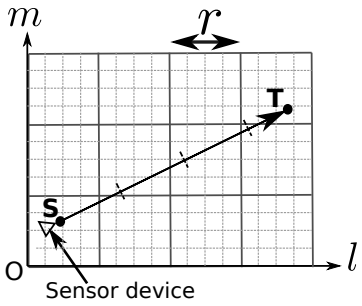


Figure 4.9 – The discrete frame of reference (O, l, m) and the resolution r

Let us replace the continuous frame (O, x, y) by a discrete frame [1] (O, l, m). The latter is shown on fig. 4.9. Coordinates in (O, x, y) belongs to $\mathbb{R}^2$ while those in (O, l, m) lays within $\mathbb{Z}^2$. Consider a point within the plane. Let (x, y) denotes its coordinates within the continuous frame, and (l, m) within the discrete frame. The relation between (x, y) and (l, m) is:

$$\forall (x, y) \in \mathbb{R}^2, x = l \cdot \delta \quad \text{and} \quad y = m \cdot \delta \tag{4.3}$$

The symbol δ denotes a discretization-step which measures the spatial precision of the discrete frame.

The value of δ is measured in meter (m). It is chosen such that the length β of sides of 2D cells becomes a multiple of δ. The value of δ is determined by the traversal resolution defined as follows.

[1] The application of discrete frame for performing range mapping was published in [Rakotovao 2016b].

Definition 4.2.1. *Let β denotes the length of sides of 2D cells. The* ***traversal resolution*** *denotes a positive integer r. It allows to determine the spatial precision δ of the discrete frame as follows:*

$$\delta = \frac{\beta}{r}$$

Hence, given the size of cells, the value of the traversal resolution is first determined in order to know the value of δ. The traversal resolution determines how precise is the discrete frame of reference while the value of δ determine exactly how long is the precision. For instance, if cells measure 10 cm-by-10 cm, a traversal resolution of 10 means that the discrete frame of reference is precise at 1 cm. A higher traversal resolution of 100 makes increases the precision at 1 mm.

The above definition implies that the length of the side of cell within the discrete frame is equal to r. By replacing x with m, y with l and β with r, the continuous traversal algorithm becomes discrete. It engenders the discrete traversal algorithm presented on Algorithm 4 (page 98). The latter follows exactly the same principles as the continuous traversal algorithm, except that it manipulates exclusively coordinates in integers.

Remark Algorithm 4 (page 98) works only for a grid traversal oriented towards the North-East. The general discrete algorithm that works in all direction is described in Section B.3 (page 135).

4.2.2.3 Analysis of the discrete traversal algorithm on a LIDAR

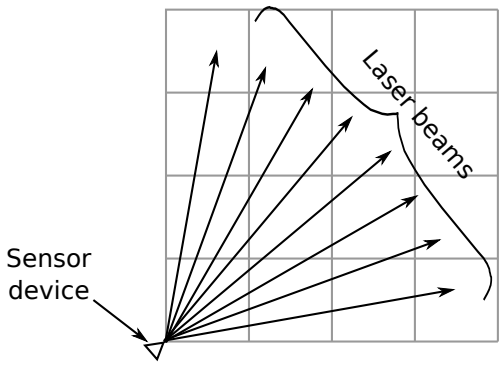


Figure 4.10 – A LIDAR emitting laser beams towards cells of a 2D grid

Using a discrete frame allowed to design a traversal algorithm that process only integer arithmetic. However with the discrete frame, does the discrete algorithm

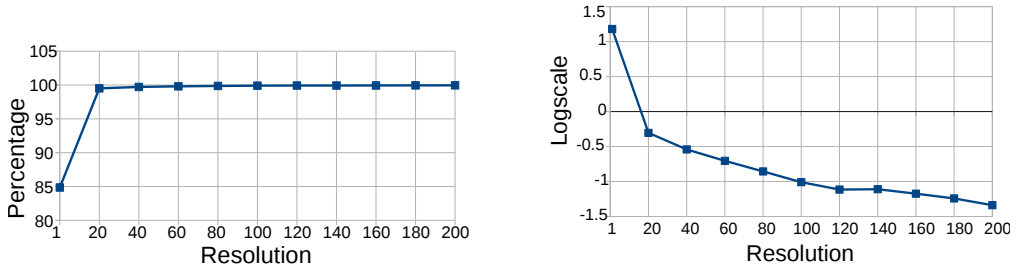
Algorithm 4 Discrete Traversal Algorithm

function TRAVERSE(Point (l_S, m_S), Point (l_T, m_T), Point (l_0, m_0), Point (l_1, m_1), Integer r)
  ▷ (l_S, m_S): coordinates of point S
  ▷ (l_T, m_T): coordinates of point T
  ▷ (l_0, m_0): location of the cell containing the point S
  ▷ (l_1, m_1): location of the cell containing the point T
  ▷ r: resolution of the discrete frame of reference
  $\Delta l \leftarrow l_T - l_S$
  $\Delta m \leftarrow m_T - m_S$
  Point (l, m) ▷ Location of the currently traversed cell c
  Integer e ▷ Error variable
  $(l, m) \leftarrow (l_0, m_0)$
  $e \leftarrow (l + \beta - l_S) \times \Delta m - (m + \beta - m_S) \times \Delta l$
  while $(l, m) \neq (l_1, m_1)$ **do**
    if $e = 0$ **then**
      $(l, m) \leftarrow (l + r, m + r)$ ▷ c_{xy} is traversed
      $e \leftarrow e + r \times (\Delta l - \Delta m)$
    else if $e > 0$ **then**
      $l \leftarrow l + r$
      $e \leftarrow e - r \times \Delta m$ ▷ c_x is traversed
    else
      $m \leftarrow m + r$ ▷ c_y is traversed
      $e \leftarrow e + r \times \Delta l$
    end if
    ▷ The cell traversed at the current iteration is located at (l, m)
    $SampleISM(l, m)$
  end while
end function

find out exactly all cells that are discovered by the continuous one? And what is the impact of the introduction of the discrete frame on the execution time?

An experiment on a fictive LIDAR is conducted to answer to the above questions. Both continuous and discrete traversal algorithms are applied for finding out cells traversed by the laser beams of the fictive LIDAR. Both algorithms are implemented on the SABRE Lite platform. The continuous algorithm is implemented with floating-points.

The LIDAR is placed at the Sout-West corner of a 2D grid as illustrated on fig. 4.10. Laser beams return a range of 50 m. They are separated with an angular step of 0.5°. These parameters are chosen to match with the characteristics of Ibeo LUX LIDARs. The grid measures 50 m-by-50 m with cells of 10 cm-by-10 cm.



(a) (b)

Figure 4.11 – Comparison of the accuracy of the discrete algorithm to that of the continuous algorithm

Accuracy To measure the accuracy of the discrete traversal algorithm, we count how many percent of the cells traversed by the continuous algorithm are also traversed by the discrete one. An accuracy of 100% would mean that the discrete algorithm has discovered all cells traversed by the continuous one. To evaluate the impact of the discrete frame on the accuracy, several values of the traversal resolution have been tested. The evolution of the accuracy as a function of the traversal resolution is shown on fig. 4.11a.

The traversal algorithm is accurate at 85% when the traversal resolution is equal to 1. A resolution of 1 means that the precision of discrete frame is equal to the length of a cell side ($\delta = \beta$). With a traversal resolution greater than 100, the discrete algorithm is accurate at more than 99.9%. The higher is the traversal resolution, the more accurate is the discrete traversal algorithm. For a deeper analysis, consider now the percentage of misses, that means 100% minus the accuracy. The logarithm of the percentage of misses is plotted on fig. 4.11b. The figure shows that when the resolution increases, the log scale of the percentage of misses continues to decrease. The percentage of misses tends then towards 0.

Execution time & speedup Let us now study the execution time of the discrete algorithm compared to the continuous one. The evolution of the execution time as a function of the traversal resolution is shown on Table 4.2 (page 100). The discrete traversal algorithm is 2.8× faster than the continuous traversal algorithm. This shows that the SABRE Lite platforms have a better support of integers compared to floating-points. Moreover, the traversal resolution does not influence the execution time. That means, the traversal resolution can be chosen as higher as possible.

Choosing the traversal resolution In practice, range sensors have a limited spatial precision of the order of centimeters. The traversal resolution is chosen as a function of the precision of range sensors. The discrete algorithm can work with a precision that is better than the precision of sensors. Having a "too high" traversal resolution is however not required. For instance, if 2D cells measure 10 cm-by-10 cm,

Algorithm	Cont.	Disc. ($r = 20$)	Disc. ($r = 100$)	Disc. ($r = 200$)
Exec. time	20 ms	7 ms	7 ms	7 ms
Speedup	1	2.8	2.8	2.8
Correctness	100%	99.50%	99.90%	99.95%

Table 4.2 – Performance of the discrete traversal algorithm compared to the continuous one

a traversal resolution of 100 means that the discrete frame has a spatial precision of 1 mm. Such a resolution is enough for range sensors which measurements are precise at an order of centimeters.

Once the methods for computing ISMs are now available, let us move towards the SW/HW integration of the integer occupancy grid framework in the next section.

4.3 HW/SW integration of multi-sensor Integer Occupancy Grids

The LIDARs mounted on the prototype car produce scan points within a period of 25 Hz. The objective consist in fusing these points into a 2D integer occupancy grid at each period. The grid measures 102.4 m-by-102.4 m, with cells of 10 cm-by-10 cm. The number of cells is 1024×1024, which is more than 1 Million. As shown on fig. 4.12, the prototype car is located at the center of the grid.

When the car moves, the grid is attached to the vehicle and follows its motion. The grid is placed at 20 cm from the ground. It is parallel to the chassis of prototype car. This section presents the implementation of the integer occupancy grid framework on the SABRE Lite platform. Experimental results will be also presented and analyzed. The SW/HW integration is tested on the experimental data described in Section 4.1.3 (page 88).

4.3.1 Implementation of integer occupancy grids

For fusing measurements from the four LIDARs, the integer occupancy grid framework is implemented as follows. Regardless of the data structure used for storing the grid, occupancy indexes of 2D cells are set to 0 at the beginning of a scan period. Occupancy index of 0 is equivalent to an occupancy probability of $1/2$. That means, the occupancy states of cells are initialized to unknown. After that, the scan points produced within a period are sequentially integrated into the 2D integer occupancy grid for performing the fusion. Let us explain the integration of a single scan point.

Consider the scan point on fig. 4.13. It is generated by a laser beam which line-of-sight is subdivided into the local grid. The latter measures 50 m. This length corresponds to the maximal range of a laser beam of the LIDAR. By taking into

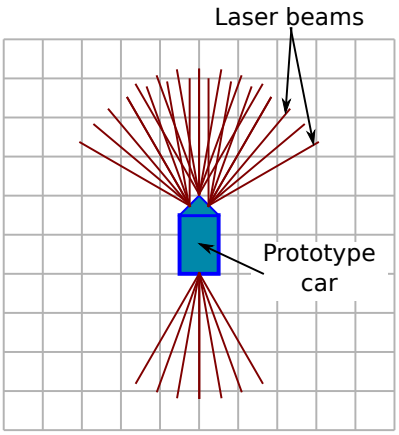


Figure 4.12 – Top view of the prototype car with the 2D grid and the laser beams from the four LIDARs (three on the front bumper and one on the back bumper)

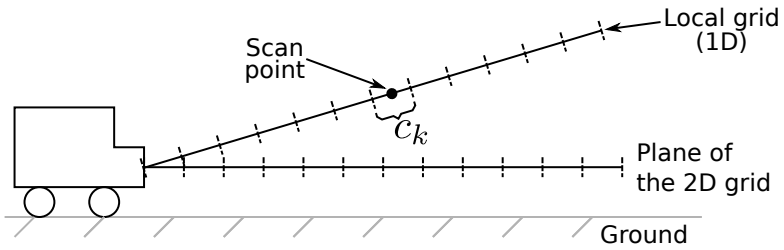


Figure 4.13 – A scan point with its corresponding local grid

account the discussions in Section 4.2.1.2 (page 91), the cell size of the local grid is set to be equal to the precision of the sensors. Therefore, 1D cells of the local grid have 10 cm of length.

To integrate a scan point into the 2D integer occupancy grid, two steps are required. First, the local integer occupancy grid is computed. Second, the local integer occupancy grid is projected onto the 2D grid.

4.3.1.1 Computation of the local integer occupancy grid

The local integer occupancy grid is defined over the local grid that corresponds to the scan point. It is built by finding out first the index k of the cell where the scan point is located. After that, the occupancy indexes of cells over the local grid are directly given by the lookup table on Table 4.3 (page 102).

The look up table is filled once before the integration of any sensor measurement and cached in the memory for further utilisation. The occupancy indexes within the table are computed by quantizing ISMs. Both blurring policy and nearest policy are experienced during the quantization. In practice, only a single quantization

Cell index	$\leq k-3$	$k-2$	$k-1$	k	$k+1$	$\geq k+2$
ISM	0.05	0.18	0.48	0.6	0.54	0.5
Occupancy index (Blurring policy)	-14	-7	0	2	0	0
Occupancy index (Nearest policy)	-15	-7	0	2	1	0

Table 4.3 – Lookup table for accelerating the computation of local 1D integer occupancy grids

policy is applied but we have implemented both for experimental purposes. ISMs are computed thanks to Theorem 6 (page 91).

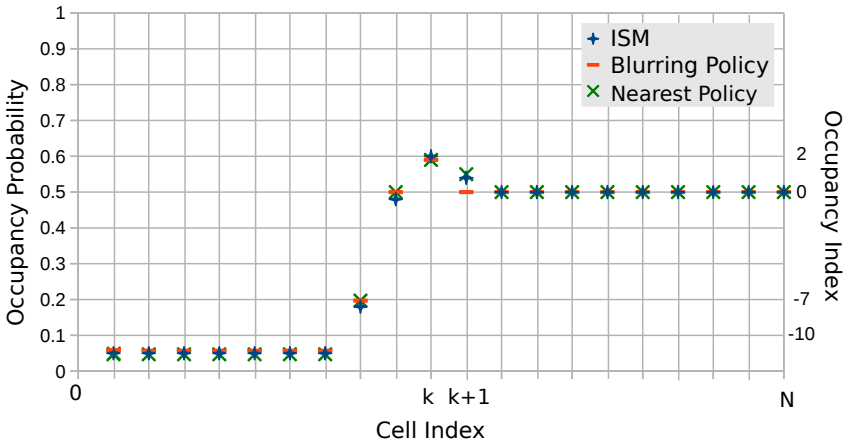
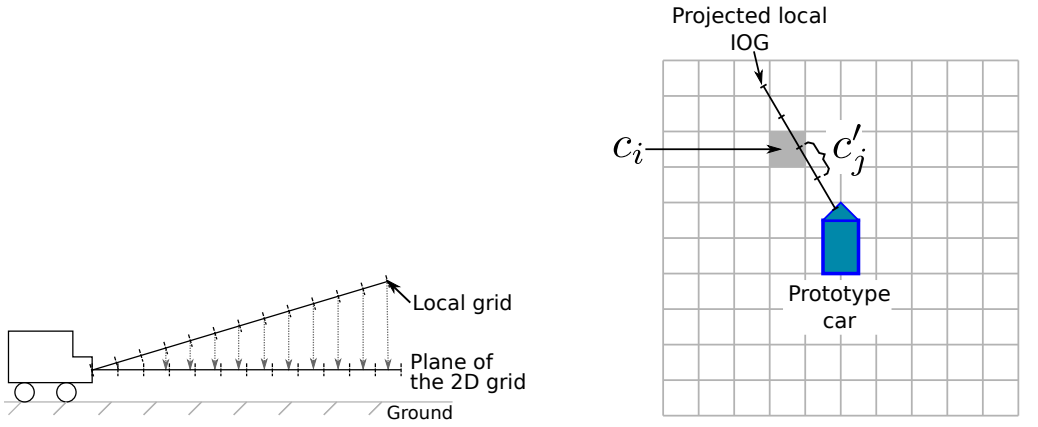


Figure 4.14 – The ISMs over the local 1D grid and its quantization by both blurring policy and nearest policy

The profiles of the ISMs and their quantization are plotted on fig. 4.14. The blurring policy approximates ISMs towards 1/2. The nearest policy approximates an ISM with the element of the recursive set that is nearest to the numerical value of the ISM. The figure also shows that ISMs have a minimum non-null value. It avoids, in practice, an overestimation of the emptiness of a cell given a single scan point.

4.3.1.2 Projection of the local integer occupancy grid

Once the local integer occupancy grid is computed, it is projected vertically on the 2D grid as depicted on fig. 4.15a. The scan point only updates the occupancy indexes of 2D cells that are traversed by the projected local grid. The discrete traversal algorithm is used for finding out the traversed cells.



(a) Project the local grid on the 2D grid

(b) Update the occupancy index of 2D traversed cells

Figure 4.15 – Projection of the local integer occupancy grid and update of the occupancy index of a 2D cell traversed by the projection

Figure 4.15b illustrates the update of a 2D cell traversed by the projected local grid. Consider a 2D cell c_i that is traversed by the projection. Let us note by c'_j the cell of the local grid that is projected on cell c_i. Both cells c_i and c'_j are presented on the figure. The occupancy index of c_i is updated by the occupancy index of c'_j as follows:

$$I(o_i|z_1 \wedge \ldots \wedge z_k) = I(o_i|z_1 \wedge \ldots \wedge z_{k-1}) + I(o'_j|z_k) \tag{4.4}$$

4.3.2 Experimental results and analysis

Three implementations were experienced. The first one implements integer occupancy grids stored within an array. It serves for evaluating both execution time and power efficiency of the SW/HW integration. The second implementation stores integer occupancy grids within a quadtree. It allows to assess quantitatively the compaction of integer occupancy grid offered by 2^d-trees compared to arrays. The third implementation computes directly occupancy grids rather than integer occupancy grids. The Bayesian fusion is implemented with floating-points. This implementation allows to compare integer occupancy grids with respect to occupancy grids computed with floating-points. It also enables to validate experimentally the numerical quality of the integer occupancy grid framework.

4.3.2.1 Analysis of the array-based implementation

Figure 4.16 shows an example of a 2D integer occupancy grid fusing the scan points of the four LIDARs at a given period. The scenario of the traffic is at the top of the figure. On the bottom left is the integer occupancy grid. The equivalent occupancy grid is depicted on the bottom right of the figure. On both grids, dark

color corresponds to cells likely occupied by obstacles while bright color to likely empty cells. Gray color corresponds to cells with unknown occupancy states. These are cells that are in majority hidden or outside the field-of-view of the LIDARs.

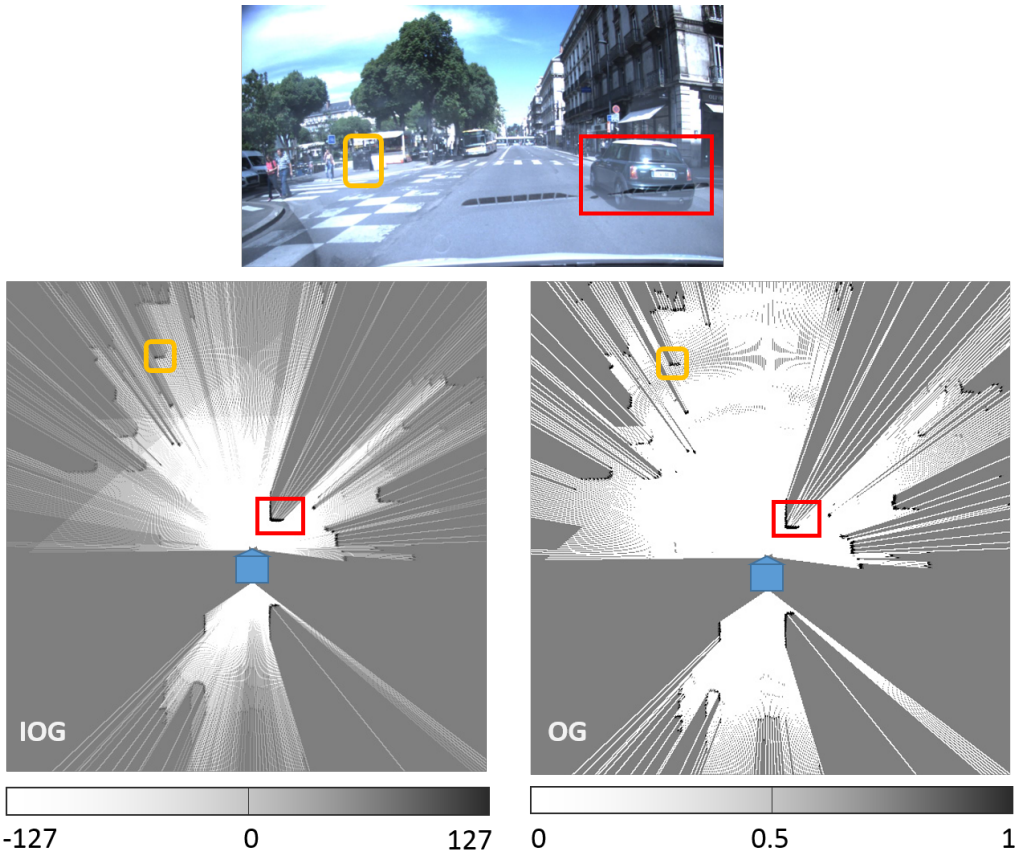


Figure 4.16 – Example of an urban scenario on top. The corresponding integer occupancy grid is on the bottom left. The parameter ε is set to 0.05. The occupancy grid corresponding to the scenario is on the bottom right.

The integer occupancy grid shows the presence of obstacles by cells of dark colors. For instance, according to the integer occupancy grid, an obstacle is likely situated in the front right of the prototype car. The image of the scenario shows that this obstacle actually corresponds to a car. Besides, another obstacle is also present in the front left of the prototype car even if its nature is hardly recognizable on the image of the scenario. Actually, this obstacle is a garbage bin placed on the sidewalk. Notice that the LIDARs do not provide information about the nature of objects hit by laser beams. Obstacles are just modeled by likely occupied cells.

Integer occupancy grids estimate the occupancy states of cells with occupancy indexes. For occupancy grids, occupancy states are estimated with occupancy probabilities. A comparison between the images on the bottom left and the bottom right of fig. 4.16 shows that integer occupancy grids provide more differentiated estima-

tion than standard occupancy grids. On the integer occupancy grid, empty cells do not have the same level of brightness. Some empty cells are brighter than others. Such difference of brightness is not however remarkable on the occupancy grid.

This difference highlights that there are regions that are seen as empty by only one LIDAR devices. It is for instance the case of the region at the left of the ego vehicle. This region has a lower brightness. At the same time, there are other regions that are seen as empty by two or three LIDAR devices. The region just in the front of the ego vehicle has a higher brightness since it is covered partially by the left and right LIDARs, and totally by the LIDAR on the center of the bumper.

Number of bits for occupancy indexes By shrinking occupancy indexes between -127 and 127, occupancy probabilities are saturated between p_{-127} and p_{127}. As explained in Section 3.4.6.3 (page 74), p_{-127} is less than 10^{-11} while p_{127} is greater than $1 - 10^{-11}$ with an ε equal to 0.05. This value ε enables to encode occupancy indexes in 8-bits. The indexes correspond to probabilities that are spread over $]0, 1[$, with values close to 0 and 1.

For smaller value of ε, occupancy probabilities are encoded in 32-bits. This allows to have a wide range of occupancy indexes (larger than $[-127, 127]$). In fact, if the occupancy indexes are still shrunk between -127 and 127 while ε is less than 0.05, the probabilities that correspond to the indexes would not reach value close to 0 or 1 (see Section 3.3.3.5 (page 63)).

Performance analysis [2] Three metrics are used for measuring the performance of the array-based implementation. The first one is the average output rate at which integer occupancy grids are produced by the SABRE Lite. The output rates versus the number of LIDAR devices are shown on Table 4.4 (page 105). Since the Ibeo LUX LIDARs produce complete scan at 25 Hz, Table 4.4 (page 105) shows that the implementation fuses the scan points into an occupancy grid in **real-time**.

Nb of LIDARs	4	3	2
OG Rate(Hz)	28	47	66

Table 4.4 – OG output rate on embedded CPU.

To compare the computation time with those of the state-of-the-art, the second metric is constituted by the product of the number of scan points with the number of cells and the output rate ($points \cdot cell \cdot Hz$). This metric shows how many points an implementation can process in one second. Table 4.5 (page 106) compares the performance of the array-based implementation to the performance of implementations of occupancy grids on a GPU in [Homm 2010] and on a desktop in [Nuss 2015]. The implementation of a GPU serves as a reference of comparison. The 6-th row of

[2] Experimental results published in [Rakotovao 2016b].

Table 4.5 (page 106) shows that the array-based implementation is 5× faster than the implementation on a GPU, and up to 10× faster than that on a desktop.

If the power consumption is also taken into account, the third metric measures the efficiency of the array-based implementation. This metric is measured in $points \cdot cell \cdot Hz/W$. It expresses the energy required for processing a given number of scan points within one second. With this metric, the array-based implementation is 1000× more efficient than the implementation of standard occupancy grids on GPU and desktop.

Criteria	**[Homm 2010]**	**[Nuss 2015]**	**Present Work**
HW	Nvidia GeForce 268GTX	Desktop	Single ARM A9
Power cons.	204W	∼80W	1W
Nb layers	2	4	16
Grid cells	512x512	533x533	1000x1000
Normalized $point \cdot cell \cdot Hz$	1	0.4	5
Normalized $point \cdot cell \cdot Hz/W$	1	∼1	1030

Table 4.5 – Performance of the array-based implementation of Integer Occupancy Grids compared with the state-of-the-art

Numerical quality Section 3.4.6.2 (page 74) showed that within the integer occupancy grid framework, only the step of quantization introduces a numerical error that is bounded by ε. The step of fusion does not introduces additional errors. Integer occupancy grids can be transformed into standard occupancy grids by replacing an occupancy index by its corresponding probability within the recursive set.

Let us evaluate the cell-by-cell absolute difference between occupancy grids computed by the array-based implementation of integer occupancy grids and the occupancy grids computed by the floating-point implementation of Bayesian fusion. Both implementations use exactly the same grid parameters and process the same set of data.

To evaluate the impact of ε on the values of occupancy probabilities computed from the integer occupancy grid framework, several values of ε were tested. Table 4.6 (page 107) shows statistics on the difference of probabilities between the two approaches for different values of ε. Both blurring quantization policy and nearest quantization policy are represented. Regardless of the quantization policy, the mean of the differences and the standard deviation are at least in order of $10^2\times$ lower than the value of ε. Table 4.6 (page 107) also shows that the differences decrease with ε. In fact, the lower is ε, the more accurate are the occupancy probabilities computed from the integer occupancy grid framework (Section 3.4.6.3 (page 74)).

	Mean of differences		Standard deviation	
ε	**Blurring pol.**	**Nearest pol.**	**Blurring pol.**	**Nearest pol.**
10^{-1}	4.65×10^{-4}	4.65×10^{-4}	1.58×10^{-3}	1.58×10^{-3}
10^{-2}	1.12×10^{-4}	7.19×10^{-5}	5.48×10^{-4}	3.61×10^{-4}
10^{-3}	2.81×10^{-6}	2.56×10^{-6}	2.65×10^{-5}	2.12×10^{-5}
10^{-4}	3.28×10^{-7}	2.56×10^{-7}	4.33×10^{-6}	2.96×10^{-6}
10^{-5}	1.65×10^{-7}	1.69×10^{-8}	5.89×10^{-7}	3.21×10^{-7}
10^{-6}	1.38×10^{-8}	5.31×10^{-9}	7.16×10^{-8}	3.46×10^{-8}

Table 4.6 – Statistics on differences between floating-point implementation of fusion and the proposed index fusion

4.3.2.2 Analysis of the tree-based implementation

This subsection discusses the implementation of integer occupancy grids based on 2^d-trees. A quadtree is experienced since the integer occupancy grid is based on a 2D grid. The compactness of quadtrees compared to the array-based implantation will be analyzed.

An example of a real traffic scenario with its corresponding integer occupancy grid is presented on fig. 4.17. To highlight how the occupancy indexes of cells are stored within a quadtree, the part of the integer occupancy grid encircled in red is zoomed. The zoom shows out the regions of the leaves that store the occupancy indexes of cells. The regions of the leaves are of different sizes. Some regions cover a unique cell while others cover 4, 8, 16 or more number of cells. This makes the quadtree more compact than arrays.

Lossless compaction To validate that the compaction offered by quadtrees are lossless, the occupancy indexes of cells stored within the quadtree are compared to the occupancy indexes of cells computed by the array-based implementation. The comparison has showed that the occupancy indexes stored within both quadtrees and arrays are cell-by-cell equal. Hence, regardless of the data storage, the produced integer occupancy grids are equal. Then, the compaction offered by the quadtrees is lossless.

Compactness To measure the compactness of quadtrees, the number of nodes within quadtrees is compared to the number of cells. Both blurring quantization policy and nearest quantization policy were experienced. In order to study the influence of ε on the compactness, several values of ε were tested. The compactness is expressed by the number of cells divided by the number of nodes, and by the number of cells divided by the number of leaves.

The compactness of quadtrees with respect to the quantization policy and the value of ε are shown on Table 4.17 (page 108). The number of cells is 1024×1024 cells. The table shows that the number of nodes is at least 2.8× less than the number

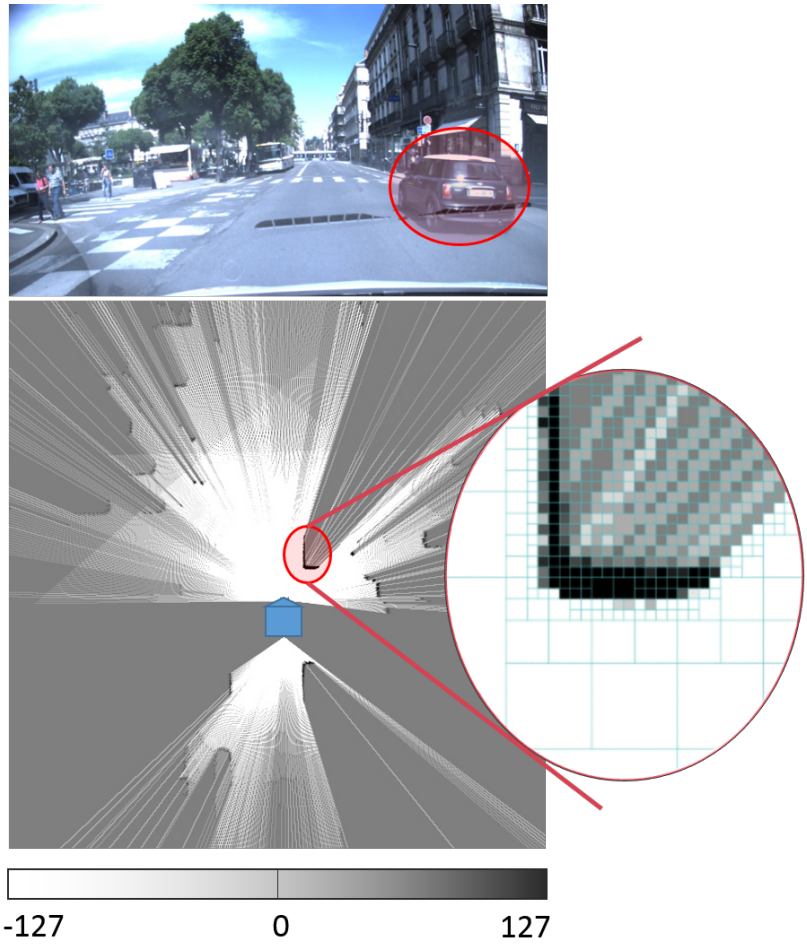


Figure 4.17 – Traffic scenario and the corresponding integer occupancy grid stored within a quadtree

of cells. Moreover, the number of leaves is at least 3.7× less than the number of cells. These proportion proves experimentally that the quadtree is more compact than the arrays. In fact, the number of array elements must be equal to the number of cells. Furthermore, Table 4.17 (page 108) shows that neither the quantization policy nor ε influence considerably the compactness of quadtrees.

Memory consumption. A quadtree is more compact than an array in term of number of nodes versus number of array elements, but how about the memory consumption? Let us analyze the memory consumed by quadtrees compared to memory consumed by arrays for storing the same integer occupancy grids.

A pointer-based implementation of quadtrees have been realized. A node is implemented as a data structure composed of two pointers and an integer. The first pointer points to the address of the children of the nodes, the second pointer points

	Blurring policy		Nearest Policy	
ε	**#cells/#nodes**	**#cells/#leaves**	**#cells/#nodes**	**#cells/#leaves**
10^{-1}	2.81	3.75	2.82	3.75
10^{-2}	2.81	3.74	2.81	3.74
10^{-3}	2.81	3.74	2.81	3.74
10^{-4}	2.84	3.78	2.81	3.74
10^{-5}	2.83	3.77	2.80	3.73
10^{-6}	2.83	3.77	2.83	3.77

Table 4.7 – Compactness of quadtrees compared to arrays

to the parent node. The integer stores the index of the node. On the SABRE Lite platform, a pointer occupies 4 Bytes. The index of node is either 1 byte if occupancy indexes are shrunken in $[-127, 127]$, or 4 Bytes otherwise. Hence, a node occupies 9 Bytes or 12 Bytes of data.

On the array-based implementation, an array elements store the occupancy indexes of all cells. An array element occupies either 1 byte or 4 Bytes, depending on the size of an occupancy index in memory. When storing an integer occupancy grid of 1024×1024, an array consumes 1 MByte or 4 MBytes. For the same grid size, a quadtree consumes in turn 3.15 MByte or 4.2 MByte. Hence, if an occupancy index is encoded in 8-bit, quadtrees are $3\times$ more memory consuming than arrays. The difference of memory consumption is reduced when occupancy indexes are encoded in 32-bit.

The memory consumption can be improved by adopting other implementations of quadtrees such as the linear quadtrees ([Samet 1988, Holroyd 1990]), or even by using other data structure like R-trees ([Guttman 1984]) or adaptive rectangular cuboids ([Khan 2015]). Furthermore, a 2D integer occupancy grid can be considered as an image by making an analogy between cells and pixels. Hence, any technique used for a lossless compression of images can be explored.

On an application viewpoint, the access and efficiency of the exploitation of integer occupancy grids depends first on the compactness of the data structure ([Soucy 2004]). The memory amount introduces though a challenge on embedded systems where memory resource is limited. A high memory consumption can also hinder the exchange of integer occupancy grids through the network. Assume that integer occupancy grids are computed by the MSF module. After that, they are transfered to an autonomous navigation module for making decision. The size of the data structure that stores integer occupancy grids plays a role in the speed of this transfer through a network.

Computation time The average output rate of integer occupancy grids stored within quadtrees is only 5 Hz in our implementation. This is $5\times$ slower than the frequency of complete scans produced by the Ibeo LIDARs. Hence, the quadtree-

based implementation does not reach a real-time performance.

Two points cause this low performance. First, the traversal algorithm is optimized for the array-based implementation. Designing a traversal algorithm optimized for quadtree is essential. Such algorithm can exploit the different size of the regions of leaves to accelerate the traversal. For instance, a traversal algorithm specifically designed for octree is presented in [Revelles 2000].

The second cause of the low performance is that, during a cell update, the operation of split and merge are always executed. These operations are though recursive and takes long execution time even if they are implemented as a loop. As a solution, the operation of merge can be deferred as in [Fairfield 2007]. That means, during a cell update, only the operation of split is performed. The operation of merge is executed less frequently, for instance after the insertion of the complete scan points from a single LIDAR device.

Both operations can be also accelerated by optimizing the data structure for storing a node. When nodes are pruned out of the tree during the operation of merge, they are deallocated in the memory. When children of a node are created during the operation of split, they are allocated in the memory. Hence, the memory allocation plays an important role in the speed of split and merge. Integer occupancy grids stored in quadtree or octree need an advanced memory allocation technique to accelerate the operations of split and merge.

4.4 Summary

To summarize, this chapter presented the application of the integer occupancy grid framework for performing automotive multi-sensor fusion.

- A formula for computing ISMs over a 1D grid given a measurement from a single-target sensor was proposed.
- A discrete traversal algorithm that works exclusively with integers was designed and studied experimentally.
- Integer occupancy grids were applied for fusing four LIDARs mounted on a prototype car.
- Integer occupancy grids were integrated on a low-cost and low-power hardware dedicated for automotive application.
- The numerical accuracy of integer occupancy grids have been studied experimentally.
- Both arrays and 2^d-trees have been experienced for storing integer occupancy grids.
- The lossless compaction of integer occupancy grids with 2^d-trees have been validated.

- The array-stored integer occupancy grids were computed in real-time. Real-time performance is however still missing for 2^d-trees.

Chapter 5

CONCLUSION AND PERSPECTIVES

This chapter concludes the present manuscript. It presents a summary of the Integer Occupancy Grid framework followed by a conclusion. Insights about perspectives and future works are listed thereafter.

5.1 Summary of Integer Occupancy Grids

In this manuscript, we introduced the Integer Occupancy Grid framework. The later enables to process fusion of range sensors and to build an environment model based on occupancy grids in an efficient way.

Like traditional occupancy grids, Integer Occupancy Grids are a tessellated probabilistic model of a physical environment. The latter is subdivided into multiple cells. A cell is either occupied by an obstacle or empty. The occupancy state of a cell is estimated by an occupancy probability for traditional occupancy grids, and by an occupancy index for Integer Occupancy Grids.

The occupancy index of a cell is paired with the value of its occupancy probability thanks to a set of probabilities. Occupancy indexes are integers while occupancy probabilities are real-numbers. The fusion based on occupancy indexes requires only integer arithmetic. Integer arithmetic are advantageous in term of HW/SW integration. They are exact, fast, power efficient and supported by the majority of modern computing platforms, even the embedded ones.

For fusing multiple measurements, Integer Occupancy Grids computed independently from individual measurements are combined cell-by-cell. The combination of occupancy indexes is equivalent to computing their sum. The computation of integer occupancy grids involves though a numerical error. Nevertheless, this error is known, bounded and chosen by the application designer.

Integer Occupancy Grids can be stored within arrays or within 2^d-trees. The occupancy index of a cell is the same whatever is the used data structure. Tree structures enable a lossless compaction of integer occupancy grids. The maintenance of the tree structure introduces though an overhead that makes trees slower when updating occupancy indexes.

5.2 Conclusion

To conclude, this thesis proposes the Integer Occupancy Grids as a new framework for processing the fusion of range sensors mounted on a car. The framework was developed by taking into account upstream both safety requirements and embedded hardware constraints. It enables the HW/SW integration of multi-sensor fusion on an embedded low-cost and low-power platform.

Integer Occupancy Grids enable to process Bayesian fusion with simple integer arithmetic. The numerical error involved by the framework is known, bounded and parametrized by the user. This allows to guaranty the quality, the safety and the robustness of the HW/SW integration of the fusion, especially when the later is used for safety critical tasks such as automotive perception.

5.3 Perspectives

The envisioned perspectives based on this thesis can be grouped as follows.

Short-term perspective

- In this thesis, Integer Occupancy Grids were experienced over 2D grids. An extension to 3D grids is required for being able to model overhanging road structures such as bridges and underground parking.

- A large 3D grid cannot be stored within arrays anymore. Tree-based structures enable to save memory but involve longer update time. Adequate algorithms of split and merge, advanced memory allocator for nodes, and traversal algorithm adapted to the tree structure are required to reach real-time performance with 2^d-trees.

Mid-term perspective

- While the multi-sensor fusion based on integer arithmetic is exact, it requires first a computation of mono-sensor occupancy grids. In this thesis, only mono-sensor occupancy grids based on a single-target sensor were considered. Single-target sensors return spatial points. This is not the case for other sensors like radars and ultrasonic sensors. Future work may focus on these sensors.

- In the present work, no notion of time is considered. Integer Occupancy Grids have to be converted into traditional occupancy grids before being injected into the Bayesian Occupancy Filter (BOF). The BOF also perform intensive probabilistic calculus and require efficient HW/SW support of real-number operations. An integer-based approach like the Integer Occupancy Grids can be studied to perform the filtering.

Long-term perspective

- When Integer Occupancy Grids will be able to handle the main sensors used for automotive or in robotics in general, when they will be able to map 3D environment with a compact data structure, a kind of operating system dedicated for multi-sensor fusion can be designed. Such operating system can be integrated on different kinds of processing platforms. It can be applied in different domains such as autonomous cars, underwater robots, mining robots, agriculture robots or aerial vehicles.

- Like traditional occupancy grids, Integer Occupancy Grids are still too dense and contain too much details for performing efficiently navigation tasks. A more abstract environment model is still required. The introduction chapter presented some of them. The conversion of an Integer Occupancy Grid into one of these models needs to be explored. The conversion may take advantage of the compactness of Integer Occupancy Grids.

- The technique based on integer arithmetic for combining occupancy probabilities can be applied to other problems different to environment modeling. This requires that the studied problem is based on the estimation of a binary state variable. The latter should be estimated from independent source of information. For instance, in an industrial production chain, this technique can be applied for verifying whether a product is defective or not. This can be done through a successive sensor observations. After the observations, a conclusion about the state of the product must be known. This problem can effectively treated with the Bayesian fusion based on integer arithmetic.

Publications

The present thesis has lead to the following publications.

Published papers

- R. Dia, J. Mottin, T. Rakotovao, D. Puschini and S. Lesecq. *Evaluation of Occupancy-Grid Resolution through a Novel Approach for Inverse Sensor Modeling.* 20th IFAC World Congress. 2017. ([Dia 2017]).

- T. Rakotovao, J. Mottin, D. Puschini and C. Laugier. *Integration of multi-sensor occupancy grids into automotive ECUs.* 2016 ACM/EDAC/IEEE Design Automation Conference (DAC) ([Rakotovao 2016b]).

- T. Rakotovao, J. Mottin, D. Puschini and C. Laugier. *Multi-sensor fusion of occupancy grids based on integer arithmetic.* 2016 IEEE International Conference on Robotics and Automation (ICRA) ([Rakotovao 2016a]).

- T. Rakotovao, J. Mottin, D. Puschini and C. Laugier. *Real-time power-efficient integration of multi-sensor occupancy grid on many-core.* 2015 IEEE International Workshop on Advanced Robotics and its Social Impacts([Rakotovao 2015a]).

- T. Rakotovao, D. Puschini, J. Mottin, L. Rummelhard, A. Negre and C. Laugier. *Intelligent Vehicle Perception: Toward the Integration on Embedded Many-core.* 2015 ACM Workshop on Parallel Programming and Run-Time Management Techniques for Many-core Architectures ([Rakotovao 2015b]).

- M. Louvel, A. Molnos, J. Mottin, F. Pacull and T. Rakotovao. *Poster abstract: Distributed coordination of sub-sys-tems power-modes and software-modes.* 2014 ACM/IEEE International Conference on Cyber-Physical Systems (ICCPS) ([Louvel 2014]).

Patents

- Julien Mottin, Diego Puschini Pascual, Tiana Rakotovao Andriamahefa and Olivier Debicki. *Procédé et système de détermination de cellules traversées par un axe de mesure ou de visualisation.* 2016. Dépôt France Ref 16 55104.

- Julien Mottin, Diego Puschini Pascual and Tiana Rakotovao Andriamahefa. *Procédé et système de perception de corps matériels.* 2015. Dépôt France Ref 15 58919.

Appendix A

Integer Occupancy Grids

This appendix presents the demonstration of some properties used in Chapter 3.

A.1 Examples of set of probability

Property A.1.1. *Let $a \geq 1$ be a positive and non-null integer. Consider the sequence $(p_n)_{n\in\mathbb{N}}$ such that:*

$$p_n = \frac{a \cdot n + 1}{a \cdot n + 2}, n \in \mathbb{N} \tag{A.1}$$

The sequence $(p_n)_{n\in\mathbb{N}}$ is monotonically increasing and $\forall n \in \mathbb{N}, 1/2 \geq p_n$.

Proof ■ By applying eq. (A.1), we get:

$$p_{n+1} - p_n = \frac{a}{(a \cdot n + 1) \cdot (a \cdot n + a + 2)}$$

Since $a \geq 1$, then $p_{n+1} - p_n > 0$. That means $p_{n+1} > p_n$, thus the sequence $(p_n)_{n\in\mathbb{N}}$ is monotonically increasing. Consequently, $\forall n \in \mathbb{N}, p_n \geq p_0$, then $p_n \geq 1/2$. ■

A.2 Definition of the recursive set

The recursive set S_ε was defined in Theorem 3 (page 60). The set was defined as follows.

Theorem. *(Theorem 3 (page 60)) Let ε be a real-number such that $\varepsilon \in]0, 1/2[$. Let $(a_n)_{n\in\mathbb{N}}$ and $(b_n)_{n\in\mathbb{N}}$ be infinite sequences of numbers defined as follows:*

$$a_n = \begin{cases} 1/2 & \text{if } n = 0 \\ 1/2 + \varepsilon & \text{if } n = 1 \\ a_{n-1} \odot a_1 & \text{otherwise} \end{cases} \qquad b_n = \begin{cases} 1/2 & \text{if } n = 0 \\ 1/2 - \varepsilon & \text{if } n = 1 \\ b_{n-1} \odot b_1 & \text{otherwise} \end{cases}$$

Consider the set $S_\varepsilon = \{p_n, n \in \mathbb{Z}\}$ such that:

$$p_n = \begin{cases} a_n & \text{if } n \geq 0 \\ b_{-n} & \text{otherwise,} \end{cases}$$

The set S_ε – called ***recursive set*** *– constitutes a set of probabilities equipped by the following index fusion operator:*

$$\forall m, n \in \mathbb{Z} : m \oplus n = m + n$$

A.3 Mathematical derivation of the recursive set of probabilities

This section [1] presents the mathematical derivation of the recursive set. It presents the mathematical intuition that has lead to the development of the sequences $(a_n)_{n\in\mathbb{N}}$ and $(b_n)_{n\in K}$. After that, this section proves that the set S_ε defined over both sequences (see Theorem 3 (page 60)) actually constitutes a set of probabilities.

The set of probabilities that is intended to capture the values of occupancy probabilities of cells must contain both elements less than $1/2$ and elements greater than $1/2$. This is required for capturing the occupancy probabilities of both likely empty and likely occupied cells. Let us now present the intuitive idea that allowed to get the elements greater than $1/2$.

A.3.1 Elements greater than one-half

To define a set of probabilities to be used for occupancy grids, let us assume that the elements of the set are defined by a numerical sequence $(a_n)_{n\in\mathbb{N}}$. To design the sequence, let us start from the singleton $\{1/2\}$. Assume that

$$a_0 = 1/2 \tag{A.2}$$

To ensure that the terms of the sequence are different to each others, let us define the other terms $a_n, n > 0$ such that the sequence is monotonically increasing. The order of terms are depicted on fig. A.1.

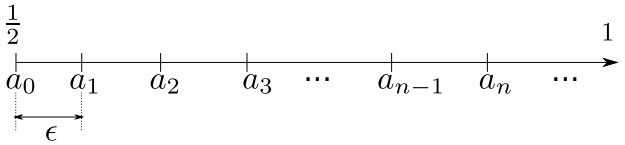


Figure A.1 – Ordering the terms of the sequence $(a_n)_{n\in\mathbb{N}}$

Let us now define the value of a_1. Assume that a_1 is the next member of the set that is closest to $1/2$. That means:

$$a_1 = 1/2 + \varepsilon \tag{A.3}$$

[1] This section can be skipped by the reader who does not need deeper mathematical details about the demonstration of the recursive set. The reader can move directly towards Section 3.4 (page 63).

where ε is a positive number. In addition, ε has to be less than $1/2$ to ensure that $a_1 < 1$:

$$0 < \varepsilon < 1/2 \tag{A.4}$$

Now, how about a_2? The above principle can be reused: after a_1, a_2 is the next member closest to $1/2$. Since $a_1 > 1/2$, the property of reinforcement of the fusion gives $a_1 \odot a_1 > a_1$ (see Property 3.1.2 (page 51)). Thus, let us consider:

$$a_2 = a_1 \odot a_1 \tag{A.5}$$

How about a_3? As above, let us assume that a_3 is the next member closest to $1/2$ after a_1 and a_2. The term a_3 can be defined by two ways: either $a_1 \odot a_2$ or $a_2 \odot a_2$. However, $a_1 \odot a_2 < a_2 \odot a_2$ (see Property A.4.4 (page 125)). Then, let us consider:

$$a_3 = a_1 \odot a_2 \tag{A.6}$$

The same reasoning can be continued by induction. By taking into account equations (A.2),(A.3),(A.5),(A.6), we obtain the general term by induction:

$$a_n = \begin{cases} 1/2 & \text{if } n = 0 \\ 1/2 + \varepsilon & \text{if } n = 1 \\ a_{n-1} \odot a_1 & \text{otherwise} \end{cases} \tag{A.7}$$

A.3.2 Elements less than one-half

Suppose that the elements of the set of probability that are less than $1/2$ form a sequence $(b_n)_{n\in\mathbb{N}}$. As for a_n, let us assign $1/2$ to the first term:

$$b_0 = 1/2 \tag{A.8}$$

Let b_1 be the next term that is closest to $1/2$:

$$b_1 = 1/2 - \varepsilon \tag{A.9}$$

The terms of $(b_n)_{n\in\mathbb{N}}$ are designed to be less than $1/2$. The sequence has to be monotonically decreasing. The order of the terms are shown on fig. A.2

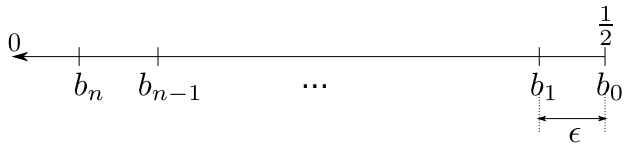


Figure A.2 – Ordering the terms of the sequence $(b_n)_{n\in\mathbb{N}}$

By applying similar induction as for the sequence $(a_n)_{n\in\mathbb{N}}$, the general term b_n becomes:

$$b_n = \begin{cases} 1/2 & \text{if } n = 0 \\ 1/2 - \varepsilon & \text{if } n = 1 \\ b_{n-1} \odot b_1 & \text{otherwise} \end{cases} \tag{A.10}$$

A.3.3 Proof of the recursive set of probabilities

After determining the formula of the sequences $(a_n)_{n\in\mathbb{N}}$ and $(b_n)_{n\in\mathbb{N}}$, let us now prove that the set $S_\varepsilon = \{p_n\}$, such that:

$$p_n = \begin{cases} a_n & \text{if } n \geq 0 \\ b_{-n} & \text{otherwise,} \end{cases} \tag{A.11}$$

, constitutes a set of probabilities. The proof is subdivided into three parts. First, we will prove that the set $\{a_n, n \in \mathbb{N}\}$ constitutes a set of probabilities called the recursive set of occupancy. Second, we will prove that the set set $\{b_n, n \in \mathbb{N}\}$ also form a set of probabilities called the recursive set of emptiness. Finally, we will demonstrate that the reunion of both sets constitutes the recursive sets and also forms a set of probabilities.

A.3.3.1 Recursive set of occupancy

Let be ε a positive number in $]0, 1/2[$. Let S^o_ε denote the set $\{a_n, n \in \mathbb{N}\}$. Let us verify if S^o_ε is a set of probability.

- **Inclusion into** $]0, 1[$**.** Property A.4.1 (page 124) shows that $a_n < 1$. Property A.4.2 (page 125) stipulates that $a_n \geq 1/2$. Then $0 < a_n < 1, \forall n \in \mathbb{N}$.

- **Countability.** Property A.4.3 (page 125) shows that $a_{n-1} < a_n$. That means $\forall m, n \in \mathbb{N}$, we get: $m \neq n \Leftrightarrow a_m \neq a_n$. Then the set S^o_ε is countable.

- **Closure.** To verify the closure of S^o_ε, let us compute the fusion of a_n and a_m, where $m, n \in \mathbb{N}$. Equation (A.7) defines a_n by the following recursion: $a_n = a_{n-1} \odot a_1$. This property can be developed as follows:

$$\begin{aligned} a_n &= a_{n-1} \odot a_1 \\ &= (a_{n-2} \odot a_1) \odot a_1 \\ &= ((a_{n-3} \odot a_1) \odot a_1) \odot a_1 \\ &\dots \\ &= \underbrace{a_1 \odot \ldots \odot a_1}_{n \text{ times}} \end{aligned} \tag{A.12}$$

Consequently, the fusion of a_m with a_n leads to:

$$\begin{aligned} a_m \odot a_n &= \underbrace{a_1 \odot \ldots \odot a_1}_{m \text{ times}} \odot \underbrace{a_1 \odot \ldots \odot a_1}_{n \text{ times}} \\ &= \underbrace{a_1 \odot \ldots \odot a_1}_{(n+m) \text{ times}} \end{aligned} \tag{A.13}$$

Finally, we obtain:

$$a_m \odot a_n = a_{m+n} \tag{A.14}$$

The above equation means:

$$\forall m, n \in \mathbb{N} : a_m \odot a_n = a_{m \oplus n} = a_{m+n} \tag{A.15}$$

Then S_ε^o is a closed set with respect to the operator $\odot$. In addition, it has an index fusion operator $\oplus$ where $m \oplus n = m + n$.

A.3.3.2 The recursive set of emptiness

Let be ε a positive number in $]0, 1/2[$. Let S_ε^e denotes the set $\{b_n, n \in \mathbb{N}\}$. The proves that S_ε^e constitutes a set of probability is similar to those of S_ε^o. Both Property A.4.5 (page 125) and Property A.4.6 (page 125) shows that $0 < b_n \leq 1/2$. Then S_ε^e is included within $]0, 1[$. Next, the countability of S_ε^e can be proved by using Property A.4.7 (page 125).

Let us now focus on the closure of S_ε^e with respect to the operator $\odot$. Like an element a_n of the set S_ε^o, an element b_n of S_ε^e verifies:

$$b_n = \underbrace{b_1 \odot \ldots \odot b_1}_{n \text{ times}} \tag{A.16}$$

By applying the same reasoning as for $a_m \odot a_n$, we obtain:

$$b_m \odot b_n = b_{m+n} \tag{A.17}$$

Therefore, the set S_ε^e is also closed under the fusion operation. The set has an index fusion operator $\oplus$ where $m \oplus n = m + n$.

A.3.3.3 The final recursive set

The recursive set S_ε is formed by the reunion of the set S_ε^o and the set S_ε^e. Like S_ε^o and S_ε^e, S_ε is included within $]0, 1[$ and is countable. It remains to verify if S_ε is closed with respect to the fusion operator $\odot$.

Theorem 3 (page 60) defines S_ε as a set of $\{p_n, n \in \mathbb{Z}\}$. An element of S_ε is defined from the sequences $(a_n)_{n\in\mathbb{N}}$ and $(b_n)_{n\in\mathbb{N}}$ such that:

$$p_n = \begin{cases} a_n & \text{if } n \geq 0 \\ b_{-n} & \text{otherwise} \end{cases}$$

Let us now study the fusion of p_m and p_n, where $m, n \in \mathbb{Z}$. Two cases are possible: m and n have the same signs or not. In the first case, if $m \geq 0$ and $n \geq 0$, then $p_m = a_m$ and $p_n = a_n$. Therefore, $p_m \odot p_n = p_{m+n}$. Similarly, if $m < 0$ and $n < 0$, then $p_m \odot p_n = p_{m+n}$.

In the second case, assume that $m \geq 0$ and $n \leq 0$. We get $p_m = a_m$ and $p_n = b_{-n}$ Consequently, the fusion of p_m and p_n gives:

$$p_m \odot p_n = \underbrace{a_1 \odot \ldots \odot a_1}_{|m| \text{ times}} \odot \underbrace{b_1 \odot \ldots \odot b_1}_{|n| \text{ times}} \tag{A.18}$$

where $|x|$ designates the absolute value of x. From here, three cases are possible:

1. if $|m| = |n|$, eq. (A.18) gives:

$$\begin{aligned} p_m \odot p_n &= \underbrace{a_1 \odot \ldots \odot a_1}_{|m| \text{ times}} \odot \underbrace{b_1 \odot \ldots \odot b_1}_{|m| \text{ times}} \\ &= \underbrace{(a_1 \odot b_1) \ldots \odot (a_1 \odot b_1)}_{|m| \text{ times}} \quad (\odot \text{ is associative}) \end{aligned} \tag{A.19}$$

   Since $a_1 \odot b_1 = 1/2$, then we finally obtain $p_m \odot p_n = 1/2$.

2. If $|m| > |n|$, rearranging eq. (A.18) leads to:

$$\begin{aligned} p_m \odot p_n &= \underbrace{(a_1 \odot b_1) \ldots \odot (a_1 \odot b_1)}_{|n| \text{ times}} \odot \underbrace{a_1 \odot \ldots \odot a_1}_{(|m|-|n|) \text{ times}} \\ &= 1/2 \odot \underbrace{a_1 \odot \ldots \odot a_1}_{(|m|-|n|) \text{ times}} \\ &= a_{|m|-|n|} \\ &= p_{|m|-|n|} \end{aligned} \tag{A.20}$$

   Since $|m| > |n|$, then $|m| - |n| = m + n$. Consequently, we get $p_m \odot p_n = p_{m+n}$.

3. Finally, if $|m| < |n|$, we can write:

$$\begin{aligned} p_m \odot p_n &= \underbrace{(a_1 \odot b_1) \ldots \odot (a_1 \odot b_1)}_{|m| \text{ times}} \odot \underbrace{b_1 \odot \ldots \odot b_1}_{(|n|-|m|) \text{ times}} \\ &= 1/2 \odot \underbrace{b_1 \odot \ldots \odot b_1}_{(|n|-|m|) \text{ times}} \\ &= b_{|n|-|m|} \\ &= p_{-(|n|-|m|)} \end{aligned} \tag{A.21}$$

   Since $|m| < |n|$, then $-(|n| - |m|) = m + n$. Therefore, $p_m \odot p_n = p_{m+n}$.

In conclusion, the fusion of two elements of S_ε gives $p_m \odot p_n = p_{m+n}$. The recursive set S_ε is closed with respect to $\odot$. It has an index fusion operator $\oplus$ such that $m \oplus n = m + n$. The recursive set forms a set of probability which elements are symmetrically spread between 0 and 1. The inverse of an element p_n with respect to $\odot$ is p_{-n}.

A.4 Properties of the recursive set

Property A.4.1. $\forall n \in \mathbb{N} : a_n < 1$

Proof ■ Proof by induction. In the base case, $\leq a_0 < 1$ is true due to (A.3). In the inductive step, assume that $a_{n-1} < 1$. Since a_1 is also less than 1, the definition of the fusion operator $\odot$ (Definition 3.1.1 (page 50)) ensures that $a_{n-1} \odot a_1 < 1$. Then $a_n < 1$. ■

Property A.4.2. $\forall n \in \mathbb{N} : a_n \geq 1/2$

Proof ■ Proof by induction. In the base case, $1/2 \leq a_0 < a_1$ is true due to (A.3). In the inductive step, assume that $a_{n-1} \geq 1/2$. Equation (A.7) gives $a_n = a_{n-1} \odot a_1$. Since both a_{n-1} and a_1 are greater than $1/2$, the property of reinforcement gives $a_{n-1} \odot a_1 > a_{n-1}$. Then $a_n \geq 1/2$. ■

Property A.4.3. $\forall n \in \mathbb{N} : a_{n-1} < a_n$

Proof ■ Proof by induction. In the base case, $a_0 < a_1$ is true due to (A.3). In the inductive step, assume that $a_{n-1} < a_n$. Equation (A.7) gives $a_{n+1} = a_n \odot a_1$. Since $a_n > 1/2$ and $a_1 > 1/2$, the reinforcement property gives $a_n \odot a_1 > a_n$. Then $a_{n+1} > a_n$. ■

Property A.4.4. $a_1 \odot a_2 < a_2 \odot a_2$

Proof ■ By utilizing the Bayesian fusion function F, we get:

$$\begin{aligned} a_1 \odot a_2 = F(a_1, a_2) &= \frac{(2 \cdot \varepsilon + 1)^3}{2 \cdot (12 \cdot \varepsilon^2 + 1)} \\ a_2 \odot a_2 = F(a_2, a_2) &= \frac{(2 \cdot \varepsilon + 1)^4}{2 \cdot (16 \cdot \varepsilon^4 + 24 \cdot \varepsilon^2 + 1)} \end{aligned}$$

Then

$$(a_1 \odot a_2) - (a_2 \odot a_2) = \frac{\varepsilon \cdot (2 \cdot \varepsilon - 1)^3 \cdot (2 \cdot \varepsilon + 1)^3}{(12 \cdot \varepsilon^2 + 1)(16 \cdot \varepsilon^4 + 24 \cdot \varepsilon^2 + 1)}$$

Since $\varepsilon < 1/2$, then $(2 \cdot \varepsilon - 1) < 0$. Finally, $(a_1 \odot a_2) - (a_2 \odot a_2) < 0$, thus $(a_1 \odot a_2) < (a_2 \odot a_2)$. ■

Property A.4.5. $\forall n \in \mathbb{N} : b_n > 0$

Proof ■ Proof by induction. In the base case, $\leq b_0 > 0$ is true due to (A.8). In the inductive step, assume that $b_{n-1} > 0$. Since b_1 is also strictly positive, the definition of the fusion operator $\odot$ (Definition 3.1.1 (page 50)) ensures that $b_{n-1} \odot b_1 > 0$. Then $b_n > 0$. ■

Property A.4.6. $\forall n \in \mathbb{N} : b_n \leq 1/2$

Proof ■ Similar to the proof of Property A.4.2 (page 125). ■

Property A.4.7. $\forall n \in \mathbb{N} : b_{n-1} > b_n$

Proof ■ Similar to the proof of Property A.4.3 (page 125). ■

Property A.4.8. *The sequence* $\{p_n\}_{n \in \mathbb{Z}}$ *is monotonically increasing:*

$$\forall n \in \mathbb{Z} : p_n < p_{n+1}$$

Proof ■ If $n \geq 0$, then $p_n = a_n$. Thus, $p_{n+1} = a_{n+1} = a_n \odot a_1$. Since both a_1 and a_n are greater than $1/2$, the property of reinforcement of $\odot$ gives $a_n < a_n \odot a_1$. Then $a_n < a_{n+1}$, and subsequently $p_n < p_{n+1}$.

If $n < 0$, then $p_n = b_{-n}$. Therefore, $p_{n+1} = b_{-n+1} = b_{-n} \odot b_1$. Since both b_1 and b_{-n} are less than $1/2$, the property of reinforcement of $\odot$ gives $b_{-n} \odot b_1 < b_{-n}$. Thus $b_{-n} < b_{-n+1}$, and then $p_n < p_{n+1}$. ■

Property A.4.9. $\forall n \in \mathbb{N} : |a_{n+1} - a_n| > |a_{n+2} - a_{n+1}|$

Proof ■ Since (a_n) is monotonically increasing, then $|a_{n+1} - a_n| = a_{n+1} - a_n$. Therefore:

$$\begin{aligned} |a_{n+1} - a_n| - |a_{n+2} - a_{n+1}| &= a_{n+1} - a_n - a_{n+2} - a_{n+1} \\ &= (a_n \odot a_1) - a_n - (a_n \odot a_1 \odot a_1) - (a_n \odot a_1) \\ &= -\frac{16\varepsilon^2 \cdot a_n \cdot (a_n - 1) \cdot (2 \cdot a_n + 2\varepsilon - 1)}{(4 \cdot a_n \cdot \varepsilon - 2\varepsilon + 1) \cdot (8 \cdot a_n \cdot \varepsilon + 4\varepsilon^2 - 4\varepsilon + 1)} \end{aligned}$$

By taking into account that $a_n < 1$ and $0 < \varepsilon < 1/2$, we obtain $|a_{n+1} - a_n| - |a_{n+2} - a_{n+1}| > 0$. Therefore, we finally have $|a_{n+1} - a_n| > |a_{n+2} - a_{n+1}|$. ■

Property A.4.10. $\forall n \in \mathbb{N} : |b_{n+1} - b_n| > |b_{n+2} - b_{n+1}|$

Proof ■ Similar to the proof of Property A.4.9 (page 126). ■

Property A.4.11. *(Proof of Property 3.3.1 (page 61))* $\forall n \in \mathbb{N} : 0 < p_{-n} < 1/2$ *and* $1/2 \leq p_n < 1$

Proof ■ Let $n \in \mathbb{N}$. Property A.4.5 (page 125) ensures that $b_n > 0$ while Property A.4.6 (page 125) gives $b_n \leq 1/2$. Since $p_{-n} = b_n$ (see Theorem 3 (page 60)), then $0 < p_{-n} < 1/2$.

Besides, Property A.4.1 (page 124) states that $a_n < 1$. In addition, $a_n \geq 1/2$ according to Property A.4.2 (page 125). Since $p_n = a_n$, then we have $1/2 \leq p_n < 1$. ■

Property A.4.12. *(Proof of Property 3.3.3 (page 62)) The inverse of the element p_n with respect to the operator $\odot$ is p_{-n}:*

$$\forall n \in \mathbb{Z}, p_n \odot p_{-n} = 1/2$$

Proof ■ Theorem 3 (page 60) gives $p_n \odot p_m = p_{n+m}$. By replacing m by $-n$, we get:

$$p_n \odot p_{-n} = p_{n \odot -n} = p_0 = 1/2$$

Since $1/2$ is the identity element of the operator $\odot$, p_{-n} becomes then the inverse of p_n. ■

Property A.4.13. *(Proof of Property 3.3.4 (page 62))* $\forall n \in \mathbb{N}$*:*

$$|p_{n+2} - p_{n+1}| < |p_{n+1} - p_n| \qquad |p_{-n-2} - p_{-n-1}| < |p_{-n-1} - p_{-n}|$$

Proof ■ Let n be a positive or null integer. We have $p_n = a_n$ and $|a_{n+1} - a_n| > |a_{n+2} - a_{n+1}|$ (see Property A.4.9 (page 126)). Then, we get $|p_{n+2} - p_{n+1}| < |p_{n+1} - p_n|$.

Besides, Property A.4.9 (page 126) states that $|b_{n+1} - b_n| > |b_{n+2} - b_{n+1}|$. Since $p_n = b_{-n}$, then $p_{-n} = b_n$. Consequently, we also have $p_{-n-2} = b_{n+2}$ and $p_{-n-1} = b_{n+1}$. Finally, we obtain $|p_{-n-2} - p_{-n-1}| < |p_{-n-1} - p_{-n}|$. ■

Property A.4.14. *(Proof of Property 3.3.5 (page 62))* $\forall n \in \mathbb{Z} : |p_{n+1} - p_n| \leq \varepsilon$

Proof ■

$$\begin{aligned} a_{n+1} - a_n - \varepsilon &= F(a_n, p1) - a_n - \varepsilon \\ &= \frac{(-2 \cdot a_n + 1) \cdot \varepsilon \cdot (2 \cdot \varepsilon + 2 \cdot a_n - 1)}{4 \cdot \varepsilon \cdot a_n - 2 \cdot \varepsilon + 1} \end{aligned}$$

Since, $1/2 \leq a_n < 1$ and $0 < \varepsilon < 1/2$, then

$$\begin{aligned} (-2 \cdot a_n + 1) &\geq 0 \\ (2 \cdot \varepsilon + 2 \cdot a_n - 1) &\leq 0 \\ 4 \cdot \varepsilon \cdot a_n - 2 \cdot \varepsilon + 1 &> 0 \end{aligned}$$

Consequently, $a_{n+1} - a_n - \varepsilon \leq 0$, then $a_{n+1} - a_n \leq \varepsilon$. ■

Appendix B

Inverse sensor model for single-target sensors

This appendix presents the demonstration of theorem and algorithms presented in Section 4.2 (page 89) These theorems and algorithms serve for computing the ISM of a 1D cell or that of 2D cell.

B.1 Mathematical derivation of the ISM of 1D cell

This section demonstrates Theorem 6 (page 91) for calculating the ISM of a linear cell given a measurement from a single-target sensor. Consider a single-target sensor that observes a physical environment through a line-of-sight. The sensor returns a measurement z. The sensing process is modeled by the sensor model $p(z|d)$ where d denotes the distance to the nearest obstacle. The line-of-sight of the sensor is subdivided into a 1D grid $\mathcal{G}$ composed of N number of cells. The cell c_1 is the cell closest to the sensor and cell c_N is the furthest. Let d_i denote the distance of cell c_i from the sensor. Let us compute the ISM $P(o_i|z)$ of a cell c_i.

By applying the Theorem of Bayes (eq. (2.13)), we get:

$$P(o_i|z) = \frac{p(z|o_i)P(o_i)}{p(z|o_i)P(o_i) + p(z|e_i)P(e_i)} \tag{B.1}$$

Under the non-informative prior, $P(o_i) = P(e_i) = 1/2$ (Hypothesis 2.3.1 (page 28)). Then, the ISM becomes:

$$P(o_i|z) = \frac{p(z|o_i)}{p(z|o_i) + p(z|e_i)} \tag{B.2}$$

The Bayesian approach (Section 2.3.1.1 (page 29)) introduced the notion of grid configurations for computing $p(z|s_i), s_i \in \{o_i, e_i\}$. Let us utilize the same approach. Equation (2.33) (Section 2.3.1.1 (page 29)) gave:

$$p(z|s_i) = \sum_{g^{s_i}} p(z|g^{s_i})P(g^{s_i}) \tag{B.3}$$

The symbol g^{s_i} denotes a grid configuration in the form of:

$$g^{s_i} \triangleq x_1 \wedge \ldots, x_{i-1}, s_i, x_{i+1} \wedge \ldots \wedge x_N \qquad \text{where } x_j \in \{o_j, e_j\} \tag{B.4}$$

Since a cell state x_j can be either o_j or e_j, then there exist 2^{N-1} number of grid configurations g^{s_i}. The Bayesian approach assumes that these configurations are equiprobable. That means $P(g^{s_i})$ is equal to $1/2^{N-1}$. Equation (B.3) becomes:

$$p(z|s_i) = \sum_{g^{s_i}} \frac{p(z|g^{s_i})}{2^{N-1}} \tag{B.5}$$

Equation (2.34) in Section 2.3.1.1 (page 29) proposed to compute $p(z|g^{s_i})$ by using the sensor model as follows. For any configuration g^{s_i}, there exists a cell index h such that:

$$p(z|g^{s_i}) = p(z|d_h) \tag{B.6}$$

If the configuration of the grid was known, a single-target sensor would sense the first occupied cell towards the ray. That means, h is the index of the first occupied cell of the grid configuration seen by the sensor.

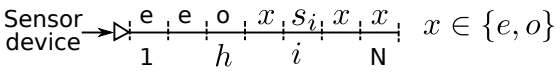


Figure B.1 – Configurations that share the same value of h_j

Several grid configurations may share the same value of h. Figure B.1 illustrates such configurations. The label e indicates empty cells while o denotes occupied cells. The state of cell c_i is set to s_i. A cell with a symbol x can be either occupied or empty. The number of such cell is $N - h - 1$. Consequently, this figure illustrate 2^{N-h-1} number of grid configurations. For all these configurations, $p(z|g^{s_i}) = p(z|d_h)$. Let us exploit this property to factorize the sum on eq. (B.5) for computing $p(z|s_i), s_i \in \{o_i, e_i\}$.

Let us consider $s_i = o_i$ and let us compute $p(z|o_i)$. That means the cell c_i is occupied, then h cannot be greater than i. Two cases are possible with respect to the value of h:

- If $h < i$, then there exist $N - h - 1$ cells labeled with x. Thus, 2^{N-h-1} grid configurations have $p(z|g^{s_i}) = p(z|d_h)$.
- If $h = i$, then there are $N - h$ cells labeled with x. Hence, 2^{N-i} grid configurations have $p(z|g^{s_i}) = p(z|d_i)$.

By taking into account both cases, we obtain:

$$p(z|o_i) = \sum_{h=1}^{i-1} \left[2^{N-h-1} \times \frac{p(z|d_h)}{2^{N-1}} \right] + 2^{N-i} \times \frac{p(z|d_i)}{2^{N-1}} \tag{B.7}$$

$$= \sum_{h=1}^{i-1} \left[\frac{p(z|d_h)}{2^h} \right] + \frac{p(z|d_i)}{2^{i-1}} \tag{B.8}$$

Consider now that $s_i = e_i$ and let us compute $p(z|e_i)$. That means the cell c_i is empty, then h cannot be equal to i. Thus, three cases are possible regarding the value of h:

1. If $h < i$, then the number of cells labeled with x is $N - h - 1$. Thus, 2^{N-h-1} grid configurations have $p(z|g^{s_i}) = p(z|d_h)$.
2. If $h > i$, then the number of cells labeled with x is $N - h$. Hence, 2^{N-h} grid configurations have $p(z|g^{s_i}) = p(z|d_h)$.

By taking into account both cases, we get:

$$p(z|e_i) = \sum_{h=1}^{i-1} \left[2^{N-h-1} \times \frac{p(z|d_h)}{2^{N-1}} \right] + \sum_{h=i+1}^{N} \left[2^{N-h} \times \frac{p(z|d_h)}{2^{N-1}} \right] \tag{B.9}$$

$$= \sum_{h=1}^{i-1} \left[\frac{p(z|d_h)}{2^h} \right] + \sum_{h=i+1}^{N} \left[\frac{p(z|d_h)}{2^{h-1}} \right] \tag{B.10}$$

The sum of (B.8) and eq. (B.10), we obtain:

$$p(z|o_i) + p(z|e_i) = \sum_{h=1}^{N} \left[\frac{p(z|d_h)}{2^{h-1}} \right] \tag{B.11}$$

Finally, by inserting eq. (B.8), eq. (B.10) and eq. (B.11) into eq. (B.2), we obtain:

$$P(o_i|z) = \frac{\sum_{h=1}^{i-1} \left[\frac{p(z|d_h)}{2^h} \right] + \frac{p(z|d_i)}{2^{i-1}}}{\sum_{h=1}^{N} \left[\frac{p(z|d_h)}{2^{h-1}} \right]} \tag{B.12}$$

Notice that the above formula is not valid if $i = 1$ since the sum on the numerator would be inversed. Let us compute $p(z|s_1)$ where $s_1 \in \{o_1, e_1\}$. If $s_1 = o_1$, the same principle as in the computation of $p(z|o_i)$ is applied, except that h cannot be less than i. This principle gives:

$$p(z|o_1) = \frac{p(z|d_1)}{2^{1-1}} = p(z|d_1) \tag{B.13}$$

If $s_1 = e_1$, the same principle as for $p(z|e_i)$ is also applied, except that h still cannot be less than i. This gives:

$$p(z|e_1) = \sum_{h=2}^{N} \left[\frac{p(z|d_h}{2^{h-1}} \right] \tag{B.14}$$

Finally, we get:

$$P(o_1|z) = \frac{p(z|d_1)}{\sum_{h=1}^{N} \left[\frac{p(z|d_h)}{2^{h-1}} \right]} \tag{B.15}$$

Both eq. (B.12) and eq. (B.15) constitute the Theorem 6 (page 91) for computing the ISM $P(o_i|z)$.

B.2 Mathematical derivation of the continuous range-mapping algorithm

This section provides the demonstration of Algorithm 3 (page 95). The algorithm works within the situation illustrated on fig. B.2 where a single-target sensor observes a physical environment. The environment is subdivided into a 2D grid denoted by $\mathcal{G}$. The sensor observes the environment along a line-of-sigh modeled by a line segment $\overline{ST}$. The sensor is localized at point S. The objective of Algorithm 3 (page 95) is to find out all squared cells traversed by the line segment $\overline{ST}$.

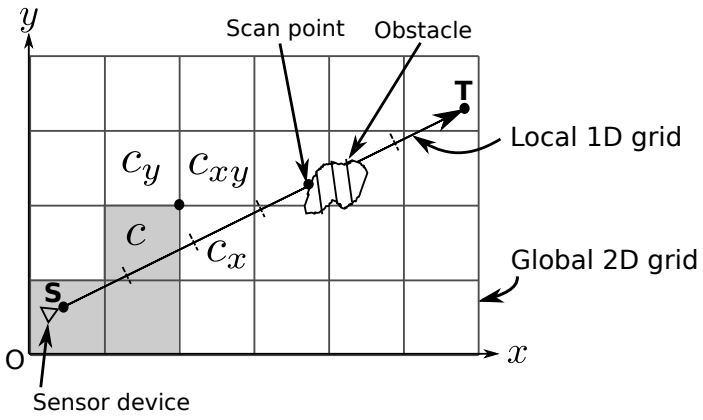


Figure B.2 – The traversal algorithm

The line segment $\overline{ST}$ traverses multiple cells of the global grid. Let us design an algorithm that finds out the traversed cells, starting from point S towards point T. On fig. B.2, some cells of the global grid are colored in gray, others are not. Assume that the gray color designates cells that are already known to be traversed by $\overline{ST}$. Knowing that the cell c is traversed, what will be the next traversed cell?

According to fig. B.2, if the traversal direction starts from S towards T, the next cell to be traversed will be one among cell c_x, cell c_y, and cell $c_{x,y}$. To determine mathematically which of them is the next cell, let us consider a cartesian frame of reference (O, x, y) attached to the global grid. Assume that the point S has coordinates (x_S, y_S) and (x_T, y_T) for the point T. Any point with coordinates (x, y) belongs to the straight line $\overleftrightarrow{ST}$ if and only if:

$$(y - y_S) \cdot (x_T - x_S) - (x - x_S) \cdot (y_T - y_S) = 0 \quad \text{iff } (x, y) \in \overleftrightarrow{ST} \tag{B.16}$$

The above equation constitutes the equation of the line $\overleftrightarrow{ST}$.

Definition B.2.1. *Let $\overleftrightarrow{ST}$ be a straight line. The **error function** designates the following function evaluated on any point (x, y) of the plan:*

$$Err_{ST}(x, y) = (y - y_S) \cdot \Delta x - (x - x_S) \cdot \Delta y$$

where $\Delta x = (x_T - x_S)$ and $\Delta y = (y_T - y_S)$.

A plane $\mathcal{P}$ is defined as a set of spatial points (x, y). The definition of the error function leads to the following property.

Property B.2.1. *A straight line* $\overleftrightarrow{ST}$ *divides a plane* $\mathcal{P} = \{(x, y) \in \mathbb{R}^2\}$ *into two three disjoint regions composed of:*

- *the straight line itself:* $\overleftrightarrow{ST} = \{(x, y) \in \mathbb{R}^2$ *such that* $Err_{ST}(x, y) = 0\}$
- *the* ***upper half-plane****:* $\mathcal{P}^+ = \{(x, y) \in \mathbb{R}^2$ *such that* $Err_{ST}(x, y) > 0\}$
- *the* ***lower half-plane****:* $\mathcal{P}^- = \{(x, y) \in \mathbb{R}^2$ *such that* $Err_{ST}(x, y) < 0\}$

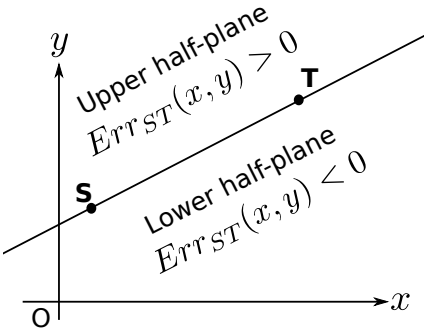
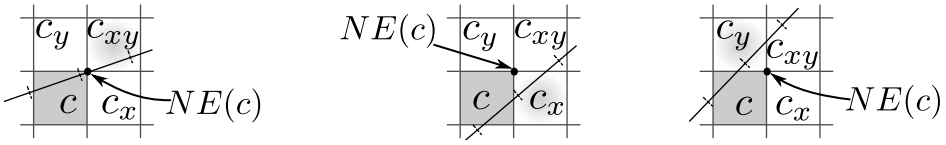


Figure B.3 – The upper and lower half-planes

The upper and lower half-planes are depicted on fig. B.3. Knowing that a cell c is traversed by $\overline{ST}$, the next traversed cell is among c_x, c_y and c_{xy}. The three cases are illustrated on fig. B.4. The symbol $NE(c)$ designate the North-East corner of cell c. The figure shows that:

- the next cell is c_{xy} if $NE(c)$ belongs to $\overline{ST}$,
- the next cell is c_x if $NE(c)$ belongs to the upper half-plane,
- the next cell is c_y if $NE(c)$ belongs to the lower half-plane.



(a) Next cell is c_{xy} (b) Next cell is c_x (c) Next cell is c_y

Figure B.4 – Next cell to be traversed knowing that cell c is already traversed

Algorithm 5 Continuous Traversal Algorithm

```
1: function TRAVERSE(Point (x_S, y_S), Point (x_T, y_T), Point (x_0, y_0), Point (x_1, y_1), Real β)
2:     ▷ (x_S, y_S): coordinates of point S
3:     ▷ (x_T, y_T): coordinates of point T
4:     ▷ (x_0, y_0): location of the cell containing the point S
5:     ▷ (x_1, y_1): location of the cell containing the point T
6:     ▷ β: length of a side of a squared cell
7:     Point (x, y)                          ▷ Location of the currently traversed cell c
8:     Real Δx = x_T − x_S
9:     Real Δy = y_T − y_S
10:    Real e                                ▷ Error function Err_ST(NE(c))
11:    (x, y) ← (x_0, y_0)
12:    e ← Err_ST(NE(x, y))
13:    while (x, y) ≠ (x_1, y_1) do
14:        if e = 0 then
15:            (x, y) ← (x + β, y + β)       ▷ c_xy is traversed
16:            e ← e + β × (Δx − Δy)
17:        else if e > 0 then
18:            x ← x + β
19:            e ← e − β × Δy                ▷ c_x is traversed
20:        else
21:            y ← y + β                     ▷ c_y is traversed
22:            e ← e + β × Δx
23:        end if
24:        ▷ The cell traversed at the current iteration is located at (x, y)
25:        SampleISM(x, y)
26:    end while
27: end function
```

Let us denote the next cell by c'. By applying Property B.2.1 (page 133), we get:

$$c' = \begin{cases} c_{xy} & \text{if } Err_{ST}(NE(c)) = 0 \\ c_x & \text{if } Err_{ST}(NE(c)) > 0 \\ c_y & \text{otherwise} \end{cases} \tag{B.17}$$

The above equation determines c' by evaluating the sign of $Err_{ST}(NE(c))$. It can be applied iteratively. To identify c', the sign of $Err_{ST}(NE(c))$ is evaluated. After that, the sign of $Err_{ST}(NE(c'))$ is evaluated to determine the next traversed cell after c', and so on.

Equation (B.17) implies:

$$NE(c') = \begin{cases} NE(c_{xy}) & \text{if } Err_{ST}(NE(c)) = 0 \\ NE(c_x) & \text{if } Err_{ST}(NE(c)) > 0 \\ NE(c_y) & \text{otherwise} \end{cases} \tag{B.18}$$

and then:

$$Err_{ST}(NE(c')) = \begin{cases} Err_{ST}(NE(c_{xy})) & \text{if } Err_{ST}(NE(c)) = 0 \\ Err_{ST}(NE(c_x)) & \text{if } Err_{ST}(NE(c)) > 0 \\ Err_{ST}(NE(c_y)) & \text{otherwise} \end{cases} \tag{B.19}$$

Let β be the length of a side of cell. By applying the initial definition of the error function on Definition B.2.1 (page 132), the error of the North-East corner of c_x, c_y and c_{xy} can be computed as a function of the error of the North-East corner of c. This techniques gives:

$$Err_{ST}(NE(c')) = \begin{cases} Err_{ST}NE(c) + \beta \cdot \Delta x - \beta \cdot \Delta y & \text{if } Err_{ST}(NE(c)) = 0 \\ Err_{ST}NE(c) - \beta \cdot \Delta y & \text{if } Err_{ST}(NE(c)) > 0 \\ Err_{ST}NE(c) + \beta \cdot \Delta x & \text{otherwise} \end{cases} \tag{B.20}$$

Finally, both eq. (B.17) and eq. (B.20) can be computed in an iterative manner. Both equations can constitute a traversal algorithm that finds out exactly the cells traversed by the line segment $\overline{ST}$.

The final continuous traversal algorithm is presented on Algorithm 5 (page 134). The algorithm takes as input the coordinates of both point S and point T, the locations of the cells containing both points, and the resolution of the discrete frame. At line 11, the first cell known to be traversed is the cell that contains the point S. The error variable is then initialized by the error of the North-East corner of that cell. Assume that a cell is located by its South-West corner. The North-East corner of the cell located at (x, y) is at coordinates $(x + \beta, y + \beta)$. Then, we have:

$$Err_{ST}(NE(x, y)) = Err_{ST}(x + \beta, y + \beta) \tag{B.21}$$

The continuous traversal algorithm has a unique main loop. Regarding the value of the error, lines 14 to 23 determine the location of the next traverse cell. The location of the cell is computed by applying eq. (B.17). The error is updated with respect to eq. (B.20). Line 25 calls a function $SampleISM()$ on the cell traversed at a given iteration of the algorithm. The loop iterates until the algorithm reaches the cell that contains the point T.

B.3 Generalization of the discrete range-mapping algorithm

The discrete traversal algorithm presented on Algorithm 4 (page 98) in Section 4.2.2.2 (page 96) finds out all cells traversed by a line segment $\overline{ST}$ provided that

$\Delta l \geq 0$ and $\Delta m \geq 0$. The following algorithm presents the general case, regardless of the sign of both Δl and Δm.

```
1: function TRAVERSE(Point (l_S, m_S), Point (l_T, m_T), Point (l_0, m_0), Point (l_1, m_1), Integer r)
2:     ▷ (l_S, m_S): coordinates of point S
3:     ▷ (l_T, m_T): coordinates of point T
4:     ▷ (l_0, m_0): location of the cell containing the point S
5:     ▷ (l_1, m_1): location of the cell containing the point T
6:     ▷ r: resolution of the discrete frame of reference
7:     Integer δl, δm, Δl, Δm
8:     Point (l, m)                          ▷ Location of the currently traversed cell c
9:     Integer e                             ▷ Error function Err_ST(NE(c))
10:    Δl ← l_T − l_S
11:    Δm ← m_T − m_S
12:    if Δl > 0 then
13:        δl ← r                            ▷ Traverse towards East.
14:    else
15:        δl ← −r                           ▷ Traverse towards West.
16:    end if
17:    if Δm > 0 then
18:        δm ← r                            ▷ Traverse towards North.
19:    else
20:        δm ← −r                           ▷ Traverse towards South.
21:    end if
22:    (l, m) ← (l_0, m_0)
23:    if sign(Δl) = sign(Δm) then
24:        if Δl ≥ 0 then
25:            e ← Err_ST(NE(x, y))
26:        else
27:            e ← Err_ST(SW(x, y))
28:        end if
29:        while (l, m) ≠ (l_1, m_1) do
30:            if e = 0 then
31:                (l, m) ← (l + δl, m + δm)          ▷ c_xy is traversed
32:                e ← e + δm × Δl − δl × Δm)
33:            else if e > 0 then
34:                l ← l + δl
35:                e ← e − δl × Δm                    ▷ c_x is traversed
36:            else
37:                m ← m + δm                         ▷ c_y is traversed
38:                e ← e + δm × Δl
39:            end if
40:            ▷ The cell traversed at the current iteration is located at (l, m)
```

```
41:             SampleISM(l, m)
42:         end while
43:     else
44:         if Δl ≥ 0 then
45:             e ← Err_ST(SE(x, y))
46:         else
47:             e ← Err_ST(NW(x, y))
48:         end if
49:         while (l, m) ≠ (l_1, m_1) do
50:             if e = 0 then
51:                 (l, m) ← (l + δl, m + δm)                ▷ c_xy is traversed
52:                 e ← e + δm × Δl − δl × Δm)
53:             else if e > 0 then
54:                 m ← m + δm                               ▷ c_y is traversed
55:                 e ← e + δm × Δl
56:             else
57:                 l ← l + δl
58:                 e ← e − δl × Δm                          ▷ c_x is traversed
59:             end if
60:             ▷ The cell traversed at the current iteration is located at (l, m)
61:             SampleISM(l, m)
62:         end while
63:     end if
64: end function
```

Appendix C

Résumé en français: Grille d'occupation entière

C.1 Introduction

Dans le domaine du transport moderne, un des défis primordiaux est la sécurité routière. Les accidents routiers coûtent des vies et ont des impacts significatifs sur la vie économique et sociale d'un pays. La consommation d'énergie et le respect de l'environnement deviennent aussi des défis majeurs dans l'ère du réchauffement climatique. Les voitures sont au centre de ces défis majeurs. Les voitures automatisées sont proposées comme étant une solution technologique pour améliorer la sécurité routière, pour réduire la consommation d'énergie et pour améliorer le respect de l'environnement.

Une voiture automatisée est une voiture équipée de plusieurs types de capteurs pour mettre en place des systèmes d'aide à la conduite ou pour la navigation autonome. Pour réaliser de tels systèmes, la voiture a besoin de d'observer et de surveiller en permanence l'environnement de conduite afin de le comprendre et de prendre des décisions de conduite par la suite. Par exemple, avant d'effectuer un virage, la voiture doit vérifier d'abord toutes les conditions requises (voie libre, feu vert, absence de risque de collision, *etc*) pour faire une telle action. Si les conditions sont satisfaites, le virage peut être effectué avec sécurité.

C.1.1 Perception multi-capteur

Les informations sur l'environnement de conduite sont obtenues à partir des lectures de capteurs perceptifs. Un **système de perception** interprète les données des capteurs afin de créer un modèle de l'environnement. Le **modèle d'environnement** est une représentation mathématique de l'environnement physique de conduite. Il localise l'emplacement des différents obstacles autour du véhicule et peut contenir

d'autres informations (localisation des espaces vides, identification des obstacles, couleurs, *etc*). Le modèle d'environnement sera par la suite utilisé pour prendre les décisions de conduite.

Plusieurs capteurs types de capteurs peuvent être utilisés pour réaliser la perception. Les capteurs les plus communs sont les radars, les LIDARs (Light Detection and Ranging), les capteurs de vision et les capteurs ultrasoniques. Ces capteurs sont capables d'estimer avec une précision limitée la position d'un obstacle par rapport à eux même.

Dans une voiture automatisées, plusieurs nombre de capteurs sont utilisés en même temps pour la tâche de perception. Chaque capteur délivre continuellement des mesures (distances, images, vitesses, *etc*) sur l'environnement de conduite. Les mesures des capteurs sont fusionnées continuellement et en temps-réel.

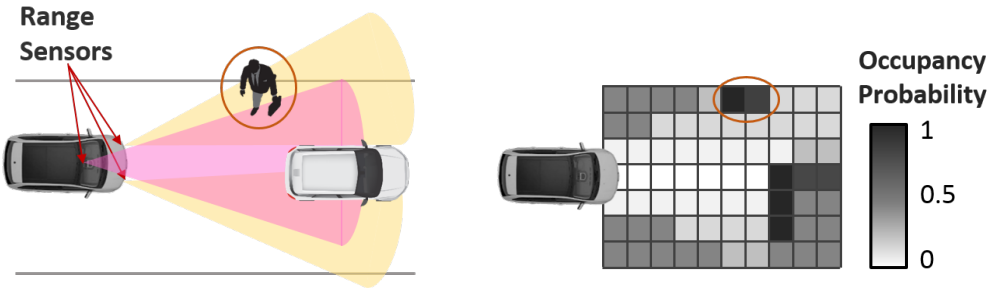
La fusion de multiple capteurs représente des avantages majeurs par rapport à l'utilisation d'un seul capteur. Premièrement, les capteurs ont leur limites physiques. Les mesures ont des incertitudes. La fusion permet de s'affranchir des incertitudes d'un seul capteur afin d'améliorer la robustesse et la fiabilité du système de perception. Deuxièmement, la fusion favorise une redondance d'information qui permet de s'affranchir au risque de défaillance d'un des capteurs. Enfin, l'utilisation de plusieurs capteurs permet de maximiser la couverture de l'environnement de conduite par les champs de vision des capteurs.

C.1.2 Grille d'occupation

Cette thèse se focalise sur la construction d'un modèle d'environnement appelé **grille d'occupation** à partir de la fusion de multiple capteurs. Les grilles d'occupation ont été introduites par Moravec et Elfes dans le milieu des années 80s ([Moravec 1985, Elfes 1987, Moravec 1988, Elfes 1989b, Elfes 1989a]). Dans le contexte de l'automobile, une grille d'occupation est une représentation de l'environnement de conduite sous forme de collection de cellules. Pour chaque cellule est estimée une **probabilité d'occupation** à partir des mesures de capteur. Plus simplement, la probabilité d'occupation d'une cellule vaut 1 si la cellule est occupée par un obstacle, 0 si elle est vide, 0.5 si l'état d'occupation de la cellule est inconnue. Les valeurs intermédiaires exprime l'incertitude sur l'état d'occupation de la cellule (plutôt vide ou plutôt occupée).

La Figure C.1 montre un exemple de grille d'occupation. La voiture automatisées se trouve à gauche de l'image. Les capteurs de perception montées sur la voiture observe l'environnent de devant et retourne des mesures (fig. C.1a). La grille d'occupation sur la fig. C.1b modélise l'environnement physique en une grille plate subdivisées en cellules carrées. La plateforme de grille d'occupation calcule la probabilité que chacune des cellules soit occupée par un obstacle en se basant sur les mesures des capteurs.

L'utilisation des grilles d'occupation comme modèle d'environnement confère plusieurs avantages. D'abord, la plateforme de grille d'occupation tient en compte en amont l'incertitude des capteurs. L'incertitude est reflétée par la suite au niveau



(a) L'environnement de conduite (b) La grille d'occupation correspondante

FIGURE C.1 – Un exemple d'environnement de conduite avec la grille d'occupation correspondante

des probabilités d'occupation des cellules. Ensuite, les grille d'occupation peuvent être calculées à partir des capteurs communément utilisés en robotique. Elles sont capables de fusionner des capteurs basés sur des technologies hétérogènes. Enfin, une grille d'occupation représente à la fois les obstacles, les régions vides et les régions inconnues des capteurs. La notion de cellule offre une bonne abstraction de tout type d'obstacles pouvant être rencontrés sur la route.

C.1.3 Objectif de la thèse

La grille d'occupation initialement développée par Moravec ([Moravec 1985]) est statique. Elle ne gère pas l'évolution de l'environnement au cours du temps et ne renferme donc pas d'information sur la dynamique des obstacles. D'autres algorithmes plus récents sur les grilles d'occupation permettent d'estimer la dynamique des obstacles au niveau des cellules. Un exemple de tel algorithme est la famille du BOF (Bayesian Occupancy Filter) développée au sein de l'INRIA Rhône-Alpes ([Coué 2006, Nègre 2014]). Le BOF fonctionne en deux étapes. D'abord, à chaque instant, une première étape de **fusion multi-capteur** ou "**Muti-sensor Fusion (MSF)**" fusionne tous les mesures provenant de tous les capteurs en une grille d'occupation instantanée statique. Par la suite, les grilles d'occupation instantanées passent à travers un filtre bayésien afin d'estimer la vitesse des obstacles au niveau des cellules. Avant cette thèse, l'objectif initial de l'INRIA Rhône-Alpes était de mettre en œuvre un système embarqué de perception basé sur la famille d'algorithme BOF. L'intégration de tels algorithmes passe par l'intégration des deux étapes, le MSF et le filtrage, dans un calculateur embarqué.

La présente thèse a pour objectif de réaliser l'intégration du MSF dans un calculateur embarqué. Le MSF est calculé par le module logiciel (SW) et matériel (HW) présenté sur la fig. C.2. Le module prend comme entrées les mesures instantanées provenant de multiple capteurs. Ces mesures sont par la suite instantanément fusionnées en une grille d'occupation.

L'intégration doit respecter les contraintes suivantes. Les grilles d'occupations

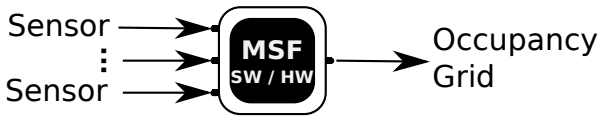


FIGURE C.2 – Le module de fusion multi-capteur

doivent être calculées en temps-réel. La référence temps-réel ici est la fréquence de lecture des capteurs. L'intégration doit être réalisée sur des matériels informatiques embarqués à bas coût. Un faible coût permet le déploiement de la fusion dans le marché de masse de l'automobile. L'intégration doit être réalisée sur des plateformes de calcul à basse consommation énergétique. L'intégration doit être sûre, robuste et maitrisée. L'incertitude des capteurs doit être prise en compte explicitement. Tout erreur numérique lors du calcul doit être connu et borné afin de valider l'exactitude de l'algorithme de fusion et de son implémentation sur du matériel embarqué.

C.1.4 Problèmes adressés

Respecter les contraintes posées ci-dessus constitue cependant un défis majeur. Comment calculer les grilles d'occupation en temps-réel sur du matériel embarqué à faible coût et à faible consommation d'énergie ? D'autant plus que des contraintes de qualité numérique, de robustesse et de sureté sont exigées.

En effet, la charge de calcul demandée par les grilles d'occupation dépend de la taille de la grille (mesurée en nombre de cellules), du nombre de mesures de capteurs et de la fréquence à laquelle ces mesures sont délivrées. Par exemple, une grille de 100 m-by-100 m avec des cellules de 10 cm de côtés contienne 1 Million de cellules. Une grille d'occupation basée sur une telle grille, calculée à partir de 2000 nombres de mesures de capteurs produites à 25 Hz, requiert d'effectuer 100 Milliards d'opérations par seconde. Ces paramètres sont typiques d'un cas d'utilisation réaliste pour des grilles d'occupation.

Pour effectuer de telle quantité d'opérations, les grilles d'occupation ont étés implémentées sur des stations de travail avec des CPUs et des GPUs (Graphical Processing Unit) avancés ([Yguel 2006, Homm 2010, Adarve 2012]). Ces matériels informatiques ne respectent cependant pas les contraintes de coût et de consommation énergétique. D'autres travaux ont essayé d'intégrer les grilles d'occupation sur des GPUs embarqués ([Nègre 2014]) et des many-cores embarquées ([Rakotovao 2015b, Rakotovao 2015a]). Toutefois, ces matériels ne sont pas encore certifiés pour être utilisés pour des applications critiques où les erreurs peuvent coûter des vies.

Les plateformes de calcul certifiées pour être sûres pour les applications automobiles sont basées sur des microcontrôleurs et des CPUs multi-coeurs embarqués([Heinecke 2004, Monot 2010]). Cependant, ces calculateurs ne disposent pas de puissance de calcul requise pour calculer les grilles d'occupation en temps-

réel. De plus, les grilles d'occupation sont basées sur des calculs de probabilités. Ces derniers requièrent une simulation de calculs de nombre réels sur le matériel informatique. La simulation introduit forcément des erreurs numériques qui ne dépendent pas de l'algorithme mais qui dépend plutôt du matériel sous-jacent. Ces erreurs peuvent être considérées comme petits, mais elles demeurent inconnues. L'objectif étant de maitriser les erreurs numériques, de nouveaux algorithmes de calcul des grilles d'occupation avec des erreurs numériques connues et maitrisées sont nécessaires.

C.1.5 Contributions de la thèse

Pour s'affranchir des problèmes ci-dessous, cette thèse présente la plateforme **grille d'occupation entière**. Cette plateforme sert à fusionner plusieurs capteurs en se basant sur les même principes probabilistes des grille d'occupation. L'originalité des grilles d'occupation entière est qu'elles permettent de calculer la fusion de manière beaucoup plus précise, sûre et robuste au niveau de l'intégration matérielle. Les contributions de la thèse sont les suivantes :

- la formulation du fondement mathématique des grilles d'occupation entière
- l'exploration des structures de données informatiques pour manipuler efficacement les grilles d'occupation entière
- le développement des algorithmes pour calculer les grilles d'occupation entière
- l'étude théorique et expérimentale des erreurs numériques introduites par les grilles d'occupation entière
- l'application des grille d'occupation entière pour fusionner des LIDARs montés sur un véhicule de test. La fusion est intégrée dans un calculateur embarqué à bas coût et faible consommation. Elle atteint une performance temps-réel sur des données d'expérience collectées dans des vrais trafics routiers en ville et sur autoroute.

Afin de présenter avec plus de détails les grilles d'occupation entière, la section suivante revisitera d'abord l'état de l'art sur les grilles d'occupation statique. Par la suite, les fondements mathématiques des grilles d'occupation entière seront détaillés, suivi de leur application pour la fusion multi-capteur pour l'automobile.

C.2 État de l'art sur les grilles d'occupation

Cette séction présente rapidement les méthodes dans l'état de l'art pour calculer les grilles d'occupation. Elle commence par une définition des grilles d'occupation, puis le calcul des grilles d'occupation grâce à un seul capteur. Ensuite, le calcul des grilles d'occupation multi-capteurs sera présenté, suivi des structures de données pour stocker les grilles d'occupations.

Les méthodes citées seront analysés sous deux axes : l'efficacité et la sureté. L'efficacité désigne la complexité de calcul ainsi que d'autres propriétés liés à l'implémentation des approches sur du matériel informatique embarqué. La sureté analyse les propriétés qui font qu'une méthode est sûre pour être utilisée pour les voitures automatisées.

C.2.1 Définition des grilles d'occupation

Considérons une région finie dans l'espace. Une **grille** est une subdivision de cette région en un nombre finie de sous-régions adjacentes. Une sous-région est appelée **cellule**. Dans ce qui suit, $\mathcal{G}$ désignera une grille et c_i une cellule dans la grille.

Une cellule a un état binaire. Elle est soit *occupée* soit *vide*. Une cellule est dite occupée si elle intersecte avec un obstacle, c'est-à-dire, si un ou plusieurs obstacles occupent partiellement ou totalement la sous-région correspondant à la cellule.

Quand une grille d'occupation modélise l'environnement de conduite d'une voiture, l'objectif est de déterminer l'état de chaque cellule. Pour ce faire, des capteurs de perception (LIDAR, radar, caméra, *etc*) sont montées sur le véhicule. Un capteur délivre des mesures sur l'environnent et les obstacles dans son champ de vision.

Soi z désigne une mesure d'un capteur. Le terme **modèle inverse de capteur** désigne par définition la probabilité $P(o_i|z)$. Il s'agit de la probabilité que la cellule c_i soit occupée sachant la mesure z du capteur. Quand plusieurs mesures provenant d'un ou de plusieurs capteurs sont disponibles, la probabilité $P(o_i|z_1 \wedge \ldots \wedge z_K)$ désigne la **probabilité d'occupation** de la cellule c_i sachant les K nombre de mesures.

Une **grille d'occupation** définit l'ensemble de tous les probabilités d'occupation des cellules, étant donné une ou plusieurs mesures de capteurs. Le nombre de probabilités dans une grille d'occupation est donc égal au nombre de cellule dans la grille. Notons qu'à partir d'une seule mesure, une grille d'occupation peut être construite. Une telle grille contient les valeurs des modèles inverses de capteur sur chaque cellule.

C.2.2 Modèle inverse de capteur

Considérons un cas simple de grille d'occupation à une dimension (1D). La figure C.3 montre l'exemple d'un capteur (sensor device) qui observe un obstacle le long d'un axe de vision (ray). Un obstacle est situé à une distance d du capteur. Après avoir observé l'obstacle, le capteur retourne une mesure sous forme d'un scalaire z. Le symbole d_i désigne la distance de la cellule c_i par rapport au capteur.

L'objectif consiste à calculer la probabilité $P(o_i|z)$. Plusieurs approches sont utilisées dans l'état de l'art. Elles peuvent être groupées en trois : l'approche bayésienne, l'approche analytique et l'approche basée sur les réseaux de neurones.

L'approche bayésienne calcule le modèle inverse de capteur en se basant sur des principes probabilistes, notamment la formule de Bayes ([Elfes 1989a]). Avec cette approche, le modèle inverse de capteur est calculé via le **modèle directe** de

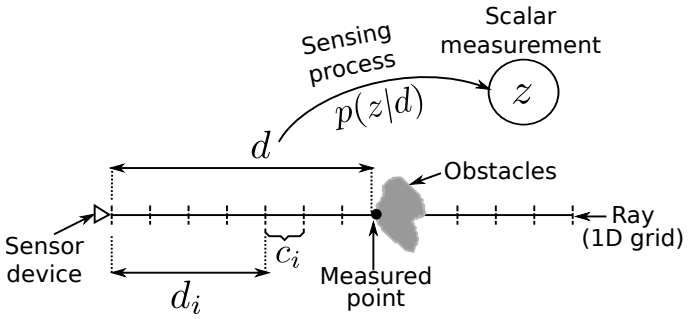


FIGURE C.3 – Modélisation d'un environnement le long d'un axe de vision

capteur. Le modèle directe désigne la densité de probabilité $p(z|d)$. Si le modèle directe modélise la réponse du capteur sachant l'état physique de l'environnement (qui est la distance d), le modèle inverse modélise l'état de l'environnement (qui est l'occupation d'une cellule) sachant la réponse du capteur.

Le modèle directe de capteur modélise les incertitudes des mesures. Ce densité de probabilité peut être calculé expérimentalement. Ainsi, l'approche bayésienne a l'avantage de tenir en compte explicitement les incertitudes des capteurs. Cette propriété permet aux grilles d'occupation d'être fidèle aux mesures de capteur, ce qui renforce la sureté du module de perception.

L'inconvénient de l'approche bayésienne est la complexité de temps qui est exponentiel en fonction du nombre de cellules. En pratique, une grille d'occupation même 1D contienne des centaines de cellules. Donc la complexité est difficilement intraitable en temps-réel.

Deux solutions sont généralement proposées dans l'état de l'art. La première consiste à modéliser directement le modèle inverse de capteur par des fonctions continues ([Payeur 1998, Gartshore 2002, Homm 2010, Einhorn 2011, Adarve 2012, Hornung 2013]). La seconde consiste modéliser le modèle inverse de capteur par des réseaux de neurones([Thrun 1993, Kortenkamp 1998, Thrun 2001b]). Dans les deux cas, la complexité devient constante, mais le modèle inverse n'est plus calculé à partir du modèle directe de capteur. Autrement dit, les incertitudes des capteurs ne sont plus tenues en compte de manière explicite, ce qui diminue la confiance sur la sureté du module de perception.

C.2.2.1 Probabilité d'occupation à partir de multiples mesures

Les méthodes pour calculer la probabilité d'occupation d'une cellule à partir de multiples mesures peuvent être classifiées en deux groupes. Le premier groupe assume que l'état d'une cellule est indépendant de celui de son voisin. Cette hypothèse est réfuté dans le deuxième groupe.

Avec l'hypothèse d'indépendance, le calcul de la probabilité d'occupation d'une cellule à partir de multiples mesures s'effectue en deux étapes. D'abord, un modèle inverse est calculé individuellement pour chacune des mesures. Ensuite une formule

de **fusion** est appliquée pour combiner les modèles inverses afin d'obtenir un unique probabilité d'occupation.

Trois formules de fusion sont les plus communes dans l'état de l'art : la fusion bayésienne ([Moravec 1988, Elfes 1989a]), la moyenne pondérée ([Thrun 1993, Adarve 2012]), et la politique du maximum ([Payeur 1997, Thrun 2005]). Soient z_1 et z_2 deux mesures. La fusion bayésienne sous l'hypothèse de non information ($P(o_i) = 1 - P(o_i) = 1/2$)) donne :

$$P(o_i|z_1 \wedge z_2) = F(P(o_i|z_1), P(o_i|z_2)) \quad \text{où} \quad F(x,y) = \frac{xy}{xy + (1-x)(1-y)} \tag{C.1}$$

La fusion bayésienne peut être aussi exprimé sous forme de logit :

$$l(o_i|z_1 \wedge z_2) = l(o_i|z_1) + l(o_i|z_2)) \quad \text{où} \quad l(x) = log\frac{P(x)}{1-P(x)} \tag{C.2}$$

La fusion bayésienne dispose des propriétés suivantes : le renforcement des mesures non conflictuelles et l'atténuation des mesures conflictuelles. Si deux mesures estiment qu'une cellule est occupée. Alors, la fusion résulte à une probabilité d'occupation supérieur aux modèles inverses calculées à partir de mesures individuelles. Si la cellule est plutôt vide selon les deux mesures, alors la fusion résulte à une probabilité inférieur aux modèles inverses. Par contre, si la cellule est plutôt occupée pour une des mesures et plutôt vide pour l'autre, alors, la fusion résultera à une probabilité qui tend vers 1/2. C'est-à-dire, l'état d'occupation de la cellule tend à être incertain vue que les mesures sont en conflit.

Le renforcement et l'atténuation sont deux propriétés qui permettent à un système de perception multi-capteur d'être sûre. Ces propriétés signifie que pour avoir une estimation correcte de l'environnement, il faut ajouter plus de nombre capteur ([Elfes 1989a]). Si les capteurs ne sont pas conflictuels (ce qui est supposé le cas en pratique sinon le système n'est pas fiable), la certitude sur l'état estimé de l'environnement est renforcée. En cas de capteurs conflictuels, la certitude diminue. La fusion par moyenne pondérée des modèles inverses de capteurs n'a pas la propriété de renforcement. La politique de fusion qui consiste à prendre la valeur maximale des modèles inverses ne supporte pas l'atténuation.

En terme d'efficacité, la version logit de la fusion bayésienne requière moins de calcul de la version manipulant directement les probabilités. Toutefois, quand il est nécessaire de récupérer la valeur de la probabilité qui correspond à un logit, il est nécessaire d'appliquer l'équation suivante :

$$P(o_i|z_1 \wedge z_2) = 1 - \frac{1}{1 + exp(l(o_i|z_1 \wedge z_2))} \tag{C.3}$$

Cette équation appelle à la fonction exponentielle. Du point de vue informatique, l'exactitude de telle fonction dépend de son implémentation et de la précision du matériel à simuler les opérations sur les nombres réels. Ces aspects ne sont pas toujours à la portée du programmeur. Donc, la précision du calcul des probabilités d'occupation n'est pas ajustable en amont.

Finalement, une méthode de fusion de multiple mesure appelée "*forward sensor model*" refute l'hypothèse que l'état d'occupation d'une cellule ne dépende pas de celui de ses voisins ([Thrun 2001a]). Cependant, la dépendance entre cellules au voisinage entraine une explosion combinatoire de la complexité de calcul des probabilités d'occupation. Cette approche n'est pas adapté pour les systèmes où une capacité à calculer les grilles d'occupation en temps-réel grâce à la fusion d'un grand nombre de mesures de capteurs est requise.

C.2.2.2 Structure de données pour les grilles d'occupation

Une fois calculées, les probabilités d'occupation doivent être stockées dans des structures données pour être utilisées par la suite par des applications de localisation, de suivi de cible, de navigation, *etc.* L'efficacité de ces applications dépendent donc de celle de la structure de données qui stocke les probabilités d'occupation. D'un point de vue informatique, ces applications émettent des requêtes à la structure de données pour récupérer la probabilité d'occupation d'une cellule, ou celles de cellules avoisinantes. Si la réponse à de telle requête est lente, cela ralentira aussi par conséquent les applications qui exploitent la grille.

Deux types de structure de données sont communément utilisées pour les grilles d'occupation : les tableaux, et les 2^d-arbres. Dans un tableaux, la probabilité d'une cellule est stockée dans un élément du tableaux. Cela permet un calcul rapide de la grille d'occupation. Cependant, les tableaux sont moins efficaces quand il s'agit de répondre à des requêtes avancées. Par exemple, récupérer les probabilités d'occupation le long d'une direction dans un tableaux requière d'effectuer plus de calcul ([Kambhampati 1986, Soucy 2004]).

Les 2^d-arbres s'avèrent plus efficaces pour ces genres de requêtes. Les 2^d-arbres utilisées pour les grilles d'occupation sont les quadtrees et les octrees ([Samet 1990]). Si la grille est de deux dimensions, un *quadtree* est utilisé. En trois dimensions, un *octree* est utilisé ([Wurm 2010, Hornung 2013]). Une feuille de ces arbres stocke au maximum une seule probabilité. Cette dernière peut cependant représenter les probabilités d'occupation de plusieurs cellules avoisinantes.

Cette technique permet donc un stockage plus compacte des grilles d'occupation. Cependant, cette compacité se fait avec perte puisque la probabilité stockée dans une feuille est en réalité une approximation qui représente les vraies probabilités d'occupation de plusieurs cellules avoisinantes. Autrement dit, une fois approximées, il devient impossible de récupérer les vraie valeurs des probabilités d'occupations à partir d'un arbre. Cette perte diminue la sureté d'un système de perception qui stockerai les probabilités d'occupation dans un 2^d-arbre.

C.3 Les grille d'occupation entière

La section précédente montre le manque de sureté et d'efficacité des méthodes pour calculer les grilles d'occupation. La présente section propose les **grilles d'occupation entières** pour rendre sûre et efficace le calcul des grilles d'occupation. Cette

section montre les fondements mathématiques des grilles d'occupation entières. Ces dernières permettent de calculer la fusion de capteurs grâce à de simple addition de nombre entier. Elles permettent de même une vraie compacité sans perte.

Les grilles d'occupation entières reposent sur la notion d'ensemble de probabilités dans le paragraphe suivante.

C.3.1 Ensemble de probabilités

Soit l'opérateur de fusion $\odot$ défini comme suit :

$$\begin{array}{rccc} \odot & : &]0,1[\times]0,1[& \mapsto &]0,1[\\ & & (p,q) & \mapsto & p \odot q = F(p,q) \end{array} \tag{C.4}$$

où la fonction F est celle introduite dans l'équation (C.1).

Un **ensemble de probabilités** S est un ensemble de nombre réels tel que :

$$\text{(Inclusion dans }]0,1[) \qquad S = \{p_n \in]0,1[, \forall n \in \mathbb{Z}\} \tag{C.5a}$$

$$\text{(Dénombrabilité)} \qquad \forall m,n \in \mathbb{Z} : p_m \neq p_n \Leftrightarrow m \neq n \tag{C.5b}$$

$$\text{(Stabilité)} \qquad \forall p_m, p_n \in S : p_m \odot p_n \in S \tag{C.5c}$$

La première équation signifie que les éléments de S appartiennent à l'intervalle $]0,1[$. La deuxième stipule qu'un ensemble de probabilités est dénombrable. Enfin, la troisième équation indique qu'un ensemble de probabilités est stable par rapport à l'opérateur de fusion. Cette propriété signifie que la fusion d'un élément de S avec un deuxième élément produit un troisième élément de S.

Soient p_m et p_n deux éléments d'un ensemble de probabilité. L'équation (C.5c) stipule que la fusion des deux éléments retourne un troisième élément noté par $p_{m\oplus n}$:

$$p_m \odot p_n = p_{m\oplus n} \tag{C.6}$$

L'opérateur $\oplus$ est appelé **opérateur de fusion entière**. Il combine les indexes des éléments à fusionner. Cet opérateur est associative et commutative.

Les ensembles de probabilités existent. Un exemple trivial est le singleton $S = 1/2$. La fusion de $1/2$ avec lui même étant $1/2$. Ce singleton n'est pas évidemment suffisant pour exprimer l'état occupé ou l'état vide d'une cellule. C'est pourquoi cette thèse introduit un ensemble de probabilités particulier appelé **ensemble de probabilités récursif**

C.3.2 Ensemble de probabilités récursif

L'ensemble de probabilités récursif est défini de manière récursive par le théorème 7.

Theorem 7. Ensemble de probabilités récursif

Soit $\varepsilon \in]0, 1/2[$. Soient $(a_n)_{n\in\mathbb{N}}$ et $(b_n)_{n\in\mathbb{N}}$ des suites définies comme suit :

$$a_n = \begin{cases} 1/2 & \text{si } n = 0 \\ 1/2 + \varepsilon & \text{si } n = 1 \\ a_{n-1} \odot a_1 & \text{sinon} \end{cases}$$

$$b_n = \begin{cases} 1/2 & \text{if } n = 0 \\ 1/2 - \varepsilon & \text{if } n = 1 \\ b_{n-1} \odot b_1 & \text{sinon} \end{cases}$$

Soit l'ensemble $S_\varepsilon = \{p_n, n \in \mathbb{Z}\}$ tel que :

$$p_n = \begin{cases} a_n & \text{si } n \geq 0 \\ b_{-n} & \text{sinon} \end{cases}$$

L'ensemble S_ε – appelé **ensemble récursif** – constitue un ensemble de probabilités tel que :

$$\forall m, n \in \mathbb{Z} : p_m \odot p_n = p_{m+n} \tag{C.7}$$

L'ensemble récursif possède un opérateur de fusion entière $\oplus$ tel que :

$$\forall m, n \in \mathbb{Z} : m \oplus n = m + n \tag{C.8}$$

Cela signifie qu'au lieu de calculer la fusion de p_m et p_n à partir de l'opérateur de fusion $\odot$, il suffit d'additionner les indexes m et n pour obtenir le résultat de la fusion.

L'ensemble récursif possède les propriétés suivantes. Premièrement, la valeur de p_0 vaut $1/2$. Les éléments avec un indexe négatif ont des valeurs inférieur à $1/2$. Ceux avec des indexes positifs sont supérieur à $1/2$. Deuxièmement, la fusion de p_n avec p_{-n} retourne $1/2$. C'est à dire que p_n est l'inverse de p_{-n} par rapport à l'opérateur de fusion $\odot$. Troisièmement, en partant de $1/2$ et en se déplaçant vers 0 ou 1, la distance entre deux éléments consécutifs décroit. Plus généralement, la distance entre deux éléments consécutifs est majorée par ε.

C.3.3 Définition des grilles d'occupation entière

Pour définir les grilles d'occupation entière, supposons que l'hypothèse suivante est vraie : les valeurs numériques des probabilités d'occupation appartiennent à un ensemble de probabilité.

Soit S un ensemble de probabilités. Soient $z_1 \wedge \ldots \wedge z_K$ des mesures de capteur, et $P(o_i|z_1 \wedge \ldots \wedge z_K)$ la probabilité d'occupation de la cellule c_i sachant les mesures des capteurs. Supposons que la valeur de $P(o_i|z_1 \wedge \ldots \wedge z_K)$ est égal à p_n, un élément de S. Alors, nous définissons par **indice d'occupation** de la cellule c_i sachant les

mesures $z_1 \wedge \ldots \wedge z_K$ le nombre entier relatif $I(o_i|z_1 \wedge \ldots \wedge z_K)$ tel que :

$$I(o_i|z_1 \wedge \ldots \wedge z_K) = n \Leftrightarrow P(o_i|z_1 \wedge \ldots \wedge z_K) = p_n \tag{C.9}$$

Par exemple, si la probabilité d'occupation d'une cellule vaut p_1, alors son indice d'occupation est égal à 1.

Une **grille d'occupation entière** désigne l'ensemble des indices d'occupation de toutes les cellules dans une grille. La différence entre les grilles d'occupation et les grilles d'occupation entière est donc trivial. Une grille d'occupation contient toutes les probabilités d'occupation des cellules d'une grille, tandis qu'une grille d'occupation entière contient touts les indices d'occupation de toutes les cellules.

L'avantage des grilles d'occupation entière est de permettre de calculer la fusion à partir d'un opérateur de fusion entier :

$$I(o_i|z_1 \wedge \ldots \wedge z_k) = I(o_i|z_1) \oplus \ldots \oplus I(o_i|z_k) \tag{C.10}$$

Cette équation signifie que pour calculer l'indice d'occupation d'une cellule sachant multiples mesures, il suffit de combiner les indices d'occupation sachant chaque mesure individuelle. Dans le cas de l'ensemble récursif, l'opérateur de fusion entier est équivalent à une addition. Par conséquent le calcul de l'indice d'occupation sachant multiples mesures devient :

$$I(o_i|z_1 \wedge \ldots \wedge z_k) = I(o_i|z_1) + \ldots + I(o_i|z_k) \tag{C.11}$$

L'équation (C.11) montre que, pour pouvoir utiliser les grilles d'occupation entière pour faire de la fusion, il faut une méthode pour calculer les indices d'occupation sachant les mesures mesures prises individuellement. Le calcul des indices d'occupation sachant une mesure nécessite de trouver un indice n tel que la valeur du modèle inverse de capteur $P(o_i|z)$ soit égal à p_n. Pour n'importe quelle valeur de la mesure z, il n'y a pas de raison qu'un tel n existe toujours. La valeur de $P(o_i|z)$ pourrait tomber entre deux éléments p_n et p_{n+1} de l'ensemble :

$$p_n \leq P(o_i|z) \leq p_{n+1} \tag{C.12}$$

Par conséquent, nous proposons de *quantifier* $P(o_i|z)$ par l'une des valeurs p_n ou p_{n+1}. Plusieurs politiques de quantification peut être adoptées : une quantification au plus proche, au plus grand ou une quantification qui tends vers $1/2$. Après la quantification, la valeur de l'indice d'occupation $I(o_i|z)$ devient n ou $n+1$ selon le résultat de la quantification.

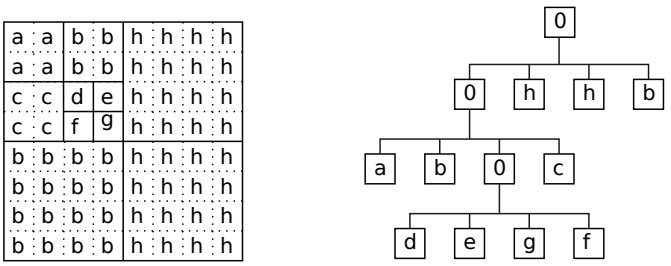
Finalement, le calcul des grilles d'occupation entière suit les étapes suivantes. D'abord, le modèle inverse est calculé puis quantifié. Cet étape est répété pour toutes les cellules, pour chaque mesure individuelle. Puis la fusion est calculée par l'équation (C.11).

La quantification introduit une erreur. Mais cette erreur est majorée par la distance maximale entre deux éléments consécutifs de l'ensemble récursif. Cette distance est inférieur au paramètre ε. Autrement dit, plus ε est petit, plus l'erreur de

quantification est minimisée. Remarquons cependant que le calcul de la fusion sur une architecture matérielle n'introduit pas d'erreur. En effet, dès que le matériel supporte le calcul des entiers relatifs, un tel calcul est exacte. La somme d'entier n'introduit pas d'erreurs numériques comme lors de la simulation des calculs sur les nombres réels sur les matériels informatiques.

C.3.4 Structure de donnée compacte pour les grilles d'occupation entière

Une grille d'occupation entière est donc un ensemble de nombre entiers relatifs qui sont les valeurs des indices d'occupation de chaque cellule. Pour stocker les grilles d'occupation entière, un tableau peut être utilisé. Chaque élément du tableau stockera un indice d'occupation d'une cellule.



(a) Grille d'occupation entière

(b) Un quadtree pour stocker la grille d'occupation entière

FIGURE C.4 – Exemple de grille d'occupation entière stockée dans un quadtree

La figure C.4a montre l'exemple d'une grille d'occupation entière définie sur une grille à deux dimensions. Les cellules sont limitées par les lignes pointillées et les lignes épaisses. Cette figure montre que dans une grille d'occupation entière, les cellules adjacentes peuvent avoir exactement les même indices d'occupation. Ainsi d'autres structures de données avancées peuvent être utilisées pour stocker de manière compacte les grilles d'occupation entière.

Cette thèse propose d'utiliser les *quadtree* pour les grilles 2D et les *octree* pour les grilles 3D. La figure C.4b montre l'exemple d'un quatree qui stocke la grille d'occupation entière de la figure C.4a. Les valeurs des indices d'occupation sont stockées uniquement au niveau des feuilles du quadtree. Les indices d'occupation des cellules adjacentes sont stockées sur la même feuille si et seulement si leurs valeurs sont égales.

Cette condition permet de garantir que la structure d'arbre ne modifie pas la valeur des indices d'occupation. En même temps, cette condition implique que le nombre de feuille dans l'arbre est inférieur ou égale au nombre de cellules. Ainsi, les quadtrees permettent une compacité sans perte des grilles d'occupation entières. Dans l'état de l'art, une compacité sans perte des grilles d'occupation n'existe pas

encore.

C.4 Application des grilles d'occupation entières pour la fusion de capteurs dans l'automobile

Une intégration logicielle des grilles d'occupation entières a été réalisée au cours de cette thèse. Le but est d'étudier expérimentalement les grilles d'occupation entières. Ces dernières servent à fusionner des LIDARs montés sur les pare-chocs avant et arrière d'un véhicule d'expérimentation. L'implémentation a été réalisée sur une plateforme matérielle embarquée et traite des données de trafics routiers réelles en ville et sur autoroute. L'environnement de conduite du véhicule d'expérimentation est modélisé sur une grille d'occupation entière à deux dimensions. L'efficacité et la sureté ont été étudiés.

C.4.1 Plateformes d'expérimentation

Les grilles d'occupation entières ont été testées sur une véhicule de l'Institut de Recherche Technologique (IRT) NanoElec ([IRT NanoElec]). Comme montré sur la figure C.5, la voiture est équipée de quatre LIDARs ibeo LUX, trois devant et un derrière. Un LIDAR renvoie des faisceau laser dans des directions fixées connue dans l'environnement. Les points d'impacts des lasers sur des obstacles de l'environnement sont alors retournés. Chaque LIDAR délivre plus de 800 points d'impacts à une fréquence de 25 Hz.

(a) Trois LIDARs ibeo LUX devant (b) Un LIDAR ibeo LUX derrière

FIGURE C.5 – Les quatres LIDARs ibeo LUX sur la voiture d'expérimentation

L'intégration logicielle des grilles d'occupation entière a été réalisée sur la carte de développement i.MX6. Il s'agit d'une carte développée par Freescale, basée sur un quad-coeur ARM cortex A9. Le processeur tourne à 1 GHz. Sur la carte est

installée un système d'exploitation à base de la distribution linux Ubuntu. La carte est conçue pour mettre au point rapidement des applications industrielles et dans le domaine de l'automobile.

C.4.2 Intégration logicielle des grilles d'occupations entières

L'objectif est de fusionner les mesures des LIDARs produites au cours d'une période sur une grille d'occupation entière à deux dimensions. Le véhicule est placée au milieu de la grille comme indiquée sur la figure C.6. Les faisceaux lasers produits par les quatres LIDARs sont représentés par les faisceaux rouges.

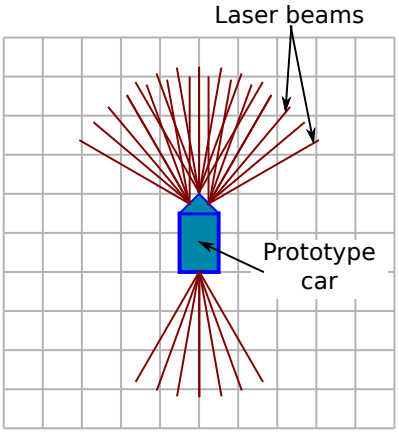


FIGURE C.6 – Vue d'en haut du véhicule d'expérimentation et les faisceau laser des quatre LIDARs

En absence de toute mesure de capteurs, les indices d'occupation des cellules sont égales à 0, correspondant à une probabilité d'occupation 0.5, soit une occupation inconnue. Le calcul des grilles d'occupation entières se fait en trois étapes.

- D'abord, pour un faisceau de laser, une grille d'occupation entière 1D le long du faisceau est calculé sachant le point d'impact correspondant au faisceau.
- Ensuite, la grille 1D est projeté sur la grille 2D. Seule les cellules 2D traversée par la grille 1D sont mis à jour.
- Enfin, la fusion est réalisées au niveau des cellules traversées. L'indice d'occupation d'une cellule 2D traversée est additionnée par l'indice d'occupation de la cellule 1D qui la traverse.

La première étape nécessite d'abord de calculer le modèle inverse de capteur le long de la grille 1D, puis d'appliquer la quantification. Une formule pour calculer le modèle inverse a été développée. Elle est basée sur l'approche bayésienne. Cette formule a permis de mettre en évidence la relation entre la taille des cellules, l'incertitude des capteurs et la valeur numérique des probabilités d'occupation. Cette

propriété rend sûr le calcul de la fusion des LIDARs du véhicule d'expérimentation. La méthode de calcul du modèle inverse est publiée dans [Dia 2017].

Le calcul de la projection des grilles 1D sur la grille 2D est coûteuse vue le nombre élevé des mesures à fusionner. Ce calcul doit se faire de manière la plus efficace possible sur une matérielle embarqué. Pour cela, un algorithme de traversée de grille 2D a été développé ([Rakotovao 2016b]. Cet algorithme de traversée utilise uniquement des calculs sur des nombres entiers. Cela implique un temps d'exécution rapide. En même temps, l'algorithme a été conçu de telle façon à ce que les erreurs de calculs soient maitrisée en amont et ajustables selon la précision spatiale voulue pour la traversée. Une telle propriété rend la traversée paramétrable avec une précision spatiale connue.

C.4.3 Résultats d'expérimentation et analyse

Pour analyser la performance de l'implémentation des grilles d'occupation entières, trois implémentations ont été réalisées. La première est une implémentation des grilles d'occupation entière où les indices d'occupation sont stockée dans un tableau. Dans la deuxième implémentation, les indices d'occupation sont stockée dans un quadtree. La troisième implémentation est une implémentation des grilles d'occupation standard où les probabilités d'occupation sont implémentées avec du calcul flottant. Il s'agit de l'implémentation classique des grilles d'occupation dans l'état de l'art. Une telle implémentation permet d'analyser la performance des grilles d'occupation entières.

La figure C.7 montre une scène de trafic urbain avec la grille d'occupation entière et la grille d'occupation standard correspondante. Les obstacles sur la scène correspondent aux cellules de couleur noire sur les grilles. Les cellules blanches sont plutôt vides. Celles en gris ont une occupation inconnues. Elles sont, soit cachées des capteurs par un obstacle, soit hors du champ de vision des capteurs. La différence de couleur entre les deux grilles montre que la grille d'occupation entière met en valeur de manière claire la différence entre une région couverte par un seul LIDAR (comme celle d'à gauche) et une région couverte par trois LIDARs (comme celle du centre). Sur la grille d'occupation standard, ces régions ont les même intensités de couleurs puisque leurs probabilités d'occupation sont toutes proches de 1.

L'exécution de l'implémentation des grille d'occupation entière à base de tableau atteint une performance temps-réel sur le matériel embarqué. En effet, le matériel est capable de fusionner les quatre LIDARs avec une fréquence de 28 Hz, tandis que les mesures des LIDARs sont produites à 25 Hz. Si la puissance électrique consommée par le processeur pendant le calcul de la fusion est prise en compte, l'implémentation des grilles d'occupation entières sur la plateforme embarquée est 1000 fois énergétique ment efficace par rapport aux implémentations des grilles d'occupation faites dans la littérature. Pour vérifier la qualité numérique des grilles d'occupation entières, une comparaison avec l'implémentation en calcul flottante des grilles d'occupation a été réalisée. La comparaison a montré une différence moyenne dans l'ordre du centième de la valeur choisie du paramètre ε. Ce résultat coïncide avec

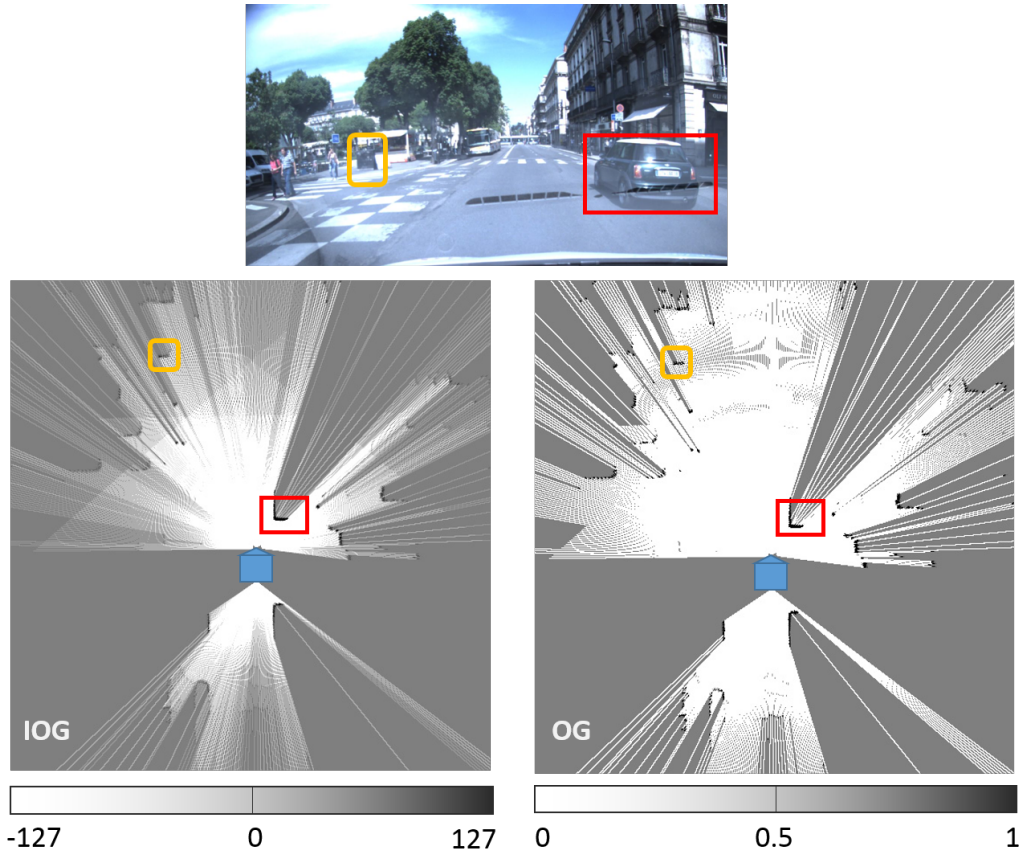


FIGURE C.7 – Exemple d'un scène de trafic en haut. La grille d'occupation entière correspondante est en bas à gauche. La grille d'occupation standard correspondante est en bas à droite.

le fait que plus ε est petit, plus l'erreur introduite au cours de la quantification est minimisée. Cela montre la précision numérique des grilles d'occupation entières, qui de plus est paramétrable.

Finalement, une implémentation des grilles d'occupation à base des quadtrees a été aussi réalisée. Cette implémentation a montré expérimentalement que l'arbre est effectivement plus compacte que le tableau pour stocker les grilles d'occupation entières. En effet, le nombre de noeuds dans l'arbre quadtree est jusqu'à trois fois moins que le nombre de cellules. Cependant, la construction et la maintenance de la structure d'arbre du quadtree est couteuse. Ces opérations internes au quadtree, nécessaires à sa maintenance, empêchent la fusion d'atteindre une performance temps-réel.

C.5 Conclusion

Cette thèse a introduit les grilles d'occupation entières. Ces dernières permettent de fusionner des mesures de capteurs télémétriques. Elles permettent ainsi de calculer de manière efficace un modèle d'environnement basé sur les grilles d'occupation.

Comme les grilles d'occupation traditionnelles, les grilles d'occupation entières sont un modèle d'environnement basé sur une grille. Chaque cellule de la grille peut être occupée ou vide. L'état d'occupation d'une cellule est estimée par une probabilité d'occupation dans une grille d'occupation, et par une indice d'occupation pour une grille d'occupation entière.

Une indice d'occupation correspond précisément à une unique valeur entre 0 et 1. La correspondance entre l'indice d'occupation et sa valeur est maintenue à travers la définition des ensembles de probabilités. Une indice d'occupation est un entier tandis qu'une probabilité d'occupation est un réel. La fusion basée sur les indices d'occupation nécessite donc uniquement des opération sur les nombres entiers. L'intégration des grilles d'occupation entières sur du matériel de calcul ont ainsi un temps d'exécution rapide et est efficace en terme de rapport de consommation d'énergie et vitesse de calcul.

Pour fusionner plusieurs mesures de capteur, les grilles d'occupation entières sont d'abord calculées à partir de chaque mesure prise individuellement. Après, les indices d'occupation sont fusionner cellule par cellule. Avec l'ensemble de probabilité récursif, la fusion est calculée via de simple addition d'entiers. Le calcul introduit une erreur qui est cependant connue, majorée et paramétrable.

Les grilles d'occupation peuvent être stockées dans des tableaux ou dans des structures d'arbres quadtree ou octree. L'indice d'occupation des cellules reste inchangé quelque soit la structure de donnée utilisée. Les structures d'arbre permet une compacité sans perte des grilles d'occupation entières. La maintenance de la structure d'arbre introduit cependant un coût de calcul additionnel qui augmente le temps de calcul des la fusion.

Pour conclure, cette thèse a proposé les grilles d'occupation entière en tant que nouvelle plateforme de calcul de la fusion des capteurs télémétriques de perception pour les voitures. La plateforme a été conçue en prenant en compte en amont les contraintes de sûreté et les contraintes embarquées de l'intégration matérielle/logicielle de la fusion. Elle permet l'intégration de la fusion dans du matériel embarqué, à bas-coût et à faible consommation d'énergie, tout en atteignant une performance en temps-réel.

Les grilles d'occupation entières permet de calculer la fusion bayésienne avec de simples additions d'entiers. Les erreurs numériques introduites sont connues, bornées et paramétrables. Cela permet de garantir la qualité, la sûreté et la robustesse de l'implémentation de la fusion, surtout quand cette dernière est réalisée pour des tâches critiques comme la perception dans l'automobile.

Bibliography

[Abou-Of 2016] Mona A. Abou-Of, Ahmed H. Taha and Amr A. Sedky. *Trade-off between low power and energy efficiency in benchmarking.* In 2016 7th International Conference on Information and Communication Systems (ICICS), pages 322–326. IEEE, apr 2016.

[Adarve 2012] Juan David Adarve, Mathias Perrollaz, Alexandros Makris and Christian Laugier. *Computing Occupancy Grids from Multiple Sensors using Linear Opinion Pools.* In IEEE ICRA, 2012.

[Aeberhard 2015] Michael Aeberhard, Sebastian Rauch, Mohammad Bahram, Georg Tanzmeister, Julian Thomas, Yves Pilat, Florian Homm, Werner Huber and Nico Kaempchen. *Experience, Results and Lessons Learned from Automated Driving on Germany's Highways.* IEEE Intelligent Transportation Systems Magazine, vol. 7, no. 1, pages 42–57, jan 2015.

[Amanatides 1987] John Amanatides and Andrew Woo. *A Fast Voxel Traversal Algorithm for Ray Tracing.* In In Eurographics '87, pages 3–10, 1987.

[Barth 2009] Alexander Barth, David Pfeiffer and Uwe Franke. *Vehicle tracking at urban intersections using dense stereo.* In 3rd Workshop on Behaviour Monitoring and Interpretation, BMI, pages 47–58, 2009.

[Bayes 1763] Thomas Bayes. *An Essay Towards Solving a Problem in the Doctrine of Chances.* Philosophical Transactions of the Royal Society of London, 1763.

[Behringer 2005] R. Behringer, W. Travis, R. Daily, D. Bevly, W. Kubinger, W. Herzner and V. Fehlberg. *RASCAL - An autonomous ground vehicle for desert driving in the DARPA grand challenge 2005.* volume 2005, pages 644–649, 2005. cited By 6.

[Berger 1985] James O. Berger. Statistical decision theory and bayesian analysis. Springer series in statistics. Springer, New York, NY [u.a.], 2. ed édition, 1985.

[Beyeler 2014] M. Beyeler, F. Mirus and A. Verl. *Vision-based robust road lane detection in urban environments.* In 2014 IEEE International Conference on Robotics and Automation (ICRA), pages 4920–4925, May 2014.

[Birdsall 2014] Michelle Birdsall. *Google and ITE: The Road Ahead for Self-Driving Cars.* ITE Journal, vol. 84, 2014.

[Bishop 1995] Christopher M. Bishop. Neural networks for pattern recognition. Oxford University Press, Inc., New York, NY, USA, 1995.

[Borenstein 1991] J. Borenstein and Y. Koren. *The vector field histogram-fast obstacle avoidance for mobile robots.* IEEE Transactions on Robotics and Automation, vol. 7, no. 3, pages 278–288, jun 1991.

[Bresenham 1965] J. E. Bresenham. *Algorithm for computer control of a digital plotter.* IBM Systems Journal, vol. 4, no. 1, pages 25–30, mar 1965.

[Broggi 1995] Alberto Broggi. *Massively parallel approach to real-time vision-based road markings detection.* pages 84–89, 1995.

[Broggi 2013] Alberto Broggi, Michele Buzzoni, Stefano Debattisti, Paolo Grisleri, Maria Chiara Laghi, Paolo Medici and Pietro Versari. IEEE Transactions on Intelligent Transportation Systems, vol. 14, no. 3, 2013.

[Burgard 1999] Wolfram Burgard, Armin B. Cremers, Dieter Fox, Dirk Hähnel, Gerhard Lakemeyer, Dirk Schulz, Walter Steiner and Sebastian Thrun. *Experiences with an interactive museum tour-guide robot.* Artificial Intelligence, vol. 114, no. 1-2, pages 3–55, oct 1999.

[Burgard 2008] Wolfram Burgard and Martial Hebert. World modeling, pages 853–869. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[Burke 2007] A. F. Burke. *Batteries and Ultracapacitors for Electric, Hybrid, and Fuel Cell Vehicles.* Proceedings of the IEEE, vol. 95, no. 4, pages 806–820, April 2007.

[Catling 1991] I. Catling and B. McQueen. *Road Transport Informatics in Europe-Major Programs and Demonstrations.* IEEE Transactions on Vehicular Technology, vol. 40, no. 1, pages 132–140, 1991. cited By 14.

[Christensen 2008] Henrik I. Christensen and Gregory D. Hager. Sensing and estimation, pages 87–107. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[Cit] *CityMobil2.* [Online]. Available at `www.citymobil2.eu/en`.

[Cleary 1988] John G. Cleary and Geoff Wyvill. *Analysis of an algorithm for fast ray tracing using uniform space subdivision.* The Visual Computer, vol. 4, no. 2, pages 65–83, mar 1988.

[Coué 2006] Christophe Coué, Cédric Pradalier, Christian Laugier, Thierry Fraichard and Pierre Bessiere. *Bayesian Occupancy Filtering for Multitarget Tracking: an Automotive Application.* International Journal of Robotics Research, vol. 25, no. 1, pages 19–30, January 2006.

[Danescu 2011] Radu Danescu, Florin Oniga and Sergiu Nedevschi. *Modeling and Tracking the Driving Environment With a Particle-Based Occupancy Grid.* IEEE Transactions on Intelligent Transportation Systems, vol. 12, no. 4, pages 1331–1342, dec 2011.

[Dargie 2015] Waltenegus Dargie. *A Stochastic Model for Estimating the Power Consumption of a Processor.* IEEE Transactions on Computers, vol. 64, no. 5, pages 1311–1322, may 2015.

[Dempster 1977] A. P. Dempster, N. M. Laird and D. B. Rubin. *Maximum likelihood from incomplete data via the EM algorithm.* JOURNAL OF THE ROYAL STATISTICAL SOCIETY, SERIES B, vol. 39, no. 1, pages 1–38, 1977.

[Dia 2017] Roxana Dia, Julien Mottin, Tiana Rakotovao, Diego Puschini and Suzanne Lesecq. *Evaluation of Occupancy-Grid Resolution through a Novel Approach for Inverse Sensor Modeling.* 20th IFAC, 2017.

[Einhorn 2010] Erik Einhorn, Christof Schröter and Horst-Michael Groß. *Building 2D and 3D adaptive-resolution occupancy maps using Nd-Trees*, nov 2010.

[Einhorn 2011] Erik Einhorn, Christof Schroter and Horst-Michael Gross. *Finding the adequate resolution for grid mapping - Cell sizes locally adapting on-the-fly.* In 2011 IEEE International Conference on Robotics and Automation, pages 1843–1848. IEEE, may 2011.

[element14] element14. *MCIMX6Q-SL: element14 development platform for i.MX 6Quad. Built to Freescale SABRE Lite design. User Manual v1.2.*

[Elfes 1987] A. Elfes. *Sonar-based real-world mapping and navigation.* IEEE Journal on Robotics and Automation, vol. 3, no. 3, pages 249–265, jun 1987.

[Elfes 1989a] A. Elfes. *Using occupancy grids for mobile robot perception and navigation.* Computer, vol. 22, no. 6, pages 46–57, jun 1989.

[Elfes 1989b] Alberto Elfes. *Occupancy Grids: A Probabilistic Framework for Robot Perception and Navigation.* PhD thesis, Carnegie Mellon University, Pittsburgh, PA, USA, 1989. AAI9006205.

[Enzweiler 2009] M. Enzweiler and D. M. Gavrila. *Monocular Pedestrian Detection: Survey and Experiments.* IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 31, no. 12, pages 2179–2195, Dec 2009.

[Ernst 1999] Stefan Ernst, Christoph Stiller, Jens Goldbeck and Christoph Roessig. *Camera calibration for lane and obstacle detection.* pages 356–361, 1999. cited By 18.

[Eskandarian 2012] Azim. Eskandarian. Handbook of intelligent vehicles. Springer, 2012.

[Fairfield 2007] Nathaniel Fairfield, Kantor George and David (Canergie Mellon University) Wettergreen. *Robotics Institute: Real-Time SLAM with Octree Evidence Grids for Exploration in Underwater Tunnels*, 2007.

[Finkel 1974] R. A. Finkel and J. L. Bentley. *Quad trees a data structure for retrieval on composite keys.* Acta Informatica, vol. 4, no. 1, pages 1–9, 1974.

[Foley 1990] James D. Foley, Andries van Dam, Steven K. Feiner and John F. Hughes. Computer Graphics, Principles and Practice. Addison-Wesley Publ. Comp., Reading, Massachusetts, 2nd édition, 1990.

[Fournier 2007] Jonathan Fournier, Benoit Ricard and Denis Laurendeau. *Mapping and Exploration of Complex Environments Using Persistent 3D Model.* In Fourth Canadian Conference on Computer and Robot Vision (CRV '07), pages 403–410. IEEE, may 2007.

[Freescale 2015] Freescale. *i.MX 6Dual/6Quad Applications Processors for Industrial Products. Document Number: IMX6DQIEC. Rev. 4.* Freescale Semiconductor Inc., July 2015.

[Gartshore 2002] R. Gartshore, A. Aguado and C. Galambos. *Incremental map building using an occupancy grid for an autonomous monocular robot.* In 7th International Conference on Control, Automation, Robotics and Vision, 2002. ICARCV 2002., volume 2, pages 613–618. Nanyang Technological Univ, 2002.

[Gindele 2009] Tobias Gindele, Sebastian Brechtel, Joachim Schroder and Rudiger Dillmann. *Bayesian Occupancy grid Filter for dynamic environments using prior map knowledge.* In 2009 IEEE Intelligent Vehicles Symposium, pages 669–676. IEEE, jun 2009.

[Guttman 1984] Antonin Guttman. *R-trees: A Dynamic Index Structure for Spatial Searching.* In Proceedings of the 1984 ACM SIGMOD International Conference on Management of Data, SIGMOD '84, pages 47–57, New York, NY, USA, 1984. ACM.

[Harris 2014] Mark Harris. *How Google's Autonomous Car Passed the First U.S. State Self-Driving Test*, 2014. [Online]. Available at http://spectrum.ieee.org/transportation/advanced-cars/how-googles-autonomous-car-passed-the-first-us-state-selfdriving-test.

[Hav] *HAVEit*. [Online]. Available at www.haveit-eu.org/.

[Heinecke 2004] Harald Heinecke, Klaus-Peter Schnelle, Helmut Fennel, Jürgen Bortolazzi, Lennart Lundh, Jean Leflour, Jean-Luc Maté, Kenji Nishikawa and Thomas Scharnhorst. *AUTomotive Open System ARchitecture - An Industry-Wide Initiative to Manage the Complexity of Emerging Automotive E/E-Architectures*. oct 2004.

[Himmelsbach 2009] M. Himmelsbach, T. Luettel and H. J. Wuensche. *Real-time object classification in 3D point clouds using point feature histograms*. In 2009 IEEE/RSJ International Conference on Intelligent Robots and Systems, pages 994–1000, Oct 2009.

[Hohm 2014] Andree Hohm, Felix Lotz, Oliver Fochler, Stefan Lueke and Hermann Winner. *Automated Driving in Real Traffic: from Current Technical Approaches towards Architectural Perspectives*. apr 2014.

[Holroyd 1990] FC Holroyd and DC Mason. *Efficient linear quadtree construction algorithm*. Image and Vision Computing, vol. 8, no. 3, pages 218 – 224, 1990.

[Homm 2010] Florian Homm, Nico Kaempchen, Jeff Ota and Darius Burschka. *Efficient occupancy grid computation on the GPU with lidar and radar for road boundary detection*. In IEEE IV, pages 1006–1013, June 2010.

[Hornung 2013] Armin Hornung, KaiM. Wurm, Maren Bennewitz, Cyrill Stachniss and Wolfram Burgard. *OctoMap: an efficient probabilistic 3D mapping framework based on octrees*. Autonomous Robots, vol. 34, no. 3, pages 189–206, 2013.

[HoseinNezhad 2002] R. HoseinNezhad, B. Moshiri and M.R. Asharif. *Sensor fusion for ultrasonic and laser arrays in mobile robotics: a comparative study of fuzzy, Dempster and Bayesian approaches*. In Proceedings of IEEE Sensors, pages 1682–1689. IEEE, 2002.

[Ibeo 2010] Ibeo Automotive Systems GmbH Ibeo. *Operating Manual ibeo LUX 2010 (R) Laserscanner v1.6*. 2010.

[Ibeo 2013] Ibeo Automotive Systems GmbH Ibeo. *Ibeo LUX - Technical facts*. 2013.

[IEA 2010] IEA. *Energy Efficiency Series – Transport Energy Efficiency*. International Energy Agency (IEA), 2010.

[IRT NanoElec] IRT NanoElec. *Institut de Recherche Téchnologique NanoElec*. [Online]. Available at www.irtnanoelec.fr.

[ISO 2011] ISO. *ISO 26262-1:2011(en) – Road vehicles - Functional safety - Part 6: Product development at the software level*, 2011. [Online]. Available at www.iso.org/obp/ui/#iso:std:iso:26262:-6:ed-1:v1:en.

[Jaynes 2003] E. T. Jaynes. Probability Theory: The Logic of Science. Cambridge University Press, April 2003.

[Jessup 2014] J. Jessup, S. N. Givigi and A. Beaulieu. *Robust and efficient multi-robot 3D mapping with octree based occupancy grids*. In 2014 IEEE International Conference on Systems, Man, and Cybernetics (SMC), pages 3996–4001. IEEE, oct 2014.

[Kambhampati 1986] S. Kambhampati and L. Davis. *Multiresolution path planning for mobile robots.* IEEE Journal on Robotics and Automation, vol. 2, no. 3, pages 135–145, Sep 1986.

[Khan 2015] Sheraz Khan, Dirk Wollherr and Martin Buss. *Adaptive rectangular cuboids for 3D mapping.* In 2015 IEEE International Conference on Robotics and Automation (ICRA), pages 2132–2139. IEEE, may 2015.

[Kolmogorov 1956] A. N. Kolmogorov. *Foundations of the Theory of Probability.* Chelsea Publishing Company, 1956.

[Konolige 1997] Kurt Konolige. *Improved Occupancy Grids for Map Building.* Auton. Robots, vol. 4, no. 4, pages 351–367, October 1997.

[Kortenkamp 1998] David Kortenkamp, R. Peter Bonasso and Robin Murphy, editors. Artificial intelligence and mobile robots: Case studies of successful robot systems. MIT Press, Cambridge, MA, USA, 1998.

[Kraetzschmar 2004] Gerhard K. Kraetzschmar, Guillem Pagès Gassull and Klaus Uhl. *Probabilistic Quadtrees for Variable-Resolution Mapping of Large Environments.* Proceedings of the 5th IFAC/EURON Symposium on Autonomous Vehicles, 2004.

[Laplace 1812] Pierre-Simon Laplace. Théorie analytique des probabilités. Courcier, Paris, 1812.

[Laugier 2011] C. Laugier, I. E. Paromtchik, M. Perrollaz, M. Yong, J. D. Yoder, C. Tay, K. Mekhnacha and A. Nègre. *Probabilistic Analysis of Dynamic Scenes and Collision Risks Assessment to Improve Driving Safety.* IEEE Intelligent Transportation Systems Magazine, vol. 3, no. 4, pages 4–19, winter 2011.

[Le Quotidient 2015] Le Quotidient. *Estimations de la population du Canada : âge et sexe, 1er juillet 2015* , September 2015.

[Li 2013] You Li and Yassine Ruichek. *Building variable resolution occupancy grid map from stereoscopic system - A quadtree based approach.* In 2013 IEEE Intelligent Vehicles Symposium (IV), pages 744–749. IEEE, jun 2013.

[Lorei 1999] Marcus Lorei and Christoph Stiller. *Visual sensing in electronic truck coupling.* volume 2, pages 303–307, 1999. cited By 0.

[Louvel 2014] M. Louvel, A. Molnos, J. Mottin, F. Pacull and T. Rakotovao. *Poster abstract: Distributed coordination of sub-systems power-modes and software-modes.* In Cyber-Physical Systems (ICCPS), 2014 ACM/IEEE International Conference on, pages 220–220, April 2014.

[Meagher 1982] Donald Meagher. *Geometric modeling using octree encoding.* Computer Graphics and Image Processing, vol. 19, no. 2, pages 129–147, jun 1982.

[Mekhnacha 2008] Kamel Mekhnacha, Yong Mao, David Raulo and Christian Laugier. *Bayesian occupancy filter based "Fast Clustering-Tracking" algorithm.* In IROS 2008, Nice, France, September 2008.

[Melpignano 2012] Diego Melpignano, Luca Benini, Eric Flamand, Bruno Jego, Thierry Lepley, Germain Haugou, Fabien Clermidy and Denis Dutoit. *Platform 2012, a Many-core Computing Accelerator for Embedded SoCs: Performance Evaluation of Visual Analytics Applications.* In Proceedings of the 49th Annual Design Automation Conference, DAC '12, pages 1137–1142, New York, NY, USA, 2012. ACM.

[Miller 2005] Jason Miller, Henry de Plinval and Kaijen Hsiao. *Mapping Contoured Terrain: A Comparison of SLAM Algorithms for Radio-Controlled Helicopters*, 2005.

[Moivre 1718] Abraham de Moivre. The Doctrine of Chances or a Method of Calculating the Probability of Events in Play. 1718.

[Monot 2010] Aurélien Monot, Nicolas Navet, Bernard Bavoux and Françoise Simonot-Lion. *Multicore scheduling in automotive ECUs*, may 2010.

[Moras 2014] J. Moras, V. Cherfaoui and P. Bonnifait. *Evidential grids information management in dynamic environments.* 17th International Conference on Information Fusion, pages 1–7, 2014.

[Moravec 1985] H.P. Moravec and A. Elfes. *High resolution maps from wide angle sonar.* In Proceedings. IEEE ICRA, volume 2, pages 116–121, Mar 1985.

[Moravec 1988] Hans Moravec. *Sensor Fusion in Certainty Grids for Mobile Robots.* AI Mag., vol. 9, no. 2, pages 61–74, July 1988.

[Moravec 1996] Hans P Moravec. *Robot spatial perceptionby stereoscopic vision and 3d evidence grids.* Perception, 1996.

[Nègre 2014] Amaury Nègre, Lukas Rummelhard and Christian Laugier. *Hybrid Sampling Bayesian Occupancy Filter.* In IEEE Intelligent Vehicles Symposium (IV), Dearborn, United States, June 2014.

[Nguyen 2009] T. N. Nguyen, M. M. Meinecke, M. Tornow and B. Michaelis. *Optimized grid-based environment perception in advanced driver assistance systems.* In Intelligent Vehicles Symposium, 2009 IEEE, pages 425–430, June 2009.

[Nguyen 2012] Thien-Nghia Nguyen, Bernd Michaelis, Ayoub Al-Hamadi, Michael Tornow and Marc-Michael Meinecke. *Stereo-Camera-Based Urban Environment Perception Using Occupancy Grid and Object Tracking.* IEEE Transactions on Intelligent Transportation Systems, vol. 13, no. 1, pages 154–165, mar 2012.

[Noykov 2007] Sv. Noykov and Ch. Roumenin. *Occupancy grids building by sonar and mobile robot.* Robotics and Autonomous Systems, vol. 55, no. 2, pages 162–175, 2007. cited By 22.

[Nuss 2015] Dominik Nuss, Ting Yuan, Gunther Krehl, Manuel Stuebler, Stephan Reuter and Klaus Dietmayer. *Fusion of laser and radar sensor data with a sequential Monte Carlo Bayesian occupancy filter.* In IEEE IV, 2015.

[NVIDIA 2015] NVIDIA. *White paper. NVIDIA Tegra X1. NVIDIA'S New Mobile Superchip*, 2015. [Online]. Available at www.nvidia.com.

[Oniga 2010] F. Oniga and S. Nedevschi. *Processing Dense Stereo Data Using Elevation Maps: Road Surface, Traffic Isle, and Obstacle Detection.* IEEE Transactions on Vehicular Technology, vol. 59, no. 3, pages 1172–1182, March 2010.

[Paliotta 2016] John Paliotta. *Paying off technical debt in safety-critical automotive software.* Embedded Computing Design. Connecting Silicon,Software, and Strategies for Intelligent Systems., May 2016.

[Paromtchik 1996] Igor E. Paromtchik and Christian Laugier. *Autonomous parallel parking of a nonholonomic vehicle.* pages 13–18, 1996.

[Pathak 2007] Kaustubh Pathak, Andreas Birk, Jann Poppinga and Soren Schwertfeger. *3D forward sensor modeling and application to occupancy grid based sensor fusion.* In 2007 IEEE/RSJ International Conference on Intelligent Robots and Systems, pages 2059–2064. IEEE, oct 2007.

[Payeur 1997] P. Payeur, P. Hebert, D. Laurendeau and C. M. Gosselin. *Probabilistic octree modeling of a 3D dynamic environment.* In Robotics and Automation, 1997. Proceedings., 1997 IEEE International Conference on, volume 2, pages 1289–1296 vol.2, Apr 1997.

[Payeur 1998] P. Payeur, D. Laurendeau and C.M. Gosselin. *Range data merging for probabilistic octree modeling of 3D workspaces.* In Proceedings. 1998 IEEE International Conference on Robotics and Automation (Cat. No.98CH36146), volume 4, pages 3071–3078. IEEE, 1998.

[Pfeiffer 2011] D. Pfeiffer and U. Franke. *Modeling Dynamic 3D Environments by Means of The Stixel World.* IEEE Intelligent Transportation Systems Magazine, vol. 3, no. 3, pages 24–36, Fall 2011.

[Pitteway 1967] M.L.V. Pitteway. *Algorithm for drawing ellipses or hyperbolae with a digital plotter.* Computer J., 10(3):282-289, 1967.

[Pomerleau 1989] Dean A. Pomerleau. *Advances in Neural Information Processing Systems 1.* chapter ALVINN: An Autonomous Land Vehicle in a Neural Network, pages 305–313. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1989.

[Pre] *PREVENT.* [Online]. Available at www.transport-research.info/project/preventive-and-active-safety-application.

[Press-Courier 1960] The Press-Courier. *Reporter Rides Driverless Car.* 7 June 1960.

[RAC-F 2014] RAC-F. *La transition énergétique du secteur des transports – Un plan d'action : comment financer l'exploitation des gisements d'efficacité énergétique du secteur ?* Réseau Action Climat France & Institut NégaWatt, May 2014.

[Rakotovao 2015a] T. Rakotovao, J. Mottin, D. Puschini and C. Laugier. *Real-time power-efficient integration of multi-sensor occupancy grid on many-core.* In 2015 IEEE International Workshop on Advanced Robotics and its Social Impacts (ARSO), pages 1–6, June 2015.

[Rakotovao 2015b] Tiana A. Rakotovao, Diego P. Puschini, Julien Mottin, Lukas Rummelhard, Amaury Negre and Christian Laugier. *Intelligent Vehicle Perception: Toward the Integration on Embedded Many-core.* In Proceedings of the 6th Workshop on Parallel Programming and Run-Time Management Techniques for Many-core Architectures, PARMA-DITAM '15, pages 7–12, New York, NY, USA, 2015. ACM.

[Rakotovao 2016a] T. Rakotovao, J. Mottin, D. Puschini and C. Laugier. *Multi-sensor fusion of occupancy grids based on integer arithmetic.* In 2016 IEEE International Conference on Robotics and Automation (ICRA), pages 1854–1859, May 2016.

[Rakotovao 2016b] Tiana Rakotovao, Julien Mottin, Diego Puschini and Christian Laugier. *Integration of Multi-sensor Occupancy Grids into Automotive ECUs.* In Proceedings of the 53rd Annual Design Automation Conference, DAC '16, pages 27:1–27:6, New York, NY, USA, 2016. ACM.

[Revelles 2000] J. Revelles, C. Ure na and M. Lastra. *An Efficient Parametric Algorithm for Octree Traversal.* In Journal of WSCG, pages 212–219, 2000.

[Ribo 2001] M. Ribo and A. Pinz. *A comparison of three uncertainty calculi for building sonar-based occupancy grids.* Robotics and Autonomous Systems, vol. 35, no. 3-4, pages 201–209, 2001. cited By 60.

[Rummelhard 2015] Lukas Rummelhard, Amaury Negre and Christian Laugier. *Conditional Monte Carlo Dense Occupancy Tracker.* 18th IEEE International Conference on Intelligent Transportation Systems, 2015.

[SAE 2014] SAE. *Automated Driving.* SAE international J3016, January 2014.

[Samet 1984] Hanan Samet. *The quadtree and related hierarchical data structures.* ACM Computing Surveys (CSUR), vol. 16, no. 2, pages 187–260, 1984.

[Samet 1988] H. Samet and R.E. Webber. *Hierarchical data structures and algorithms for computer graphics. I. Fundamentals.* IEEE Computer Graphics and Applications, vol. 8, no. 3, pages 48–68, may 1988.

[Samet 1990] Hanan Samet. The design and analysis of spatial data structures. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1990.

[Sar] *SARTRE.* [Online]. Available at www.sartre-project.eu.

[Schmid 2010] M. R. Schmid, M. Maehlisch, J. Dickmann and H. J. Wuensche. *Dynamic level of detail 3D occupancy grids for automotive use.* In Intelligent Vehicles Symposium (IV), 2010 IEEE, pages 269–274, June 2010.

[Shade 2011] R. Shade and P. Newman. *Choosing where to go: Complete 3D exploration with stereo.* In Robotics and Automation (ICRA), 2011 IEEE International Conference on, pages 2806–2811, May 2011.

[Siciliano 2008] Bruno Siciliano and Oussama. Khatib. Springer handbook of robotics. Springer, 2008.

[Soucy 2004] M. Soucy and P. Payeur. *Robot path planning with multiresolution probabilistic representations: a comparative study.* In Canadian Conference on Electrical and Computer Engineering 2004 (IEEE Cat. No.04CH37513), volume 2, pages 1127–1130. IEEE, 2004.

[Souza 2015] Anderson Souza and Luiz M. G. Gonçalves. *Occupancy-elevation grid: an alternative approach for robotic mapping and navigation.* Robotica, vol. FirstView, pages 1–18, 4 2015.

[Stepan 2005] P. Stepan, M. Kulich and L. Preucil. *Robust data fusion with occupancy grid.* IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews), vol. 35, no. 1, pages 106–115, Feb 2005.

[Stiller 1997] Christoph Stiller, Werner Poechmueller and Bernd Huertgen. *Stereo vision in driver assistance systems.* pages 888–893, 1997. cited By 2.

[Stroia 2015] Iulian Stroia, Lucian Itu, Cosmin Nita, Laszlo Lazar and Constantin Suciu. *GPU accelerated geometric multigrid method: Performance comparison on recent NVIDIA architectures.* In 2015 19th International Conference on System Theory, Control and Computing (ICSTCC), pages 175–179. IEEE, oct 2015.

[Sun 2006] Z. Sun, G. Bebis and R. Miller. *On-road vehicle detection: A review.* IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 28, no. 5, pages 694–711, 2006. cited By 572.

[Tao 2013] Z. Tao, P. Bonnifait, V. Fremont and J. Ibanez-Guzman. *Lane marking aided vehicle localization.* pages 1509–1515, 2013. cited By 6.

[Thorpe 1997] C. Thorpe, T. Jochem and D. Pomerleau. *The 1997 automated highway free agent demonstration.* In Proceedings of Conference on Intelligent Transportation Systems, pages 496–501. IEEE, 1997.

[Thrun 1993] Sebastian Thrun. *Exploration and Model Building in Mobile Robot Domains.* In In Proceedings of the IEEE International Conference on Neural Networks, 1993.

[Thrun 1998] Sebastian Thrun. *Learning metric-topological maps for indoor mobile robot navigation.* Artificial Intelligence, vol. 99, no. 1, pages 21–71, feb 1998.

[Thrun 2001a] Sebastian Thrun. *Learning Occupancy Grid Maps with Forward Sensor Models.* Proceedings of the Conference on Intelligent Robots and Systems (IROS), 2001.

[Thrun 2001b] Sebastian Thrun, Dieter Fox, Wolfram Burgard and Frank Dellaert. *Robust Monte Carlo localization for mobile robots.* Artificial Intelligence, vol. 128, no. 1-2, pages 99–141, may 2001.

[Thrun 2003] Sebastian Thrun. *Exploring Artificial Intelligence in the New Millennium.* chapter Robotic Mapping: A Survey, pages 1–35. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2003.

[Thrun 2005] Sebastian Thrun, Wolfram Burgard and Dieter Fox. Probabilistic robotics (intelligent robotics and autonomous agents). The MIT Press, 2005.

[Thrun 2006] Sebastian Thrun*et al.* *Stanley: The Robot That Won the DARPA Grand Challenge: Research Articles.* J. Robot. Syst., vol. 23, no. 9, pages 661–692, September 2006.

[Triebel 2006] R. Triebel, P. Pfaff and W. Burgard. *Multi-Level Surface Maps for Outdoor Terrain Mapping and Loop Closing.* In 2006 IEEE/RSJ International Conference on Intelligent Robots and Systems, pages 2276–2282, Oct 2006.

[Ulmer 1994] Berthold Ulmer. *VITA II - active collision avoidance in real traffic.* pages 1–6, 1994.

[Urmson 2008] Chris Urmson*et al.* *Autonomous Driving in Urban Environments: Boss and the Urban Challenge.* J. Field Robot., vol. 25, no. 8, pages 425–466, August 2008.

[Van Aken 1984] J.R. Van Aken. *An efficient ellipse-drawing algorithm.* CG & A, 4(9):24-35, 1984.

[Vatavu 2012] A. Vatavu and S. Nedevschi. *Real-time modeling of dynamic environments in traffic scenarios using a stereo-vision system.* In 2012 15th International IEEE Conference on Intelligent Transportation Systems, pages 722–727, Sept 2012.

[Vatavu 2014] Andrei Vatavu, Radu Danescu and Sergiu Nedevschi. *Modeling and tracking of crowded traffic scenes by using policy trees, occupancy grid blocks and Bayesian filters.* In 17th International IEEE Conference on Intelligent Transportation Systems (ITSC), pages 1948–1955. IEEE, oct 2014.

[Weiss 2007] Thorsten Weiss, Bruno Schiele and Klaus Dietmayer. *Robust Driving Path Detection in Urban and Highway Scenarios Using a Laser Scanner and Online Occupancy Grids.* In 2007 IEEE Intelligent Vehicles Symposium, pages 184–189. IEEE, jun 2007.

[WHO 2015] WHO. *Global Status Report on Road Safety.* World Health Organization (WHO), 2015.

[Wurm 2010] Kai M. Wurm, Armin Hornung, Maren Bennewitz, Cyrill Stachniss and Wolfram Burgard. *OctoMap: A probabilistic, flexible, and compact 3D map representation for robotic systems.* In In Proc. of the ICRA 2010 workshop, 2010.

[Yeomans 2014] Gillian Yeomans. *Autonomous vehicles - Handing over control: Opportunities and risks for insurance.* Lloyd's, pages 1–27, 2014.

[Yguel 2006] 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, pages 105–110. IEEE, oct 2006.

[Yguel 2008] Manuel Yguel, Olivier Aycard and Christian Laugier. *Update Policy of Dense Maps: Efficient Algorithms and Sparse Representation.* In Field and Service Robotics, pages 23–33. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[Yu 2015] C. Yu, V. Cherfaoui and P. Bonnifait. *Evidential occupancy grid mapping with stereo-vision.* volume 2015-August, pages 712–717, 2015. cited By 0.

[Ziegler 2014] J. Ziegler*et al.* *Making bertha drive-an autonomous journey on a historic route.* IEEE Intelligent Transportation Systems Magazine, vol. 6, no. 2, pages 8–20, 2014. cited By 82.