# Lifelong Exploratory Navigation : integrating planning, navigation and SLAM for autonomous mobile robots with finite resources

Fabrice Mayran de Chamisso

NNT : 2016SACLS413

# THESE DE DOCTORAT
DE
L'UNIVERSITÉ PARIS-SACLAY
PRÉPARÉE À
L'UNIVERSITÉ PARIS-SUD

ECOLE DOCTORALE N° 580
Sciences et technologies de l'information et de la communication

Spécialité robotique

Par

**Monsieur Fabrice Mayran de Chamisso**

## Lifelong Exploratory Navigation
Integrating planning, navigation and SLAM
for autonomous mobile robots with finite resources

**Thèse présentée et soutenue à Palaiseau, le 18 novembre 2016**

**Composition du Jury :**

| | | |
|---|---|---|
| M. Filliat David | Professeur, ENSTA ParisTech | Président |
| M. Chatila Raja | Directeur de recherche, Université Pierre et Marie Curie | Rapporteur |
| M. Lacroix Simon | Directeur de recherche, LAAS - CNRS | Rapporteur |
| M. Rives Patrick | Directeur de recherche, INRIA Sophia Antipolis | Examinateur |
| Mme. Serfaty Véronique | Docteur, correspondante Direction Générale de l'Armement | Examinatrice |
| M. Aupetit Michaël | Chercheur HDR, Qatar Computing Research Institute | Directeur de thèse |
| M. Soulier Laurent | Ingénieur de recherche, CEA LIST | Encadrant |

université PARIS-SACLAY

ÉCOLE DOCTORALE
Sciences et technologies de l'information et de la communication (STIC)

**Titre :** Navigation exploratoire au long de la vie : une approche intégrant planification, navigation, cartographie et localisation pour des robots mobiles disposant de ressources finies

**Mots clés :** robot, autonome, intelligence, apprentissage, SLAM, planification

**Résumé :** Il est fondamental pour un robot d'être capable de se déplacer de manière complètement autonome afin d'accomplir une mission qui lui a été confiée, et ce avec un budget énergétique fini, dans un laps de temps contraint et sans connaissances préalables de l'environnement. Afin d'atteindre un objectif dans le plan ou l'espace, un robot doit à minima être capable d'accomplir quatre tâches : maintenir une représentation abstraite de l'environnement (une carte), être capable de se localiser à l'intérieur de cette représentation, utiliser la représentation pour planifier des itinéraires et naviguer le long de la trajectoire prévue tout en s'adaptant aux dynamiques de l'environnement et en évitant les obstacles. Chacun de ces problèmes a été étudié par la communauté de la robotique. Cependant, ces quatre composants sont en général étudiés séparément et sont par conséquent incompatibles entre eux pour l'essentiel. De plus, étant donné qu'humains et robots ne disposent que de ressources computationelles et mémorielles finies, les algorithmes de planification, navigation et SLAM devraient être capables de fonctionner avec des données incomplètes ou compressées tout en garantissant que le ou les objectifs fixés soient atteints. Dans cette thèse, la planification, la navigation et le SLAM dans des environnements arbitrairement grands et avec des ressources computationelles et mémorielles finies sont vues comme un seul problème, créant un nouveau paradigme que nous appelons Navigation Exploratoire au long de la Vie ou Lifelong Exploratory Navigation.

**Title:** Lifelong Exploratory Navigation: integrating planning, navigation and SLAM for autonomous mobile robots with finite resources

**Keywords:** robot, autonomous, intelligence, learning, SLAM, planning

**Abstract:** One of the yet unresolved canonical problems of robotics is to have robots move completely autonomously in order to accomplish any mission they are charged with, with time and resource constraints and without prior knowledge of the environment. Reaching a goal requires the robot to perform at least four tasks: maintaining an abstract representation of the environment (map), being able to localize itself within this representation, using the representation to plan paths and navigating on the planned paths while handling dynamics of the environment and avoiding obstacles. Each of these problems has been studied extensively by the robotics community. However, the four components are usually studied separately, and as a result are mostly incompatible with each other. Additionally, since humans as well as robots have to operate with finite memory and computing resources, long running planning, navigation and SLAM algorithms may have to operate on incomplete or compressed data while guaranteeing that the goal(s) can still be reached. In this thesis, planning, navigation and SLAM in arbitrarily large environments with finite computing resources and memory are considered as one single problem, for a new bio-inspired paradigm which we call Lifelong Exploratory Navigation.

**Université Paris-Saclay**
Espace Technologique / Immeuble Discovery
Route de l'Orme aux Merisiers RD 128 / 91190 Saint-Aubin, France

# Thanks

This thesis is dedicated to the late Janine Thévenet, who hosted me for multiple years and ended up passing away just two days after I came back from presenting my work at IJCAI. Janine did not like maths (especially three-digit divisions) and had absolutely no idea what I could be doing all day.

I would like to thank all the people who supported me during my PhD years and especially during the last months where work started to pile up. This includes my parents who supported my bad mood during the weekends and had me do some gardening to stop thinking about the thesis. At the end, the garden got flooded and all the work got lost. I prefer this to a hard drive failure, though. I also thank the people from the Algorithm and Architecture Codesign Laboratory who supported me during the rest of the week while maintaining a constant level of wry humor and providing numerous gif comics. Thanks to Laurent Soulier for trying to help me solve hard probability problems but ending up filling whiteboards with equations resulting in stuff completely unrelated to the original question (note that I did no better and ended up proving lots of weird theorems and designing spiking neuron Turing machines). Thanks to Grégory Vaumourin for making a lot of posters for conferences, which I got jealous of, encouraging me to design better posters, and to Karl-Eduard Berger for the interesting discussions on various stuff (from finding algorithmic complexities to origami peacocks). Thanks to Pierre Sauleau for programming the core of the interface between my code and the Robot Operating System (ROS), also introducing a bug related to the export folder which made me lose one important robot run. Kudos also to Michaël Aupetit for suggesting an impressive amount of very accurate corrections for each single line of text I wrote, occasionally correcting himself multiple times in the process.

I have to thank my computer for having raised the temperature of the office by around two degrees Celsius during the heat wave doing nothing but an anti-virus scan (there was of course no virus but that thing runs once a week, always at the moment where I need computing power and fast disk access). Working in a sauna is very pleasing and you have absolutely no problem typing or writing on paper (stolen from the printer) with wet hands. I thank DGA (French equivalent to DARPA) for providing financing and applications to my work. I finally thank my doctoral school and the university of Paris-Saclay for forcing me to undergo professional and scientific training and trying to bury me under red tape. Both of which may prove useful later, I guess.

# Contributions and structure of this thesis

## 1 Structure

This thesis is structured in three parts:

- **Part I**: introducing Lifelong Exploratory Navigation (LEN) as a new paradigm inspired by animal behaviors, and studying why it is necessary.
- **Part II**: describing the integration of Planning, Navigation and SLAM into a single paradigm called PNSLAM.
- **Part III**: implementing resource management into PNSLAM, as required for LEN of mobile robots, and concluding on the approach.

## 2 Contributions

***Introducing PNSLAM and Lifelong Exploratory Navigation*** **(Part I, chapter 1)** explains why Planning, Navigation and Simultaneous Localization and Mapping (SLAM) should be integrated for autonomous operation of mobile robots. The combined problem is called PNSLAM. Chapter 1 also introduces Lifelong Exploratory Navigation as a new paradigm allowing PNSLAM in robots running for a long time in environments whose size may exceed their memory and processing capacities.

***Lifelong Exploratory Navigation - a system view*** **(Part I, chapter 2)** studies how Lifelong Exploratory Navigation is performed by animals and introduces a mobile robot software architecture compatible with Lifelong Exploratory Navigation.

***Exploratory Planning*** **(Part II, chapter 3)** introduces a new planning algorithm compatible with PNSLAM, EDNA*. This chapter builds on the IJCAI-15 paper "Exploratory Digraph Navigation Using A*" (Mayran de Chamisso, Soulier, and Aupetit, 2015) and the associated patent PCT/FR2016051039. The IJCAI paper can be cited as:

> Fabrice Mayran de Chamisso, Laurent Soulier and Michaël Aupetit (2015). "Exploratory Digraph Navigation using A*". In: *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence*. IJCAI. AAAI Press /

International Joint Conference on Artificial Intelligence. URL: http://ijcai.org/proceedings/2015.

***Navigation, obstacle avoidance and topology extraction*** **(Part II, chapter 4)** introduces topology-based navigation and topology extraction for PNSLAM and builds on the RFIA/RFP 2016 paper "Robust topological skeleton extraction from occupancy grids for mobile robot navigation", which can be cited as:

Fabrice Mayran de Chamisso, Laurent Soulier and Michaël Aupetit (2016). "Robust topological skeleton extraction from occupancy grids for mobile robot navigation". In *Proceedings of the twentieth national congress on Shape Recognition and Artificial Intelligence*. RFIA'16

***A hybrid metrical/topological SLAM for Lifelong Exploratory Navigation*** **(Part II, chapter 5)** develops a SLAM framework for PNSLAM. This chapter builds on a paper called "A rigorous hybrid metrical-topological SLAM framework for autonomous mobile robot navigation in large-scale highly cyclic environments" submitted to the *International Journal of Robotics Research*.

***Assembling PNSLAM components - experiments*** **(Part II, chapter 6)** presents PNSLAM simulations and robot experiments using the components described in part II. We intend to publish these experiments along with chapter 5.

***Handling finiteness of computing power*** **(Part III, chapter 7)** studies the computing power required by components described in part II and describes a refinement of chapter 5 bringing the worst-case algorithmic complexity to constant in the size of the environment the robot evolves in.

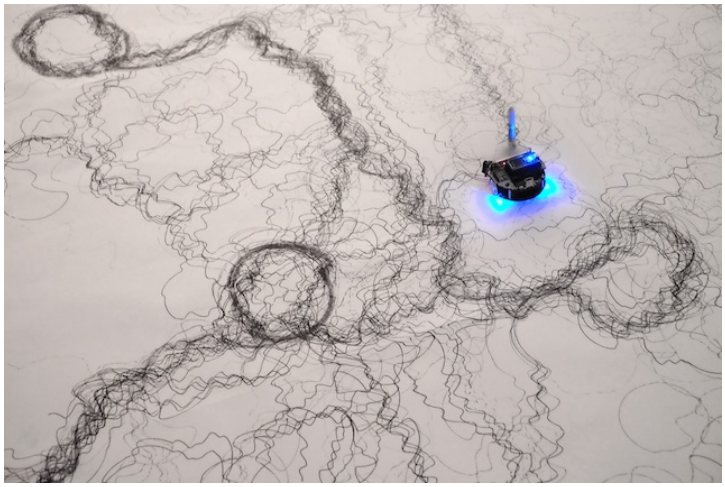
***Handling finiteness of memory*** **(Part III, chapter 8)** studies lossy compression of the map of an environment stored in memory, with the aim to achieve a high compression ratio with minimal impact on navigation performances. This chapter builds on a paper called "Lossy graph compression for efficient navigation using smart exploration" submitted to *Artificial Intelligence*.

***Conclusion*** **(Part III, chapter 9)** concludes on LEN and proposes improvements to the approach that could be implemented in later works.

# Contents

# Part I

# The Lifelong Exploratory Navigation paradigm



A robot builds niches in the environment, mimicking the behavior of rodents and other animals in a primitive form of **Lifelong Exploratory Navigation**.
Picture: J. McCormack (2010; 2009) for the IJCAI 2015 "AI and the arts" exhibition, Buenos Aires, http://jonmccormack.info/

# 1 Introducing PNSLAM and Lifelong Exploratory Navigation

Mobile robots are currently used for a large number of missions: go to a single location, find something or someone, explore an environment, draw a map of an environment... In this chapter, we show that the lack of integration between planning, navigation and SLAM is a major hurdle in achieving these complex tasks autonomously.

## 1.1 Mobile robots and the challenges of movement

### 1.1.1 Robots are moving entities with reasoning capacities

Giving a definition of what a *robot* is and understanding what makes it different from a *machine* or *computer* is a key aspect in understanding *what a robot can do* and *how it should do it*. Rather than using peer reviewed research papers to try and derive a definition, let us use some commonly admitted definitions to try and derive what most people expect of robots, expressed in everyday language on Wikipedia :

- "A *robot* is a mechanical or virtual artificial agent, usually an electro-mechanical machine that is guided by a computer program or electronic circuitry." (https://en.wikipedia.org/wiki/Robot),
- "A *machine* is a tool containing one or more parts that uses energy to perform an intended action." (https://en.wikipedia.org/wiki/Machine),
- "A *computer* is a general purpose device that can be programmed to carry out a set of arithmetic or logical operations automatically." (https://en.wikipedia.org/wiki/Computer) and
- "An *embedded system* is a computer system with a dedicated function within a larger mechanical or electrical system, often with real-time computing constraints." (https://en.wikipedia.org/wiki/Embedded_system).

Looking for these words in a dictionary or encyclopedia such as Universalis or Britannica does not result in significantly different definitions.

According to these definitions, **a robot is a machine enhanced with reasoning capacities, or an embedded computer enhanced with the ability to physically interact with matter**. As such, the missions a robot may accomplish require interaction and integration of intellectual (or *computational*) and physical (or *mechanical*) abilities. This idea of computational-mechanical integration raises a few issues: how to smoothly integrate physical actions and reasoning? Is there some balance between physical actions and computations? Do physical actions impose a precision and time limit on computations or computations impose a limit on physical actions?

Note that we do not include *bots* (or web robots) in the definition of robots. Even though some results and algorithms such as graph planning methods may be of use for both bots and robots, bots are pieces of software while robots are mechanical entities.

### 1.1.2 Movement scales

While some robots do not move and only induce changes to the environment surrounding them through wireless actions and energy transfers, the immense majority of robots moves in some way. We can even say that *movement* is the primary way robots have to interact: industrial robots move their tools to paint cars or transfer cargo around warehouses. Autonomous vacuum cleaners travel through houses to do the cleaning. Humanoid robots reproduce human poses, gestures and mimics to have people feel at ease. Rovers explore Mars to look for water and life sources while search and rescue robots use similar algorithms to look for survivors of an earthquake...

We propose an intuitive classification of robotic movements based on their scale:

1. *Large-scale movements* correspond to changes in the position of the robot whose amplitude is high compared to the size of the robot. For instance, an automatic vacuum cleaner will move from room to room or a mobile robot will explore a building.

2. *Maneuvers* are movements occurring at scales around that of the robot and up to a few times the size of the robot. One typical maneuver is to turn around in a narrow place for a non-holonomic robot (a holonomic robot would just turn in place). Avoidance of dynamic obstacles is performed at this scale.

3. *Reconfigurations* are movements of a part of the robot whose result is a change of the robot's shape or orientation with little to no change in the robot's position. Robotic arms and manipulation tools are typical examples of this scale of

movement.

4. *Expressions* are small movements of a part of the robot whose primary goal is not to modify the robot's shape or to physically interact with an object in the world. Instead, the robot uses motion to display something or to warn someone (or some other robot) of something. This movement scale is essential for robots interacting with humans such as talking robotic museum guides.

While works on problems such as Simultaneous Localization and Mapping (SLAM) including Kuipers' Spatial Semantic Hierarchy (2000) or Bosse et al.'s ATLAS framework (2004) tend to separate large-scale movements from small-scale movements such as maneuvers, we did not find the above classification or a similar one in the mobile robot literature. This classification resembles those reviewed and proposed by Daniel R. Montello (1993) for human perception. However, while Montello bases his classification on *perceptions*, we build ours around *movements*. Both classifications use equivalent descriptions for the three first scales (respectively called "environmental", "vista" and "figural" space in (Montello, 1993)), while our scale 4 has no real equivalent. Interestingly, works in the field of cognitive psychology reviewed by Montello are pretty unanimous on the distinction between large-scale perceptions, which require substantial movement to build a map, and small-scale perceptions which require much less movement to construct a local space representation.

Scales 1 and 2 define the field of *mobile robotics*, which describes how robots can and/or should move within an environment. Scale 3 essentially includes every physical task a robot may have to carry that does not imply navigation. Finally, scale 4 has strong connections to animal and human psychology and physiology. A typical workflow for a robot is the following: the robot goes to a place (scale 1) and positions itself in order to accomplish a task (scale 2). While manipulating objects (scale 3), it acknowledges the presence of a human in its working area by nodding and emitting some sound (scale 4). Figure 1.1 shows scales 1, 2 and 3 for a typical mobile robot.

The scope of this thesis is limited to scales 1 and 2, with the assumption that later integration of scales 3 and 4 will be possible with minimum effort given a robust implementation of scales 1 and 2. While scales 3 and 4 are considered out of scope, the components we implement in Part II allow scales 3 and 4 to run on top of scales 1 and 2, in a hierarchical approach.

### 1.1.3 The four challenges of mobile robot movement

In order to plan and execute large-scale movements and maneuvers, the robot may have to overcome four challenges, represented on Figure 1.2:

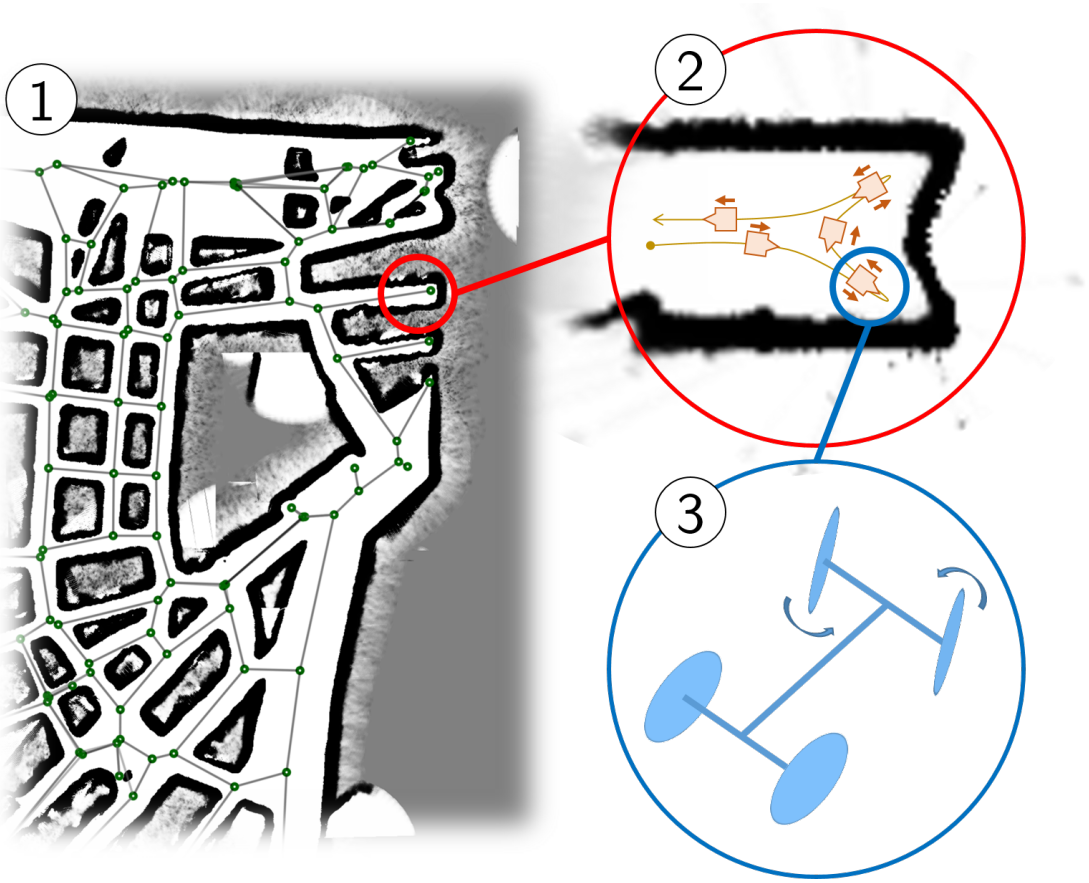



**Figure 1.1:** A schematic view of the three first movement scales for a mobile robot: (1) large-scale movements, (2) maneuvers and (3) reconfigurations.

- *mapping*: being able to draw and maintain a map of the environment,
- *localization*: being able to localize itself within the map and to maintain this localization up to date,
- *planning*: being able to plan a path using the map and
- *(physical) navigation*: being able to physically move in the environment along a planned path and collect sensor data along this path.

Note that there are multiple definitions of the term "navigation", even within robotics. For instance, the "navigation" of (Milford and Wyeth, 2012) is actually topological mapping while Crowley (1985) includes mapping, localization and planning in his "navigation" component. Within this thesis, we define navigation as the component responsible for *physical* movements along a path towards a goal, as opposed to mapping, localization and planning which are concerned with *abstract data* such as maps or itineraries. The main goal of navigation is thus to control actuators that are responsible for executing the movement. In addition to actuator control, we also integrate sensor control and low level sensor processing into navigation, since on the one hand it is not possible to move consistently in the world without any feedback from sensors and on the other hand active sensors do include actuators (think orientable cameras).

We will show in the following section that the four challenges of mapping, localization, planning and navigation need to be simultaneously addressed in order for the robot to move in large-scale and/or complex environments. Additionally, we will question the independence of these challenges and introduce the idea that it may be necessary to consider all four at the same time in order to correctly implement the movement capacity of a mobile robot.

## 1.2 On the necessity of integrating localization, mapping, planning and navigation

### 1.2.1 Navigation without localization, mapping and planning?

It is possible to perform navigation without mapping (without *memory*) and without localization or advanced path planning techniques when certain conditions are met.

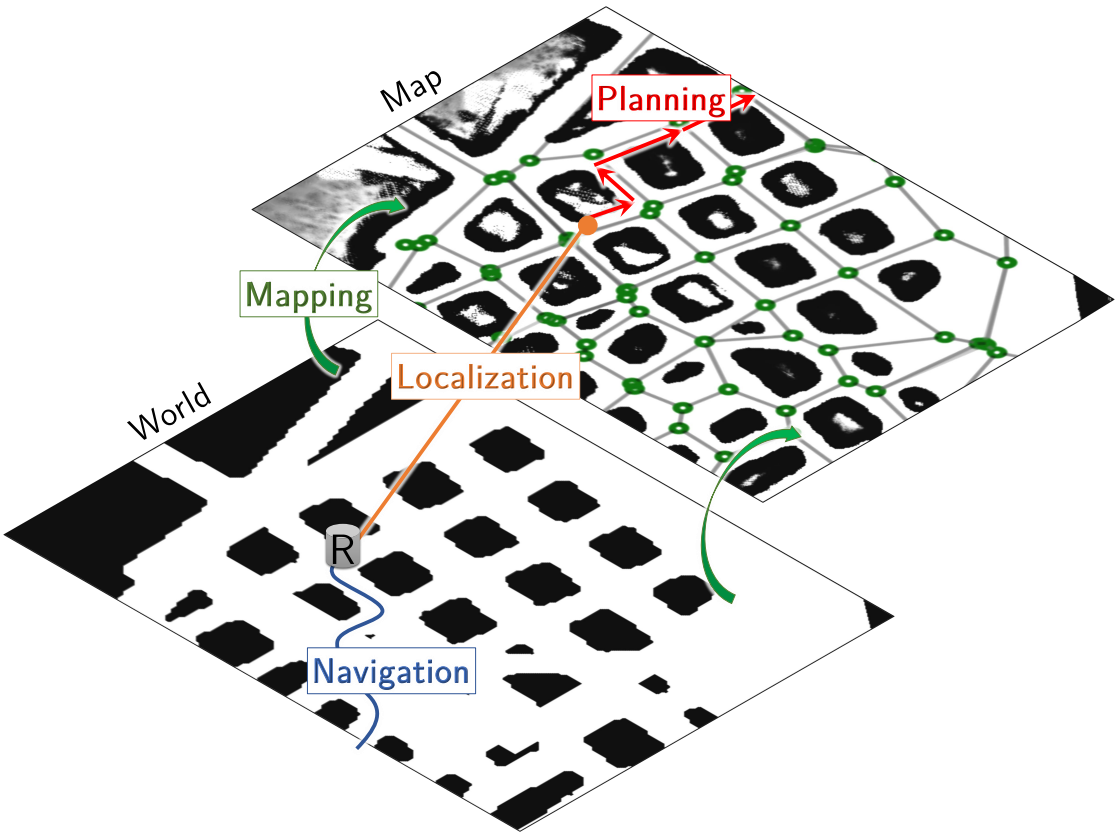



**Figure 1.2:** A schematic view of *mapping*, *localization*, *planning* and *navigation*.

### Greedy navigation

Greedy navigation is a simple method which consists in the robot always trying to minimize its distance to destination. This is done iteratively by moving straight towards the goal, following the boundary of obstacles in the way if any. Locally, if multiple movements lead to the same distance to destination, one movement is chosen randomly. We call $G$-environments environments supporting greedy navigation from any reachable position to any other reachable position. As shown in appendix 9.2, there are only few $G$-environments.

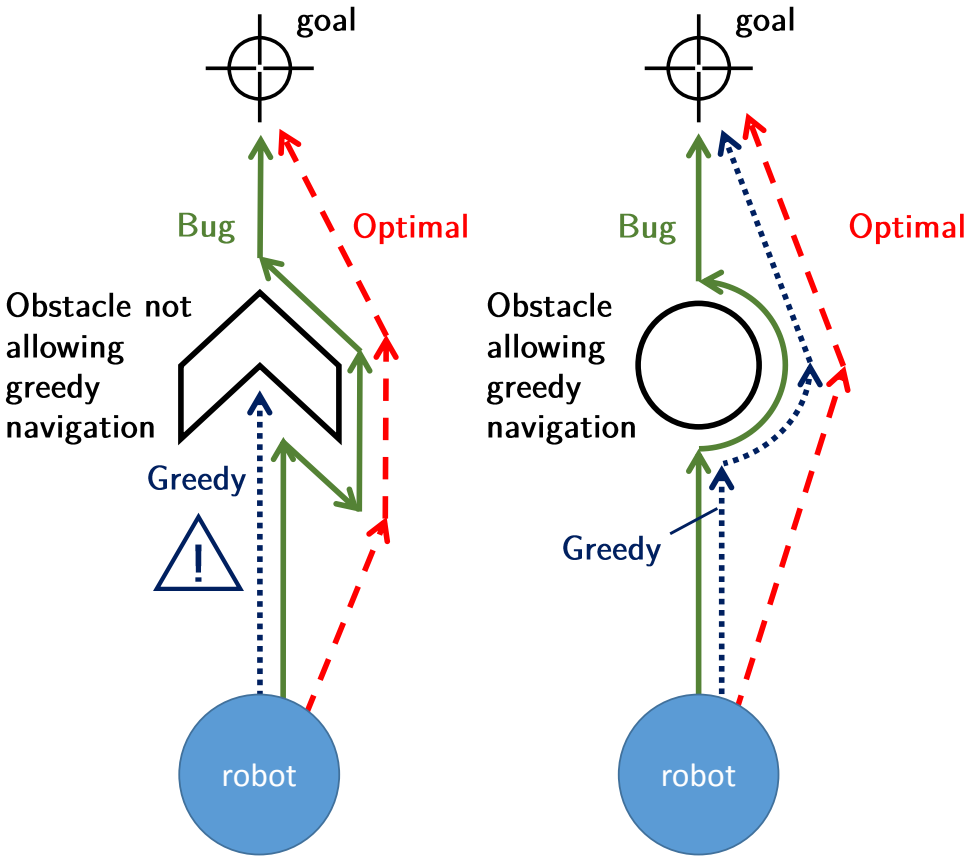
It should be noted that environments may fulfill $G$ but not be planar (for instance, greedy navigation works in 2D but also in 3D - and higher for that matters). Inversely, planar environments may not respect $G$, causing greedy navigation to fail. Indeed, the robot may get stuck in a cavity of an obstacle as on Figure 1.3(a). Additionally, even when $G$ is true, the greedy path is not necessarily the shortest one (Figure 1.3(b)).

### Planar environments, Bug and GPSR

Another example of map-less navigation is when the environment the robot evolves in is planar, when each place can be traversed in every non-blocked direction and when the robot's mission is to reach a set geographic position, Maze-solving algorithms such as Pledge, variants of Bug (Rao et al., 1993) or their graph equivalent Greedy Perimeter Stateless Routing (GPSR) (Bose et al., 2001; Karp and Kung, 2000) are guaranteed to lead the robot to the goal. Bug and GPSR are simple variants of Greedy which implement a "recovery phase" for objects breaking $G$: each time greedy navigation is locally impossible (local minimum of the distance to destination), the robot circles the obstacle right (or left, but always the same) and looks for a point of the obstacle closer to the goal than the point where contact with the obstacle was initially made. Once a point closer to the goal is reached, greedy navigation is used until a new local minimum is found (see Figure 1.3). Of course and as visible on the figure, if an accurate map of the environment was available, using it would lead to shorter paths to the goal than provided by Bug, Pledge or GPSR. The constraint of each place being traversable in every non-blocked direction means that one-way paths such as one-way streets or escalators are forbidden.

### On the necessity of mapping

In 3D non-$G$-environments, in nearly 2D environments with locally multiple layers (such as cities with bridges and tunnels) and in environments with one-way paths, there exists to our knowledge no algorithm ensuring that a robot can reach arbitrary (reachable) GPS




**(a)** In a non-$G$-environment, greedy navigation (dotted blue) may fail to lead to the goal

**(b)** Even in a $G$-environment, neither greedy navigation (dotted blue) nor Bug (plain green) will follow a trajectory whose length is the shortest (dashed red)

**Figure 1.3:** Greedy navigation and Bug follow suboptimal paths, Greedy additionally fails when there are local minima of the distance to some destination position

coordinates without using and/or building a map of the environment. Additionally, even in environments where a map is not necessary, not using one may drastically reduce navigation performances. In chapter 8, we show that not using a map may lead to average path lengths being multiplied by 2.5 or more. Consequently, using/constructing a map and finding/updating the position of the robot within said map are convenient, if not necessary for navigation (Filliat, 2001).

The fact that there is no map-less (no *memory-less*) algorithm for navigation in arbitrary environments implies that any algorithm guaranteeing navigation in an arbitrary environment must use some form of memory. One fundamental question would then be: "How much memory is required to reach a single destination/to reach an arbitrary destination". Or more generally, "What is the relation between *memory* and *navigation performances*". As far as we know, this question was never formulated as such, let alone answered, in the field of robotics. The interested reader will find an experimental study of the link between *memory* and *navigation performances* for various algorithms in chapter 8.

### 1.2.2 From mapping and localization to SLAM

Suppose that a robot simultaneously builds a map of an environment and localizes itself within the map. Theoretically, the robot would want the map and its own position as two separate entities, independent of each other. However, intuitively, localization is always relative to the map, which in turn is constructed around a position. This intuition was notably developed in early works by Chatila and Laumond (1985), Crowley (1985), Smith and Cheeseman (1986) and Leonard and Durrant-Whyte (1991). Durrant-Whyte, Rye and Nebot (1996) formulated the intuition mathematically and devised the simultaneous localization and map building problem as the estimation of a joint robot-position-and-map-state probability density. With an infinite number of observations, the *relative* positions of observable features in the environment (the map) will become more and more precise, with the robot's localization being defined relative to the map, and not relative to the environment. The actual position of the map relative to the environment, while bounded, may not converge.

This seminal work of Durrant-Whyte et al. kickstarted the study of a problem now known as *Simultaneous Localization and Mapping* or *SLAM*. SLAM is one of the pillars of mobile robotics if not the "holy grail" of mobile robotics (Durrant-Whyte and Bailey, 2006). In part II, chapter 5, we give an extensive review of existing SLAM algorithms and implement our own. The important element for now is that mapping and localization should be treated as one single problem: SLAM.

### 1.2.3 Adding planning and navigation to SLAM: PNSLAM

SLAM describes how to build a map from a set of observations and localize the robot on said map. It does not describe how these observations should be made or how the map can be used. Intuitively, both mapping and localization depend heavily on sensed data (observations), which triggers the question: should there be a dual-way link between data acquisition (sensors) and SLAM?

In the vast majority of works on SLAM, a "robot" is guided in an environment, with sensors gathering data. Data are then processed either online or offline (after the run) to produce both map and robot trajectory. Note that we put quotes around the term "robot". Indeed, the sensor platform does not need to be a robot: the sensor platform does not take a specific action in response to a stimulus and cannot interact with the world. Consequently, it is possible to do SLAM just letting a human or a human-driven vehicle carry a camera around in the environment (see (Lim, Lim, and Kim, 2014) for instance).

In order for a robot to move efficiently and autonomously in an initially (partially or totally) unknown environment, we showed that using a map was mandatory. Since the environment is initially at least partially unknown, the robot has to construct the map using SLAM. Then, it must be able to plan paths using this map and to follow the plan to reach its goal. The integration of navigation, planning and SLAM is critical since:

- The result of a planning algorithm heavily depends on the map and on the starting position of the robot. As a minimum requirement, the map has to be usable for navigation, and a starting position should be provided.
- While navigating along the path, the robot may find that the environment does not correspond to what's written on the map, because the map is either obsolete or plainly wrong.
- While navigating along the path, the robot should update the map, its localization on the map and possibly its itinerary according to the updated localization and map.
- What the map contains depends on the trajectory of the robot.

We call *PNSLAM* the integration of planning, navigation and SLAM, as sketched on Figure 1.4. PNSLAM is a superset of SLAM. PNSLAM covers a family of existing mobile robotics problems:

- navigating from point A to point B,

- reaching a target known by its approximate coordinates (see *Exploratory Digraph Navigation*, chapter 3),
- finding a "treasure" in a labyrinth,
- exploring an initially unknown environment in one or multiple runs (piecemal exploration, see (Awerbuch et al., 1999)) and
- building niches and finding food in order to simulate real life.

We believe that the use of PNSLAM may lead to the discovery of new problems and of solutions to them.

### 1.2.4 Considering finite resources: Lifelong Exploratory Navigation

One fundamental hypothesis of this thesis is that a robot should not maintain a connection to an external resource such as a database or a command server. Indeed, connection to external resources such as a cloud server for storage and computations or even just the GPS is not an option in many applications due to latency, bit-rate, eavesdropping or interception issues, risks of loosing the connexion or sheer impossibility to maintain a connexion. Marine, mine or extraterrestrial environments are typical examples where on the one hand, robots are much needed and on the other hand, maintaining a stable high-speed connexion is nearly impossible using state-of-the-art technology. Even in simple indoor environments, Wi-Fi roaming and GPS signal degradations make it difficult to download data from external entities. Consequently, even though external resources may sometimes be accessed, it is necessary to make the pessimistic hypothesis that the robot must operate on its own, without external help. When the robot operates on its own, it has to consider its resources as finite and manage them carefully in order to maintain an operational state.

Even with integrated planning, navigation and SLAM, a mobile robot will not be able to move in extremely large environments, since no matter how much memory and computing power the robot has, there are environments too big to fit into memory, to be processed in real time or to be explored in a single run due to limited battery capacity. While for a single robot mission (like "find an object" or "go to a specific position"), the map of the environment acquired by SLAM may not exceed the memory capacities of the robot, lifelong operation with diversified missions in potentially extremely large environments is not possible without careful memory and computing power management as well as a mission unit to choose which planning algorithm to use and prioritize missions.

Typically, if the size of the environment the robot may have to operate in is defined as a

number $n$ of features, we wouldn't like memory or computing power to grow in $\mathcal{O}(n^{p\geq 1})$. An $\mathcal{O}(log(n))$ or less growth should be acceptable though. As an underlying theoretical question, one can wonder what is the minimum amount of computing power and memory needed to travel from any point to any other point of the environment. While we do not give a conclusive answer to this question in this thesis, the relation between planning, navigation and resource management is one key aspect of this work.

Speaking of resource management, probably the most important resource for robot navigation is *energy*. Even though energy handling is critical, it is relatively easy for a robot to know when to pause its current task to look for an energy source. For instance, if energy sources are written on the map, the robot can estimate (overestimate if robustness is needed) how much energy is needed to reach an energy source, and never go too far from any energy source according to this energy budget. While we do not address the issue of *energy management* explicitly in this thesis, the algorithms developed in the following chapters (especially EDNA*, chapter 3) are compatible with energy management in addition to being energy-friendly (by reducing the amount of movements required to reach a goal and allowing foraging for energy sources). Energy management is handled at the *mission* level in the schematic view of Figure 1.4.

We decided to call the ensemble {PNSLAM + resource management + mission control} *Lifelong Exploratory Navigation* (Figure 1.4), where Lifelong denotes the fact that this paradigm is intended for robots performing multiple missions over a long period of time. We believe that this new paradigm can be used to create robots much more autonomous and versatile than the current ones.

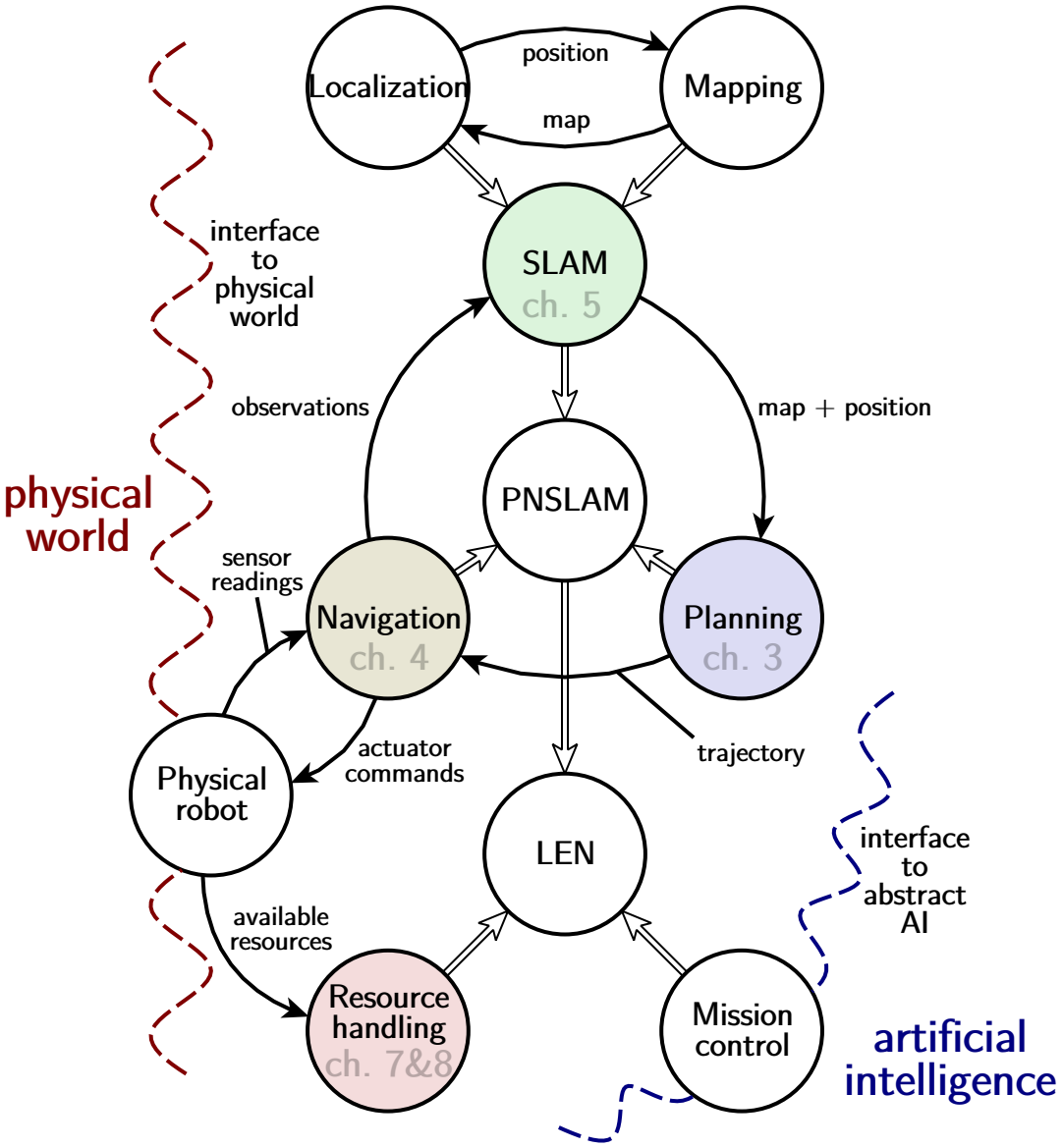



**Figure 1.4:** A schematic view of PNSLAM, which unifies (P)lanning, (N)avigation and (SLAM) and of Lifelong Exploratory Navigation (LEN), which adds in resource management, mission control and interfaces to higher level intelligence. Plain arrows show data dependency while double arrows show integration of components. Available resources include memory, battery and processing power. An implementation of each of the main components (Planning, Navigation, SLAM and Resource handling) is provided in parts II and III of this thesis.

# 2 Lifelong Exploratory Navigation - a system view

Lifelong Exploratory Navigation is a new paradigm whose definition includes planning, navigation, SLAM, memory and computing power management. While the previous chapter was focused on what a mobile robot should be able to do, this chapter gives an insight on how to perform the tasks, from a structural point of view. In this chapter, we give an overview of various components required for LEN and explain how they can interact. Each component operates with specific timing, complexity and safety constraints. The components are structured within a hierarchy based on time and space scales and inspired by animals and humans.

## 2.1 High-level view of a robot

Figure 2.1 presents a very high level view of a robot. We made the user optional because the definition of a robot, given in the previous chapter, does not mention a user. Indeed, some robots do not have users, such as the Ecobots produced at Bristol university, whose aim is just to live eating organic matter (Ieropoulos et al., 2010). While a robot without user interaction may not seem very useful, there are potentially a huge number of uses for such robots such as autonomous treatment and transformation of waste, interaction with wildlife and completely autonomous cleaning agents. Science fiction authors would add to these more exotic missions such as terraforming. However, most currently used robots are equipped with some form of reporting mechanism, whence the inclusion of a user on the sketch.

In the rest of this chapter, we will explore how the "robot" block of Figure 2.1 should be structured in order for the robot both to preserve its physical integrity and to accomplish a mission it was given, or a mission it has decided to undertake on its own. Deciding which mission to undertake and how to undertake it is the role of the "mission" unit on Figure 1.4.

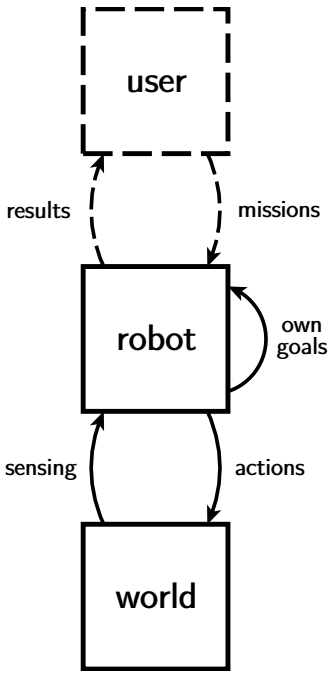



**Figure 2.1:** Very high level view of a robot. The presence of a user is optional.

## 2.2 LEN in animals

The first sources of inspiration to try and derive the internal components of the "robot" on Figure 2.1 are animals and humans. While the structure and functioning of the brain have not been fully puzzled out yet, existing studies on planning, navigation or SLAM do give us hints on how PNSLAM may be implemented in our own brain.

### 2.2.1 Perceiving distances and directions

The seminal work of Tolman et al. (1948) reports experiments in which rats are trained to go to a food source in a simple environment. Then, the environment is made more complex. In the complex environment, rats tend to choose paths whose angle best matches the direction of the food source. Thus, it can be deduced that rats have some notion of orientation and angles. The works of Alyan et al. (1994), Etienne et al. (2004; 1996) as well as other works listed in (Touretzky and Redish, 1996) show that rodents not only have a notion of angles, but also of distances. This gives them a basic sense of vector arithmetics (additions (Touretzky and Redish, 1996) and subtractions (Etienne and Jeffery, 2004; Etienne, Maurer, and Séquinot, 1996)) and enables them to do path integration, that is to determine their position based on their past movements.

Alyan et al. (1994) suggest that path integration is used when the animal only has poor knowledge of the environment and is replaced by place or beacon recognition when knowledge about the environment increases. When there is a conflict between positions returned by both methods, path integration is privileged in case the conflict is major and place or beacon recognition is privileged when the conflict is minor. Touretzky (1996) suggests that since the process of recognizing beacons and localizing relative to them is quite expensive computationally wise, rodents prefer using odometry for short journeys, rarely updating their absolute position by observing beacons. Hamsters and ants can use odometry to reach the vicinity of their goal and processes implying environment recognition to finally home in on the goal (Etienne and Jeffery, 2004).

Path integration is performed by animals using their vestibular organs to detect accelerations in 3D as well as efferent motor signals. Some animals such as blind mole rats use a compass sense to improve the precision of their odometric sensing (Kimchi, Etienne, and Terkel, 2004), which comes of use in large environments. Visual optic flow is also used (Gibson, 1950) to find relative movements. Path integration computations are believed to happen within the hippocampus (McNaughton et al., 2006).

The experiments of Byrne (1979) on humans show that our notion of angles and lengths is approximative: angles are most of the time approximated to multiples of 90° and distance estimates get biased by path curvature and subjective utility of a path. Similarly, the study of McNamara et al. (1989) shows that our estimate of distance between objects depends on membership of these objects to the same or different semantic groups.

### 2.2.2 Path planning

Path planning in animals and humans is less studied than localization, mapping and navigation. The only papers we could find on the subject are theoretical papers by Ponulak et al. (2013) and Rueckert et al. (2016) on wavefront propagation in an artificial spiking neuron network. The observed propagation scheme resembles what can be obtained using Dijkstra's shortest path algorithm. However, there is no proof that such a propagation scheme is actually used in the living world.

### 2.2.3 SLAM

#### A graph-like map

The experiments of O'Keefe et al. (1987), Whishaw et al. (1991) and Etienne et al. (1996) on rodents as well as that of Collett et al. (1997) on bees tend to prove that these animals rely on a map of the environment stored as a graph of places (vertices)

and links between them (for instance, distances and changes in angles). A graph-based model can easily integrate semantic data describing vertices and links between them, which Byrne et al. (1979) use to support the idea of semantic graph-based mapping in humans. In the same vein, bees locate themselves relatively to successive beacons (graph-based localization) until the goal is near enough to be directly recognized and tracked by the vision system.

One weakness of beacon-based localization is its dependency to point-of-view. Thus, a drastic decrease of localization performances should be expected when the point of view changes, even when the scene itself does not change. This decrease is observed by Simons et al. (1998). Since planning, navigation and localization need to abstract away orientation, it is expected for vertices of the graph map to be more complex entities than just point-of-view dependent perceptions (Touretzky and Redish, 1996). In the mammal brain (at least), the hippocampus contains a zone called PPA for Parahippocampal Place Area containing neurons which only fire when the animal is in a specific place (*place units*), moves between places (*displace units*) or does not detect a place it expected to find at the current location (*misplace units*) (Epstein and Kanwisher, 1998; O'Keefe and Speakman, 1987; O'Keefe, 1976; Touretzky and Redish, 1996). This area of the hippocampus is only sensitive to places, not to faces or to objects. Consequently, place units continue emitting neuronal spikes when the animal is currently at the place they are describing, even though the immediate environment has changed (things have been removed or added). Place units also fire in the dark, despite the fact that no visual cue is available. We can deduce from these observations that each place unit is associated to a physical position defined by multiple criteria and observations (similarly to the local maps used in robotic SLAM by Bosse et al. (2004)).

Finally, a graph-like map offers the ability to model a hierarchy in places and place characteristics, including membership relations. This is compatible with the experiments of McNamara et al. (1989) where humans had to estimate distances between objects and estimations were biased by objects belonging or not to a common semantic group.

### A dense map

In rats and humans at least, it was observed (Doeller, Barry, and Burgess, 2010; Ekstrom et al., 2003; Kuipers, 2000; O'Keefe, 1976) that certain zones of the hippocampus had neurons forming a metrical map of the environment, in the sense that excitation of a neuron at position $(f(x), g(x))$ in the hippocampus corresponded to the animal or human being at position $(x, y)$ of the environment, with $f$ and $g$ two bijective functions (ideally). Compared to graph-like maps, these neurons form a dense *metrical* map of the environment.

Dense and graph-like (or topological) maps are not mutually exclusive. Indeed, a dense

map is similar to a graph with many vertices and metrical relations between vertices (Figure 2.2). The idea that there may be multiple interacting representations is also present in the work of Newman et al. (2007). In this work, Newman et al. observed the behavior of taxi drivers having to localize themselves in changing virtual environments. Their conclusion was that there exists a hierarchy between localization strategies: when layout data is available as well as visual cues and there is a contradiction between both, visual cues are privileged. When one of the representations is missing, the other one will be used. Steck et al. (2000) add that human navigation uses both local beacons (whose visibility is limited to a certain distance) and global cues (objects and properties visible or accessible from a large fraction of the environment, such as the sun, a tower or a compass sense). When one source of information is not available, the other one is used.

### 2.2.4 Resource handling

The memory and computing capacities of the brain are still widely unknown and under investigation (Bartol et al., 2015). For instance, the rate at which living creatures learn and forget things is still widely undetermined. What we can be sure of is that living creatures are able to navigate in almost any environment that is presented to them, no matter how *old* they are, with the hypothesis that old creatures will have perceived more things and thus should have more memory issues. Thus, either the brain's computational capacities are so high that the size of the map in memory will never be a problem or there is some kind of forgetting mechanism. Such a forgetting mechanism can be observed with Artificial Neural Networks (ANNs) when the amount of data to learn far exceeds the number of neurons or connections in the network. In this case, useful knowledge may be forgotten (catastrophic forgetting, see (Robins, 1995)).

### 2.2.5 Summary of animal LEN capacities

Animals (or at least some species) are capable of:

- Traversing an environment from any known place to any other known place (which includes planning and navigation capacities),
- avoiding obstacles (which requires a local representation of occupied space including zones immediately behind the animal),
- maintaining a topological representation of the environment,
- maintaining a metrical representation of the environment,

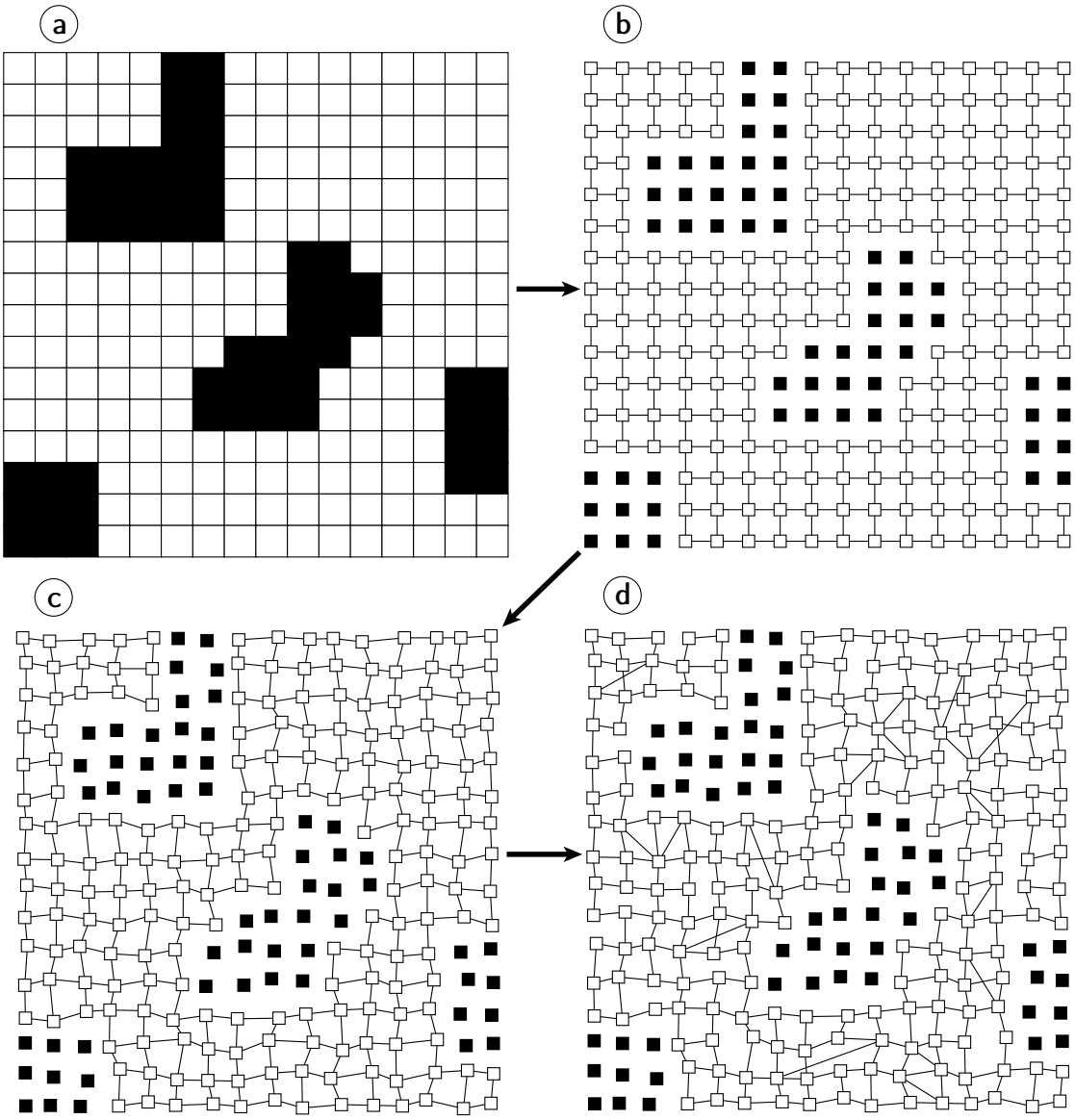



**Figure 2.2:** A dense map (a) is similar to a grid-graph (b). Even if the graph is deformed (c) and if topology changes slightly (d), the map still looks similar. Case (d) is probably closer to how maps are stored in the hippocampus than cases (a), (b) and (c).

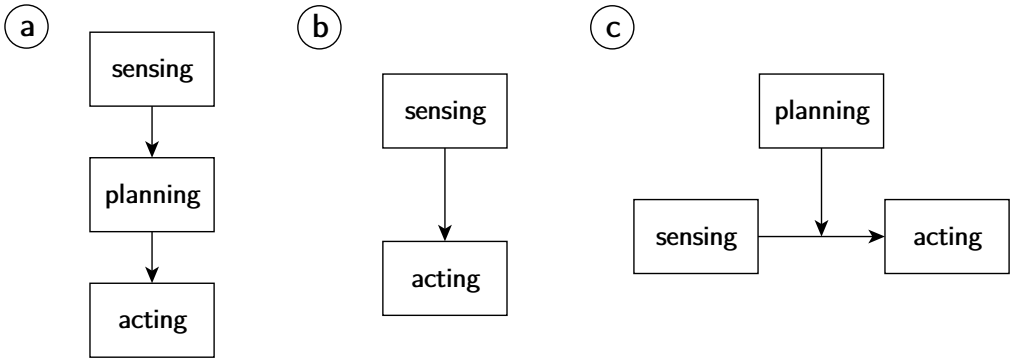



**Figure 2.3:** Primitive control architectures from (Filliat, 2001): (a) hierarchical, (b) reactive and (c) hybrid.

- performing localization within the environment,
- accomplishing various missions implying movements such as go to a target, explore, find food or find the closest hideout and
- (probably) forgetting or compressing unused map data

These capacities give us a hint on what a mobile robot should be able to do, and how to do it. They are also a proof that the Lifelong Exploratory Navigation paradigm can be implemented. Reproducing the tasks performed by animals requires a robust robot control architecture describing time and space scales as well as the interaction between various subparts of a robot (sensors, actuators, artificial intelligence, ...), which is the purpose of the next section.

## 2.3 The Hybrid Spatial Semantic Hierarchy and other models

### 2.3.1 Primitive architectures

Various robot architectures have already been proposed. Filliat (2001) for instance lists three of them: hierarchical, reactive and hybrid (see Figure 2.3).

While the "reactive" architecture is clearly not sufficient for PNSLAM in huge environments, let alone LEN, the two other architectures are in fact similar, with planning as a supervising step.

### 2.3.2 Hierarchical SLAM and navigation structures

We can track down the origins of modern robot control architectures for SLAM and navigation to a paper written by Crowley (1985). This paper formulated the following ideas:

- Sensing and control should be done within an (at least) three-level model: a sensor model, a (composite) local model and a global model.
- Path planning is done at two levels: global path planning, requiring a global map of the environment, and local path planning, navigation and obstacle avoidance, requiring a representation of nearby obstacles.
- The global map is represented as a network of places.

A similar, albeit more complex control architecture was formulated years later by Kuipers (2000). The proposed hierarchy of space models is known as the Spatial Semantic Hierarchy (SSH). The SSH models environmental representations in a hierarchical approach (Figure 2.4). These representations are, from the least abstract to the most abstract:

1. the sensor level, which is basically sensor values;
2. the control level, which describes low-level control laws;
3. the causal level, which includes the notion of view, actions and odometry;
4. the topological level, which describes places, paths and connectivity and
5. the global 2D geometry, which describes accurate global metrical data

The key idea of the SSH is to extract the topology of the environment from a local view and to build a global metrical map by some optimization process using the topological map. Compared to the work of Crowley (1985), the SSH distinguishes the topological and metrical components of the global map. The SSH was used in later works by Kuipers and his team (Beeson, Jong, and Kuipers, 2005; Beeson, Modayil, and Kuipers, 2010; Kuipers et al., 2004). It is also implicitly or explicitly used in nowadays' most advanced SLAM approaches such as hybrid metrical/topological mapping (Bailey, 2002; Bosse et al., 2004) and graph SLAMS (Gutmann and Konolige, 1999; Pinies, Paz, and Tardos, 2009; Schuster et al., 2015; Thrun and Montemerlo, 2006; Wagner, Frese, and Bauml, 2014).

The SSH was later improved (Beeson, Modayil, and Kuipers, 2010; Kuipers et al., 2004)

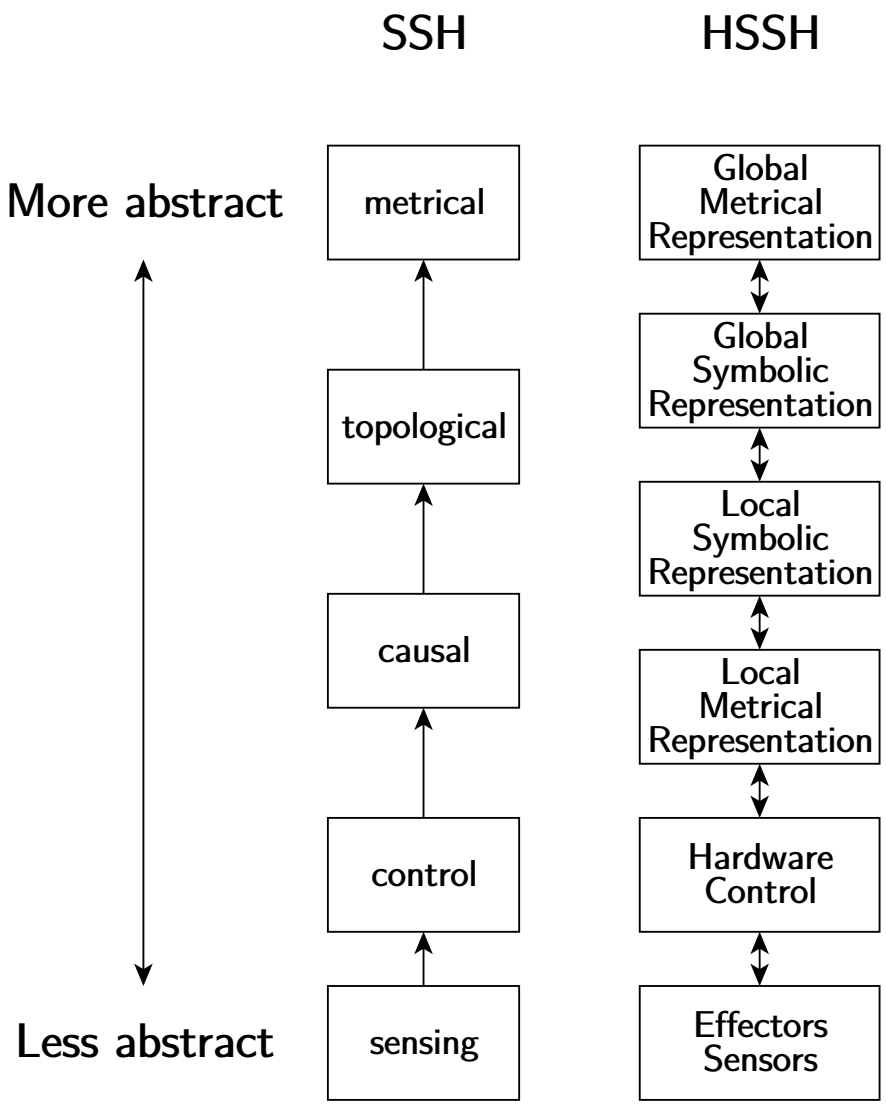



**Figure 2.4:** (simplified view) The Spatial Semantic Hierarchy (SSH) and the Hybrid Spatial Semantic Hierarchy (HSSH) are multilevel SLAM architectures from sensors to large-scale optimized maps. The main advantage of the HSSH over the SSH is the use of a local perceptual model or local map, that is a local metrical representation from which a local symbolic representation is extracted.

into an "Hybrid Spatial Semantic Hierarchy" or HSSH (Figure 2.4), whose main advantage over the SSH is the use of a local perceptual model or local map, that is a local metrical representation from which a local symbolic representation is extracted.

While the HSSH was proven to be a robust architecture for SLAM, it lacks integration of planning and navigation, reflecting the common practice in SLAM literature to let aside planning and navigation as external issues.

## 2.4 System view of our Lifelong Exploratory Navigation approach

We propose a Lifelong Exploratory Navigation mobile robot architecture based on the HSSH (for its SLAM capacities) and on a hierarchical control structure. Our architecture is sketched on Figure 2.5. This architecture has a few notable properties.

First, the (hard) real-time algorithms necessary for safety (obstacle avoidance) are decoupled from Artificial Intelligence (AI) processes such as planning, topological SLAM, mission unit, etc., which means that no matter the algorithmic complexity or running time (or crashes) of the high-level AI, safety is still enforced. In fact, this robot control architecture matches the timescales observable in human beings:

- The sensorymotor level is where reflexes occur: fast movements in direct response to a stimulus.
- The local occupancy grid and topology extraction levels maintain a local map of nearby obstacles, as needed for obstacle avoidance and fast action.
- The topological SLAM and Exploratory Planner as well as mission unit and abstract AI are the common "thoughts" of a human.
- Map optimization and compression are steps optional for navigation and PNSLAM but necessary for LEN that would typically be executed during a "dream" phase in a human brain.

Note that movement scales defined in paragraph 1.1.2 are also visible in this structure: large-scale movements correspond to the components running in non-constant complexity (especially topological SLAM and planning) while maneuvers correspond to all components up to topology extraction and local topological navigation.

Second, this structure is compatible with the PNSLAM behavior observed in animals: there are both local-dense and global-topological maps. As in the HSSH, the topological

map is augmented with approximate metrical properties.

Third, it is possible to switch a specific component without changing the rest of the structure. Programmatically, minimum standard interfaces allow smooth integration of components. It is possible to run different components in different places: sensors on the robot, AI on a dedicated computer, ... Our whole LEN approach may also run on a single computer aboard the robot.

Finally and as visible on Figure 2.5, AI functions not related to large-scale movements and maneuvers are completely separated from the Lifelong Exploratory Navigation architecture. In fact, the LEN architecture can be viewed as a service offering an interface to physical movements to a client AI process. This client AI process may request movements and get feedback from the LEN architecture in a transactional way. For instance, in order to find a specific item in the environment, the AI process may require the robot to travel to a certain approximate location, a task which is forwarded to the *mission unit* of Figure 2.5. While the robot is moving to the desired location or exploring around the location, the AI process may use an object detector to look for its target item. Another AI process independent from LEN may handle physical manipulations of the item once found. This new AI process may request further movements from the LEN architecture in order to position the robot more accurately relative to the item. Then, another AI process may order the LEN architecture to return to a place where the item should be deposited, etc.

The rest of this thesis derives the blocks marked as “new” or “improved” on Figure 2.5.

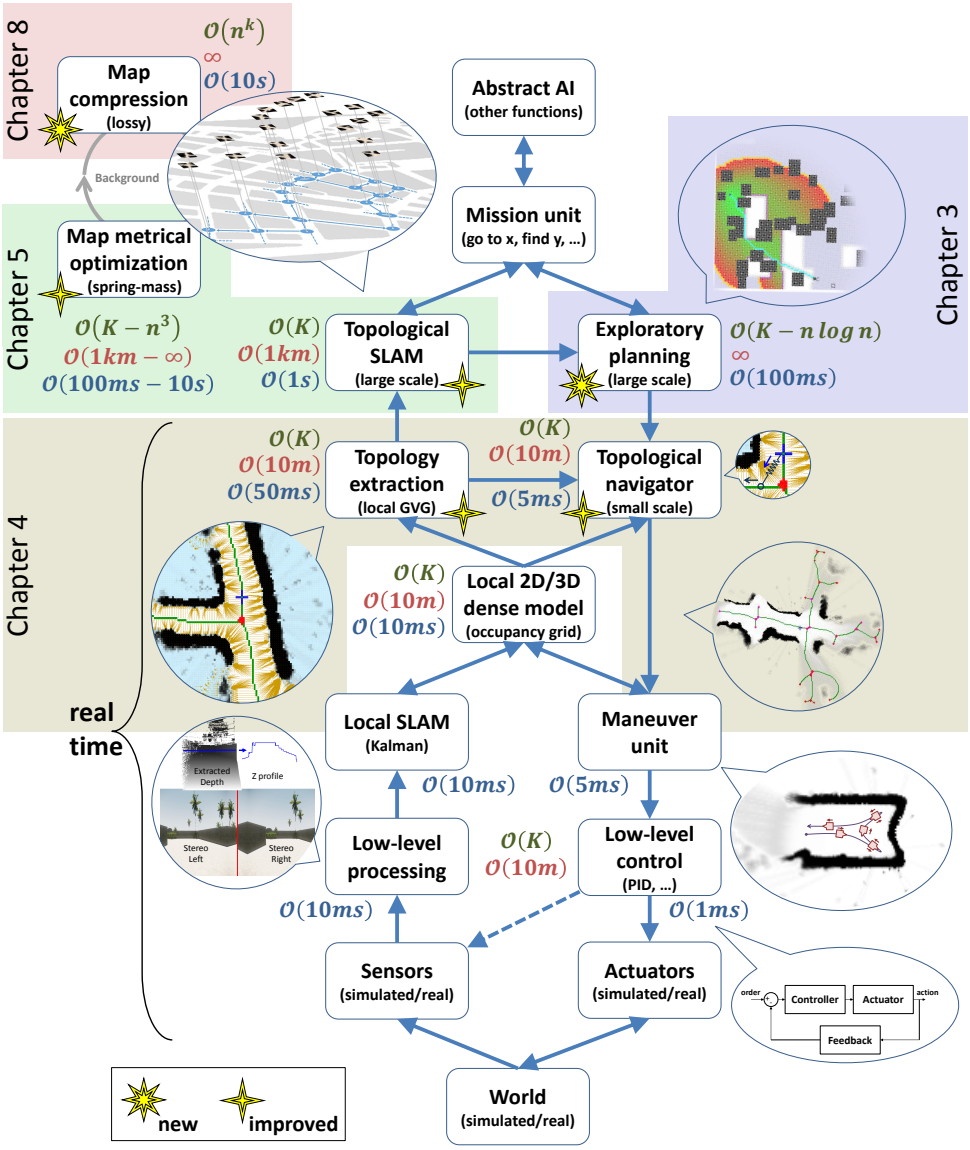



**Figure 2.5:** The mobile robot architecture used in this thesis is derived from the Hybrid Spatial Semantic Hierarchy (Beeson, Modayil, and Kuipers, 2010). In addition to the HSSH, our framework integrates navigation and planning as well as mission handling and map compression. This sketch is a detailed version of Figure 1.4. Algorithmic complexities (green), space scales (red) and time scales (blue) are given for the main components ($\mathcal{O}(K)$ means constant time). Our main contributions, developed in part II, are marked with yellow stars (new components) and crosses (improved components).

# Part II

# Planning, Navigation and SLAM

The pioneer 3 robot equipped for mapping experiments. On top of a long cardboard box, the magnetometer is protected from the metallic parts of the robot.

# 3 Exploratory Planning

*"All right, we're here! Let us never speak of the shortcut again."*
**Homer**, The Simpsons, *Itchy & Scratchyland*
http://tvtropes.org/pmwiki/pmwiki.php/Main/ShortCutsMakeLongDelays

In this chapter, we define an Exploratory Planner (EP) as a path planning algorithm operating on an incomplete map and able to take advantage of yet unmapped areas. We also introduce a PNSLAM approach called Exploratory Digraph Navigation (EDN). EDN is a planning-centered view of PNSLAM which uses navigation, SLAM and EP as external components (Figure 3.1).

The key idea of an exploratory planner is that it can choose to resort to an exploration phase if it sees a sufficient benefit in it. In order to improve a given heuristic, the path planner may choose to privilege exploration of uncharted space or reuse of known paths. Exploration is riskier (it may lead to dead ends) but may also lead to shortcuts. An exploratory planner can smoothly transition from pure path planning to pure exploration. We develop the EDNA* (Mayran de Chamisso, Soulier, and Aupetit, 2015) exploratory planner as a simple modification of A* (Hart, Nilsson, and Raphael, 1968). This modification can be ported (see appendix 2) to other path planners such as (Lazy) Theta* (Nash et al., 2007; Nash, Koenig, and Tovey, 2010).

## 3.1 Notations

Let $\mathcal{W}$ be a physical environment which can be completely described by a map in the form of a directed graph (or *digraph*) $\mathcal{G}^R$. The $R$ of $\mathcal{G}^R$ stands for "real" (ground truth). Let $\mathcal{V}^R$ be the set of vertices of $\mathcal{G}^R$ and $\mathcal{E}^R \subset \mathcal{V}^R \times \mathcal{V}^R \times \mathbb{N}$ be the set of edges of $\mathcal{G}^R$. An edge $e \in \mathcal{E}^R$ is uniquely identified by its origin or "source" vertex $S(e) \in \mathcal{V}^R$, its

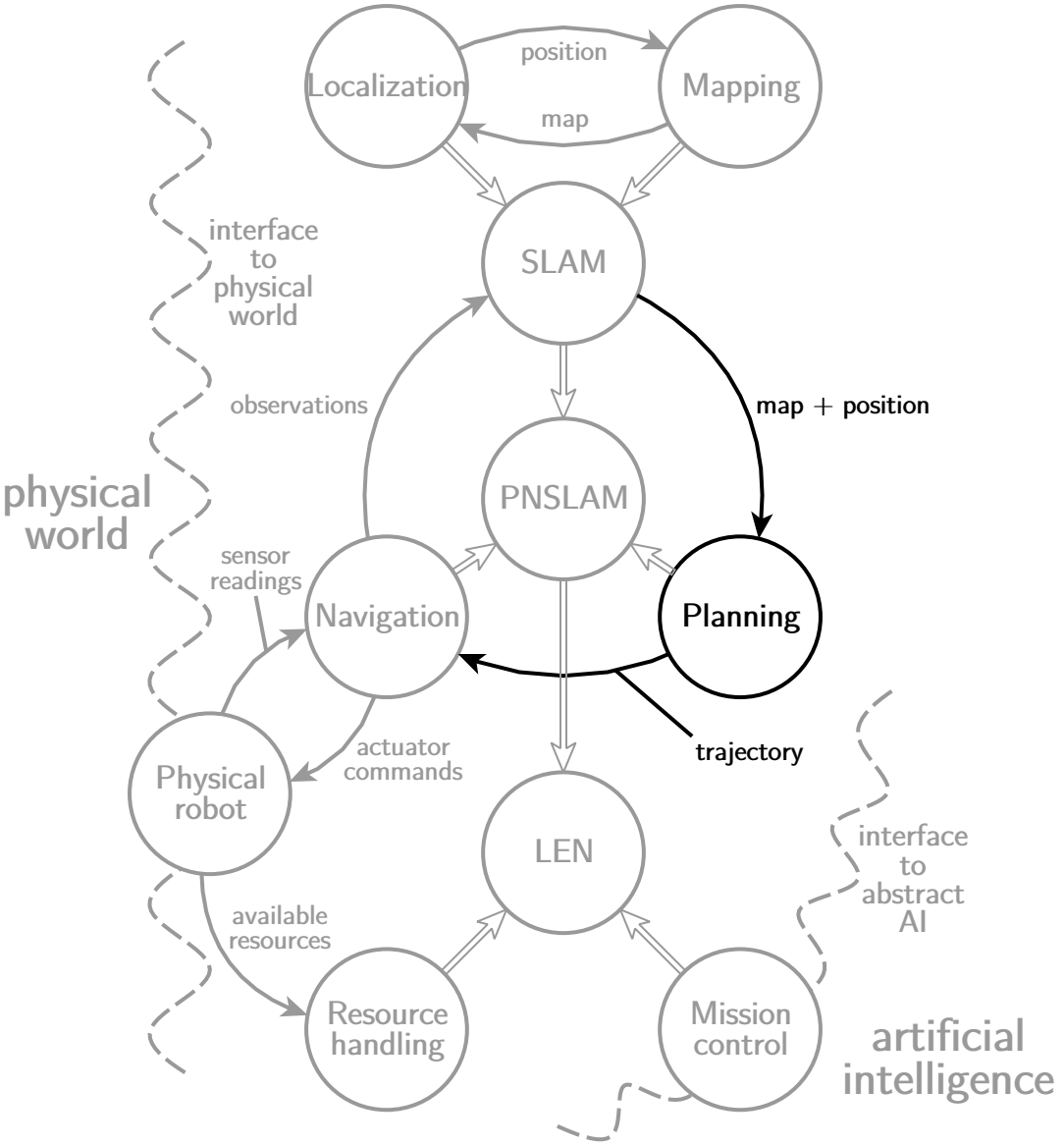



**Figure 3.1:** Exploratory Digraph Navigation (EDN) is a planning-centered view of PNSLAM and LEN.

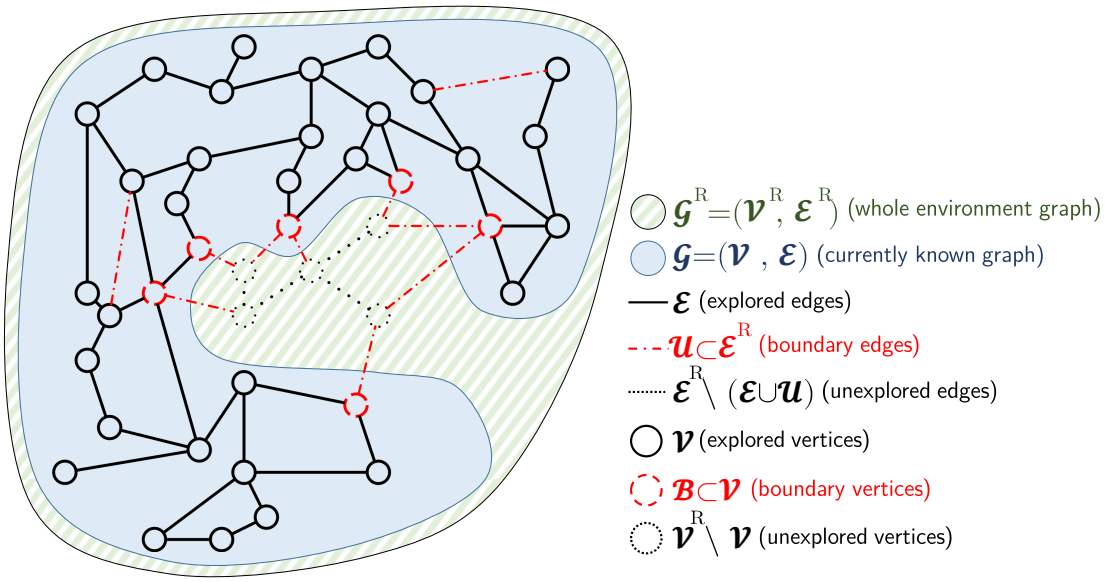



**Figure 3.2:** Sketch of the notations used in this chapter: explored and unexplored space and boundary vertices and edges.

destination or "termination" vertex $T(e) \in \mathcal{V}^R$ and an integer identifier used only when multiple edges link the same origin vertex to the same destination vertex.

The robot navigates in $\mathcal{W}$ while running SLAM to maintain a map of the environment $\mathcal{G} \subset \mathcal{G}^R$, so that $\mathcal{G}^R \backslash \mathcal{G}$ represents space not explored yet. Felner et al. (2004) call $\mathcal{G}$ the *currently known graph*. Let $\mathcal{V}$ be the set of vertices of $\mathcal{G}$ and $\mathcal{E} \subset \mathcal{V} \times \mathcal{V} \times \mathbb{N}$ be the set of edges of $\mathcal{G}$. In addition to these vertices and edges, called internal vertices and explored edges respectively, we also define boundary vertices ($\mathcal{B}$) and unexplored edges ($\mathcal{U}$). Unexplored edges are edges whose origin is in $\mathcal{B}$ but whose destination is unknown (it may be in $\mathcal{V}^R \backslash \mathcal{V}$ or in $\mathcal{B}$). Formally, the set of boundary vertices is $\mathcal{B} \subset \mathcal{V}$ and the set of unexplored edges is $\mathcal{U} \subset \mathcal{B} \times ((\mathcal{V}^R \backslash \mathcal{V}) \cup \mathcal{B}) \times \mathbb{N}$. Boundary vertices are vertices which possess at least one unexplored outgoing edge: $\mathcal{B} = \{v \in \mathcal{V}, \exists e \in \mathcal{U}, \exists v' \in ((\mathcal{V}^R \backslash \mathcal{V}) \cup \mathcal{B}), S(e) = v, T(e) = v'\}$. The boundary state of these vertices is known at planning time. From the definition follows that $\mathcal{G} = \mathcal{G}^R \Leftrightarrow \mathcal{U} = \varnothing \Leftrightarrow \mathcal{B} = \varnothing$. $\mathcal{V} \neq \mathcal{V}^R$ and $\mathcal{E} \cup \mathcal{U} \neq \mathcal{E}^R$ in general since some vertices and edges of $\mathcal{G}^R$ may be missing (not yet discovered) in $\mathcal{G}$. All notations are sketched on Figure 3.2.

## 3.2 Introducing Exploratory Planning and Exploratory Digraph Navigation

### 3.2.1 Definitions

From its current position, the robot wants to reach a goal position with minimal effort. While planning its path to the goal on $\mathcal{G}$, it realizes that it essentially has two choices represented on Figure 3.3: using a known (safe) path, which may be quite long, or trying to find a *shortcut* thanks to places whose knowledge is incomplete (examples of vertices in $\mathcal{B}$ are rooms containing doors not yet opened or junctions with corridors not yet explored). However, taking a shortcut is risky since the expected shortcut may actually be a dead end, forcing the robot to turn back and replan its path. The said shortcut may also actually be longer than the safe path. Depending on its internal state (battery level, mission, ...), the robot may want to privilege the safe path or hope for luck and try an exploratory path which may lead to a decrease of traveled distance relative to the safe path while increasing the robot's knowledge of its environment. We call the algorithm responsible for path planning while considering shortcuts an *Exploratory Planner (EP)*. At the heart of Exploratory Planning is a trade-off between safety guaranteed by exploitation of existing knowledge and short and long term efficiency obtained through exploration of uncharted territory, described for example by Argamon-Engelson et al. (1998).

We additionally define Exploratory Digraph Navigation (EDN) as the PNSLAM problem of physically reaching a known destination $F \in \mathcal{G}$ from a known origin $O \in \mathcal{G}$ with decent navigational and computational cost. Exploratory Planning is the *Planning* component of EDN. In addition to EP, navigation and SLAM are required in order to physically traverse yet unexplored space, model it as vertices and edges and add these to $\mathcal{G}$. $\mathcal{G}^R$ is the theoretical limit of $\mathcal{G}$ when all available space has been explored.

We developed exploratory planning for use in Exploratory Digraph Navigation and more widely PNSLAM problems where a robot models its environment as a graph (Beeson, Jong, and Kuipers, 2005; Bosse et al., 2004; Choset and Nagatani, 2001) or as an occupancy grid (Elfes, 1989). Indeed, an occupancy grid can be treated as a graph where vertices are pixels and each vertex is connected to its neighbors if space is traversable (chapter 2, Figure 2.2). Other problems related to graph theory but outside the field of robot navigation such as the Traveling Salesman's Problem or routing problems in changing networks may also benefit from Exploratory Planning. In these problems, the *computational cost* or algorithm execution time associated to planning is small but not necessarily negligible compared to the *navigational cost* associated to navigation and exploration, where the navigational cost is defined as the energy used or distance physically traveled summed on each move until destination is reached.

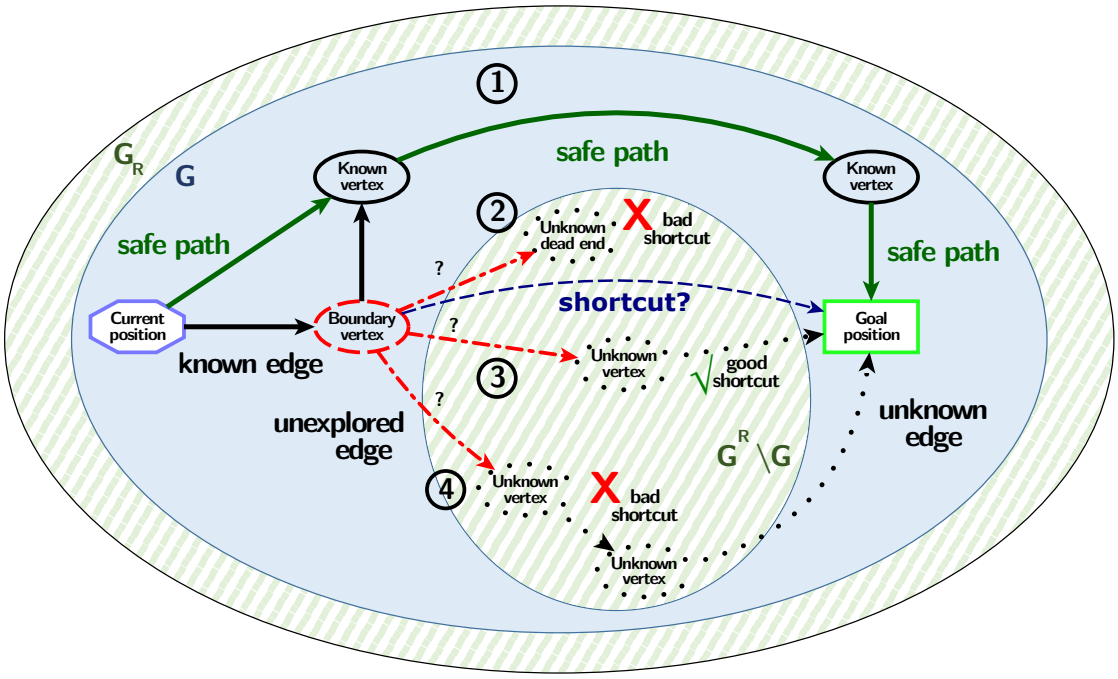



**Figure 3.3:** Typical example of Exploratory Planning (EP). At planning time, the navigating agent knows that the boundary vertex has yet unexplored edges, because for instance there is a door it has never opened. Compared to using an already known and safe path (1) to the goal, traversing yet unexplored space may (3) or may not (2, 4) reduce the navigational cost. EP heuristically decides whether or not to use exploration.

### 3.2.2 What Exploratory Digraph Navigation is not

To our knowledge, EDN has never been formulated as such. However, a number of close PNSLAM problems have already been formulated. Most of them are still open as of 2014.

The Canadian Traveler Problem (CTP) (Bar-Noy and Schieber, 1991; Karger and Nikolova, 2007) and the Stochastic Shortest Path Problem with Recourse (SSPPR) (Polychronopoulos and Tsitsiklis, 1996) are concerned with graphs where some edge traversal costs may be unknown (edges may not even exist) at planning time. However, the edge traversal cost is unknown but the extremities of explored and unexplored edges are still known, which is not true for unexplored edges in EDN. D* and D*Lite (Koenig and Likhachev, 2005; Koenig, Likhachev, and Furcy, 2004; Stentz, 1995) are known approximate CTP/SSPPR solvers used for mobile robot planning and exploration (see (Koenig, Tovey, and Smirnov, 2003) for instance). EDN can solve navigation problems on graphs $\mathcal{G}$ with unknown edge costs but is not restricted to graphs with all possible edges or all possible topologies known at planning time.

EDN is a generalization of graph exploration (Awerbuch et al., 1999; Betke, 1991; Dessmark and Pelc, 2004; Panaite and Pelc, 1999; Smirnov et al., 1996) since increasing the knowledge of $\mathcal{G}$ relative to $\mathcal{G}^R$ is not mandatory in EDN but any EDN solver should be able to perform exploration at least when $O$ and $F$ initially belong to different components of $\mathcal{G}$.

The shortest path between $O$ and $F$ in a partially unknown environment is provably computed by PHA* (Felner et al., 2004) for future use at the expense of immediate navigational cost. EDN is a navigation problem aiming at *reaching $F$ with minimum navigational cost*. As a side effect, a reusable path from $O$ to $F$ is found which hopefully exhibits a low (but not necessarily minimal) navigational cost. As opposed to the approach of Argamon-Engelson et al. (1998), EDN does not require the whole graph to be considered for each planning step and only considers exploration reducing the navigational cost to the goal (immediate reward).

### 3.2.3 Exploratory Digraph Navigation as a stochastic problem

EDN can be treated stochastically (stochastic edge or vertex traversal cost). The study of pathfinding on graphs with stochastic properties, pioneered notably by Loui (1983) is an active field of research, notably with the Markov Decision Process formulation (Puterman, 1994). EDN can conform to the formalism of stochastic graphs if every boundary vertex is connected to every other boundary vertex or directly to $F$ with virtual edges whose traversal costs are described by probability distributions. The resulting

problem can then be solved using a utility criterion (Loui, 1983) or other techniques out of scope of this thesis. However, the choice of the probability distribution for each virtual edge is not trivial notably due to destinations of unexplored edges possibly already belonging to $\mathcal{G}$ (unexplored loops). We therefore prefer a non-stochastic approach with a single number instead of a probability distribution to represent the length of virtual edges. Our approach of EDN nonetheless shares with a recent approach of the stochastic shortest path problem by Trevizan and Veloso (2014) its tunable search horizon and penalization of intermediary goal states.

## 3.3 EDNA*

We introduce a new algorithm, EDNA* (Exploratory Digraph Navigation using A*) which achieves EDN using an improved version of A* for planning.

### 3.3.1 From A* to EDNA*

Let $O_0$ be the starting position of the navigating agent in the usual Euclidean space $\mathbb{R}^2$ or $\mathbb{R}^3$. $O_n$ with $n > 0$ is the position of the navigating agent after the $n^{th}$ run of EDNA*. $O$ is always the starting position of the current run. We need the hypothesis that $\mathcal{G}^R$ is static (it does not change during the experiments) for the theoretical proofs. SLAM experiments reported in chapter 6 show that EDNA* may also work on dynamically changing graphs though.

A path (sequence of edges from one vertex to another) is said to be admissible in a graph if it has the lowest navigational cost amongst all possible paths. Multiple paths can be admissible. $\forall (X, Y) \in \mathcal{V}^2$, let $D(X, Y)$ be the (non-commutative) navigational cost from $X$ to $Y$ following an admissible path on $\mathcal{G}$. $\mathcal{G}$ can carry arbitrary per-vertex and per-edge traversal costs, as long as each vertex traversal cost is greater or equal to zero and each edge traversal cost is strictly positive. If no path exists from $X$ to $Y$, we take the convention that $D(X, Y) = \infty$.

EDNA*, like A*, requires an admissible but not necessarily consistent heuristic $\mathcal{H}$ (Dechter and Pearl, 1985) estimating the distance between two vertices. For given vertices $(X, F) \in \mathcal{V}^2$, the value of the heuristic is written $\mathcal{H}(X, F, \mathcal{G})$, abridged to $\mathcal{H}(X)$ if unambiguous. The admissibility constraint ($\mathcal{H}$ never overestimates navigational costs) writes $\forall (X, Y) \in \mathcal{V}^2, \mathcal{H}(X, Y, \mathcal{G}) \leq D(X, Y)$. When not mentioned otherwise, the $\mathbb{R}^2$ metric space with $\mathcal{H}(X, F, \mathcal{G}) = ||F - X||$ ($L_2$ norm) is used since it is the most common space in path-planning applications.

It is not possible to add a shortcut discovery strategy to A* by modifying $\mathcal{H}$ for two

reasons. First, taking a shortcut is risky so that $\mathcal{H}$ should be increased, which in turn is in conflict with the admissibility constraint on $\mathcal{H}$. Second, a boundary vertex can be found on an admissible path, in which case there is no reason for penalizing it. For these reasons, EDNA* defines a second heuristic, $\mathcal{R}$, the *shortcut risk heuristic*, on boundary vertices. $\mathcal{R}$ estimates the navigational cost of a shortcut while taking into account the risk that this cost was underestimated. Given the origin $O$ of an EDNA* planning step and $Z \in \mathcal{B}$, the value of the heuristic writes $\mathcal{R}(O, F, Z, \mathcal{G})$. We insist on the fact that $\mathcal{R}$ does not replace $\mathcal{H}$, even on boundary vertices where it is used to add information to $\mathcal{H}$. $\mathcal{R}(O, F, Z, \mathcal{G})$ is abridged to $\mathcal{R}(O, Z)$ or $\mathcal{R}(Z)$ if there is no contextual ambiguity. There is no constraint on $\mathcal{R}$ (it can be negative or even infinite) even though we will see that there is no interest in having $\mathcal{R}(O, F, Z, \mathcal{G}) < D(O, Z) + \mathcal{H}(Z, F, \mathcal{G})$. The value of $\mathcal{R}$ relative to $D(O, Z) + \mathcal{H}(Z, F, \mathcal{G})$ determines how reluctant the navigating agent is to try and discover shortcuts. The idea of using multiple heuristics in a single A* run has also been developed by Aine et al. (2016) under the name "Multi-Heuristic A*". However, while Aine et al. use the heuristics to guide the search in order to reduce the computational cost of the approach, EDNA* uses the new heuristic to find shorter paths going through yet unexplored space. Additionally, the $\mathcal{H}$ and $\mathcal{R}$ heuristics in EDNA* compete with each other, the first one describing a purely navigational strategy and the second one describing a partially exploratory strategy. The heuristics of Multi-Heuristic A* all describe purely navigational strategies.

Finally, we define an EDNA* run as the combination of a planning phase, purely computational, where a path to some vertex $X \in \mathcal{V}$ (not necessarily $F$) is computed on $\mathcal{G}$, and of a plan execution (or navigation) phase, purely navigational, where the agent physically navigates on $\mathcal{W}$ from its current position to the position $x \in \mathcal{W}$ corresponding to $X \in \mathcal{G}^R$ while updating its current position on $\mathcal{G}$ through SLAM. When at $X$, if $X \neq F$ then $X \in \mathcal{B}$ and the agent takes the "most promising" unexplored edge simultaneously on $x$ and $X$ and follows it until it reaches another vertex. We embed the concept of "most promising" or "best" edge into $\mathcal{R}$ by saying that for each boundary vertex, its $\mathcal{R}$ value is computed with the best edge in mind. For instance, the best edge may be the one whose angular difference with the goal vertex $F$ in a metric space is the best amongst all unexplored edges on $X$. The most promising edge can also be chosen randomly, possibly resulting in higher navigational costs on average.

### 3.3.2 Exploratory Planning with EDNA* (Algorithm 1)

Like A* (Hart, Nilsson, and Raphael, 1968), EDNA* planning has two phases. The first one is a vertex expansion phase where vertices $X$ are considered in order of increasing expected navigational cost $\delta(X) = D(O, X) + \mathcal{H}(X, F)$ and starting with $O$. The priority queue used to store vertices and their navigational cost $(X, \delta(X))$, also known as *open set*, is called $Q$. If $\mathcal{H}$ is not consistent, a list $S$ (called *closed set*) of already encountered vertices is maintained since vertices can be encountered again with a lower $\delta$. In the

second phase, an actual path is computed starting at $F$ or $Z \in \mathcal{B}$ and going back to $O$. The interested reader will report to open-source implementations of A* for details and implementation of the second phase. Planning is done on $\mathcal{G}$ and does not imply any movement in $\mathcal{W}$.

EDNA* has the same algorithmic complexity as A*, that is up to exponential in the size of the graph (Pearl, 1984; Russell and Norvig, 2009). On graphs describing physical environments, vertices have a bounded number of outgoing edges (or out-degree) and the time complexity of the algorithms can be measured in terms of vertices expanded during the vertex expansion phase. Figure 3.6 shows that expanded vertices are located within a kind of ellipse whose focal points are $O$ and $F$. Consequently, the effective time complexities of A* and EDNA* are described by the surface of the ellipse, that is between linear and quadratic in the distance to destination $||F - O||$ depending on the layout of obstacles (supposing uniformly distributed vertices in the graph).

**Algorithm 1** Exploratory Planning with EDNA*

**Input:** $O$, $F$, $\mathcal{G}$, $\mathcal{H} : X \to \mathcal{H}(X, F, \mathcal{G}), \mathcal{R} : Z \to \mathcal{R}(O, F, Z, \mathcal{G})$
  dest $\leftarrow \varnothing$; best_distance $\leftarrow \infty$; $Q \leftarrow (O, \delta(O))$; $S \leftarrow \varnothing$
  **while** $Q \neq \varnothing$ **do**
  | pop $(X, \delta(X))$ from $Q$; $S \leftarrow (X, \delta(X))$
  | **if** $\delta(X) >$ best_distance **then**
  | | **break** //EDNA* early stopping criterion
  | **if** $X = F$ **then**
  | | dest $\leftarrow F$; best_distance $\leftarrow \delta(F)$
  | | **break** //traditional A* stopping criterion
  | **if** $X \in \mathcal{B}$ **then**
  | | **if** $\mathcal{R}(O, F, X) \leq$ best_distance **then**
  | | | dest $\leftarrow$ X; best_distance $\leftarrow \mathcal{R}(O, F, X)$
  | **for all** neighbors $T$ of $X$ **do** compute $\delta(T)$
  | | **if not** ( $(T \in S$ **and** $\delta(T) \geq \delta(T)_S)$ **or**
  | | $(T \in Q$ **and** $\delta(T) \geq \delta(T)_Q)$ ) **then**
  | | | $Q \leftarrow (T, \delta(T))$
  **if** dest $= \varnothing$ **then return** failure //no path can be found
  unroll a path from dest back to $O$ and return the path found

### 3.3.3 Properties of the EDNA* exploratory planning algorithm

At the end of the algorithm, EDNA* returns a path to $F$ or to a boundary vertex $Z$ from which exploration must take place. If no path from $O$ to $F$ exists on $\mathcal{G}$ and there is no boundary vertex accessible, the algorithm cannot find a path. If no path exists from $O$ to $F$ and there is at least one boundary vertex, the algorithm will always give the shortest

path leading to the boundary vertex $Z$ with the lowest $\mathcal{R}(O, F, Z)$ encountered during expansion. It will thus perform similarly to Agent-Centered A* (Smirnov et al., 1996) if $\forall\ Z \in\ \mathcal{B}, \mathcal{R}(O, F, Z, \mathcal{G}) <= D(O, Z) + \mathcal{H}(Z, F, \mathcal{G})$. EDNA* is a greedy algorithm since the path choice is always optimized for the current origin $O$. Finally, EDNA* always expands less vertices than A* during one planning phase since vertices expanded by EDNA* would be expanded by A* but the early stopping criterion based on $\mathcal{R}$ may stop algorithm 1 before the whole A* search space has been expanded. However, $F$ may not be reached in one run, so that EDNA*'s search space may end up being bigger than A*'s.

### 3.3.4 Navigation with EDNA* (Algorithm 2)

In order for the agent to actually move in the world, a *topological navigation* algorithm compatible with the EDN paradigm is required. The simple algorithm 2 performs the navigation task while being independent of the underlying planner (here EDNA*). Physically following a path or exploring an edge means that the navigating agent simultaneously moves in $\mathcal{W}$, which incurs a navigational cost, and updates its current position on $\mathcal{G}$. Vertex recognition and map update must be handled by a separate algorithm, such as those used for topological SLAM (see (Bosse et al., 2004) or chapter 5 for example). Figure 3.4 shows an example of EDNA* reducing the navigational cost compared to A* thanks to a shortcut. Figure 3.5 shows a more complete EDNA* run highlighting the steps of the algorithm and the role of $\mathcal{R}$. Finally, Figure 3.6 gives a geometrical interpretation of A*'s guiding heuristic $\mathcal{H}$ and EDNA*'s risk heuristic $\mathcal{R}$.

**Algorithm 2** Navigation with EDNA*

**Input:** $\mathcal{W}$, $O$, $F$, $\mathcal{G}$, $\mathcal{H} : X \rightarrow \mathcal{H}(X, F, \mathcal{G})$
**Input:** $\mathcal{R} : Z \rightarrow \mathcal{R}(O, F, Z, \mathcal{G})$
$n = 0, O_n \leftarrow O$ // origin
**while** $O_n \neq F$ **do**
  **call** EDNA* Planning$(O_n, F, \mathcal{G}, \mathcal{H}, \mathcal{R})$
    $\rightarrow$ Returns path to $O_{n+1}$
  physically follow path from $O_n$ to $O_{n+1}$ //takes time
  $n \leftarrow n + 1$
  **if** $O_n = F$ **then break** //successfully reached $F$
  physically explore best edge $E$ to reach a vertex $O_{n+1}$
  $n \leftarrow n + 1$, insert $E$ in $\mathcal{G}$ //new edge
  **while** $(O_n \notin \mathcal{G})$ **do**
    insert $O_n$ in $\mathcal{G}$ //new vertex
    **if** $O_n \notin \mathcal{B}$ **then break** //dead end
    physically explore best edge $E$ to reach $O_{n+1}$
    $n \leftarrow n + 1$, insert $E$ in $\mathcal{G}$ //new edge

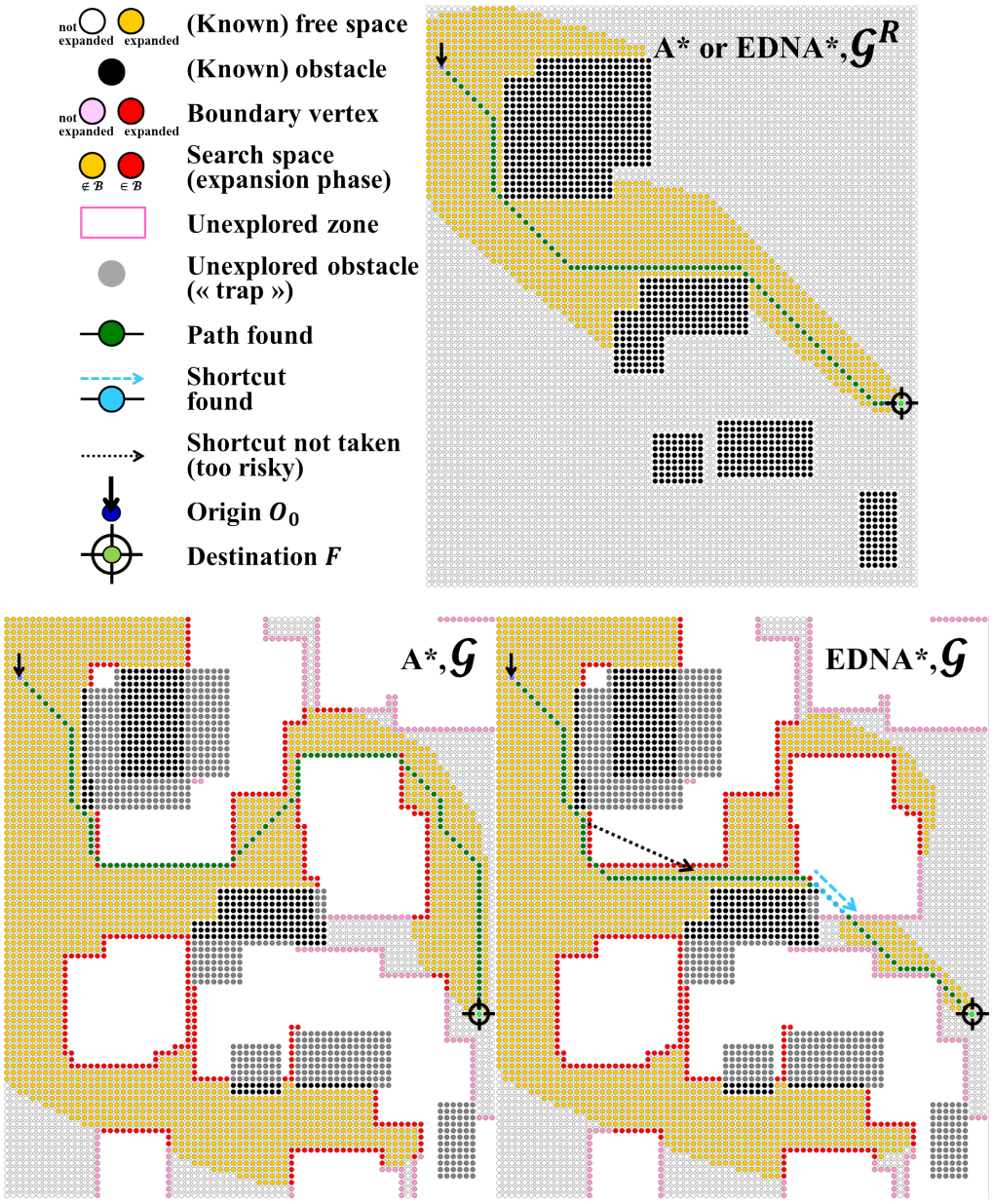



**Figure 3.4:** EDNA* shortcut discovery as described in algorithms 1 and 2. 8-way connected grid-like graphs used for a better understanding of the principles at work.

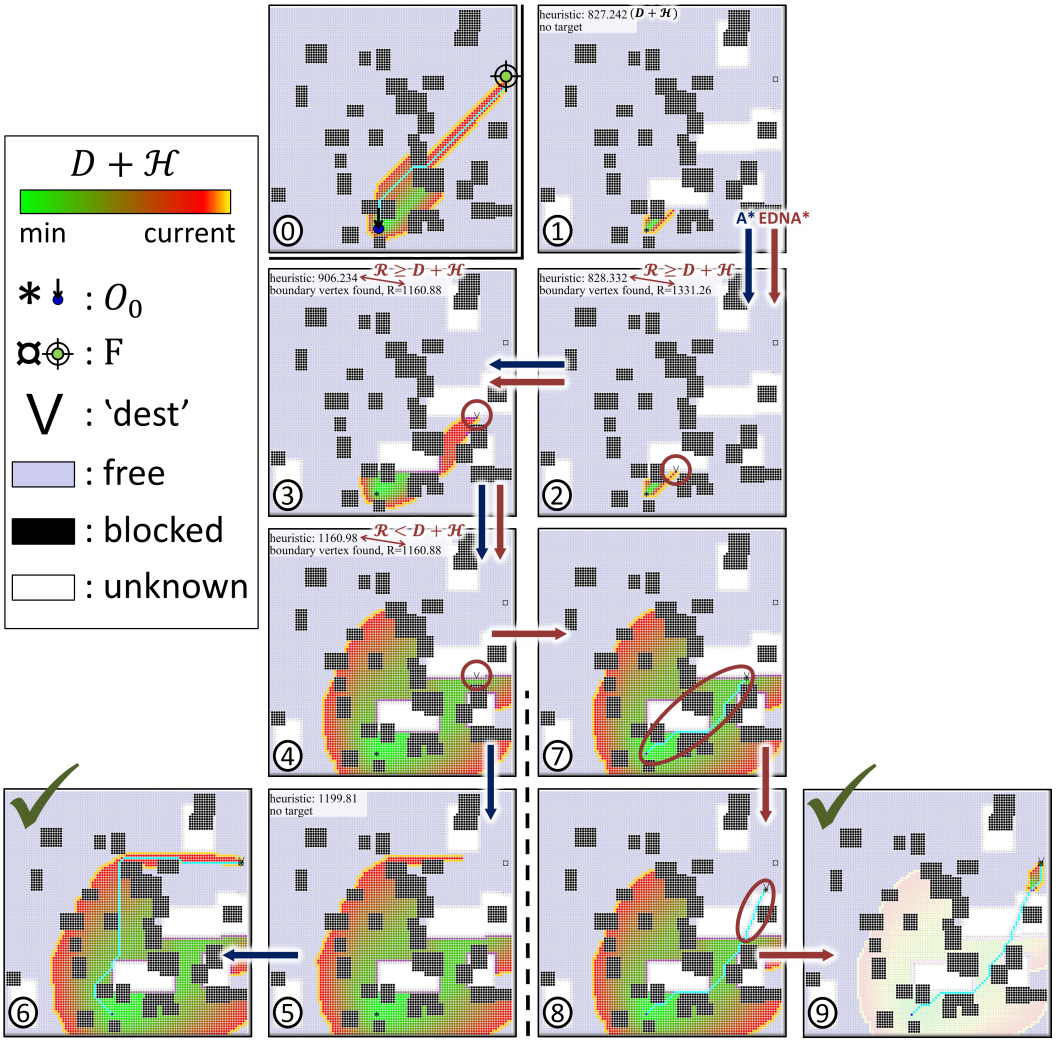



**Figure 3.5:** EDNA* shortcut discovery as described in algorithms 1 and 2. 8-way connected grid-like graphs used for a better understanding of the principles at work. EDNA* starts exactly like A* (1). During step (2), a shortcut hypothesis is made. This shortcut is obsoleted by a better one in step (3). Later (4) and as detailed on Figure 3.6, $\mathcal{R}$ reaches $D+\mathcal{H}$, at which point EDNA* forks from A* by taking the shortcut (7) while A* keeps trying to reach the destination (5). A* finally reaches the goal (6), which EDNA* also does (9) after an exploration phase (8). Compared to the theoretically optimal path (0), EDNA* returns a marginally longer path while A* leads to a non-negligible detour.

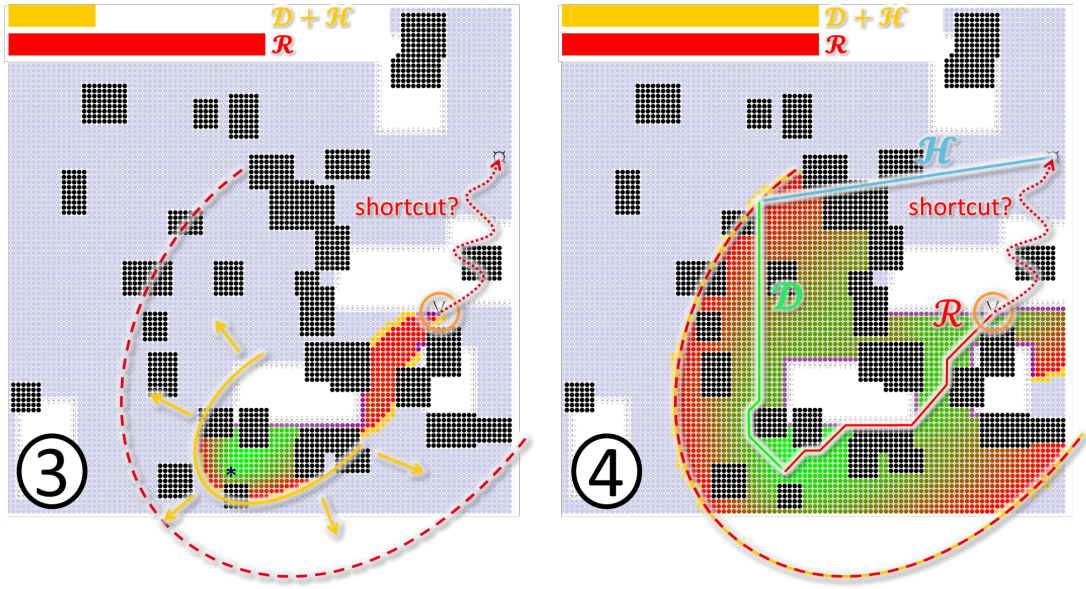



**Figure 3.6:** This Figure is a zoom on steps (3) and (4) of Figure 3.5. Vertices which are going to be expanded next by A* or EDNA* (colored in yellow on the figure) verify $D + \mathcal{H} = K$ with $K$ a constant. $K/\mathcal{H}(O, F)$ represents the minimum detour to reach the destination given the current state of the algorithm. No path from $O$ to $F$ may be shorter than $K$. The equation $D + \mathcal{H} = K$ defines a kind of ellipse whose focal points are $O$ and $F$ and whose eccentricity $e = \frac{\mathcal{H}(O,F)}{K}$ reflects the minimum possible detour given the current state of the algorithm. It is not a true ellipse since $D$ is an on-graph distance while $\mathcal{H}$ is an Euclidean distance. $\mathcal{R}$ can be represented as another ellipse with the same focal points but eccentricity $\frac{\mathcal{H}(O,F)}{\mathcal{R}}$. EDNA* chooses a shortcut as soon as both ellipses become equal.

### 3.3.5 Convergence proof and exploration variant

Let $e$ and $e_{max}$ be the amount of explored edges in the component around $O_0$ of $\mathcal{G}$ and $\mathcal{G}^R$ respectively.

**Theorem 1** (Navigation success and upper bound)**.** *If $\mathcal{G}^R$ is strongly connected, EDNA* will always lead the navigating agent to $F \in \mathcal{G}^R$ from any $O_0 \in \mathcal{G}^R$. A maximum of $1 + (e_{max} - e)$ EDNA* navigation runs will be necessary.*

**Corollary 1** (Complete graph exploration)**.** *If $F$ is unreachable from $O_0$ because both vertices belong to a different component in $\mathcal{G}^R$, at most $(e_{max} - e)$ EDNA* navigation runs will lead to exploration of the whole component containing $O_0$ before returning an error. The $1 + (e_{max} - e)$ bound is tight.*

*Proof.* An EDNA* run either leads to $F$, returns an error or registers at least one new edge in $\mathcal{G}$, so that at most $(e_{max} - e)$ runs would lead to $e = e_{max}$. As EDNA* is exactly A* when the graph is completely known ($\mathcal{G} = \mathcal{G}^R$), a last EDNA* run on $\mathcal{G}^R$ will lead to $F$ if reachable or return an error if not reachable, but after having discovered the component containing $O_0$ (otherwise, there would be a boundary vertex which EDNA* would target). The bound is tight considering a chain of vertices connected in $\mathcal{G}^R$ but not in $\mathcal{G}$ (Figure 3.7). □

Using $F \notin \mathcal{G}^R$, corollary 1 can be used to achieve complete exploration of an unknown graph prioritizing a specific position or direction ("goto approximate coordinates" mission). The position of $F$ will be used to bias shortcut discovery, but as the vertex cannot be reached, the whole subgraph around $O_0$ will be explored, starting with the part closest to $F$. In order to explore the environment radially around $O_0$, giving $O_0$ as target position but not taking $F = O_0$ is sufficient.

It should be noted that each run of EDNA* leads to up to one unit of long-term memory (one new edge) being used. Thus, EDNA* is guaranteed to reach the goal, but at the expense of using up to $(e_{max} - e)$ units of long-term memory. In a context where $Card(\mathcal{V}^R)$ is potentially huge, this may become an issue. However, most of the time, EDNA* will use much less than $(e_{max} - e)$ units of memory. The memory aspects of EDNA* are further studied in chapter 8.

### 3.3.6 Theoretical study of the risk heuristic

$D_{\mathcal{G}^R}$ indicates that the navigational cost is computed on $\mathcal{G}^R$ instead of $\mathcal{G}$. From a theoretical point of view, six cases, represented on Figure 3.8, are particularly interesting regarding the $\mathcal{R}$ heuristic:

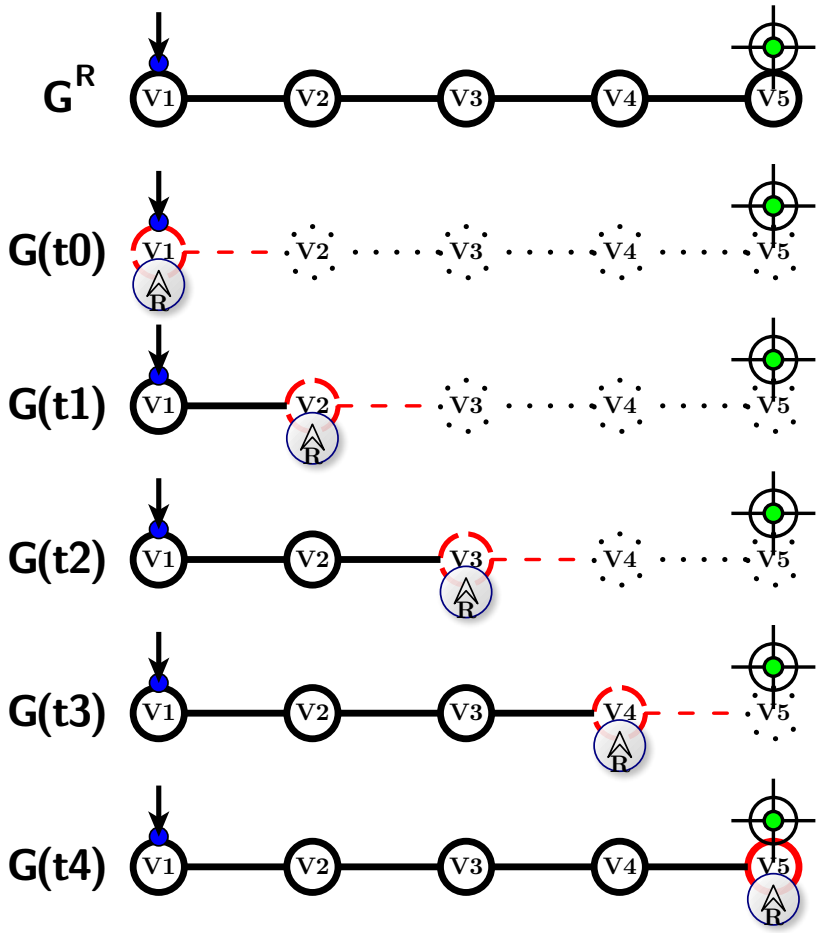



**Figure 3.7:** With a chain of vertices initially unexplored, EDNA* may need to rediscover all missing edges, so that the $(e_{max} - e)$ upper bound on the number of EDNA* runs necessary to reach $F$ is tight. Notations for this sketch are defined on Figures 3.2 and 3.4. The navigating agent "R" (for "R"obot) is always located on a boundary vertex until it reaches its destination $F = V_5$.

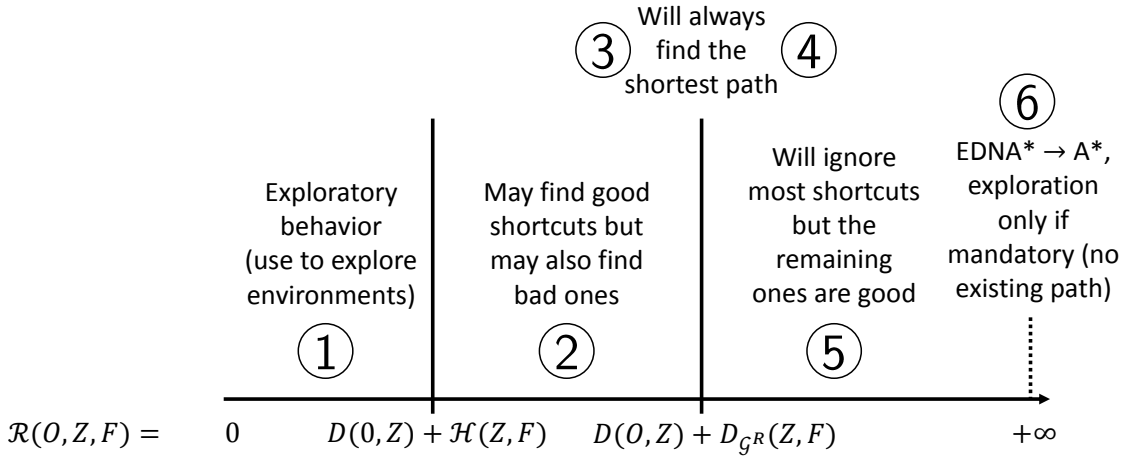



**Figure 3.8:** The behavior of EDNA* changes with the value of the risk heuristic, supposing that the criteria on $\mathcal{R}(0,F,Z)$ are valid $\forall\ Z \in\ \mathcal{B}$, which is almost never the case in practice.

1. early stopping to privilege exploration:
   $\forall\ Z \in\ \mathcal{B}, \mathcal{R}(O,F,Z,\mathcal{G}) <= D(O,Z) + \mathcal{H}(Z,F,\mathcal{G})$

2. shortcut length underestimation:
   $\forall\ Z \in\ \mathcal{B}, \mathcal{R}(O,F,Z,\mathcal{G}) < D(O,Z) + D_{\mathcal{G}^R}(Z,F)$

3. optimal case:
   $\forall\ Z \in\ \mathcal{B}, \mathcal{R}(O,F,Z,\mathcal{G}) <= D(O,Z) + \mathcal{H}(Z,F,\mathcal{G})$
   and $\forall\ X\ \in\ \mathcal{G}, \mathcal{H}(X,F,\mathcal{G}) = D_{\mathcal{G}^R}(X,F)$

4. optimal improvement over uninformed A* on $\mathcal{G}$:
   $\forall\ Z \in\ \mathcal{B}, \mathcal{R}(O,F,Z,\mathcal{G}) = D(O,Z) + D_{\mathcal{G}^R}(Z,F)$

5. improvement over uninformed A* on $\mathcal{G}$:
   $\forall\ Z \in\ \mathcal{B}, \mathcal{R}(O,F,Z,\mathcal{G}) > D(O,Z) + D_{\mathcal{G}^R}(Z,F)$

6. EDNA* reduced to A* on $\mathcal{G}$:
   $\forall\ Z \in\ \mathcal{B}, \mathcal{R}(O,F,Z,\mathcal{G}) \to \infty$

#### Case 1: early stopping to privilege exploration

Case 1 sees the expansion phase stopping at the first boundary vertex encountered due to the early stopping criterion. If $\forall X \in \mathcal{G}, \mathcal{H}(X) = 0$ (Dijkstra's algorithm), with $F$ unreachable (or no $F$ at all) and with unitary traversal costs, this would result in the

greedy algorithm *Nearest Neighbor (NN)* (Koenig and Smirnov, 1996). NN's penalty in terms of traversed edges on a non-directed graph during exploration relative to traversed edges if the graph was known is only worst-case logarithmic in the number of edges (Megow, Mehlhorn, and Schweitzer, 2011). Bounds for algorithms operating on digraphs are still an open problem (Megow, Mehlhorn, and Schweitzer, 2011) but due to its greedy nature, we expect EDNA* to reach a competitive ratio of $n-1$ with $n$ the number of vertices in $\mathcal{G}^R$ (Förster and Wattenhofer, 2012). EDNA*'s relocation strategy (using known edges to travel from a newly completed vertex to a boundary vertex) is however not specifically optimized for exploration tasks compared to that of Albers and Henzinger (1997) or Fleischer and Trippen (2005). Case 1 can be exploited to perform exploration tasks.

#### Case 2: shortcut length underestimation

In case 2, shortcut lengths are underestimated, which may lead to detours. However, compared to case 1, EDNA* does not necessarily choose the first shortcut available.

#### Cases 3 and 4: optimal improvement over uninformed A*

In cases 3 and 4, heuristic path length estimations are replaced by their actual value on $\mathcal{G}^R$, so that EDNA* is fully informed and can find shortest paths in $\mathcal{G}^R$ using only $\mathcal{G}$ for planning. In other words, the information about $\mathcal{G}^R$ missing in $\mathcal{G}$ is carried by the $\mathcal{H}$ and $\mathcal{R}$ heuristics. In case 3, the information is carried by $\mathcal{H}$ so that EDNA* is also optimal in terms of computational effort (number of vertices expanded). On the contrary, in case 4, the information on $\mathcal{G}^R$ is carried by $\mathcal{R}$. Consequently, in case 4, the early stopping criterion of algorithm 1 is triggered optimally but the number of vertices expanded before triggering it or reaching $F$ is not necessarily optimal. The properties of cases 3 and 4 can be expressed by the following theorems:

**Theorem 2** (admissible EDNA*)**.** *In cases 3 and 4, EDNA∗ on $\mathcal{G}$ follows a path admissible on $\mathcal{G}^R$.*

**Theorem 3** (minimum search space)**.** *In addition to theorem 2, EDNA* on $\mathcal{G}$ in case 3 only expands vertices on admissible paths from $O_0$ to $F$ in $\mathcal{G}^R$.*

*Proof.* [Proving theorem 2]

This proof is done in the conditions of theorem 1. Suppose that case 3 or case 4 is verified.

Let $P$ be the property "The robot (or navigating agent) is on an admissible path from $O_0$ to $F$ on $\mathcal{G}^{\mathcal{R}}$". Let us prove that if $P$ is true at the beginning of a planning step, algorithm 1 will ensure that $P$ remains true. If $P$ is true before planning step $n+1$, then $O_n$ is on a path from $O_0$ to $F$ admissible on $\mathcal{G}^R$. During planning step $n+1$, if $X \in \mathcal{V}$ is on a path from $O_n$ to $F$ admissible on $\mathcal{G}^R$, then $X$ is also on a path from $O_0$ to $F$ admissible on $\mathcal{G}^R$. Consequently, if $P$ is valid up to planning step $n+1$, then $P$ for planning step $n+1$ can be formulated "The robot (or navigating agent) is on an admissible path from $O_n$ to $F$ on $\mathcal{G}^{\mathcal{R}}$".

If EDNA* returns a path to $F$ or to $\mathcal{Z} \in \mathcal{B}$, this path is necessarily admissible on $\mathcal{G}$ due to A* expanding vertices $X$ by order of increasing $\delta(X) = D(O_n, X) + \mathcal{H}(X, F) \leq D(O_n, X) + D(X, F) \leq D_{\mathcal{G}^R}(O_n, X) + D_{\mathcal{G}^R}(X, F)$ (the proof is the same as for the unmodified A* algorithm).

During planning step $n+1$, two cases are possible:

- (i) There exists a path from $O_n$ to $F$ in $\mathcal{G}$ admissible on $\mathcal{G}^R$ or
- (ii) There exists no path from $O_n$ to $F$ in $\mathcal{G}$ admissible on $\mathcal{G}^R$. Then, since an admissible path from $O_n$ to $F$ in $\mathcal{G}^R$ exists, there must be a boundary vertex $Z_0 \in \mathcal{B}$ with an unexplored edge $u \in \mathcal{U}$ belonging to this path and such that there exists a path from $O_n$ to $Z_0$ on $\mathcal{G}$ admissible on $\mathcal{G}^R$. Expressed simply, $Z_0$ is the first boundary vertex on the path admissible in $\mathcal{G}^R$

There are also only two possibilities concerning the vertex targeted by the algorithm: $F$ or $Z \in \mathcal{B}$. Which one is chosen is conditioned by the early stopping criterion:

- (a) the early stopping criterion is not triggered (a path to $F$ admissible on $\mathcal{G}$ is returned) or
- (b) the early stopping criterion is triggered (a path to $Z \in \mathcal{B}$ admissible on $\mathcal{G}$ is returned).

Combining the two binary choices results in four cases:

(a.i) This case corresponds to EDNA* falling back to traditional A*, which is guaranteed to return a path to $F$ admissible on $\mathcal{G}^R$, so that $P$ remains true.

(b.i) This combination is impossible. Indeed, in case (b), the algorithm returns a path to $Z \in \mathcal{B}$ with $\forall X \in \mathcal{V}, D(O_n, Z) + D_{\mathcal{G}^R}(Z, F) < D(O, X) + \mathcal{H}(X, F)$. Since $F \in \mathcal{V}$, with $X = F$ and the path admissible on $\mathcal{G}^R$ being used: $D(O_n, Z) + D_{\mathcal{G}^R}(Z, F) < D_{\mathcal{G}^R}(O_n, F)$ which is impossible since no path from $O_n$ to $F$ can be shorter than the one admissible on $\mathcal{G}^R$.

(a.ii) $F$ is reached. Moreover, $\forall Z \in \mathcal{B}, D_{\mathcal{G}^R}(O_n, F) \leq D(O_n, F) \leq \mathcal{R}(O_n, F, Z, \mathcal{G}) = D(O_n, Z) + D_{\mathcal{G}^R}(Z, F)$. If $Z_0$ exists (case (ii)), then $D(O_n, Z_0) + D_{\mathcal{G}^R}(Z_0, F) = D_{\mathcal{G}^R}(O_n, F) = \min_{Z \in \mathcal{B}}(D(O_n, Z) + D_{\mathcal{G}^R}(Z, F))$, meaning that $D(O_n, F) = D_{\mathcal{G}^R}(O_n, F)$. In this case, EDNA* found a path from $O_n$ to $F$ admissible on $\mathcal{G}^R$ and $F$ is reached with $P$ still true. However, we are not in case (ii) since there exists a path from $O_n$ to $F$ in $\mathcal{G}$ admissible on $\mathcal{G}^R$.

(b.ii) Let $Z_1 = argmin_{Z \in \mathcal{B}}(\mathcal{R}(O_n, F, Z, \mathcal{G}) = D(O_n, Z) + D_{\mathcal{G}^R}(Z, F))$. We know that $\mathcal{R}(O_n, F, Z_0, \mathcal{G}) = \mathcal{R}(O_n, F, Z_1, \mathcal{G}) = D_{\mathcal{G}^R}(O, F)$ (admissible path), so EDNA* will return a path to $Z_0$ or $Z_1$ on an admissible path from $O_n$ to $F$ and $P$ remains true.

Since $P$ remains true for any planning step, EDNA* can only lead the robot (or navigating agent) on admissible paths in $\mathcal{G}^R$. The robot must move at each step. Furthermore, it cannot go backwards from $O_n$ since going backwards would increase the remaining distance to destination on $\mathcal{G}^R$ since all vertex and edge traversal costs are strictly positive. Thus, each planning/navigation step brings the robot (navigating agent) strictly closer to the goal according to the $D$ distance on $\mathcal{G}^R$.

[Proving theorem 3]

Only cases (a.i) and (b.ii) are possible according to the previous paragraphs.

(a.i) EDNA* expands vertices $X$ by order of increasing $\delta(X) = D(O_n, X) + D_{\mathcal{G}^R}(X, F)$. Since for $X = F$, $\delta(F) = D_{\mathcal{G}^R}(O_n, F)$, no vertex not on an admissible path can be expanded before $F$ gets its $\delta(F)$ set (where the algorithms stops and unrolls the path). So, the theorem is verified.

(b.ii) Using the same arguments as for theorem 2 in case (b.ii), $Z_1$ belongs to a path from $O_n$ to $F$ admissible on $\mathcal{G}^R$. In case a vertex $X$ not on an admissible path gets expanded, $D(O_n, X) + D_{\mathcal{G}^R}(X, F) > D_{\mathcal{G}^R}(O_n, F)$, the early stopping criterion gets triggered and $Z_1$ gets targeted.

□

#### Cases 5 and 6: improvement over uninformed A* and fallback to A*

Case 5 results in EDNA* on $\mathcal{G}$ always overestimating the navigational cost of shortcuts (in case 6, all shortcuts are considered of infinite cost). This means that the stopping criterion can only be triggered if an exploratory path's navigational cost is actually lower than all non-exploratory ones. However, it may not be triggered if a path is found from $O$ to $F$ with a cost between $D_{\mathcal{G}^R}(O, F)$ and the lowest $\mathcal{R}$. For a given problem which EDNA* must solve, if bounds can be given on $D(Z, F)$ for every possibly interesting

boundary vertex $Z$, then bounds can be given on EDNA*'s sub-optimality in terms of navigational cost: the worst case detour relative to A* on $\mathcal{G}^{\mathcal{R}}$ in the case 5 is the estimated $D(O,Z)+D(Z,F)$ minus the actual one.

#### Practical considerations

In practice, while it is easy to avoid cases 1 and 6, it is not possible in general to fulfill the $\forall Z \in \mathcal{B}$ conditions of cases 3 and 4. Usually, only little knowledge is available about the length of shortcuts through yet unexplored space. So, some lengths may be underestimated, some may be overestimated and some may be approximately correctly estimated. By increasing $\mathcal{R}$ globally, the lengths of more shortcuts become overestimated, up to the point where all lengths are overestimated (case 5). $\mathcal{R}(O,F,Z) = D(O,F) + max_{(A,B)\in\mathcal{V}^R\times\mathcal{V}^R,A\neq B}(\frac{D(A,B)}{\mathcal{H}(A,B)}).\mathcal{H}(F,Z)$ will always achieve case 5 (it overestimates the length of any shortcut based on the worst-case detour relative to free space $max_{(A,B)\in\mathcal{V}^R\times\mathcal{V}^R,A\neq B}(\frac{D(A,B)}{\mathcal{H}(A,B)})$). It should be noted that case 5 will probably not be optimal in terms of *average* traveled distance because most shortcuts are ignored. Lowering $\mathcal{R}$ relative to case 5 may cause EDNA* to choose shortcuts which actually increase the length of the path ("bad" shortcuts, whose length is underestimated) but may also unlock access to some "good" shortcuts which would have been ignored with a higher $\mathcal{R}$. On the contrary, if $\mathcal{R}$ is too low, EDNA* will take more "bad" than "good" shortcuts, leading to an increase of traveled distances in average. The best strategy in terms of average traveled distance is to have some shortcuts underestimated (case 2) and some overestimated (case 5). The tricky balancing between underestimation and overestimation is studied experimentally in the next section.

## 3.4 Assessing the performances of EDNA*

Our objective is to demonstrate shortcut discovery i.e. EDNA* reaching $F$ from $O_0$ with a lower navigational cost than a non-exploratory strategy. The point of the following subsection is to determine which technique EDNA* should be compared to.

### 3.4.1 Finding a reference algorithm

Both EDNA* and D* (Lite) (Koenig and Likhachev, 2005; Koenig, Likhachev, and Furcy, 2004; Stentz, 1995) can provide suboptimal (greedy) navigation in a static Canadian Traveler Problem/Stochastic Shortest Path Problem with Recourse situation, such as on grid-like graphs. D* and its variants use a free-space assumption ("unexplored space is traversable") resulting in a systematical underestimation of the navigational

cost. As a consequence, transforming unexplored vertices to vertices with all edges traversable makes EDNA* with $\mathcal{R} = 0$ follow the D*Lite path but without D*Lite's search space optimization. D* (Lite) cannot work on digraphs where nothing is known about yet unexplored vertices, so that comparing EDNA* to D* does not make sense.

It is possible to use PHA* (Felner et al., 2004) for navigation by stopping the algorithm when $F$ is reached for the first time (Felner et al. do not mention this possibility). In this situation, the path returned would be either the A* path on $\mathcal{G}$ (infinitely overestimating the length of shortcuts, obtainable with EDNA* and $\mathcal{R} \to \infty$) or that of iterated Agent-Centered A* (Smirnov et al., 1996) (obtainable with EDNA* and $\mathcal{R} = 0$). Not stopping PHA* early would result in a lot of physical movements increasing the navigational cost and unnecessary for a navigation task. PHA* cannot balance physical exploration of unknown parts and navigation on known parts even though it ends up using both. Comparing PHA* to EDNA* would come down to comparing A* to EDNA* or iterated agent-centered A* to EDNA*.

More generally, EDNA* uses the exploration risk as a degree of freedom which none of the above algorithms does. EWP (Argamon-Engelson, Kraus, and Sina, 1998) considers balancing exploration and navigation, but no algorithm to do it during planning is described. As a consequence, each iteration of EWP must compute the navigational cost from the current position to any $v_1 \in \mathcal{B}$ (algorithm not explained in the article) and estimate the cost from it to any $v_2 \in \mathcal{B}$ which is orders of magnitude more compute-intensive than our approach. When using exploration, EWP navigates to the best $v_1$ from where it must reach the best $v_2$ using exploration even though another $v \in \mathcal{V}$ is reached while going from $v_1$ to $v_2$. As a consequence, the navigational cost from $O_0$ to $F$ ends up being always higher than that of our approach in the same situation, which is why we don't compare EDNA* to EWP.

Since EDNA* is a modification of A*, we chose to compare it to A* itself in terms of search space and navigation performances (path lengths). Map knowledge improvement due to adding newly discovered zones to $\mathcal{G}$ is not evaluated due to the difficulty of finding a realistic experimental protocol to do the evaluation. It is however expected to strongly reinforce the interest of EDNA* since future traversals can benefit from currently discovered shortcuts, dead ends and loops.

### 3.4.2 A simple choice of the risk heuristic for experiments

We chose to use a risk heuristic $\mathcal{R}(O, F, Z, \mathcal{G}) = D(O, Z) + \alpha\,(1 - \beta.cos(\theta))\,\mathcal{H}(Z, F, \mathcal{G})$ in an Euclidean space with the $L_2$ norm as $\mathcal{H}$. $\theta$ is the angle between an edge going out from $Z$ and the vector from $Z$ to $F$, $\beta$ is a parameter to describe an angular penalty and $\alpha$ is a parameter to describe the total penalty in the sense that if $\beta = 0$, $\alpha$ describes by how much a shortcut's navigational cost is overestimated relative to $\mathcal{H}(Z, F, \mathcal{G})$. ($\alpha \leq 1, \beta =$

0) for instance gives the first case of the previous section. This proposal of heuristic is inspired by the way humans actually find shortcuts in an unknown environment (possible shortcuts are edges which go in the right direction and would potentially save a lot of time or distance). For a high enough risk factor $\alpha$, $\mathcal{R}$ should fall in case 5 of the previous section. It should be noted that $\mathcal{R}$ does not take advantage of the planarity of a graph.

### 3.4.3 Benchmark protocol

#### Quality measures

Let $\mathcal{N}_{A*}$ and $\mathcal{N}_{EDNA*}$ be the navigational cost from $O_0$ to $F$ using A* and EDNA* on a graph $\mathcal{G}$. The navigational cost change from A* to EDNA* on $\mathcal{G}$ is $\mathcal{N}_{\mathcal{G}} = \frac{\mathcal{N}_{A*} - \mathcal{N}_{EDNA*}}{\mathcal{N}_{A*}}$. A $\pm 0.1$ change means that if the cost following the A* path is 100 units, the one following the EDNA* path is $100 \mp 10$. Similarly, a computational cost change $\mathcal{C}_{\mathcal{G}}$ is computed using the number of vertices visited during the expansion phase of the algorithms which reflects execution time.

#### Graph generation and data collection

$n_{\mathcal{G}^R} = 200$ random planar digraphs $\mathcal{G}^R$ with about 9 000 vertices and 12 500 dual-way edges each are generated, with square-shaped possibly-overlapping (SSPO) obstacles of varying size. Generating graphs allows control over obstacle shapes and obstacle density. $\mathcal{G}^R$ intends to mimic road networks or Generalized Voronoi Graphs (Choset and Nagatani, 2001) while being more labyrinthine and misleading in order to test the worst-case behavior of EDNA*. $\mathcal{G}$ is obtained by removing a fraction $\phi$ of the edges of $\mathcal{G}^R$ to create SSPO unexplored zones. Figure 3.9 shows a small zone of one of the graphs used.

In each *experiment attempt*, $\mathcal{G}$ is created from $\mathcal{G}^R$ and $O_0$ and $F$ are randomly chosen. Due to the edge removal procedure, $O_0$ and $F$ may belong to different components of $\mathcal{G}$, leading to A* being unable to find a path but EDNA* still finding one. Such cases represent more than 50% and up to 95% (high $\phi$) of all experiment attempts. They are eliminated as giving EDNA* an infinite navigational advantage over A*, thus strongly favoring A* with respect to EDNA*. In these cases, a new experiment attempt is done until a *successful experiment* occurs, where A* finds a path from $O_0$ to $F$. Successful experiments are accumulated until an *informative experiment* occurs, where $\mathcal{N}_{\mathcal{G}} \neq 0$ ($\mathcal{N}_{\mathcal{G}} = 0$ means $\mathcal{R}$ is not used). There are $n_s$ successful experiments including the last one which is informative. On each $\mathcal{G}^R$, 10 000 informative experiments are done, uniformly sampling 100 values in $[1; 9]$ for $\alpha$ and 100 values in $[10\%; 50\%]$ for $\phi$.

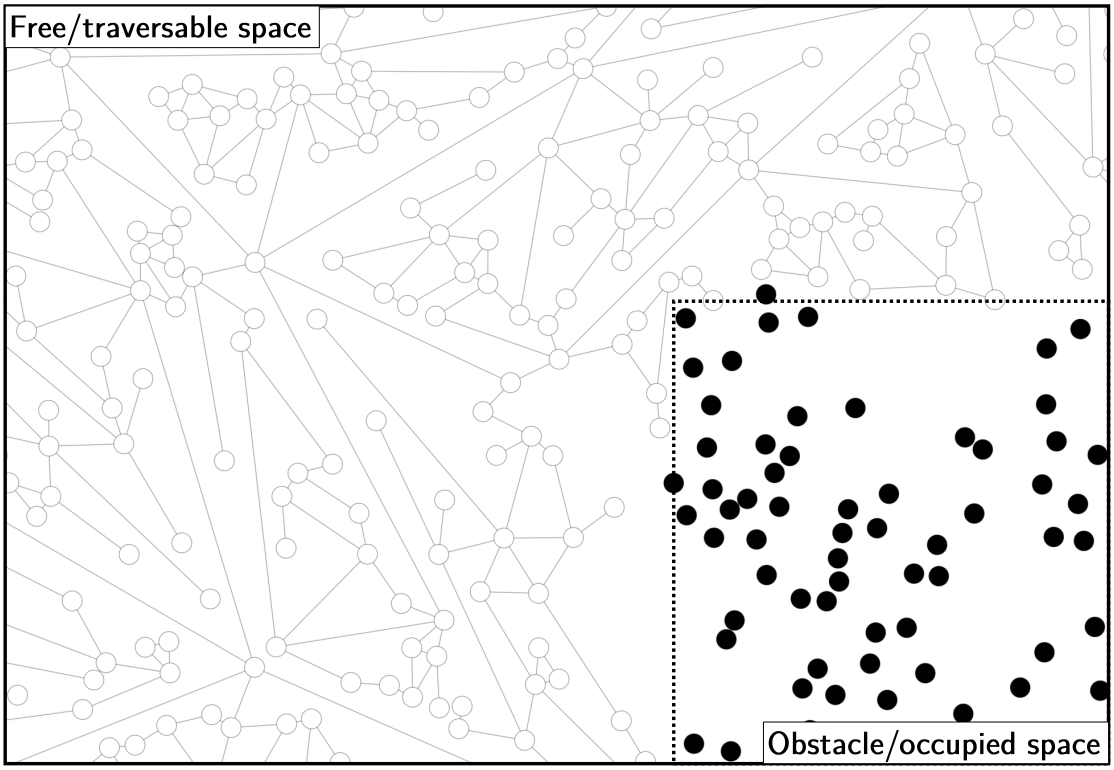



**Figure 3.9:** One of the random planar graphs used for benchmarking EDNA* (detail). Vertices in black fall in obstacle regions. Graphs are generated with 10 000 vertices and 15 000 edges inside a square box (20000 × 20000). The graph generator imposes a minimum distance between any two vertices and between any vertex and any edge ($\sqrt{5000}$), as well as a maximum edge length (1500) and a minimum angle between edges starting on the same vertex (30°). SSPO obstacles are then punched into the graph. While the initially produced graph has a single component, SSPO obstacle-punching may partition it in multiple components.

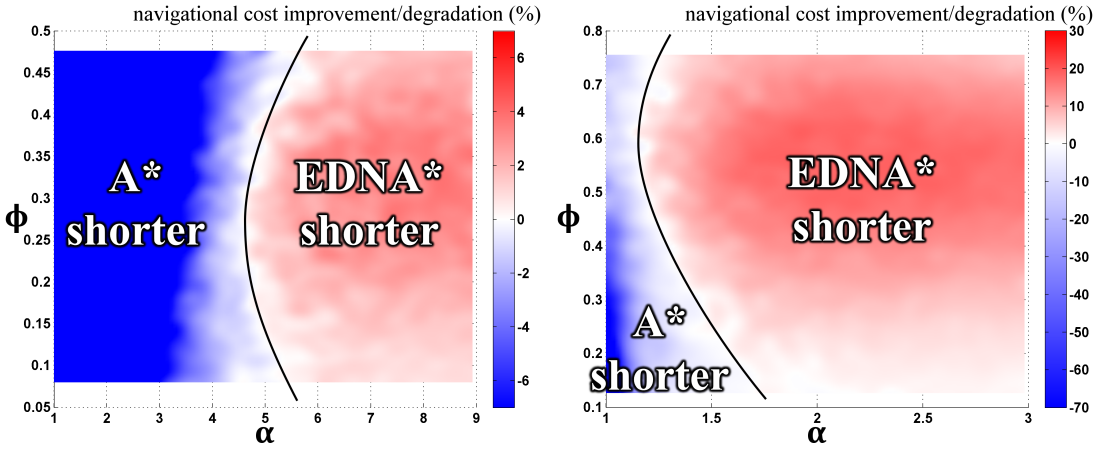



**Figure 3.10:** Average navigational cost ($\bar{\mathcal{N}}$) changes for successful experiments, random (left) and grid-like (right) graphs.

The estimators used for each $(\alpha, \phi)$ to retrieve the average navigational and computational cost changes plotted on figures 3.10 and 3.11 are $\bar{\mathcal{N}} = \frac{1}{n_{\mathcal{G}^R}} . \sum_{\mathcal{G}^R} \left( \frac{1}{n_s} . \sum_{\mathcal{G}} \mathcal{N}_{\mathcal{G}} \right)$ and $\bar{\mathcal{C}} = \frac{1}{n_{\mathcal{G}^R}} . \sum_{\mathcal{G}^R} \left( \frac{1}{n_s} . \sum_{\mathcal{G}} \mathcal{C}_{\mathcal{G}} \right)$. Both estimators average on all $\mathcal{G}^R$ the (poor) average on all successful experiments on a given $\mathcal{G}^R$. Arithmetically averaging changes is intuitive but highly unfavorable to EDNA* since a 0.9 degradation (navigational cost multiplied by 1.9) balances a 0.9 improvement (navigational cost divided by 10) of EDNA* over A*.

$\beta \sim 0.25$ was experimentally observed to give EDNA* the best navigational improvements over A*. Tests were also carried on 8-way connected grid-like graphs like that of Figure 3.4 for which $\beta \sim 0.125$ gave better results.

### 3.4.4 Results and discussion

#### EDNA* vs A* navigational cost ($\bar{\mathcal{N}}$) comparison

Figure 3.10, left shows EDNA* with $\alpha > 5$ always reducing $\bar{\mathcal{N}}$ compared to A* in average, demonstrating successful shortcut discovery. With $\alpha \sim 8$, informative experiments show navigational costs divided by up to 14 when switching from A* to EDNA*. With $\alpha > 8$, EDNA*'s success decreases with $\alpha$ because most experiments are not informative (early stopping is not triggered). The maximum performance gain of EDNA* over A* on random graphs is 3.9% with our experimental protocol unfavorable to EDNA* (since experimental results approximately follow a Gaussian distribution, the 95% confidence interval is $[0.5\%; 7.3\%]$). We observed that selection of the sole experiments where A*

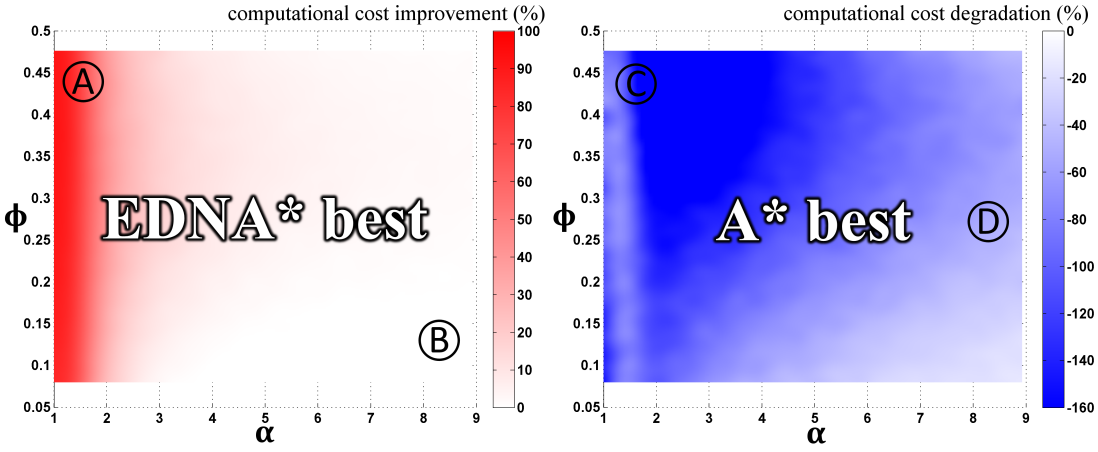



**Figure 3.11:** Average computational cost ($\bar{\mathcal{C}}$) changes for successful experiments on random graphs, up to the first boundary vertex (left) and up to $F$ (right).

reaches $F$ causes the A* shorter/EDNA* shorter limit to be biased towards high $\alpha$ when $\phi$ increases, thus further disadvantaging EDNA*. Results on grid-like graphs (Figure 3.10, right) show up to 20% improvement of navigational cost of EDNA* over A*. Finally, the curvature of the A* shorter/EDNA* shorter limit was unexpected. We think that it is an artifact due to how successful experiments are selected. So, only the low-$\phi$ part of the curve is actually valid and EDNA* finds even shorter paths than shown on the Figure for high $\phi$ values.

#### EDNA* vs A* computational cost ($\bar{\mathcal{C}}$) comparison

Figure (3.11, right) shows an increase from A* to EDNA* of $\bar{\mathcal{C}}$ when multiple EDNA* runs are required. This penalty for EDNA*, which exceeds 100% with low $\alpha$ and high $\phi$ (zone C) is most of the time located between 0 and 100%, decreasing linearly until the high $\alpha$ zone (D) (which is the interesting zone as seen in the paragraph on $\bar{\mathcal{N}}$) where it progressively tends to 0. As explained in the paragraph on navigational cost comparison, there is probably a bias in how successful experiments are selected, which causes an artificial increase of computational cost for EDNA*. Figure (3.11, left) shows the same result as Figure (3.11, right) but on a single EDNA* run where the stopping criterion allows EDNA* to always expands less space than A*. With a high $\phi$ and a low $\alpha$ (zone A), this difference reaches 100% (EDNA* explores close to no vertices). With a low $\phi$ and a high $\alpha$ (zone B), each run of EDNA* performs only marginally better than A* but there are less runs, so that EDNA*'s and A*'s performances are equivalent (with $\phi = 0$ or $\alpha \to \infty$, EDNA* would fall back to A*). EDN's focus is on navigational cost but the figures show a computational cost comparable to that of A*, which has been

proved optimal in a non-exploratory situation (Hart, Nilsson, and Raphael, 1968).

## 3.5 Conclusion on EDNA*

We presented the Exploratory Digraph Navigation problem and the EDNA* algorithm to solve it. With a $\mathcal{R}$ heuristic sufficiently overestimating shortcut lengths, the agent using EDNA* will on average move from 3.9% to more than 20% less than using A* to reach $F$, with best EDNA* runs dividing the navigational cost by up to 14. Moving less leads for instance to energy savings, especially if the path is intended to be taken multiple times like in routing problems. Shortcuts, dead ends and loops found during exploration phases are stored in $\mathcal{G}$. The impact of this knowledge reinforcement has yet to be evaluated. Minimizing computational cost is not the main focus of EDN problems where traversal is much slower than planning. Nonetheless, even if EDNA* tends to expand about twice as many vertices as A* during the whole navigation, it expands less per planning step which is interesting for real-time systems where single computations must terminate quickly. Moreover, if $O$ and $F$ are not connected on $\mathcal{G}$, A* fails and an exploration algorithm has to be used. EDNA* on the contrary does not fail and automatically resorts to exploration, thus combining the navigation (A*) and the exploration algorithm. The relative amount of navigation and exploration can be tuned through $\mathcal{R}$. Additional data such as metric or semantic properties or planarity could be integrated into $\mathcal{H}$ and $\mathcal{R}$ for better results. For instance, if $\mathcal{G}^R$ describes a transportation network with buses and trains (see chapter 8), $\mathcal{R}$ can be lowered on vertices/edges having train connections because these are typically faster than buses.

When using an exploratory strategy, it happens that the navigating agent starts to thoroughly explore a dead end. Ideas to deal with this situation have already been formulated for exploration algorithms (Smirnov et al., 1996). We implemented a non-stubbornness criterion for SLAM using EDNA* in chapter 5, which detects this situation on the current vertex $X$ by comparing $D(O_0, X) + \mathcal{H}(X, F)$ to $\mathcal{H}(O_0, F)$. Triggering this criterion leads to a replan with possibly an increase of $\mathcal{R}$, causing EDNA* to try finding a safe path instead of hoping for a non-existing shortcut. Another possible solution to the issue would be to give the navigating agent an energy budget which would decrease with each detour while $\mathcal{R}$ would increase. If it is known for sure that $\mathcal{G}^R$ is a planar graph, a face routing algorithm such as Greedy Perimeter Stateless Routing (GPSR) (Bose et al., 2001; Karp and Kung, 2000) can be used for exploration phases instead of the simple greedy algorithm used within this chapter. An idea of the possible gains using this strategy is given in chapter 8.

As a final note on Exploratory Digraph Navigation, we supposed in this chapter that there was no mapping error, which implies that $\mathcal{G} \subset \mathcal{G}^R$. If there are mapping errors, the Exploratory Digraph Navigation approach described throughout the chapter may or may

not work, depending on how severe and numerous the mapping errors are. Indeed, the $\mathcal{G} \subset \mathcal{G}^R$ hypothesis is required for theorem 1 and corollary 1 which guarantee respectively navigation success and exploration capacities. Since it is hard to model all possible mapping errors, we prefer to run actual experiments in chapter 6 with EDN, navigation (chapter 4) and SLAM (chapter 5) and check to which extent EDN is robust to mapping errors. Chapter 6 introduces metrics to evaluate topological correctness of $\mathcal{G}$ as well as navigation overheads due to incorrect maps.

## 3.6 Variants of EDNA*

The transformation from A* to EDNA* consists in adding a new heuristic $\mathcal{R}$ and computing the value of this new heuristic on certain vertices. This principle can be applied in various ways and to various algorithms. In this section, we propose two A* extensions using the new heuristic: EDN-(Lazy)theta* and greedy-A*. EDN-(Lazy)theta* is a modification of the (Lazy) Theta* (Nash et al., 2007; Nash, Koenig, and Tovey, 2010) "any-angle" path planner. Greedy-A* is a variant of A* with smart early stopping to limit the algorithmic complexity. It is notably used in chapter 8.

### 3.6.1 EDN-Theta* and EDN-Lazy Theta*

Theta* (Nash et al., 2007) and its lazy variant Lazy Theta* (Nash, Koenig, and Tovey, 2010) are A*-based algorithms which consider straight-line paths (and not only paths constrained on the graph) between two vertices of the graph used for planning (see Figure 3.12). (Lazy) Theta* uses line of sight checks between graph vertices to detect straight-line paths. The difference between Theta* and Lazy Theta* resides in how both algorithms perform these checks, with lazy theta* performing less checks for the exact same final result. Thorough explanations and examples as well as the pseudocode of both algorithms can easily be found on the internet.

The purpose of EDN-Theta* and EDN-Lazy Theta* is similar to that of EDNA*: discovering shortcuts through yet unexplored space. The EDN modification to the algorithms is implemented similarly for A*, Theta* and Lazy Theta*. That is: take the risk heuristic $\mathcal{R}$ as parameter, compute it on boundary vertices and return a path to a boundary vertex if a shortcut starting on this boundary vertex is interesting enough. For the sake of clarity, the pseudocode of these algorithms is deferred to appendix 2. We did not run extensive performance benchmarks of EDN-Lazy Theta* and EDN-Theta* relative to Lazy Theta* and Theta* but the performances of the EDN variants relative to their non-EDN base should be comparable to the relative performances of EDNA* and A* exposed in section 3.4.4.

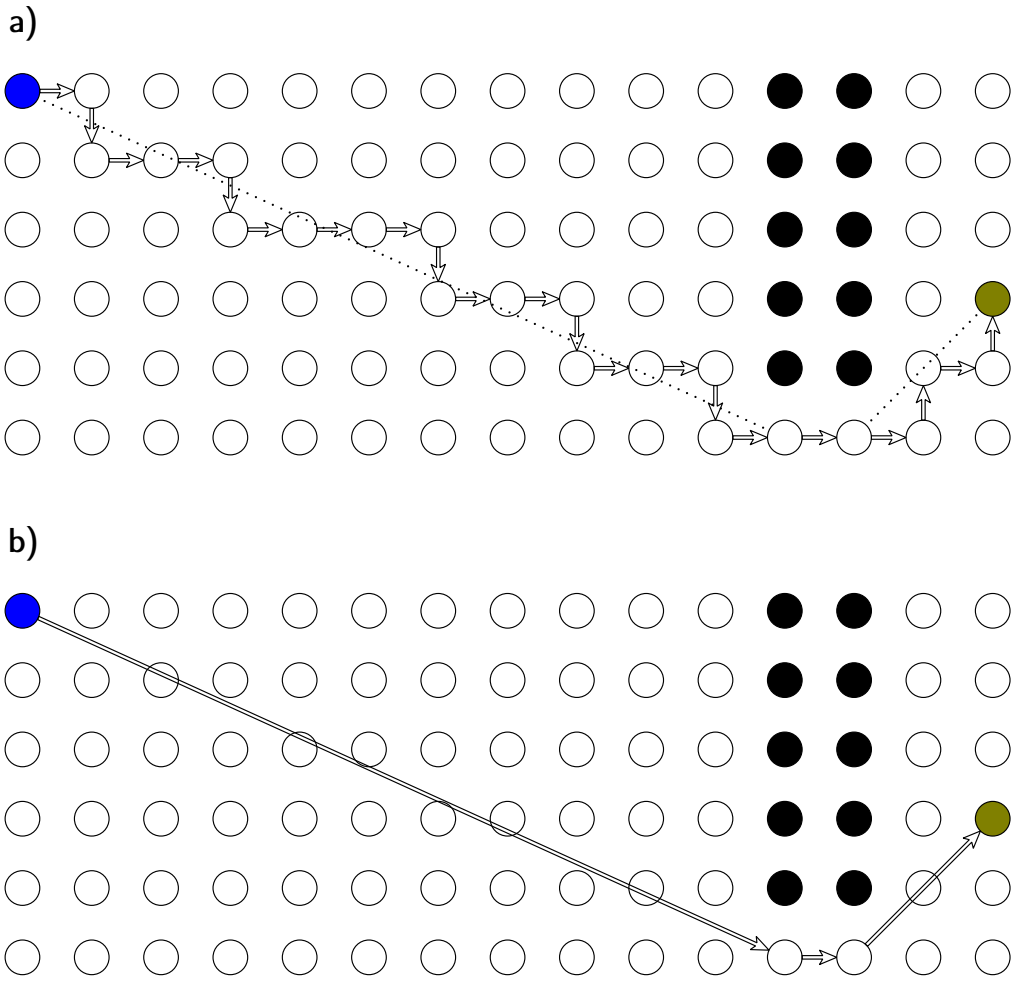



**Figure 3.12:** While A* (a) returns a "drunken man" path constrained to graph edges, (Lazy) Theta* (b) returns a path formed of straight lines traversing free space.

Following the three exploratory planners presented in this section (EDNA*, EDN-Theta* and EDNA-Lazy theta*), we believe that the majority of A*-based algorithms (and probably other algorithms) can be modified to take into account shortcuts. This shows that Exploratory Planning, Exploratory Digraph Navigation, PNSLAM and LEN are new paradigms and not just algorithms.

### 3.6.2 Greedy A*

We introduce a final EDN modification: Greedy A* (Algorithm 3). Greedy A* is not intended to discover shortcuts (it is not an exploratory planner) and should be seen as an implementation trick for use within chapter 8.

The principle of Greedy A* is to compute the risk heuristic $\mathcal{R}$ on all vertices expanded, not just on boundary vertices. The algorithm will return a path to $F$ or to a vertex $X \in \mathcal{V}$ considered to be on the way to $F$. By imposing a greedy criterion such as $||F-X|| < ||F-O_n||$, it is possible to guarantee that $F$ will eventually be reached. Like EDNA*, Greedy A* always expands less vertices than A* and may thus be preferred in situations with constrained memory.

With $\mathcal{R}(O_n, F, Z, \mathcal{G}) = \begin{vmatrix} ||F-Z|| < ||F-O_n|| & \mapsto D(O_n, Z) + K||F-Z|| \ (K \gg 1) \\ ||F-Z|| \geq ||F-O_n|| & \mapsto \infty \end{vmatrix}$,
a single run of gA* from $O_n$ will lead to $F$ if reachable, to $O_n$ if moving closer to $F$ is not possible or to $X$ so that $||F-X|| < ||F-O_n||$ otherwise. Unless $O_n$ corresponds to a local minimum of the distance to $F$, the distance to $F$ decreases monotonically with each run so that $F$ will eventually be reached.

## 3.7 Conclusion on exploratory planning and Exploratory Digraph Navigation

This chapter exposed a first perspective on PNSLAM/LEN from the point of view of planning, with SLAM and navigation as input and output (Figure 3.1). As a matter of fact, we developed EDNA* as an A* (Hart, Nilsson, and Raphael, 1968) hack while working on robot navigation because we could not find an algorithm able to plan a path on a partially unknown graph (an exploratory graph planner). Later, while working on the algorithm, we realized that the reason why such an algorithm did not already exist was probably that it required thinking about mapping, localization, navigation and planning as interdependent entities, at least within robotics. An exploratory planner relies on navigation and mapping to update the map while a non-exploratory planner only considers navigation.

**Algorithm 3** Greedy A*

**Input:** $O$, $F$, $\mathcal{G}$, $\mathcal{H} : X \to \mathcal{H}(X, F, \mathcal{G}), \mathcal{R} : Z \to \mathcal{R}(O, F, Z, \mathcal{G})$
dest $\leftarrow \varnothing$; best_distance $\leftarrow \infty$; $Q \leftarrow (O, \delta(O))$; $S \leftarrow \varnothing$
**while** $Q \neq \varnothing$ **do**
| pop $(X, \delta(X))$ from $Q$; $S \leftarrow (X, \delta(X))$
| **if** $\delta(X) >$ best_distance **then**
| | **break** //EDNA* early stopping criterion
| **if** $X = F$ **then**
| | dest $\leftarrow F$; best_distance $\leftarrow \delta(F)$
| | **break** //traditional A* stopping criterion
| **if** $\mathcal{R}(O, F, X) \leq$ best_distance **then**
| | dest $\leftarrow$ X; best_distance $\leftarrow \mathcal{R}(O, F, X)$
| **for all** neighbors $T$ of $X$ **do** compute $\delta(T)$
| | **if not** ( $(T \in S$ **and** $\delta(T) \geq \delta(T)_S)$ **or**
| | $(T \in Q$ **and** $\delta(T) \geq \delta(T)_Q)$ ) **then**
| | | $Q \leftarrow (T, \delta(T))$
**if** dest $= \varnothing$ **then return** failure //no path can be found
unroll a path from dest back to $O$ and return the path found

In the next chapter, we develop a second perspective on PNSLAM/LEN, this time from the point of view of *navigation*. A Navigation algorithm takes a path generated by EDNA* (or another exploratory planner), refines it for local navigation and obstacle avoidance and sends commands to the robot's actuators. Sensor data collected during physical movement is processed to extract the local topology of the environment and forwarded to a SLAM component which takes care of map update.

# 4 Navigation, obstacle avoidance and topology extraction

*"The greater the obstacle, the more glory in overcoming it."*
**Molière**

In this chapter, we develop the *navigation* perspective on PNSLAM (Figure 4.1). We use an occupancy grid (Elfes, 1987, 1989; Thrun, 2001) to perform SLAM at a local scale, as well as to provide a robust representation of local obstacles for navigation and obstacle avoidance. The occupancy grid allows local path planning of holonomic and non-holonomic robots. Moreover, since our SLAM component described in chapter 5 is based on topology, this chapter also describes how to perform topology extraction from low-level sensor data (Mayran de Chamisso, Soulier, and Aupetit, 2016). Navigation takes advantage of the extracted topology while obstacle avoidance uses a byproduct of topology extraction, a grid of vectors from each free pixel of the grid to the closest occupied pixel which we call the Vectorial Euclidean Distance Map.

## 4.1 Introducing the occupancy grid

### 4.1.1 Handling obstacles and goal-directed navigation

It is necessary for the robot to be aware of nearby obstacles, which implies that a local representation of free and occupied space must be kept in memory. Coherency of this local map is only required to be maintained up to some fixed distance of the robot (typically, a few meters) in order to allow maneuvering. At this local scale, odometric drift is not an issue (Thrun et al., 1998) and there are no topological loop closure issues (see chapter 5).

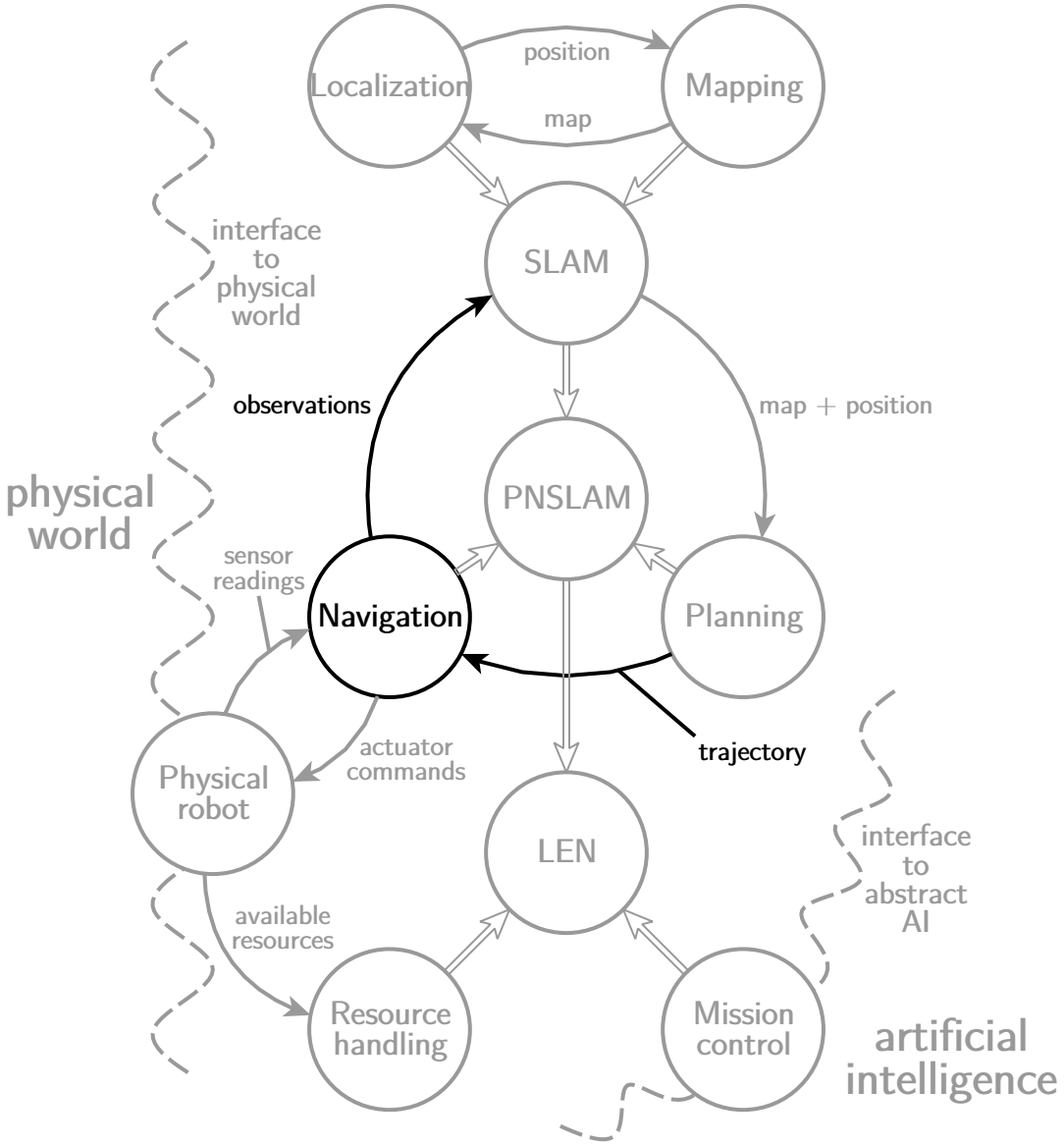



**Figure 4.1:** A navigation-centered view of PNSLAM and LEN.

Probably the most successful model to represent local obstacles and free space is the occupancy grid (Elfes, 1987, 1989; Moravec and Elfes, 1985; Thrun, 2001). An occupancy grid can be acquired using distance and movement sensors or provided to the robot as a floorplan. In the grid, each pixel encodes the probability $p_{occ}$ that the corresponding position in space is occupied as a log-odd $l = log(\frac{p_{occ}}{1-p_{occ}})$. Values above zero describe occupied space and values below zero describe free space (see Figure 4.2).

Occupancy grids can integrate data from various types of distance sensors including lasers, sonars, infrared, stereo cameras and Microsoft Kinect. Dense as well as sparse distance data can be integrated. Occupancy grids are robust to dynamic sensor and environment noise thanks to their probabilistic nature and to time integration. They are also easy to update with incoming sensor observations. An occupancy grid can be used to implement obstacle avoidance using repulsive forces (Borenstein and Koren, 1989) or potential fields (Barraquand, Langlois, and Latombe, 1992). For instance, repulsive forces are computed by finding for each free pixel the closest occupied pixel, resulting in a *Vectorial Euclidean Distance Map*. The vector to the closest occupied pixel can then be used to compute the virtual force that characterizes the free pixel. Finally, and as we will see, occupancy grids are convenient representations from which the local topology of the environment can be extracted.

In the context of this thesis, we chose to implement a *scrolling* occupancy grid of fixed size around the robot (Kuipers et al., 2004). The robot is always at coordinates $(0, 0)$ in this grid (the origin of the grid moves with the robot). The scrolling occupancy grid is used as Local Perceptual Model as in the Hybrid Spatial Semantic Hierarchy (Beeson, Modayil, and Kuipers, 2010).

When the robot moves further than a certain distance $d$ from its previous position $P_{n-1}$, an aging process is applied to the occupancy grid. This aging process is necessary since when the grid scrolls, pixels on the far left may appear on the far right (for instance) due to wrap-around addressing. The algorithm used for occupancy grid aging is described in appendix 2.

## 4.2 Topology extraction using the Vectorial Euclidean Distance Map

### 4.2.1 State of the art and motivations

Recent SLAM algorithms including ours (chapter 5) use topological or hybrid metrical-topological mapping (Bailey, 2002; Beeson, Modayil, and Kuipers, 2010; Bosse et al., 2004; Kuipers et al., 2004; Nitsche et al., 2011; Thrun, 1998) where the map is a set of

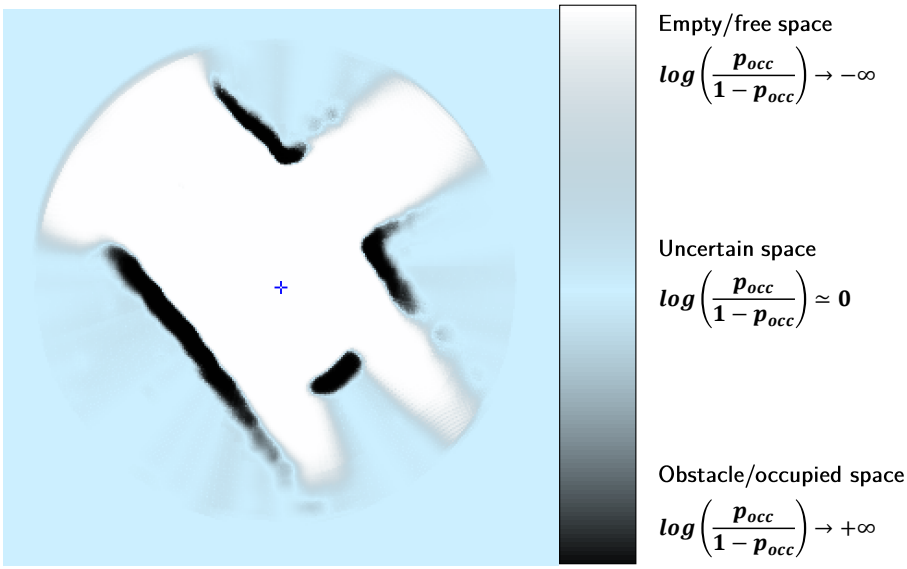



**Figure 4.2:** An occupancy grid. Free space in white, occupied space in black, uncertain zones in shades of cyan.

vertices describing discrete places and a set of edges describing transformations between vertices. While being more robust to large-scale uncertainty and odometric drift than dense representations, topological representations require vertices to be robustly defined from sensor data in the sense that multiple observations of a place will lead to the same vertices. The reliability of vertex detection is usually not considered (Bosse et al., 2004) or left aside as a secondary issue (Kuipers et al., 2004). Reliability is however discussed in depth in (Beeson, Jong, and Kuipers, 2005).

The robot wants to extract the topology of the environment around its own position. The robot is only interested in paths it can use to navigate, that is paths that:

- describe passages large enough for the robot to traverse,
- are visible and/or accessible from the robot's current position and
- are not too far away from the robot's current position, with the idea that the probabilistic data of the grid gets less reliable when the distance to the origin increases. If the robot moves, it can update its occupancy grid and extract the local topology again around its new position.

We propose a lightweight topology extraction method "from pixels to graph" using a topological skeleton computed on a local occupancy grid through the *Vectorial Euclidean*

*distance map* (VEDM) (Danielsson, 1980). The VEDM contains for each empty pixel of the grid a vector pointing to the closest occupied pixel. We use the VEDM for two main reasons:

- it can be computed using *integer arithmetics*, resulting in faster computations especially on processors without floating point units (typically, low-power processors used in embedded systems), and
- it can be reused to perform obstacle avoidance using virtual repulsive forces (Borenstein and Koren, 1989).

In addition to the topology extraction method, we propose criteria to simplify the skeleton by removing spurious edges which the robot would not be able to *physically traverse*. Compared to existing approaches such as (Choset and Nagatani, 2001) or (Beeson, Jong, and Kuipers, 2005), this removal does not make assumptions on yet unknown space and is only parametrized by the robot's size, making it both more intuitive from a robotics point of view and more *robust to occlusions* and noise. Finally, the approach achieves *close to linear time complexity* in the number of pixels in the grid on real-world datasets, exceeding 100 frames per second on square grids of size $300 \times 300$ pixels.

### 4.2.2 State of the art of topology extraction

The topological skeleton, sometimes called Generalized Voronoï Diagram (GVD) or Generalized Voronoï Graph (GVG) (Attali, 1995; Choset and Nagatani, 2001) is a convenient topological representation of an environment which has already been used for SLAM (Choset and Nagatani, 2001; Thrun, 1998). Edges of the skeleton are the set of points equidistant from two or more obstacles while vertices are the set of points where edges meet or end. Computing the skeleton from a discrete representation such as an occupancy grid is traditionally done through homotopic thinning, wavefront propagation or distance map calculation. The vast state of the art of these approaches is reviewed in (Attali, 1995). Briefly, (distance ordered) homotopic thinning, though easily parallelizable, leads to hard-to-exploit skeletons. Moreover, the skeleton simplification techniques such as only connecting the center of maximal balls (Pudney, 1998) lead to skeletons unstable with grid updates since some centers may appear or disappear. Wavefront propagation techniques need complex models to describe the wavefront. Finally, distance maps (Arcelli and Sanniti di Baja, 1989; Danielsson, 1980; Man et al., 2010) need to be computed as a first step and exploiting them requires reconstruction of ridge lines which is a process sensitive to noise and may lead to 'holes' in the skeleton, an issue addressed by Li and Vossepoel (Li and Vossepoel, 1998). A comprehensive review of existing distance transform algorithms can be found in (Grevera, 2007) and (Fabbri et al., 2008). Alternatively to a distance transform, an Euclidean feature transform (Hesselink, 2007) can be used to diagnose the ridges directly. This does not however address the

issue of noise.

One of the consequences of noise in the grid is the presence of spurious edges on the skeleton. Simplification of the skeleton is a known issue discussed notably in (Attali, 1995; Choset and Nagatani, 2001; Pudney, 1998) and (Garrido et al., 2006). Examples without simplification of the skeleton can be found in (Li and Vossepoel, 1998). They show that a simplification is indeed necessary for robot navigation. In order to remove spurious edges, Choset et al. (Choset and Nagatani, 2001) proposed to discard edges terminating near a wall (dead ends). While perfectly valid in static situations, this approach becomes unstable when the grid gets updated: sometimes, the bottom of a dead end cannot be seen from the current origin of the grid. If the robot moves back and forth, the edge may keep appearing and disappearing, causing navigation algorithms to get stuck in an infinite loop. The same issue affects the "gateway" approach of Beeson et al. (Beeson, Jong, and Kuipers, 2005) which distinguishes edges terminating in free space as possible exits and only consider shortest paths leading to the exits as valid. Indeed, the "exits" may actually be unseen dead ends. Attali (Attali, 1995), Malandain et al. (Malandain and Fernández-Vidal, 1998) and Couprie et al. (Couprie, Coeurjolly, and Zrour, 2007) proposed criteria based on bissector angles (angles between the vectors pointing to the two obstacles equidistant from an edge) to remove spurious edges. While these criteria do not make assumptions on dead ends, they require trigonometric computations which are expensive and require floating-point representations. Note that bissector angles can easily be recovered from the VEDM.

### 4.2.3 Our approach

#### Notations

Suppose that the dimensions of the grid are $(w, h)$. We write $L_1$ ("Manhattan") distances using simple bars $|.|$ and Euclidean distances using double bars $||.||$. We consider that the skeleton has to be computed only within range $r \leq min(w, h)$ of the origin (Euclidean distance used for range), represented by a dark red dashed circle on Figure 4.3. The skeleton is computed in the Euclidean space. We use the term "pixel cluster" as a set of pixels where each pixel is connected to the cluster horizontally, vertically or diagonally. Finally, we suppose that the center of the robot is not allowed to get closer to an obstacle than some fixed distance $D + S$, where $S$ is the physical radius of the robot around its center of inertia and $D$ is an additional safety zone, which we call the "minimum allowed distance to wall".

### Overview of our approach

The following steps are required in our approach to compute the GVG or Extended GVG (Beeson, Jong, and Kuipers, 2005) from an occupancy grid around a given origin and extract it in a graph format:

1. (optional) Remove hot pixels and noise. Hot pixel removal can be achieved with a simple $4 \times 4$ or $9 \times 9$ median filter or through more elaborate mathematical morphology operators. Noise removal can be obtained by convolution with a Gaussian Kernel.

2. Binarize every pixel of the grid to an 'occupied/empty' state. The simplest approach (used in this paper) is to have a fixed log-odd threshold, at 0 for instance. Thresholding can be done on the fly and does not require a buffer to store the binary version.

3. From the origin, radially propagate a wavefront (Figure 4.3). Propagation stops on a specific pixel if it is occupied or if the pixel is too far from the origin (or outside the grid). For the most part, propagation only reaches line of sight pixels. Step 3 is an optimization specific to robot navigation and could be replaced by step 3' consisting in marking all occupied pixels directly in one pass over the image (the benefit of step 3 over step 3' is visible on Figure 4.10).

4. Using the occupied pixels detected at step 3/3', compute a VEDM (see Figure 4.6), that is associate each pixel to a vector pointing to the closest occupied pixel.

5. (Figure 4.6) For each pixel with a vector computed at step 4, test whether or not it belongs to an acceptable edge (where 'acceptable' will be properly defined later).

6. (optional) Remove 'edge' pixels that are not linked to the main skeleton, which is defined either as the component of the skeleton passing closest to the origin or as the component of the skeleton counting the most pixels.

7. For each remaining 'edge' pixel, detect whether or not it is a vertex, where a vertex is defined as the intersection of two or more edges or as the end point of an edge (Figure 4.7).

8. Expand each 'vertex' pixel by some predefined amount. If the expansion zones of two vertices meet, both vertices are clustered into a single one.

9. Cluster 'edge' pixels that are not 'vertex' pixels into edge segments. Each segment necessarily has exactly two end vertices.

10. (optional) Create a graph representation of the environment with detected edges and vertices.

We tried different orders for steps 5 to 10, such as detecting vertices first (step 7), then edges linking them (steps 5 and 6). We also tried to extract skeletons without vertex clustering (step 8). However, the above order of steps was the one for which the graph representation created at step 10 best matched the topology of the environment. Up to step 5 included, our approach can be implemented storing only three integer values per pixel in the grid:

1. a state (never reached, reached at step 3, expanded at step 3, reached at step 4, belonging to an edge), which can be reused during steps 5 to 10 to store a vertex or vertex cluster identifier on the pixels classified as "vertex"

2. the $x$ and $y$ components of the vector to the closest occupied pixel (Vectorial Euclidean Distance Map)

### Computing a vector map (steps 3 and 4)

Initially, the vector field is set to $\vec{V_{xy}} = (\infty, \infty)$ or to values bigger than $\vec{V_{xy}} = (w, h)$ for every pixel $(x, y)$ in the grid. During step 3, pixels are considered in an order defined by a growing circle using the $L_1$ distance, as sketched on figure 4.3 (a). Each time an empty pixel is considered, its empty neighbors are marked to be considered during the next pass if they have not been considered already and if they are not further than $r$ from the origin (Euclidean distance). This expansion phase stops if no vertex was marked during a pass. Each time an occupied vertex is marked, its associated vector in the vector field is set to $(0; 0)$. This way of proceeding tends to confine the search for the skeleton to the zone actually visible by the robot, or in other words, the *interior* of the environment, shown in green on figure 4.3. It avoids computing the part of the skeleton that lies in zones not accessible from the origin of the grid without traversing occupied space. The added computational burden of step 3 is balanced by the simplification of steps 4 to 10. The interest of step 3 is visible on Figures 4.3 and 4.6: On Figure 4.3, there are much less pixels reached by step 3 than by state 3'. On Figure 4.6, the vector field has mostly been calculated *inside* the building, even though cracks (places where the log-odd approaches 0) in the wall on the bottom-left corner led to some calculations being carried outside the building. Figure 4.10 shows the actual decrease in computation time due to step 3.

We suppose that at least one occupied pixel was reached during step 3 (otherwise, the skeleton is empty). In environments with large patches of empty spaces, the Extended GVG (discussed in the paragraph describing step 5) can be used to prevent the skeleton from being empty.

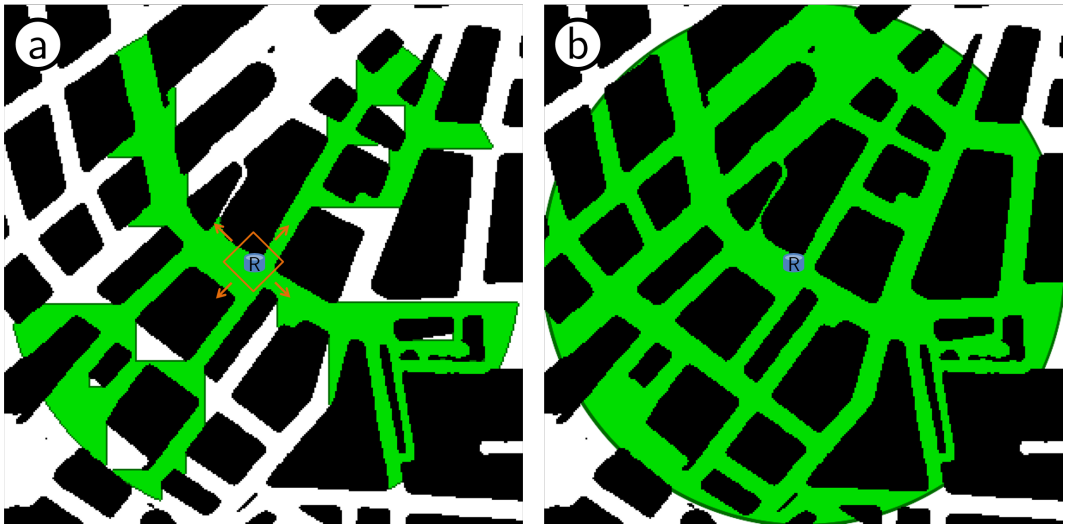



**Figure 4.3:** Step 3 (a) versus step 3' (b), reached pixels in green. Synthetic large occupancy grid used to illustrate the difference between steps 3 and 3'. In practice, the scrolling occupancy grids used are smaller. Step 3 tries to reach only pixels in line of sight of the robot, up to a certain range equal here to half the size of the map. However, due to the $L^1$ distance used for propagation (orange unit ball), step 3 also reaches pixels not in line of sight of the robot. Step 3' marks all pixels up to a certain range of the robot, equal here to half the size of the map. Compared to step 3, step 3' marks more pixels to be processed during later steps. Thus, the computational burden of steps 4+ is increased when using step 3' instead of step 3. This is not completely compensated by step 3' being faster than step 3.

Step 4 computes a field of vectors to the closest occupied pixel or *Vectorial Euclidean Distance Map (VEDM)*, which is very like distance map calculation but vectors contain both distance and direction. In a (vectorial) Euclidean Distance Map, the environment is partitioned into *Voronoï cells*, where each cell is made of one occupied pixel, the *seed*, and all empty pixels closer to the seed than to any other seed. Linear complexity algorithms exist to compute Euclidean distance maps, such as that of (Breu et al., 1995) based on incremental construction of the Voronoï cells or that of (Hirata, 1996) based on building an intermediate map of closest obstacles along the vertical axis. A review of modern parallel approaches to Euclidean distance map calculation is given in (Man et al., 2010). The difficulty of constructing a (Vectorial) Euclidean Distance Map comes from the fact that the boundary of Voronoï cells is not restricted to the boundary of pixels and the local width of a cell may be arbitrarily small (see Figure 4.4).

Instead of an exact algorithm, we chose to use the approximate algorithm of Danielsson (1980) to compute the VEDM. This algorithm propagates vectors from pixels to their neighbors, where occupied pixels start with a vector of $(0, 0)$. With an eight-neighbor propagation scheme, the algorithm was demonstrated in (Danielsson, 1980) to be completely accurate for a pixel when there exists a path from the closest occupied pixel to it within the Voronoï cell. Otherwise, the maximum error of the algorithm is 0.076 pixels (the pixel is attached to a Voronoï seed 0.076 pixels further than the true seed of its actual cell). There are extremely few cases where the algorithm is not exact, and when an error occurs, its impact is negligible compared to that of sensor noise (even a one-pixel error would probably not cause any visible change in the skeleton). One failure situation is represented on figure 4.4. This algorithm does not require dynamical allocation or sorting and does not use a stack, as opposed to Breu et al. (1995), Hirata et al. (1996) and later approaches. Moreover, the algorithm of Danielsson always returns single-cluster Voronoï cells, which alleviates the issue of isolated 'hot' pixels. Parallelization of the computation is also possible, even on embedded platforms such as ASICs or FPGAs.

Both steps 3 and 4 are efficient computationally wise since they can be coded with integer arithmetics, without dynamical allocation, divisions, square roots and trigonometry).

At the end of step 4, the VEDM is obtained as a vector field $\vec{V}$ such as that of figure 4.6.

### Detecting and pruning edges (step 5)

During step 5, each pixel of the image expanded at step 4 is tested to see if it belongs to an edge of the skeleton.

Let $J(\vec{V_{xy}}) = (\vec{V_{(x+1)y}} - \vec{V_{xy}}, \vec{V_{x(y+1)}} - \vec{V_{xy}})$ be the approximate Jacobian of the vector field

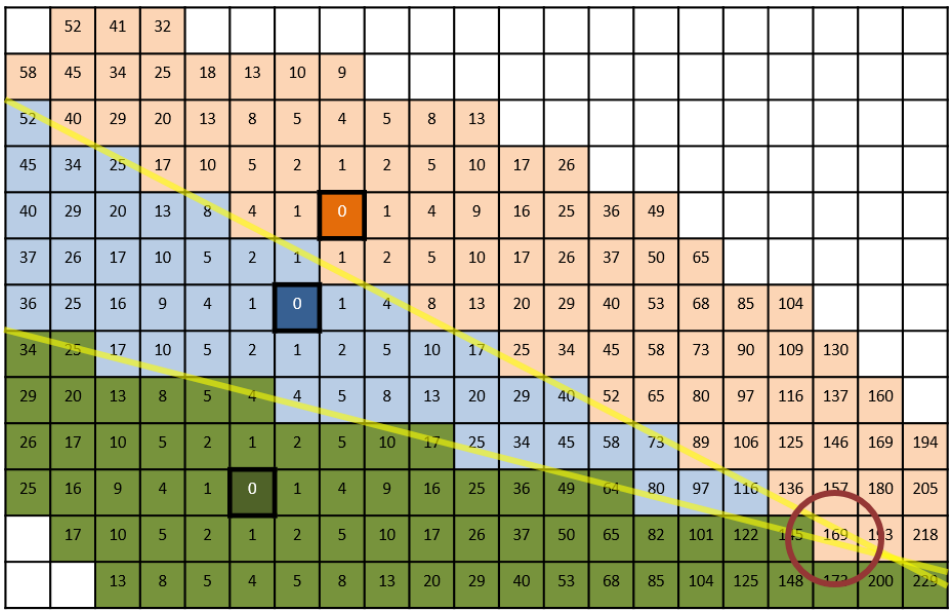



**Figure 4.4:** Local inaccuracy at *step 4*: the pixel circled in red is erroneously labeled as belonging to the orange cell (square distance 170) instead of the blue one (square distance 169). Numbers represent square Euclidean distances in pixels, colors represent Voronoï cells and yellow lines are the boundary of the cells. The seed of each cell (occupied pixel) has a distance of 0 to the closest occupied pixel (itself).

at point $(x, y)$. We propose an edge detection criterion based on the maximum norm of the Jacobian along the $x$ and $y$ directions, $M = max(||\vec{V_{(x+1)y}} - \vec{V_{xy}}||, ||\vec{V_{x(y+1)}} - \vec{V_{xy}}||$. If $M < 2(S+D)$, then the walls the edge is supposed to go between are too close for the robot to pass, so the edge is discarded. Zhang et al. (Zhang et al., 2014) seem to use this same criterion of minimal distance between the two obstacles. If $M \geq 2(S+D)$, the edge is kept and a local tangent vector to it is computed as $\frac{\vec{V_{(x+1)y}} + \vec{V_{xy}}}{||\vec{V_{(x+1)y}} + \vec{V_{xy}}||}$ if $||\vec{V_{(x+1)y}} - \vec{V_{xy}}|| >= ||\vec{V_{x(y+1)}} - \vec{V_{xy}}||$ or $\frac{\vec{V_{x(y+1)}} + \vec{V_{xy}}}{||\vec{V_{x(y+1)}} + \vec{V_{xy}}||}$ if $||\vec{V_{(x+1)y}} - \vec{V_{xy}}|| < ||\vec{V_{x(y+1)}} - \vec{V_{xy}}||$. The tangent vector is perpendicular to the difference of adjacent vectors, in the direction where the norm of the Jacobian is maximal. Note that due to the definition of the Jacobian, the skeleton is computed with a constant offset of $(0.5, 0.5)$.

We found a further refinement for mobile robot navigation: check that the pixel pointed to by the average of adjacent vectors in the $x$ or $y$ direction (subsequently called average pixel) is far enough from an obstacle, where far enough means the robot should be able to navigate up to this point without hitting an obstacle. This refinement can be understood as virtually projecting the robot forward on the hypothetic edge, supposing that the edge remains straight (first-order approximation). Formally, suppose that $||\vec{V_{xy}} - \vec{V_{(x+1)y}}|| \geq ||\vec{V_{xy}} - \vec{V_{x(y+1)}}||$ (the other case is handled by permuting $x$ and $y$). Then, let $(x', y') = \frac{(x+(x+1),y+y)+\vec{V_{xy}}+\vec{V_{(x+1)y}}}{2} = (x + 0.5, y) + \frac{\vec{V_{xy}}+\vec{V_{(x+1)y}}}{2}$. If $M' = ||\vec{V_{x'y'}}|| < S + D$, then it is likely that there is no actual path between the walls and that the presumed edge is a shallow dead end. The method is not exact since it considers that the edge remains straight when going towards an obstacle (the average of both vectors is tangent to the edge at the current pixel, so the approximation is first-order). However, experiments in real conditions proved it to be quite reliable even though we could not find a metric in literature to describe the quality of the extracted skeleton. Figure 4.6 shows the two criteria for edge validation $M \geq 2(S+D)$ and $M' \geq S+D$. $M' \geq S+D \Rightarrow M \geq 2(S+D)$, so that the first criterion is only used in order to avoid the small overhead due to computation of the second criterion.

It is easy to implement the Extended GVG (Beeson, Jong, and Kuipers, 2005) with a (vectorial) Euclidean distance map. The difference between simple and extended GVGs is that in the latter, no edge is allowed to lie more than a distance $L$ away from the closest obstacle and there must be edges at distance $L$ from any obstacle. An example of GVG and EGVG is given on Figure 4.5. With our approach, the Extended GVG is computed the following way: for each pixel at coordinates $(x, y)$,

- If $||V_{xy}|| > L$, then the pixel at $(x, y)$ cannot belong to an Extended GVG edge.
- If $||V_{xy}|| \leq L$ and if one of the eight neighbors $(x', y')$ of $(x, y)$ defined by $(x' - x)^2 + (y' - y)^2 \in \{1, 2\}$ verifies $||V_{x'y'}|| > L$, then the pixel at $(x, y)$ necessarily belongs to an Extended GVG edge.

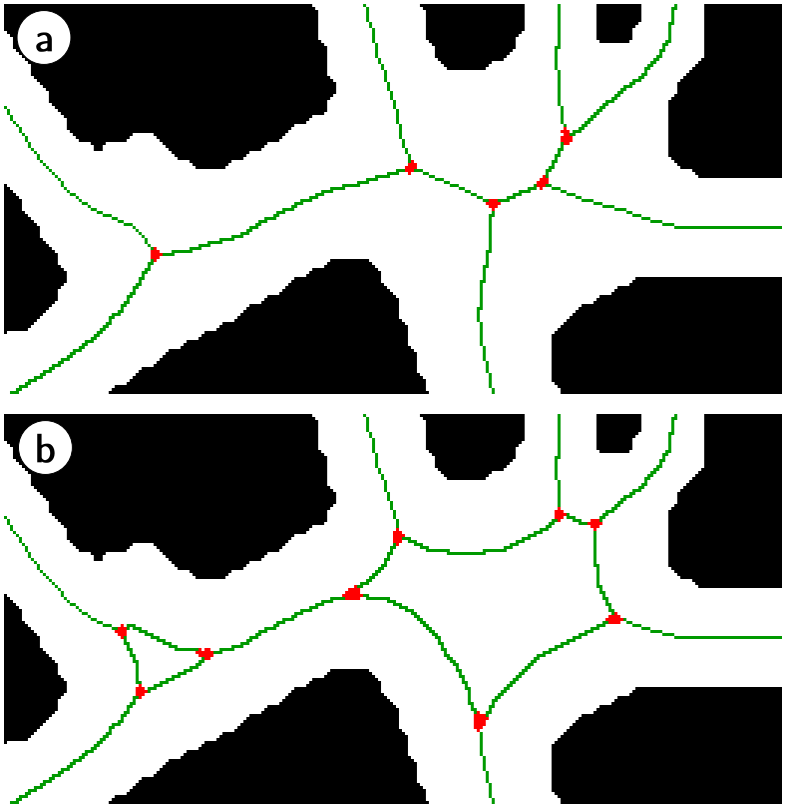



**Figure 4.5:** (a) GVG and (b) Extended GVG ($L = 25$ pixels) computed for the same environment. In the Extended GVG, no edge lies further than $L$ away from an occupied pixel, which is useful in environments containing patches of empty space whose size exceeds the range of the robot's sensors. The GVG can be seen as an Extended GVG with $L \to \infty$.

- Otherwise, the two edge validation criteria $M \geq 2(S + D)$ and $M' \geq S + D$ are used like for the regular GVG computation.

The Extended GVG is particularly useful in environments whose dimensions are much larger than the maximum range of the robot's distance sensors, such as outdoor environments. Within our implementation, the transition from Extended GVG to GVG is equivalent to changing $L$ from a finite value to positive infinity.

### Detecting vertices (steps 7 and 8)

Once edges have been detected using the above method, vertices are found as edge intersections and loose ends of edges with a single pass over the image.

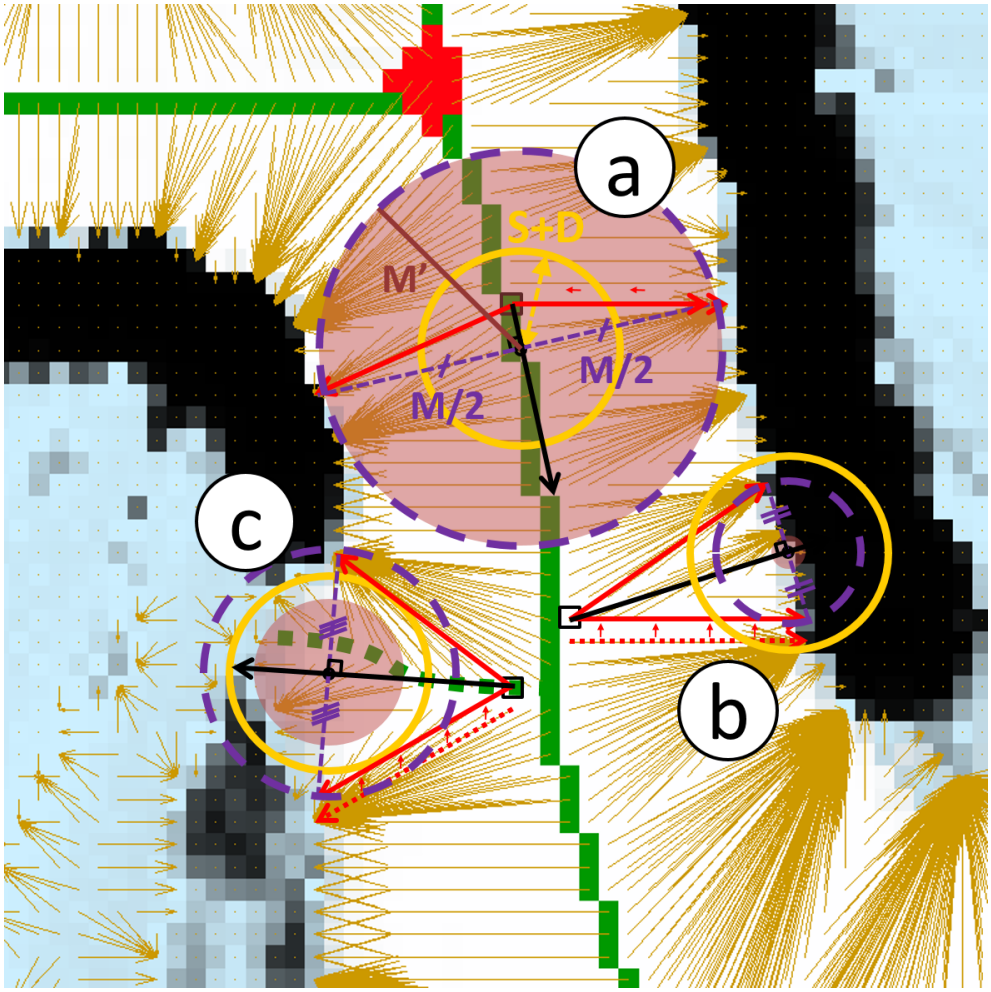



**Figure 4.6:** Removal of parasitic edges (*step 5*): each pixel (bold black square) is tested to see if it belongs to an edge or not. $S + D$ is represented with a yellow circle. In configuration (a), $M/2$ (dashed purple circle) and $M'$ (red disc) are high enough for the pixel to belong to an edge. In configuration (b), $M/2 < S + D$, so the pixel does not belong to an edge. In configuration (c), $M/2 \geq S + D$ but $M' < S + D$, invalidating the edge hypothesis in dotted green. Black arrows are the local edge tangents, computed as average of both vectors and scaled arbitrarily. Red arrows are the vectors of the Vectorial Euclidean Distance Map (in each configuration, one of both vectors was offset one pixel left or up from its original position marked with a dotted arrow. This offset is due to the discrete gradient implementation).

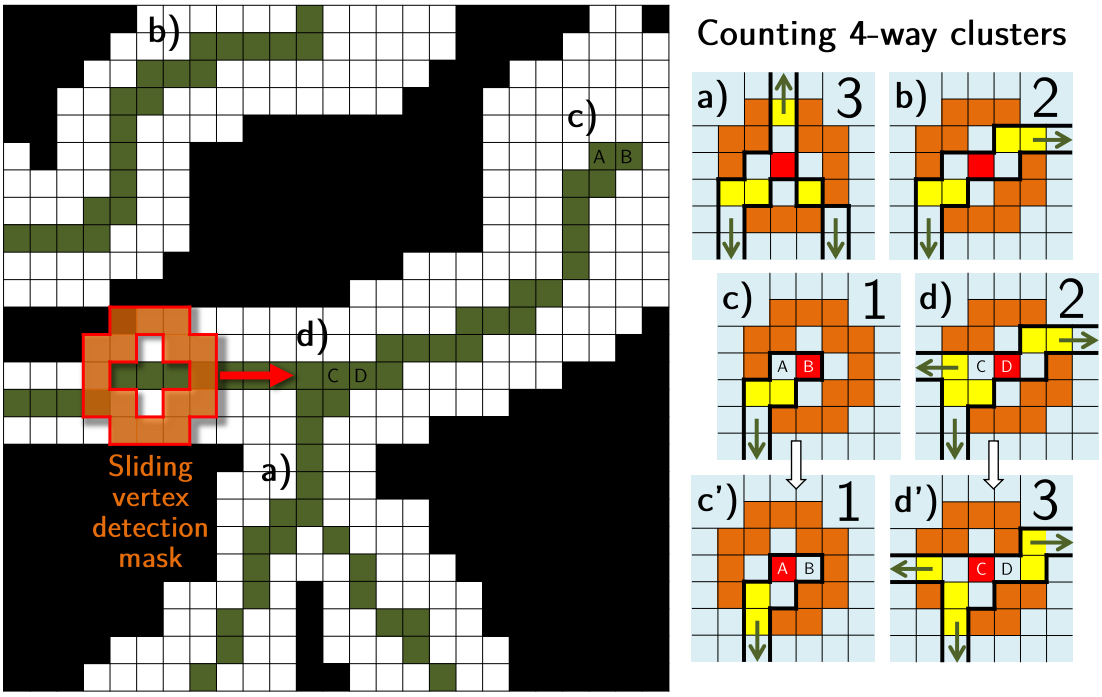



**Figure 4.7:** Vertex detection process (*step 7*): an edge pixel is a vertex if and only if one or strictly more than two edges cross the orange contour. Edges crossing the contour are counted as the number of yellow clusters, considering only 4-way connection. Errors may occur on one pixel but are likely not to occur on neighboring pixels. For instance, in case (d), 2 edges are detected on pixel $D$ while on neighboring pixel $C$, there are actually 3 edges (case (d')). This is not an issue since it is sufficient that one pixel be marked as vertex. Multiple neighboring pixels can be marked as vertex (pixels $A$ and $B$ for cases (c) and (c')), which calls for a vertex clustering process.

For each edge pixel, a test is performed to determine if it is a vertex. This test sketched on Figure 4.7 consists in checking how many edges cross a closed contour around the pixel. A pixel marked as edge is a vertex if and only if the contour crosses exactly one or strictly more than two edges. As opposed to the method proposed in (Malandain and Fernández-Vidal, 1998) or to simple point characterization (Klette, 2003), contours are not restricted to 4- or 8-neighborhoods. With a 4- or 8-neighbor scheme, no vertex would have been detected in case (d') of Figure 4.7.

This contour-based vertex detection does only work well if edges are locally one or two pixels wide, which represents the immense majority of cases. In order to handle edges locally wider than two pixels (which may happen when multiple individual edges get too close to each other and form one single apparent edge), if a pixel has more than 7 out of 8 neighbors belonging to edges, it is automatically labeled as vertex, as well as said 7 or 8

neighbors. This refinement is almost never necessary. Both the contour-based approach and the 7-8-neighbors case are heuristic choices which were observed to give visually satisfying results in a very large fraction of the thousands of skeletons extracted while working on this thesis. For reference, a 270m run of the robot with our experimental setup (see chapter 6) leads to extraction of more than 10 000 skeletons!

After the initial detection, vertices are clustered respecting the two following rules:

1. Two different clusters must be disconnected regarding 8-way connection.

2. An edge must always link exactly two vertex clusters. This means that two edges cannot "touch" each other at any place but a vertex (using 8-way connectivity).

We found that both rules were enforced by attaching vertices within 3 pixels (inclusive, using the $L_1$ distance) of each other, or within four pixels if not along the same line or column. The value of 3 was found by trial and error and validated a posteriori given that the immense majority of edges are one or two pixels wide according to the precision of vector calculation and edge detection. As a consequence, each edge intersection will cause at least one pixel to be classified as vertex. If edges are locally wider than two pixels, the refinement for pixels having 7 neighbors belonging to edges is likely to ensure that the second rule does not get violated. Once clustering is done, the center of each cluster is taken as the center of gravity of all vertex pixels belonging to the cluster.

Finally, each one of the original vertex pixels marks a zone of radius 1 pixel ($L_1$ distance) around itself as belonging to the cluster it belongs to (because anything that would be within this range would belong to the cluster). 1 is strictly lower than the integral part of 3/2 (3 being the vertex clustering radius, which can also be seen as the minimum inter-vertex distance), so that this step will not fuse clusters together. The center of the cluster is also marked as vertex, as well as the zone of 1 pixel around it, because it is conceptually impossible that the center of the cluster does not belong to the cluster. The output of the clustering step is shown on Figure 4.8.

Higher clustering radiuses simplify the graph structure (less vertices with more edges per vertex in average).

Some skeletons obtained with steps 1 to 8 are presented on Figure 4.9.

### Graph extraction (steps 9 and 10)

If the second rule of vertex clustering is verified, then edges always stop at a vertex cluster. Thus, each edge segment links exactly two vertex clusters. Edge segments,

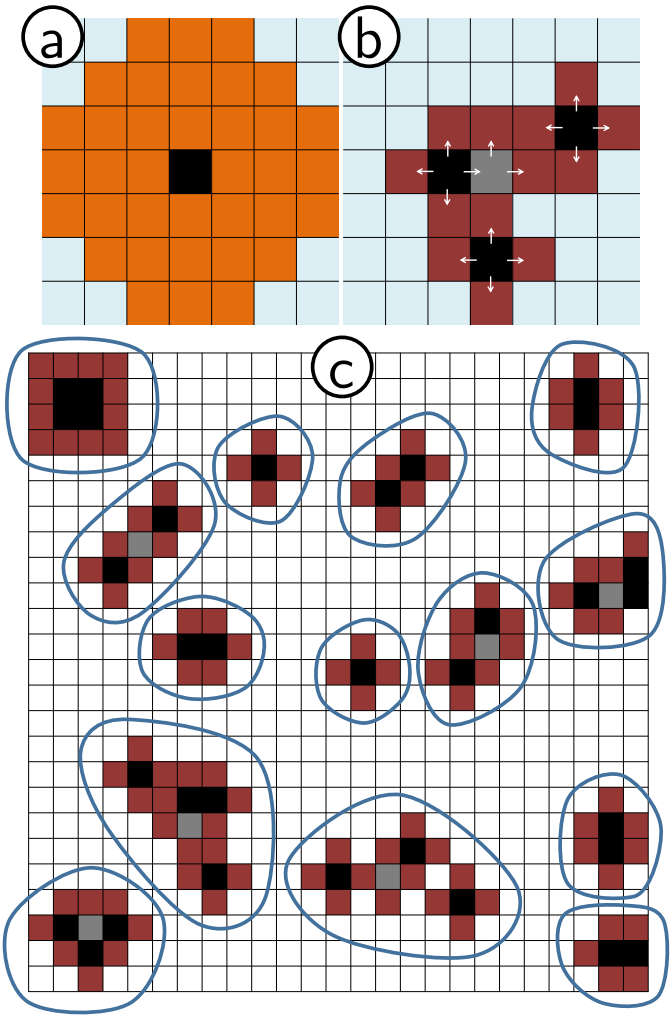



**Figure 4.8:** (a) Vertex clustering radius. (b) Final vertex (red) obtained after clustering of three vertex pixels (black). Center of the cluster in gray. The small arrows show how vertices detected at step 7 (black) as well as the cluster center gray are expanded, marking neighboring pixels as belonging to the cluster (red). The final vertex cluster (blue outline) is made of the black, gray and red pixels. (c) Vertex clustering applied on a pseudorandom vertex pixel distribution.

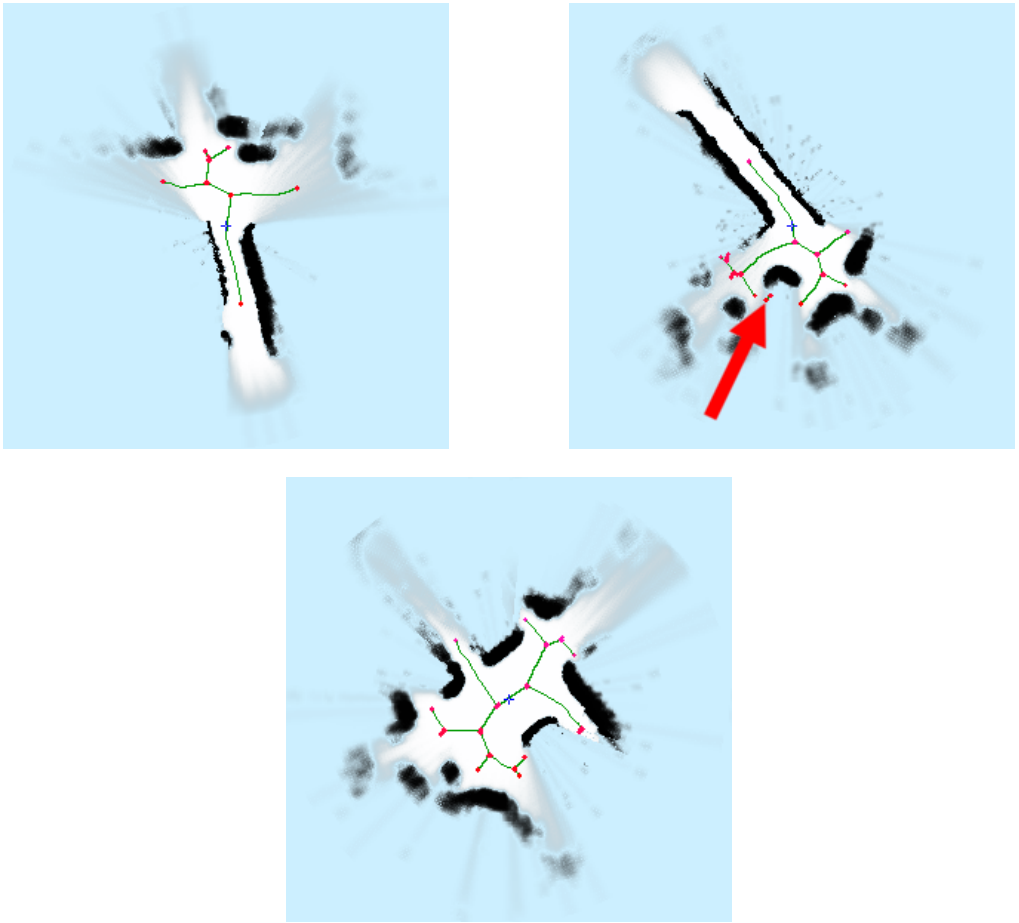


**Figure 4.9:** Some outputs of the algorithm. Edges in green and vertices in red. Removal of edges not belonging to the main cluster (step 6) was not applied, as can be seen on the middle figure where there are two disconnected graph components (red arrow). Vertices are automatically placed at the end of raw edges even though these may not lead to dead ends.

characterized by their two end vertices, are found using a simple agglomerative clustering algorithm:

For each edge pixel $P$:

- Enumerate all eight neighbors $Q$ of $P$ one after the other:
  - If $P$ does not belong to an edge cluster and $Q$ does, attach $P$ to the cluster of $Q$.
  - If $P$ and $Q$ belong to the same cluster, do nothing.
  - If $P$ and $Q$ belong to different edge clusters, fuse both clusters.
- If none of the eight neighbors of $P$ already belonged to an edge cluster, create a new edge cluster with two empty vertex slots.
- Enumerate all eight neighbors $Q$ of $P$ one after the other a second time. If $Q$ is a vertex with identifier $i$:
  - If one of the end vertices of $P$'s edge cluster already has the identifier $i$, do nothing.
  - Otherwise, $i$ is one of the end vertices of $P$'s edge cluster.

Then, edges with less than one extremal vertex get pruned, which notably eliminates the rare case where an edge links a vertex to itself. Vertices with exactly two edges are possible due to vertex clustering, as shown on figure 4.12. Path segments made of chains of such vertices can be simplified to a single edge between both extremal vertices. Skeletons shown on all figures of this chapter were all extracted without topological errors in a graph format such as that shown on Figure 4.12.

#### Implementation

The above series of algorithms was implemented in C using only integer arithmetics. Dynamical allocation was only used for clustering and graph extraction. The actual execution time of the whole computation is on the order of 6 milliseconds for a grid of $300 \times 300$ pixels. Without parallelization or further optimizations, extrapolation (from Figure 4.10) leads to typical execution times of a few seconds for a grid of 10 000 pixels. This is the order of magnitude obtained for CPUs by Man et al. (Man et al., 2010) in their approach to Euclidean distance mapping using CPUs and GPUs. However, comparing both approaches is biased since on the one hand, Man et al. recover an exact

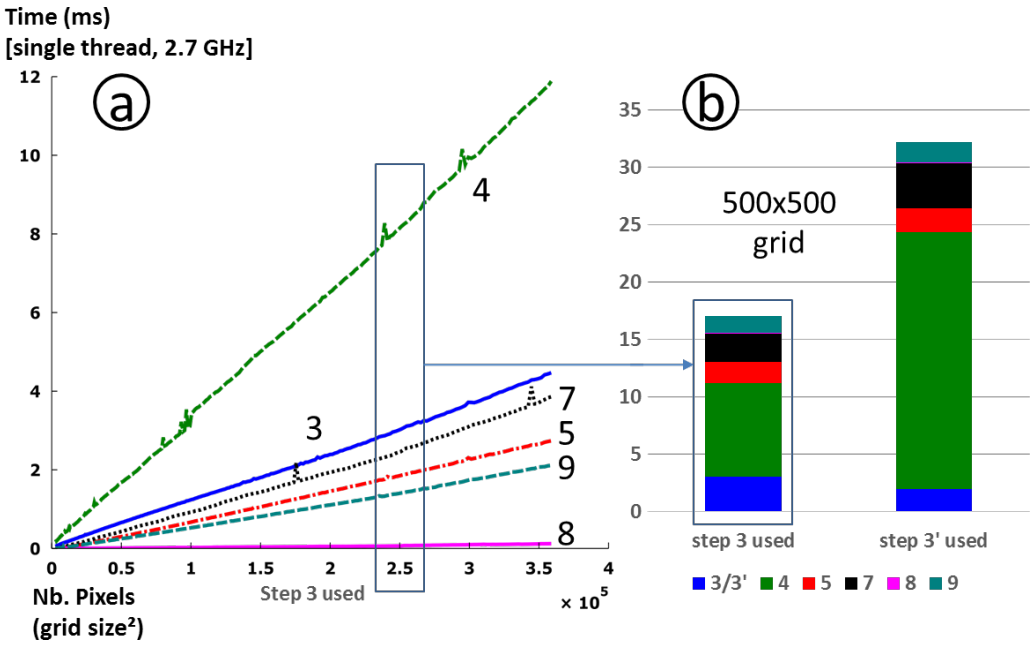



**Figure 4.10:** (a) Execution time (ms) of the steps of our method as a function of the size of the grid (in square pixels) on a 2.7 GHz core. (b) Step 3 can be replaced by step 3' where all occupied pixels are direcly marked as Voronoï seeds, which increases the total execution time. These graphs show the close-to-linear complexity of our approach and reveal that computing the VEDM (step 4) is the most time-consuming part.

Euclidean distance map while we use an approximate VEDM but on the other hand, Man et al. don't run the edge and vertex detection (steps 5 to 8). We observed that steps 6+ run almost in linear time (Figure 4.10), which was not obvious given that these steps include naively implemented clustering which may operate in quadratic complexity in the number of pixels to cluster (Sibson, 1973). The worst case for clustering is when all edges first form independent clusters ($r/2$ vertical edges of length up to $r$ pixels, separated by one pixel) and then join at the bottom of the image. This is far from an usual situation where there are only a handful of edges and clusters, so that managing clusters takes negligible time compared to scanning the image.

We compared our skeleton computation approach (steps 4 to 6) to the works of Garrido et al. (Garrido et al., 2006) on Figure 4.11 and to that of Beeson et al. (Beeson, Jong, and Kuipers, 2005) on Figure 4.12. The occupancy grids used to compare approaches are bigger than scrolling occupancy grids typically used by a robot to represent its immediate environment. Remarkably, there is almost no visible difference for a well chosen robot size ($S + D = 5$ pixels) with the works of Beeson, validating our approach. Beeson et al. don't describe how they computed the skeleton (whether a distance map was used or not and how the ridges were detected), so we cannot compare the efficiency of the algorithms.

We tested robustness of our approach by randomly offsetting pixels of the grid before detection. The result can be seen on the last image of Figure 4.12. The random offset of the pixels (up to $10px$) used to simulate noise is higher than $S + D = 5px$ and yet the simplified skeleton is mostly similar to the non-noisy one, demonstrating robustness to noise of our approach, without even applying step 1. Finally, we extracted a graph representation of the simplified skeleton of the first image, which is also shown on the figure. The graph representation does not show any topological error. This is essential to perform large-scale topological SLAM, as described in chapters 5 and 6.

### 4.2.4 Final words on topology extraction

In this section, we described how to perform robust extraction of the local topology from an occupancy grid using an approximate Euclidean skeleton whose spurious edges have been removed. As far as we know, it is the first description of a robust integrated method from pixels to topology. The steps of this approach individually exhibit multiple benefits over existing ones:

- We use a Vectorial Euclidean Distance Map to compute the skeleton. This map can be reused to achieve obstacle avoidance.

- Spurious edges are removed from the skeleton using criteria parametrized only by the robot's size (the same size being used for obstacle avoidance). These criteria

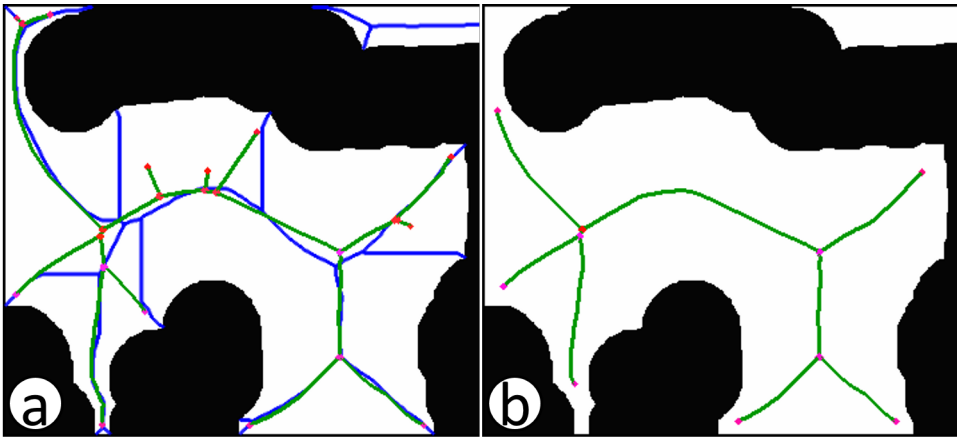



**Figure 4.11:** (a) Comparing our approach (green edges) to that of Garrido et al. (Garrido et al., 2006) (blue edges) on a dataset of (Garrido et al., 2006) reveals that the skeleton of Garrido et al. is not a true Voronoï diagram where all points are equidistant to two obstacles. (b): the robot size $S$ was increased, simplifying the skeleton due to the two edge validation criteria $M \geq 2(S + D)$ and $M' \geq S + D$. Step 3' was used in all cases.

do not make assumptions on whether edges lead to dead ends or are exits leading to free space. The approach is thus more robust to occlusions and point-of-view differences.

- Our vertex detection and clustering algorithms allow robust extraction of the simplified skeleton in a graph format for use in topological mapping.

- Detection of the skeleton (steps 1 to 5) runs in linear complexity. Later steps (steps 6 to 10) either take negligible time or run near linear complexity on real-world data. The total execution time (around 6 ms for a $300 \times 300$ pixel grid) is compatible with robotics experiments.

While our topology extraction approach is robust to noise, there are still rare situations (see Figure 4.14) where slight changes in the occupancy grid lead to large changes in the skeleton. In the context of this thesis, we did not study this issue further, since we found the current accuracy of skeleton extraction to be sufficient for large-scale SLAM (chapter 6). One possible approach to limit this issue of small local changes leading to the local topology being reported differently is to use a *two-threshold* extraction. That is, extract the skeleton once using robot size $S_1$ and minimum distance to obstacle $D_1$ and once using $S_2 = S_1 + \epsilon$ and $D_2 = D_1 + \epsilon$, with $\epsilon$ greater than the uncertainty of individual readings of distance sensors. In most cases, when $\epsilon$ is small (typically, one or two pixels), both extracted skeletons will exhibit the same topology (homeomorphic

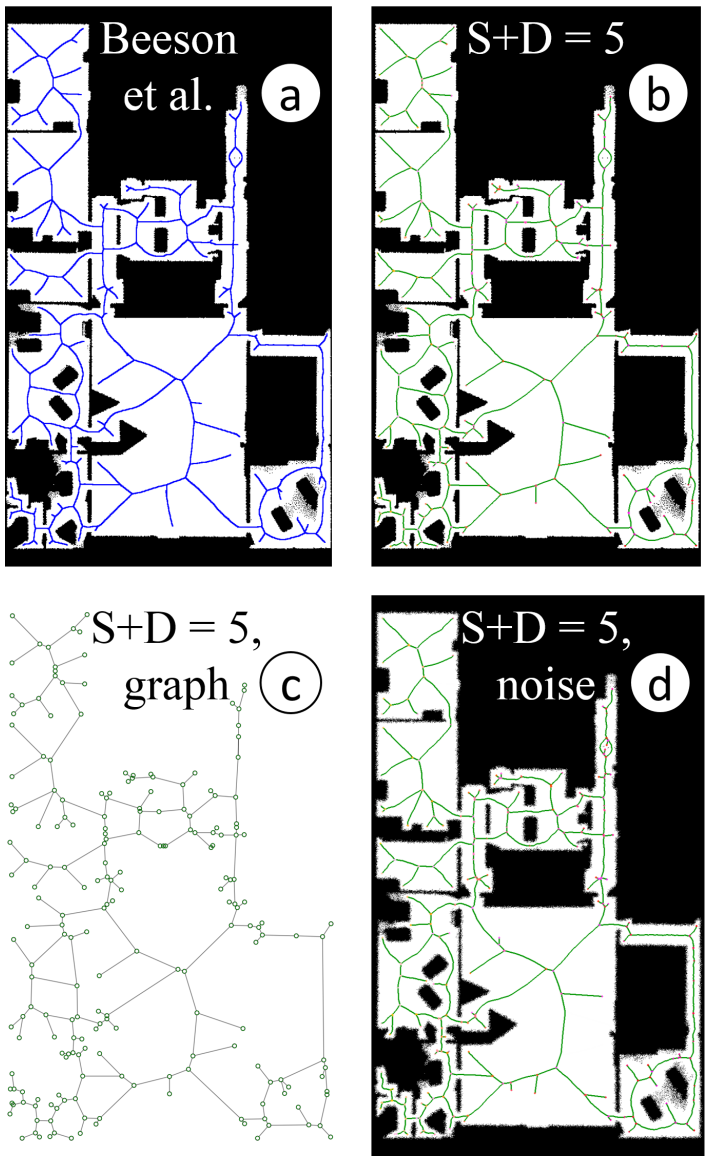



**Figure 4.12:** Our approach (b) gives similar results to that of Beeson et al. (Beeson, Jong, and Kuipers, 2005) (a) on a dataset of (Beeson, Jong, and Kuipers, 2005). Step 3' was used in order to compute the skeleton on the whole image. Image (c) shows the graph extracted from (b), preserving two-edge vertices. For image (d), before computing the skeleton, each pixel of the original image was offset by a random number up to $10 = 2(S + D)$ pixels in the $x$ and $y$ directions. This simulated noise, comparable to the size of the robot in amplitude, causes only minor modifications of the skeleton, confirming robustness of the method.

graphs). If both graphs exhibit a different topology, both can be considered as valid hypotheses for future steps using the skeleton, such as topological SLAM (chapter 5).

## 4.3 Navigation using topology and the Vectorial Euclidean Distance Map

In the previous section, we explained how to compute a Vectorial Euclidean Distance Map and how to extract the topology of the environment using this map. This section explains how to perform navigation using both local topology and the Vectorial Euclidean Distance Map.

### 4.3.1 Large-scale planning and local topological navigation

An (exploratory) planning algorithm such as EDNA* (see chapter 3) is able to compute a path using a map represented as a graph. While it is possible to run EDNA* or other (exploratory) planning algorithms on dense maps, the computational burden may quickly become intractable. Thus, it is preferable to run the planning algorithm on a sparse structure representing places and paths instead of pixels. However, this means that the planning algorithm returns a series of places and paths between places as itinerary. In order to physically execute the plan and transform the sequence of places and paths into speed or acceleration commands, a local (fine-grained) navigator is required.

The main task of the local navigator is to follow a path from one place to another, where paths and places correspond respectively to GVG edges and vertices. A naive navigation approach would be to keep the robot on the GVG, as done by Choset et al. (2001). However, discretization of the GVG would lead to the robot zigzagging around edges. Moreover, we would like the robot to be able to reach any place physically reachable in the environment, which implies that the robot should have some movement freedom around the GVG.

In order to robustly track an edge while not being constrained to navigate exclusively on GVG edges and vertices, we use a virtual ski-tow approach represented on Figure 4.13. Without additional constraints on the robot's motion system, it is sufficient for the robot to go towards the point $P$ where the tow is attached to the cable (edge), represented by a dark blue circle on the figure. The virtual cable on which the tow is attached always moves forward unless the length of the tow reaches its maximum value because the robot is lagging behind. If this is the case, the cable stops. It starts moving again as soon as the robot gets closer to $P$. In the absence of sensor noise and obstacles preventing the robot from going towards $P$, the robot's movement curve is close to

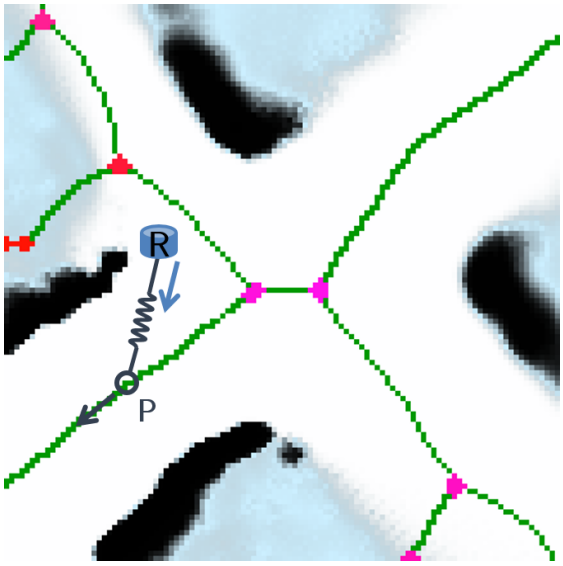



**Figure 4.13:** The robot behaves like a skier attached to a cable (edge) by a ski-tow. The robot is towed forward while keeping lateral movement capacities. If the robot happens to get too close to an obstacle (the VEDM gives the distance to the closest obstacle), a virtual force-based obstacle avoidance strategy is triggered.

a tractrix. When the pole reaches a vertex, the vertex is reported to the topological SLAM and planning algorithms, which in return give the next edge to take. It should be noted that tracking a single edge or vertex between different frames is difficult due to pixelization, especially around a vertex where multiple edges meet. Robustness to occlusions and noise of topology extraction exposed in the previous section is critical to keep the amount of navigation mistakes tractable.

As a final word on topological navigation, we have to raise a subtle issue: while navigating, the robot updates its local occupancy grid and regularly extracts a new skeleton. Since the robot must always be attached to one single edge, each time the skeleton gets updated, we have to find to which edge $E_n$ of the new skeleton $S_n$ the edge $E_o$ of the old skeleton $S_o$ to which the robot is attached corresponds. To do so, algorithm 4 finds to which edge $E_i$ of $S_n$ the individual pixels of $E_o$ correspond and associates a confidence score to the match. The edge $E_n$ of $S_n$ that got the best confidence score from all pixels $p_o$ of $E_o$ is returned.

The $\frac{1}{\sqrt{(||p_o\vec{p}_n||^2+1)(||p_o||^2+1)}}$ confidence score is inversely proportional to the distance between old ($p_o$) and new ($p_n$) pixel and also inversely proportional to the distance from the old pixel ($p_o$) to the robot, with the intuition that the grid gets less reliable when the distance to the robot increases. The +1 terms prevent the score of one pixel from

**Algorithm 4** Local edge remapping

origin of $S_o \leftarrow$ robot's position
origin of $S_n \leftarrow$ robot's position
matchbuffer $\leftarrow [0, 0, ..., 0]$, with as many zeros as edges in $S_n$.
**for all** pixels $p_o$ of $E_o$ **do**
  find edge pixel $p_n$ of $S_n$ that is closest to $p_o$. $p_n$ belongs to edge $i$ of $S_n$.
  matchbuffer$[i] \leftarrow$ matchbuffer$[i] + \frac{1}{\sqrt{(||p_o\vec{p}_n||^2+1)(||p_o||^2+1)}}$
**return** $E_n, n = argmax(\text{matchbuffer}[i])$

being infinite.

### 4.3.2 Obstacle avoidance

While the ski-tow approach allows the robot to move according to the local topology of the environment and theoretically prevents it from hitting obstacles, there are cases where ski-tow navigation fails, causing the robot to hit an obstacle. Some of these cases are:

- as shown on Figure 4.14, despite the robustness to noise of the algorithmic stack explained in section 4.2, edges may appear or disappear on the GVG due to noise and the use of a threshold $S + D$ in GVG edge pruning. This happens quite often in passages barely large enough for the robot, that is passages whose width is just above $2(S + D)$.

- the ski-tow is too loose and the robot has to go around a corner, as on Figure 4.13.

In order to provably avoid obstacles represented in its local world model, a robot can use simple physics-based techniques. Three of the most successful physics-based approaches are the real-time obstacle avoidance approach of Borenstein and Koren (1989), the potential field approach of Barraquand et al. (1992) and the elastic band approach of Quinlan and Khatib (1993).

The approach of Borenstein and Koren is based on having obstacles emit virtual repulsive forces $\vec{f_r}$ towards the robot and the goal position of the robot (if known) emit a virtual attractive force $\vec{f_a}$ towards the robot. The desired acceleration $\vec{a}$ of the robot is then computed using Newton's law $m_{robot}\vec{a} = \vec{f_a} + \sum \vec{f_r}$. This approach's large-scale use is limited by at least two factors: local non-convexities of obstacles causing the robot to get stuck (the issue is represented on Figure 1.3 (a), page 9) and tricky balancing of $||\vec{f_a}||$ and $||\vec{f_r}||$ which, if not done correctly, causes the robot either not to move at all or to hit walls. Nevertheless, if the repulsive forces $\vec{f_r}$ are chosen so that they grow to

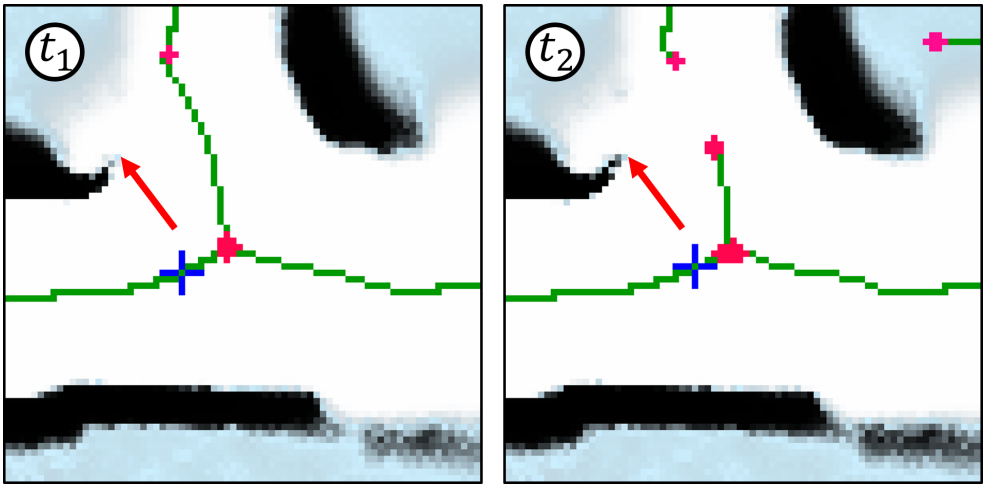



**Figure 4.14:** It may happen for GVG edges to appear or disappear between GVG extracted at successive time steps due to almost invisible differences in the occupancy grid. For instance, here, a subtle change in the zone pointed by a red arrow caused an edge present at step $t_1$ to appear broken at step $t_2$. Fortunately, this is a border case happening when the distance between obstacles is very close to $2(S + D)$ (the robot hesitates whether or not it can pass between these obstacles).

infinity when the robot approaches a wall and to 0 far from any wall (typically, forces in $1/r$ and $1/r^2$), a robot with null reaction time will never hit an obstacle.

While Borenstein and Koren use a Newtonian formulation, Barraquand et al. use a Hamiltonian or Lagrangian (aka. energy-based or integral) formulation of the differential equations of movement. Thus, a virtual potential field (corresponding to a potential energy) is constructed around the robot. This field's value at a specific point corresponds to the distance to the closest object. The ridges of the field correspond to the GVG. The potential field approach is sensitive to sensor noise and is costly computationally-wise, however it supports path planning on the skeleton.

The obstacle avoidance approach of Quinlan and Khatib (1993) is based on constructing and updating free-space "bubbles" centered on the robot's trajectory (Figure 4.15). These bubbles describe space traversable by the robot, leaving safety borders around obstacles. It is easy to recover these bubbles using the Vectorial Euclidean Distance Map: the radius of a bubble centered on pixel $x, y$ of the grid is $||\vec{V_{xy}}|| - (D + S)$.

The Vectorial Euclidean Distance Map combines the advantages all three techniques: while the ski-tow approach uses the GVG computed using non-local properties of the map, virtual repulsive forces are easily obtained from single vectors of the map. Moreover, the robot can compute zones it is not allowed to access using bubbles centered on

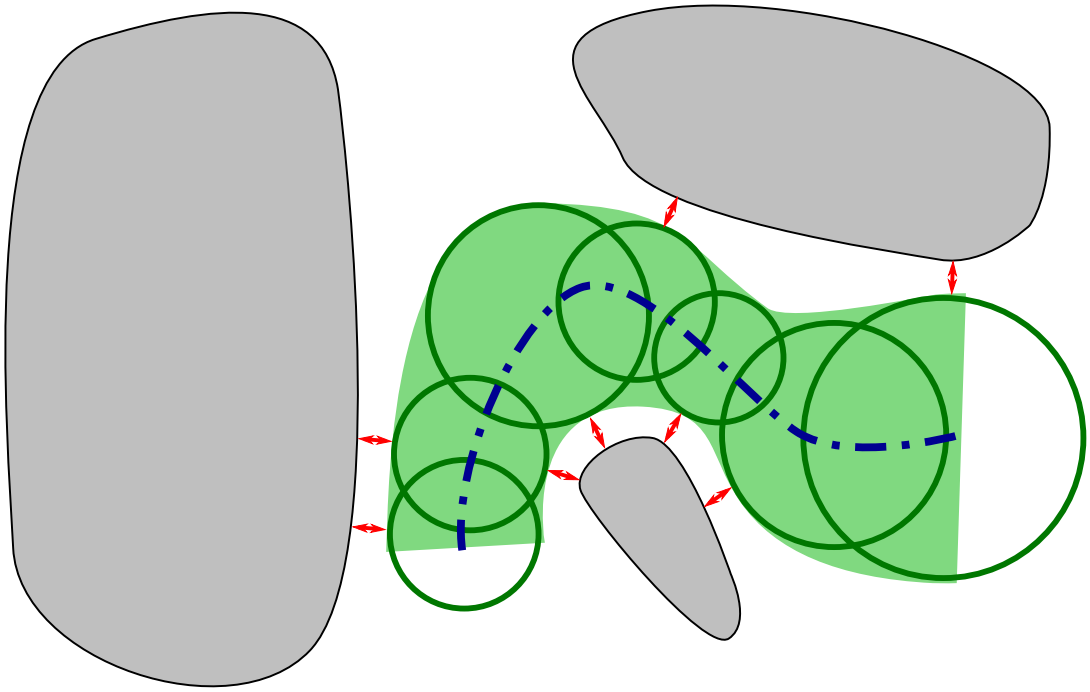


**Figure 4.15:** The "bubbles" approach to navigation and obstacle avoidance of (Quinlan and Khatib, 1993). Obstacles in gray, robot trajectory represented with a dash-dotted blue line. The green balls and the associated interpolated patch represent zones that the robot can safely traverse. These zones do never approach obstacles closer than a "safety" distance $D + S$ marked with a red arrow for each ball. The balls/safety zone approach can easily be derived from the Vectorial Euclidian Distance Map.

its current position in the VEDM. In order to guarantee that the robot never hits an obstacle, we propose the following strategy using two levels of free-space bubbles centered on the robot:

1. When the robot is further than $D+S$ away from an obstacle, the ski-tow approach alone is used to issue speed commands to the actuators. This first strategy corresponds to a free-space bubble extending up to a distance $D+S$ of any obstacle.

2. When the robot gets closer than $D+S$ away from an obstacle, the ski-tow approach is progressively replaced by a repulsive forces-based strategy which we call "obstacle avoidance mode". The influence of both strategies is represented by a single factor $\alpha$. $\alpha = 1$ corresponds to pure ski-tow and $\alpha = 0$ to pure repulsive-forces navigation. Pure repulsive-forces navigation correspond to a free-space bubble extending up to a distance $S$ of any obstacle. If the robot comes closer than $S$ to any obstacle, a collision is likely to occur.

This dual control strategy allows the robot to provably avoid static and dynamic obstacles, as will be shown in the following paragraphs.

### Static obstacles

Suppose that the robot is at position $(0,0)$ on the grid and that the ski-tow approach returns a unitary speed command $\vec{v_u}$.

In order to guarantee safety of the navigation system, if the Vectorial Euclidean Distance Map $\vec{V_{0,0}}$ at the robot's position is such that $||\vec{V_{0,0}}|| < S+D$, the robot enters an "obstacle avoidance" mode. In obstacle avoidance mode, a virtual repulsive force $\vec{f}$ is computed as $\vec{f} = -\vec{V_{0,0}}$. More robust approaches taking into account multiple nearby pixels instead of just the nearest one are possible. Let $\alpha = \frac{||\vec{V_{0,0}}||-S}{D}$.

The unitary command $\vec{v_u}$ is corrected in obstacle avoidance mode as $\vec{v_u} \leftarrow (1-\alpha^2)\vec{f_u} + \alpha^2\left(\vec{v_u} - (\vec{v_u}.\vec{f_u})\vec{f_u}\right)$ with $\vec{f_u} = \frac{\vec{f}}{||\vec{f}||}$ and normalized: $\vec{v_u} \leftarrow \frac{\vec{v_u}}{||\vec{v_u}||}$. The purpose of $\vec{v_u} - (\vec{v_u}.\vec{f_u})\vec{f_u}$ is to compute the part of $\vec{v_u}$ that is orthogonal to $\vec{f_u}$ (removing the part tangent to $\vec{f_u}$). The purpose of the $\alpha$ factor is to balance ski-tow navigation and obstacle avoidance. We found experimentally that squaring $\alpha$ resulted in better obstacle avoidance behaviors but other formulae are possible. When $||\vec{V_{0,0}}|| = S+D$, $\alpha = 1$ and the robot moves orthogonally to $\vec{V_{0,0}}$. When $||\vec{V_{0,0}}|| = S$, $\alpha = 0$ and the robot moves in direction $\vec{f_u}$ (directly away from the obstacle). Thus, a robot with sufficiently low reaction time cannot hit a static obstacle that was present on the occupancy grid at the time the Vectorial Euclidean Distance Map was computed. The obstacle avoidance strategy is represented on Figure 4.16.

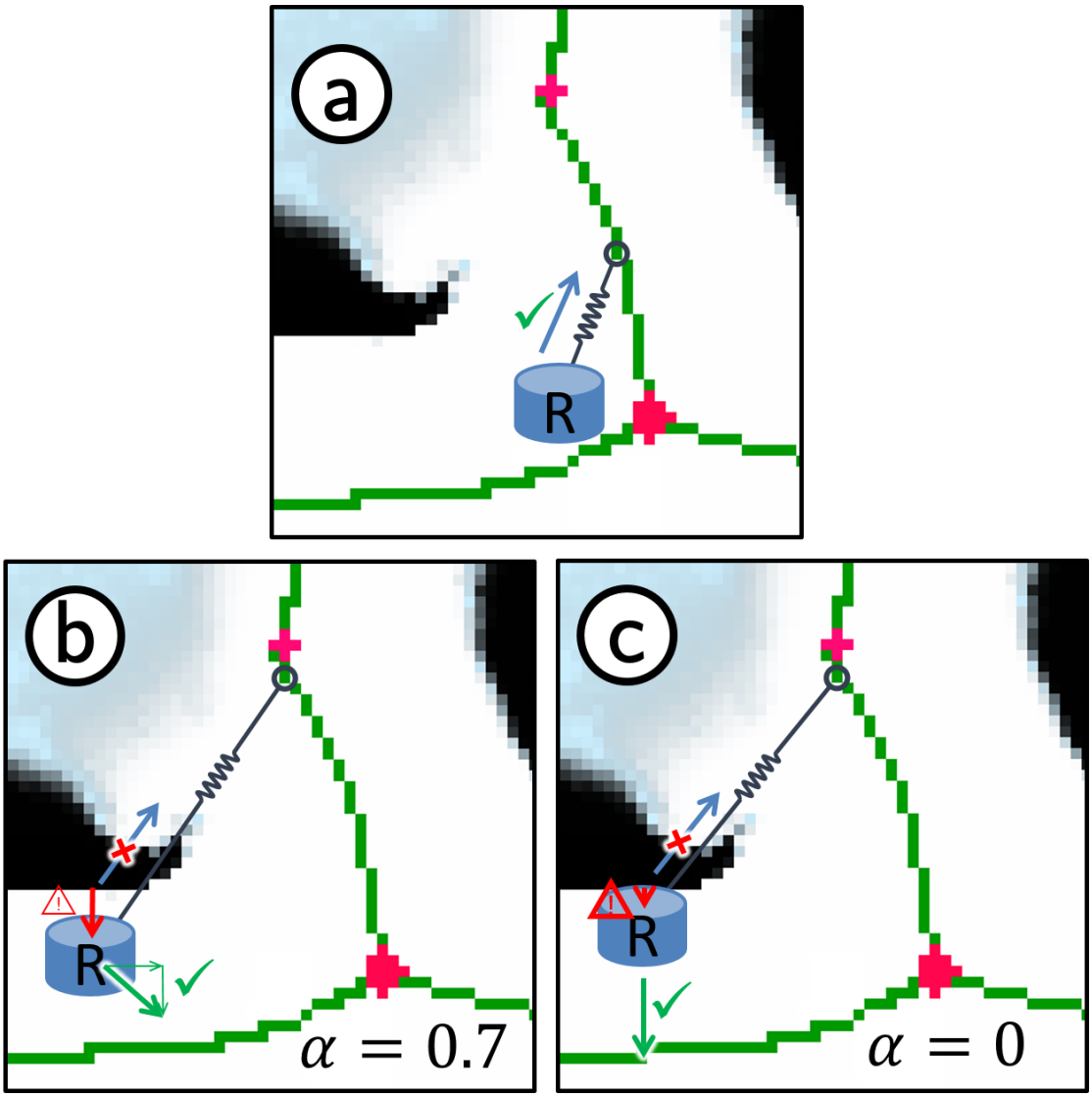



**Figure 4.16:** When the robot gets too close to an obstacle, an obstacle avoidance procedure is triggered. (a) The robot is far from any obstacle, ski-tow-based navigation is used. The ski-tow vector is represented in blue. (b) Getting closer to an obstacle: the ski-tow vector is progressively replaced by the (normalized) virtual repulsive force emitted by the obstacle (represented by a red arrow). The resulting movement vector is represented with a green arrow. (c) Almost touching a wall: when a collision is imminent, the robot uses pure obstacle avoidance and goes directly away from the obstacle.

### Dynamic obstacles

Dynamic obstacles are more of an issue. Suppose that there is a dynamic obstacle whose position is $(x_o(t), y_o(t))$ at time $t$. Suppose that the robot's position is $(x_r(t), y_r(t))$. The position of the obstacle relative to the robot is thus $(x(t), y(t)) = (x_o(t), y_o(t)) - (x_r(t), y_r(t))$. A dynamic obstacle is provably avoided if $||(x(t), y(t))|| \geq S$ at all times. The speed of the dynamic obstacle relative to the robot is $(\frac{dx(t)}{dt}, \frac{dy(t)}{dt}) = ((v_x(t), v_y(t))$.

There are various delays in the system:

- $t_d^1$: sensor delay,
- $t_d^2$: write values to occupancy grid,
- $t_d^3$: compute the Vectorial Euclidean Distance Map,
- $t_d^4$: compute a trajectory,
- $t_d^5$: send values to motors and
- $t_d^6$: physical acceleration

We call $t_s = t_d^1 + t_d^2 + t_d^3$ the sensing delay (delay necessary to detect anything). We also call $t_m = t_d^5 + t_d^6$ the motor delay (time to reach a given command). $t_d^4$ is the update rate of the speed command issued by the local navigator.

The worst-case dynamic obstacle avoidance scenario is the following: the robot and the dynamic obstacle are moving towards each other, preparing for a head-to-head collision. We take $y(t) = 0$ given the symmetry of the problem and suppose that the "intruder" (dynamic obstacle) does not change its trajectory, going along the $x$ axis at speed $v_o > 0$. The robot's speed is $v_r(t)$ along the $x$ axis. Let $v(t) = v_o - v_r(t)$. $t_s$ is modeled by saying that while the local navigator sees the intruder at distance $r$, the intruder is already at distance $r + v_o.t_s$. Let $v_{max}$ be the maximum speed of the robot. The dynamical obstacle avoidance process is sketched on Figure 4.17.

At time $t_0$, the "intruder" (dynamic obstacle) and robot are $x(t_0)$ apart with $r_0 = |x(t_0)| < S + D$. At time $t_1 = t_0 + t_s$, the local navigator "sees" the obstacle at distance $r_0$ from the robot. The minimum value of $r_0$ is $S + D - (v_o + v_{max}).t_s$ (otherwise, the obstacle would have been detected at the previous frame). At this point, robot and intruder are $r_1 = r_0 - \sum_{t'=t_0}^{t_1} v(t').t_d^4$ apart. If $r_1 < S + D$, a collision occurs. Otherwise, the obstacle avoidance mode is triggered, and a corrected unitary speed command $\vec{v_u}(t_1)$ is issued. Given how $\vec{v_u}(t_1)$ is computed, $vx_u(t_1) \geq \frac{1-\alpha^2}{\sqrt{2}} = (\frac{1}{\sqrt{2}} - \frac{(r_0-S)^2}{\sqrt{2}D^2})$. In the worst

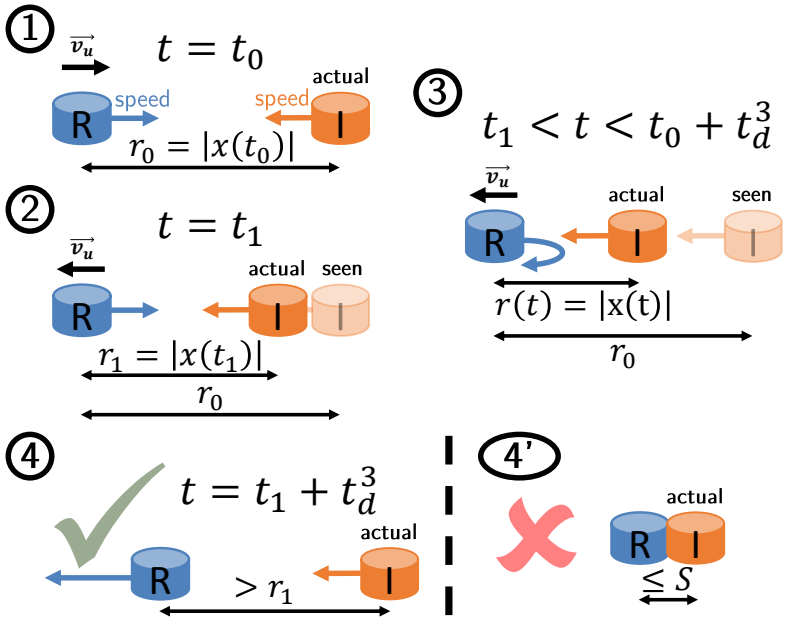



**Figure 4.17:** (1) An intruder approaches the robot at a distance $r_0 < S + D$. (2) The robot sees the intruder at $r_0$, but due to the sensing delay $t_s$, the intruder is already at $r_1$. (3) The robot tries to flee. (4) If the robot increased its distance to the intruder before the next grid update, obstacle avoidance has succeeded. (4') If obstacle avoidance fails, there is a collision. It is possible that $S < r(t_1 + t_d^3) < r_1$ due to inertia, in which case another frame of duration $t_d^3$ has to be considered.

case, the final speed command of the robot will be $v_r(t_1) = v_{max}(\frac{1}{\sqrt{2}} - \frac{(r_0-S)^2}{\sqrt{2}D^2})$.

Between $t_1$ and $t_1+t_d^3$, the speed command of the robot will be $v_r(t_1+n.t_d^4) = v_{max}(\frac{1}{\sqrt{2}} - \frac{(r_0+\sum_{t'=t_1}^{t_1+(n-1)t_d^4} v_r(t'-t_m).t_d^4-S)^2}{\sqrt{2}D^2})$ unless $r_0 + \sum_{t'=t_1}^{t_1+(n-1)t_d^4} v_r(t'-t_m).t_d^4 \geq S+D$ (which is a best-case scenario: the robot escaped the intruder). The robot successfully escapes the intruder if it manages to increase its distance to this intruder: $\sum_{t'=t_1}^{t_1+t_d^3} v_r(t'-t_m).t_d^4 \geq v_o.t_d^3$. Otherwise, the calculation is repeated for another $t_d^3$ frame. Figure 4.18 shows an example of successful and unsuccessful dynamical obstacle avoidance. Correct obstacle avoidance is observed on the figure with $v_o = 1\text{m}\,\text{s}^{-1}, v_{max} = 1.7\text{m}\,\text{s}^{-1}, t_d^3 = 200\text{ms}, t_d^4 = 10\text{ms}, t_s = 210\text{ms}, t_m = 70\text{ms}, S = 10\text{cm}$ and $D = 80\text{cm}$. It should be noted that avoidance of dynamic obstacles is only possible if $v_{max} > v_o$ by some amount dependent on the various delays. Reducing delays is always beneficial for obstacle avoidance. A typical C implementation without parallelism can easily reach $t_s < 50\text{ms}$ on a $300 \times 300$ grid, using a ring of sonars as distance sensors. (Massively-)parallel (GPU) or hardware implementations (ASIC, FPGA) may only need $20ms$ or less.

The formulae of the previous paragraph are computed for a worst-case (head-to-head) collision. Practically, the robot is extremely likely to perform lateral obstacle avoidance instead of trying to escape in the direction the intruder is moving. The additional security margin obtained thanks to lateral obstacle avoidance may be necessary for a non-holonomic robot which may need to rotate around itself before trying to escape.

Even during dynamic obstacle avoidance, the robot remains attached to the topological skeleton by the ski-tow. If the dynamic obstacle causes the edge to which the tow is attached to disappear, the tow attaches itself to the closest remaining edge.

## 4.4 Conclusion on local topology, navigation and obstacle avoidance

Local navigation and obstacle avoidance is probably one of the earliest capacity acquired by animals during evolution (think “escaping predators”). For a robot, it is critical to perform these tasks provably and in constant time, no matter the size of the environment or the mission the robot is charged with (or charged itself with). The approach we proposed in this chapter is both fast (the whole sensing-and-control loop may react faster than 50ms, down to less than 20ms with an optimized implementation) and safe (provable static and dynamic obstacle avoidance). The underlying tool, the Vectorial Euclidean Distance Map, is also used in our approach to robustly extract the local topology of the environment for use in topological SLAM and planning. The description of our (hybrid metrical/)topological framework is the point of chapter 5.

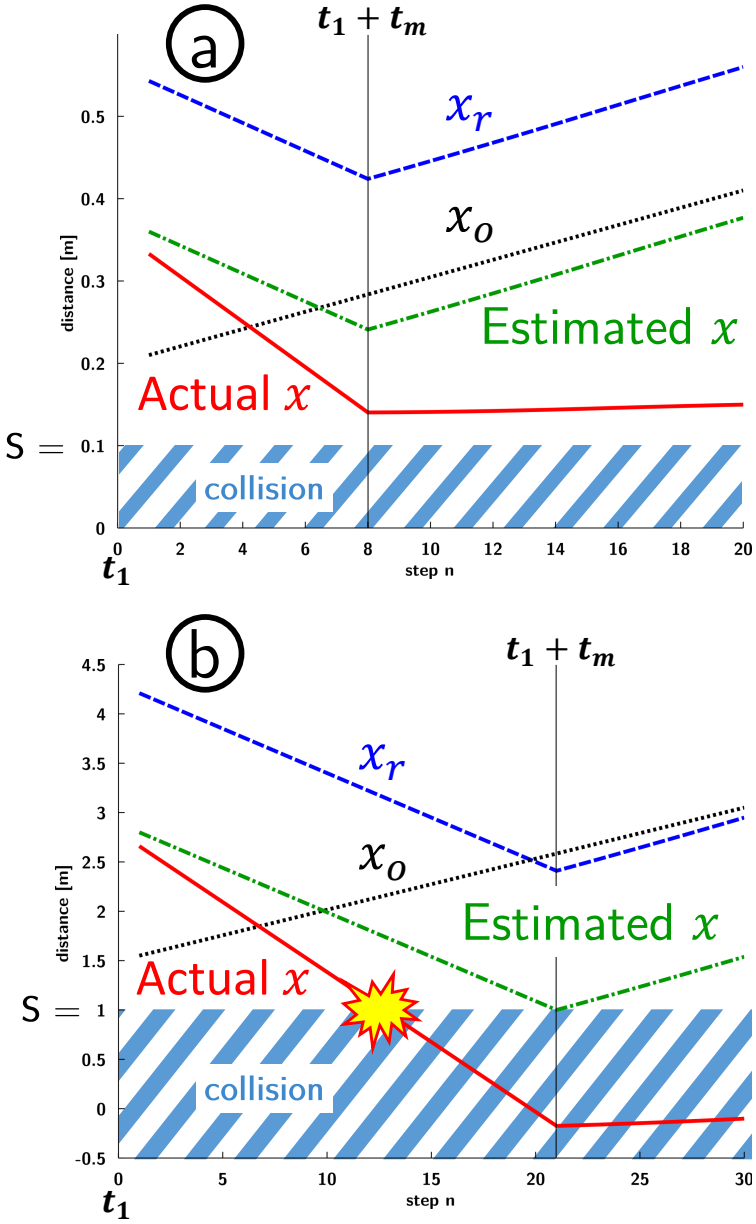



**Figure 4.18:** (a) The robot ($S = 0.1$), whose position is marked with a dashed blue line, managed to escape the dynamically moving obstacle whose position is marked with a dotted black line. Indeed, the position difference $x(t)$ (plain red line) never goes below $S = 0.1$ and increases from timestep 8 on. The dash-dotted green line represents the robot's estimation of $x(t)$ (the robot erroneously thinks the obstacle is static). (b) here, the robot did not manage to escape and collided with the obstacle (the position difference goes below $S = 0.1$). On both figures, the horizontal axis represents timestep $n$ from $v_r(t_1 + n.t_d^4)$.

# 5 A hybrid metrical/topological SLAM for Lifelong Exploratory Navigation

*"Our brain is mapping the world. Often that map is distorted, but it's a map with constant immediate sensory input."*
**E. O. Wilson**

The SLAM component is not straightforward to implement, notably because it has to cope with probabilistic sensor noise as well as semantic place recognition (a *classification* problem). Additionally, the SLAM framework is the common denominator allowing interaction of abstract intelligence, robot mission control, exploratory path planning and low-level navigation (Figure 5.1). Our SLAM framework uses a new *bounded uncertainty* model as well as *active disambiguation* in order to ensure that the robot never gets lost. Compared to other SLAM frameworks, it is designed for Lifelong Exploratory Navigation while offering additional features such as handling of *one-way paths* or recognition of already visited environments (the kidnapped robot problem: `https://en.wikipedia.org/wiki/Kidnapped_robot_problem`). This chapter explains the theoretical construction of the framework, with simulations and experiments deferred to chapter 6.

## 5.1 Introduction - Motivations

### 5.1.1 Stakes of autonomous robot navigation and SLAM

In order to build an accurate map of an environment, a reliable SLAM framework should cope with three sources of errors (Kuipers et al., 2004) represented on Figure 5.2:

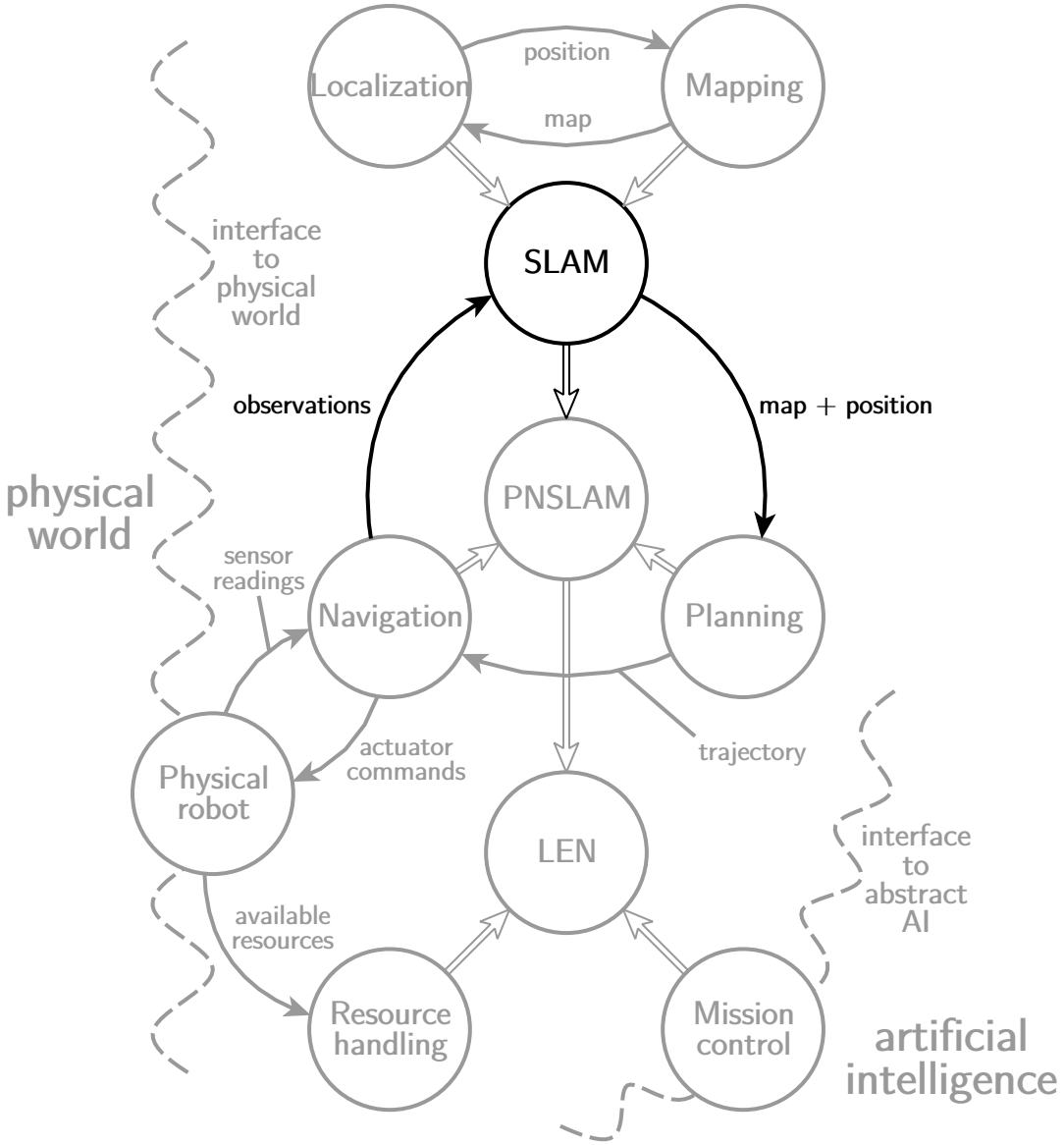



**Figure 5.1:** A SLAM-centered view of PNSLAM and LEN.

1. *Movement uncertainty*, which describes errors of the sensors determining the trajectory of the robot (typically, wheel odometry or point tracking with a camera).

2. *Pose uncertainty* which describes the fact that the robot cannot determine its exact position relative to a point of interest in real space, because its sensors are not precise enough or because the position of the point of interest is not precisely defined. For instance, a road crossing may be a conceptually well-defined and point-of-view independent place, but the exact center of the crossing is defined only within a precision of a few feets.

3. *Structural ambiguity*, also known as the *cycle detection* or *loop closure* problem and which is solved by unambiguously determining whether or not the current place is already on the map (and if yes, which place on the map it is).

In addition to these three sources of errors, a SLAM framework may also correct *global metrical uncertainty* (Kuipers et al., 2004) to produce a map visually close to ground truth.

Autonomous navigation imposes additional constraints on the SLAM framework:

- First and foremost, the SLAM framework should *run online*, which limits the execution time of any algorithm to a maximum of a few seconds.

- Then, the SLAM framework should maintain up to date *a map of the environment which represents physically traversable space* (and not only landmarks for instance).

- Additionally, path planning algorithms require a unique starting position (current position of the robot). Thus, it is not sufficient for the SLAM framework to update a set of likely positions of the robot: the framework should provide a *single robot position*.

- Finally, physical environments contain *single-way paths*, which the map produced by the SLAM framework should model. Examples of single-way paths are escalators and single-way roads.

In this chapter, we propose a new SLAM approach addressing all these constraints in a unified framework.

### 5.1.2 Hybrid metrical/topological SLAM

Our approach is based on the hybrid metrical/topological SLAM frameworks of Kuipers, Beeson et al. (Beeson, Modayil, and Kuipers, 2010; Kuipers et al., 2004; Kuipers, 2000),

Bailey (2002) and the ATLAS framework of Bosse et al. (2004). In these works, the map produced by the SLAM algorithms is a graph of submaps linked by metrical relations (see Figure 5.2).

In the works of Bailey (2002) and Bosse et al. (2004), each vertex of the graph is associated to a local coordinate frame and each edge models the transformation and accumulated uncertainty from its origin vertex to its destination vertex, in the coordinate frame of the origin vertex. Mapping does not require a global coordinate frame, but approximate absolute positions can still be obtained by means of global optimization of the position of submaps. In order to diagnose and fix *structural ambiguity*, *uncertainty projection* is used to determine places already on the map that may match the current place. Each of these places generates a loop closure hypothesis. The validity of each hypothesis will be tested against evidence acquired during later movements of the robot. If there is sufficient evidence to validate an hypothesis, then a new loop has been found and can be written down on the map. Since acquiring evidence that a specific loop closure hypothesis is valid requires physical movements and computations, there is a balance between map correctness and mapping and navigation efficiency. For performance reasons, when a loop is found in the environment, the transformations stored in the edges of the graph are not modified even though the compound transformation along the newly formed loop should be the identity. A global map may be obtained a posteriori by means of constrained optimization. All processes but *uncertainty projection* and disambiguation run in constant complexity. Uncertainty projection runs in $\mathcal{O}(N.log(N))$ complexity (Bosse et al., 2004) and disambiguation of place hypotheses runs in $\mathcal{O}(N)$ complexity, where $N$ is the number of vertices on the map at a given time, provided that the degree (number of outgoing edges) of each vertex is bounded.

### 5.1.3 Problems of existing hybrid metrical/topological SLAMs

The works of Bailey (2002) and Bosse et al. (2004) present a number of unsolved issues:

- They do not describe *traversability* of a mapped environment, only the relative position of features in this environment. Consequently, the produced map cannot be used directly by a robot to navigate in the environment.
- They are not designed to interact with approaches used for *local path planning and obstacle avoidance*, such as occupancy grids.
- The *disambiguation of place hypotheses* while mapping is not compatible with autonomous robot navigation while mapping, which requires loops to be ascertained as quickly as possible and not only when enough information is available.
- They do not consider the *kidnapped robot problem.*

- They do not meet the *single robot position* requirement.
- They cannot take into account *single-way relations* between vertices of the graph. When traversability of an environment has to be modeled, single-way paths have to be considered.
- They have trouble coping with *point-of-view dependency* of place detection.
- They do not separate *pose and movement uncertainties*, which may cause inaccuracies in uncertainty projection.
- They cannot ensure *topological correctness* of the produced map.
- Their *worst-case algorithmic complexity* is at least linear in the number of features or places in the environment.

### 5.1.4 Proposed approach

This chapter introduces a hybrid metrical-topological SLAM framework designed from the ground up for (real-time) autonomous robot navigation and exploration in extremely large, highly cyclic (hundreds of loops) and ambiguous environments, such as entire cities. A consistent map is produced after each observation, where consistent means that path planning and navigation algorithms can use the map to successfully direct the robot within the environment. Only a finite set of past movements of the robot is remembered in addition to the map itself.

We introduce a few essential modifications and additions to the concepts described by Bailey (2002) and Bosse et al. (2004):

- Our approach uses a map representing *physically traversable* places as graph vertices and paths as graph edges.
- Our framework is designed to *interact with a scrolling occupancy grid* of fixed size (Kuipers et al., 2004) or an equivalent representation of free and occupied space in the immediate vicinity of the robot.
- We introduce *active disambiguation* of place hypotheses to allow for autonomous navigation and meet the requirement of a *single robot position*.
- The method used to build and maintain the map provides an elegant solution to the *kidnapped robot problem* (recognizing whether the current environment corresponds to a map stored in memory).

- The map generated by our framework is a *directed graph* which can contain *single-way edges*.

- Using single-way edges provides an elegant solution to *point-of-view dependency* of place detection.

- While *movement uncertainty* was already handled in the works of Bailey (2002) and Bosse et al. (2004), we introduce *pose uncertainty* and propose a unified formalism (section 5.3) to handle both *pose* and *movement* uncertainties.

- When all sensor uncertainties are bounded, we demonstrate (section 5.4, theorem 7) that constructing a "provably good" map with our framework is possible under some mild assumptions.

- Under some mild assumptions, we reduce in chapter 7 the *worst-case algorithmic complexity* of the SLAM framework to constant in the number of features or places in the environment.

- In addition to these modifications, we also introduce a new set of evaluation metrics based on using the map for navigation in order to compare SLAM approaches in a quantitative and sensor-agnostic way.

Places are understood in our approach as approximate physical positions whose definition may be point of view dependent but which are detected identically when traversing the environment multiple times with the same trajectory in the absence of sensor noise. Each place is associated to its signature, which may be globally and locally ambiguous. Like in (Bosse et al., 2004), a local SLAM algorithm running in constant complexity can be used to increase the quality of odometry. However, such a local SLAM is not required in our approach.

Absolute vertex positions are not necessary for SLAM and navigation but useful for visualization of the produced map, interaction with a human operator and optimization of algorithmic complexity (see chapter 7). With our approach, approximate absolute vertex positions are obtained by means of considering the graph as a spring-mass network whose energy should be minimized. With the help of spring-mass energy minimization and *rigging* with the graph, it is possible to produce a global dense map of the environment (section 5.5).

The main components of our approach are sketched on Figure 5.2.

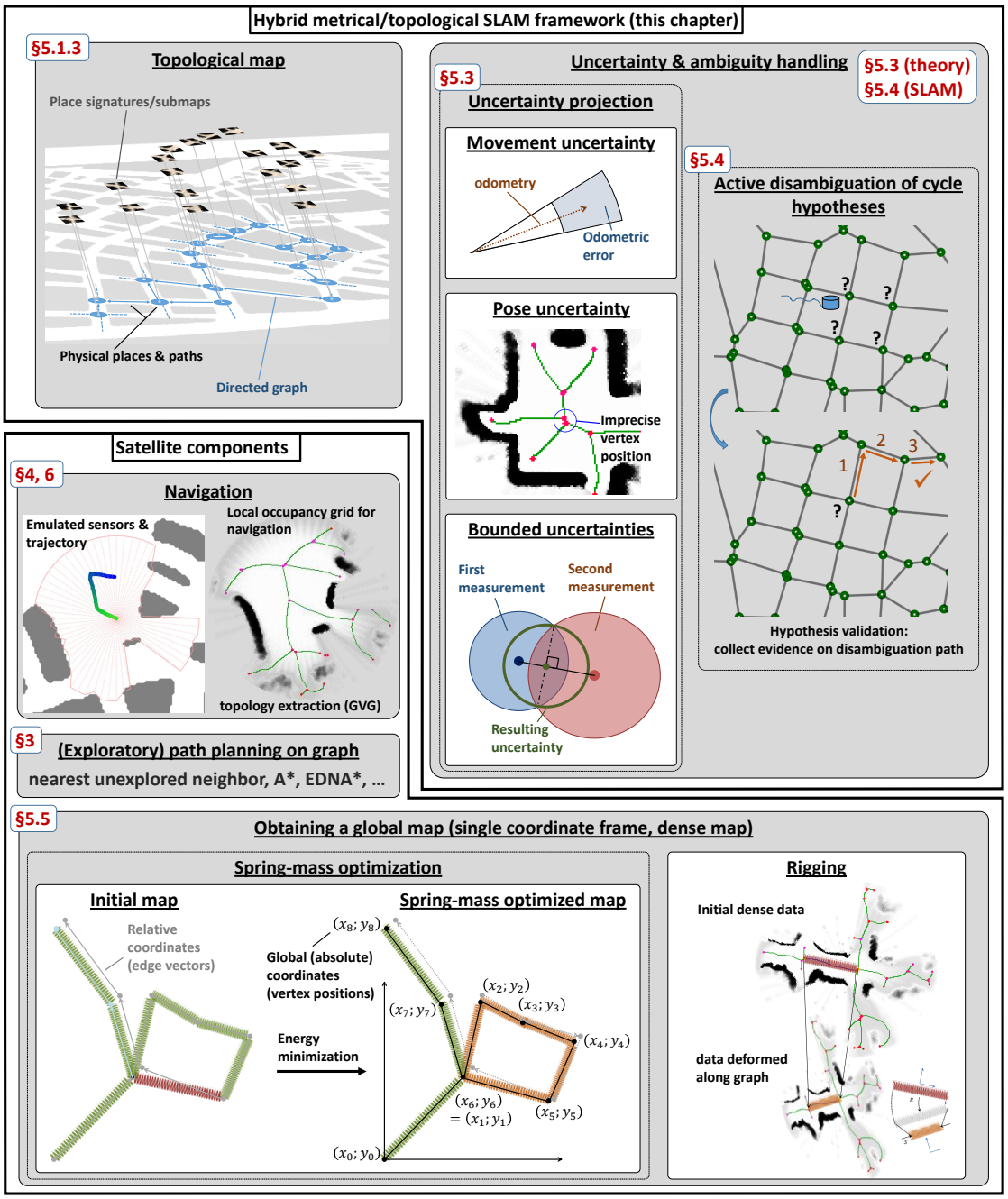



**Figure 5.2:** Overview of the components of our SLAM framework. Satellite components (exploratory) planning and navigation are respectively described in chapters 3 and 4. Spring-mass and rigging, described in section 5.5 are required for visualization of the produced map. Paragraph numbers (in red) point to the section or chapter where the corresponding component is described.

### 5.1.5 This chapter

The rest of this chapter goes as follows: in section 5.2, we briefly review existing SLAM frameworks with a focus on their structural differences and on which kinds of uncertainties or ambiguities they can tolerate. We then compare the components of our approach to that of existing SLAM frameworks. In section 5.3, we introduce a *bounded uncertainty model* and derive the relative uncertainty accumulated (or *projected*) from one vertex to another along a path. Uncertainty projection is used in section 5.4 to determine places already on the map that may match the current place. *Active disambiguation* of these place hypotheses is then used to validate loop closures, leading to theorem 7 which guarantees that *structural ambiguity* is handled correctly in our approach. Rules for choosing the parameters of the approach are also given. In section 5.5, we describe how spring-mass optimization and rigging allow computing a global map in a single reference frame. Simulations and experiments are postponed until chapter 6 since these do reflect the whole PNSLAM/LEN approach, not only the SLAM component.

## 5.2 Related work

The history of SLAM is that of progressively taking uncertainty and errors into account in order to produce accurate maps of environments of increasing size and complexity. The development of SLAM frameworks was first driven by the necessity to take into account probabilistic *movement uncertainty*. It then progressively switched focus to the loop closure or *structural ambiguity* (Kuipers et al., 2004) problem, leading to hybrid metrical/topological frameworks such as ours.

We call *topologically correct* a SLAM algorithm producing both a map homeomorphic to the environment being mapped and topologically correct positioning within said map. The notion of homeomorphism denotes the fact that the map is free of structural ambiguity (perfect loop detection). Topologically correct positioning means that the robot knows at which intersection in the topological map or along which path between intersections it is located at any moment. Topological correctness of SLAM is sufficient to achieve planning and navigation from any already explored place to any other already explored place in the environment.

### 5.2.1 probabilistic uncertainty and structural ambiguity

In the presence of noise, the works of Smith et al. (1986; 1990) and Durrant-Whyte et al. (1996) introduced a fundamental idea: the robot's position and the current map state should be expressed as a single joint probability distribution and cannot be factored into

a robot position distribution and a map state distribution. When a joint distribution is used and if there is no structural ambiguity, the map will converge with increasing number of observations.

The first probabilistic SLAM approaches such as Enhanced Kalman Filter (EKF) SLAM (Dissanayake et al., 2001; Smith, Self, and Cheeseman, 1990) considered the joint robot-position-and-map-state distribution as a whole, ignoring multimodality caused by structural ambiguity, which caused them to produce topologically inconsistent maps in all but very small environments with few or no loops. Intuitively, a probability density can be expressed as the product of peaks due to different hypotheses (structural ambiguity) and unimodal probability densities due to uncertainty. This intuition was used in Montemerlo et al.'s FastSLAM (Montemerlo et al., 2002). In FastSLAM, the distribution of possible trajectories of the robot gets sampled (to recover from structural ambiguities) while the distribution of measurements around the robot (uncertainty-driven) keeps its model-driven nature. However, FastSLAM suffers from progressive loss of diversity of the particle cloud used to represent sampled hypotheses (Havangi, Teshnehlab, and Nekoui, 2010) and its ability to map extremely large environments has yet to be demonstrated (Aulinas et al., 2008; Bosse et al., 2004). A simple description of both the EKF and FastSLAM can be found in the first part of the survey of probabilistic SLAM by Durrant-Whyte and Bailey (2006).

In parallel to the aforementioned Bayesian SLAM approaches whose internal memory is the joint probability density of the robot position and the map state at a given time, another type of SLAM methods developed: topological SLAMs. Topological SLAMs model the environment as a graph of places characterized by signatures (see for instance Dudek et al. (1991; 1993), Savelli and Kuipers (2004), Tapus et al. (2008) or Werner et al. (2009)). Since they do not consider metrical localization, topological SLAMS are robust to odometric drift and don't need to model sensor uncertainties. The use of the notion of *places* is supported by evidence from neurobiology (Ekstrom et al., 2003; O'Keefe, 1976) that the brain of mammals encodes such *places* even though the complete localization and mapping system of the brain has not been puzzled out yet. Using places raises the issue of how to reliably extract places that will be detected identically in future traversals of the environment. The recent survey of visual place recognition by Lowry et al. (2016) discusses the notion of *places*, the use of places for SLAM as well as extraction of places from visual data. Visual SLAM algorithms such as FabMap (Cummins and Newman, 2008; Cummins and Newman, 2010) or SegSLAM (Milford and Wyeth, 2012) can be categorized as topological approaches where places of the graph are individual pictures.

Purely topological SLAMs where places are only characterized by their signature are however computationally expensive since the search space to be considered for each new observation is not constrained by a position estimate. Moreover, purely topological SLAMs cannot easily disambiguate between different places with the same signature. There is also evidence from neurobiology that at least rodents (but probably other

mammals) are able to manipulate some kind of vector arithmetics (Etienne and Jeffery, 2004; Touretzky and Redish, 1996). For these reasons amongst others (Bailey, 2002), hybrid topological/metrical SLAMS (Bailey, 2002; Beeson, Modayil, and Kuipers, 2010; Bosse et al., 2004; Estrada, Neira, and Tardos, 2005; Kuipers et al., 2004; Thrun et al., 1998; Thrun and Montemerlo, 2006; Tomatis and Nourbakhsh, 2002) were developed. Hybrid frameworks store approximate metrical relations on the edges and vertices of the graph in addition to the semantic properties necessary for topological SLAM. In a hybrid metrical/topological SLAM framework, *probabilistic uncertainties* are stored on edges and vertices while *structural ambiguity* is encoded in the structure of the graph (its topology): structural errors are represented by the absence/incorrect presence of some edges and vertices.

Finally, the seminal paper of Lu and Millios (1997) led to the development of so-called Graph SLAMs (Gutmann and Konolige, 1999; Pinies, Paz, and Tardos, 2009; Schuster et al., 2015; Thrun and Montemerlo, 2006; Wagner, Frese, and Bauml, 2014) which represent the map as a graph of robot poses (vertices) and constraints describing observations (edges) (Grisetti et al., 2010). The main difference between graph SLAMs and hybrid metrical/topological (m/t) SLAMs is that hybrid m/t frameworks rely at some point on discrete places and create a sparse graph using only these places, while graph SLAMs represent each pose (or a subset of all poses) of the robot as vertices of the graph. Additionally, hybrid m/t frameworks often use two levels of mapping (local and global) while graph SLAMs usually use only one. While hybrid m/t SLAMs are focused on topological correctness (*uncertainty projection* and the *loop closure* problem), graph SLAMs try to optimize metrical correctness of the produced map relative to ground truth (which can be done as an optional post-processing step in hybrid m/t approaches). Both approaches are currently being mixed together by making loop closure more robust (Pfingsthorn and Birk, 2014) and reducing the number of vertices in the graph (Carlevaris-Bianco, Kaess, and Eustice, 2014; Mazuran, Burgard, and Tipaldi, 2016) in graph SLAMs.

### 5.2.2 Comparison to our approach

#### Structure of the map

The structure of the map as a network of submaps (places) of our approach is taken from (Bosse et al., 2004), as well as the general algorithm: uncertainty projection implemented with Dijkstra's algorithm, vector summation (expressed more generally as transformation composition in (Bosse et al., 2004)) and disambiguation. The paper of Bosse et al. describes the various advantages of such a structure over approaches with a single reference frame such as the Enhanced Kalman Filter or FastSLAM (Montemerlo et al., 2002). Similarly to the works of Bosse et al. or Thrun et al. (1998) and as opposed to that of Choset et al. (1997; 2001), local uniqueness of place signatures is not

required in our approach. Finally, like (Bosse et al., 2004), our approach can be coupled to a local mapping technique such as a scrolling occupancy grid (Kuipers et al., 2004), knowing that structural ambiguity can be ignored at a local scale (Thrun et al., 1998).

We represent physical places and paths by a graph, as in Choset et al.(1997; 2001) or in Modayil, Beeson and Kuipers (2010; 2004). These places and paths are not extracted directly from a sensor profile, but from an intermediary process such as a local SLAM and/or an occupancy grid following Kuipers and Beeson (Beeson, Jong, and Kuipers, 2005; Kuipers et al., 2004). We carried experiments (not reported here) using the approach of Choset et al. without an occupancy grid but detection of vertices proved not to be robust enough to sensor noise when mapping large environments. The issue was especially critical for vertices with more than three outgoing edges. Contrary to the works of Bosse et al. (2004) and Bailey (2002), we define our "places" as single points in space, following Kuipers et al. (2004) and Beeson et al. (2005). This alleviates the issue of "multiple overlapping submaps for the same region of the environment" and of map fusion raised but not solved by Bosse et al. The equivalent of Atlas's map-frames would be local occupancy grids centered on a physical place.

#### Autonomous robot navigation

One of the main differences between our approach and existing SLAM frameworks is that SLAM, planning and navigation are usually studied separately, with the assumption that they can act independently from each other. As a result, most SLAM approaches able to map large and highly ambiguous environments are incompatible with path planning, navigation or both. For instance, approaches that track multiple hypotheses such as Thrun et al.'s metrical topological integration (1998), Montemerlo et al.'s FastSLAM (2002) and hybrid metrical/topological frameworks (Bailey, 2002; Bosse et al., 2004) do not allow path planning in the sense that the robot almost never knows where it *is*, only where it *may be*. Additionally, not all SLAM approaches run online. For instance, Thrun and Montemerlo's GraphSLAM (2006) processes data a posteriori to produce the most likely map of the environment. Finally, navigation requires a map representing traversable and occupied space and not only landmarks as used for example by Dissanayake et al. (2001).

On the other hand, SLAM approaches designed for autonomous path planning and navigation are more limited in terms of mapping capacities or in the way the robot can move within the environment. For instance, occupancy grids (Elfes, 1989; Thrun, 2001) are easily used for path planning (using the (Lazy) Theta* (Nash et al., 2007; Nash, Koenig, and Tovey, 2010) algorithm for instance) and navigation but cannot model *structural ambiguity* as defined by Kuipers et al. (2004). Choset and Nagatani's (2001) approach of topological maps based on the Generalized Voronoï Graph (GVG) is partly successful in modeling structural ambiguity but requires the robot to stay equidistant

to two obstacles at all times. Additionally, the approach of Choset and Nagatani may fail when identical-looking places exist within the environment.

The idea of combining hybrid metrical/topological maps for large-scale mapping and path planning (using A* (Hart, Nilsson, and Raphael, 1968) or EDNA* (Mayran de Chamisso, Soulier, and Aupetit, 2015)) and local representations of physical space such as occupancy grids for autonomous navigation emerged with the works of Kuipers, Beeson et al. (Beeson, Modayil, and Kuipers, 2010; Kuipers et al., 2004; Kuipers, 2000) and was later used by Konolige et al. (2011). The approach of Kuipers, Beeson et al. does not use *uncertainty projection* to find *loop closures*. As a consequence, a tree of all loop closure hypotheses has to be maintained in (Beeson, Modayil, and Kuipers, 2010), which is not compatible with path planning during mapping and may become intractable in extremely large environments with hundreds of loops. Konolige et al. (2011) do neither use *uncertainty projection* to find *loop closures* nor advanced hypothesis testing to ascertain the validity of loop closure hypotheses. Moreover, the nodes (vertices) of the graph used as topological map in (Konolige, Marder-Eppstein, and Marthi, 2011) do not represent places (understood as the intersection of physical paths). Finally, while we were working on our SLAM framework, Carrillo et al. (2015) published an approach mixing occupancy grids (for autonomous robot navigation and exploration) and graph SLAM. The approaches of Konolige et al., Kuipers, Beeson, Modayil et al. and Carrillo et al. are tested in environments containing only a handful of topological loops, which is not sufficient to assess large-scale mapping capacities.

#### Single-way edges

As far as we know, the use of single-way edges and the associated point of view dependence of place extraction is unique to our work. However, if the transformation and accumulated uncertainty along an edge $E$ from an origin vertex $V_1$ to a destination vertex $V_2$ is the same as that on a reciprocal edge from $V_2$ to $V_1$, then both single-way edges can be merged into a single dual-way edge to return to the operating conditions of Bosse et al. (2004) or Gutmann and Konolige (1999). Merging edges is only possible if point-of view dependence is low and if space is physically traversable in both directions.

#### Pose uncertainty

Even though Kuipers et al. (2004; 2000) formulated the difference between *pose* and *movement* uncertainty, this difference was never explicitly used in the structure of a hybrid metrical/topological SLAM approach. Attaching movement uncertainty to measurement of paths/edges and pose uncertainty to detection of places/vertices seems very natural, but maintaining a model with two different sources of uncertainty requires extra

care. We show in section 5.3 that the *pose uncertainty* terms are necessary to decorrelate the map from the actual trajectory of the robot by allowing loops to be ignored during uncertainty projection.

#### Kidnapped robot problem

The *kidnapped robot problem* consists in recognizing, when the robot is switched on, whether the current environment corresponds to a known one or is completely new. In case the environment is known, the current localization of the robot within the existing map should also be provided. The kidnapped robot problem requires the robot to travel in the new environment until it can ascertain whether this new environment matches the existing map (Koenig, Mudgal, and Tovey, 2006). The kidnapped robot problem is usually solved using Kalman filtering or Monte Carlo Localization (Thrun, 2000). Vision-based techniques have also been developed (Lee, Lee, and Baek, 2011). Our framework considers the kidnapped robot problem as a particular loop closure where no hint is available initially. Our disambiguation strategy then solves the problem exactly like a regular loop closure where hints are available.

#### Bounded uncertainties

The whole idea of using bounded uncertainties is, as far as we know, new in the field of hybrid metrical/topological SLAMs. It was introduced for three purposes: reducing the computational and memory burden relative to probabilistic approaches (to equip smaller and simpler robots), simplifying uncertainty projection formulae and making topological correctness of the approach provable. It is expected that replacing edge vectors by linear transformations (and vector summation by transformation composition) and modeling uncertainties with covariance matrices, as in (Bosse et al., 2004), would not degrade the mapping capabilities of our framework and may even improve them but at the expense of losing provability of topological correctness. Since using bounded uncertainties is less precise than using a probabilistic formulation, it is important for the graph to be sparse as in hybrid metrical/topological approaches and not dense as in graph SLAM approaches.

#### Active disambiguation of place hypotheses

In hybrid metrical/topological SLAM litterature, disambiguation strategies consist of updating the likelihood of a discrete set of place hypotheses (Bailey, 2002; Bosse et al., 2004). A review of such strategies is given in (Bosse et al., 2004). Bosse et al. and Bailey defer the loop closure process in their respective approaches by maintaining

hypotheses until enough evidence is gathered, knowing that sometimes enough evidence will never be gathered. We can see three main weaknesses of strategies currently used in literature. First, there is no upper bound on the amount of space to physically traverse until enough evidence is gathered to confirm that a place was already known, a failure cause underlined but not solved by Bosse et al. The second weakness of current strategies is *branching*, that is the case where while disambiguating between hypotheses, new hypotheses arise. Branching is potentially infinite. Third, no strategy deferring a loop closure to a later moment is compatible with goal-directed path planning where one decision has to be taken even though there are multiple place hypotheses (*single robot position* requirement).

For these reasons, we choose an *active* approach, explicitly disallowing branching of hypotheses and actively looking for evidence that the current hypothesis is valid or not. An active strategy requires the navigating robot to be freely able to chose its path, whence the tight integration with (exploratory) path planning and navigation of our SLAM framework shown on Figure 5.3. The idea of physical navigation for disambiguation has also been developed by Kuipers et al. (2004; 1991). The extra evidence is payed for by disrupting the path plan for the whole duration of the disambiguation phase and possibly requiring more physical movements (trying to follow disambiguation paths and backtracking in case of failure) of the robot. Nevertheless, if the correct hypothesis is tested first and the disambiguation path is chosen in agreement with the path planning algorithm, navigation may not be affected at all. Our disambiguation strategy revises beliefs backwards in time, which is conjectured in (Thrun et al., 1998) to be essential for mapping of large environments.

When a loop closure hypothesis gets validated, the robot is not required to backtrack to the point where the loop closure was first detected, as opposed to (Tomatis and Nourbakhsh, 2002). Backtracking can sometimes fail due to dynamics of the environment and/or sensor noise, so avoiding backtracking when possible is beneficial. Note that backtracking errors are taken into account in our framework (see Figure 5.10), and even happen quite often on actual datasets such as that presented in chapter 6.

At last, our disambiguation strategy does not impose limits on the maximum uncertainty accumulated on a loop before encountering the ambiguous situation, contrary to Atlas (Bosse et al., 2004).

## 5.3 Bounded uncertainty projection

Within hybrid metrical/topological frameworks, it is possible to compound transformations along edges from a first vertex to a second vertex of the map, in order to find the coordinates of the second vertex in the reference frame of the first vertex. *Uncertainty*

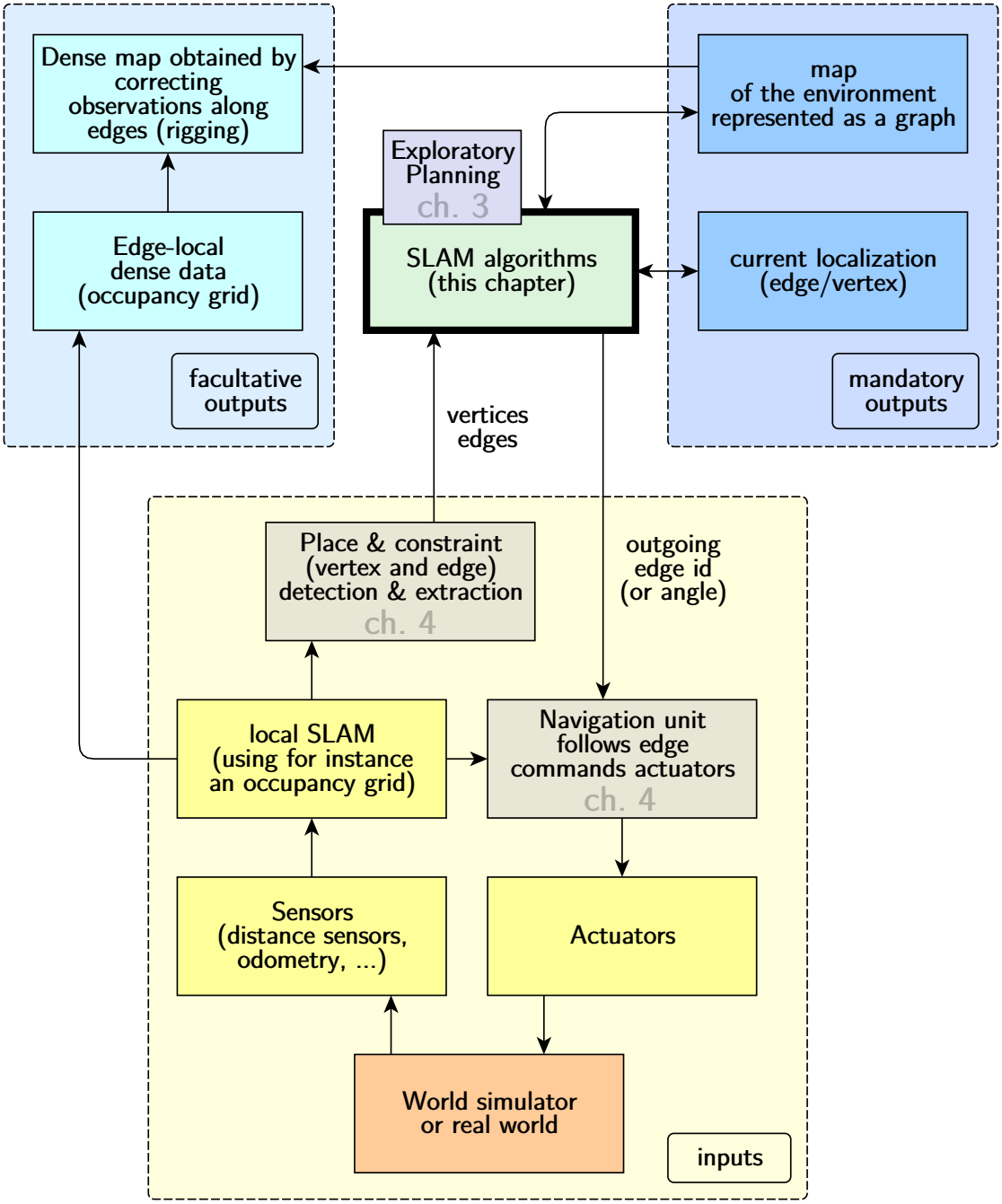



**Figure 5.3:** Our SLAM framework, its inputs and outputs. This sketch details the *SLAM* block of Figure 5.1.

*projection* consists in finding the cumulated uncertainty associated to this compound transformation. In this section, we derive uncertainty projection formulae for a single edge traversed once (equations 5.3 and 5.4), for a sequence of edges and vertices traversed once (theorem 4) and for a path made of edges and vertices traversed an arbitrary number of times (theorem 5). In the process of deriving this last projection formula, we also explain how to update edge transformations and uncertainties when new measurements are available (Figure 5.8). The uncertainty projection formulae derived in this section are used in section 5.4 to perform robust cycle detection (theorem 6).

### 5.3.1 Notations

Consider a mobile robot moving in a world $\mathcal{W}$. The robot creates a map of said world in the form of a graph $\mathcal{G}$ whose vertices $v \in \mathcal{V}$ represent places in $\mathcal{W}$ and whose edges $e \in \mathcal{E}$ represent paths or transformations from one place to another. Each vertex $v \in \mathcal{V}$, corresponding to a place at position $\mathbf{R_v}$ in $\mathcal{W}$, can be associated to its global coordinates $\tilde{\mathbf{R}_\mathbf{v}}$, computed in a post-processing step through spring-mass optimization. Each edge $e \in \mathcal{E}$ from $S(e) \in \mathcal{V}$ (for "start") to $T(e) \in \mathcal{V}$ (for "termination") is associated to the measured vector $\tilde{\mathbf{r}}_e$, where $\tilde{\mathbf{r}}_e$ is an estimate of the difference $\mathbf{r_e} = \mathbf{R_{T(e)}} - \mathbf{R_{S(e)}}$ between the positions of both its end vertices. Global vertex coordinates are used for visual output but not for SLAM, path planning or navigation. We suppose that $\mathcal{W}$ remains static while the robot is mapping it and navigating inside it. An environment is said to be static if in the absence of sensor noise, if a place is traversed twice with the same trajectory, then the sensed data describing this place both times are identical.

Each time the robot traverses an edge and a vertex, a traversal counter $t$ is increased by one unit. $V(t) \in \mathcal{V}$ is the vertex that was traversed at step $t$ when coming from $V(t-1) \in \mathcal{V}$ through edge $E(t) \in \mathcal{E}$. A real robot will need to forget old values of $V(t)$ and $E(t)$ because of limited memory. Typically, history of the trajectory is only maintained within a finite time horizon $\Delta$, so that if the current time step is $t$, only $V(i), E(i), i \geq t - \Delta$ are stored.

A single vertex $v$ is visited at timestep $t$ when $V(t) = v$. Let $Visit(v) = \{t | V(t) = v\}$. Similarly, for an edge $e$, let $Visit(e) = \{t | E(t) = e\}$.

Let $d_{min} \in \mathbb{R}^+$ be the minimum distance between any two places in $\mathcal{W}$.

### 5.3.2 Angular sensing

We assume that there is no large-scale angular odometric drift, which can be ensured using a compass in three ways compatible with each other:

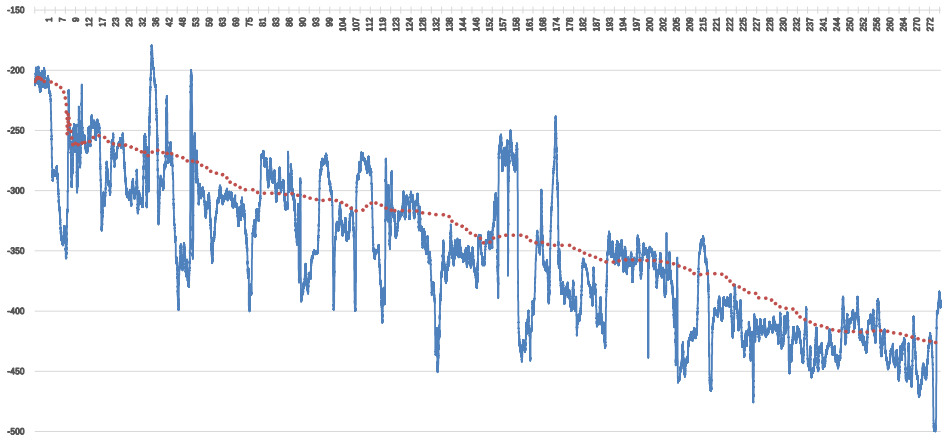



**Figure 5.4:** (plain blue line) Offset in degrees between heading deduced from wheel odometry and from compass: 49000 data points collected along a 276m trajectory using a Kobuki robot for odometry and a common MEMS magnetometer for compass. (dotted red line) Applying a sliding median (or mean) filter on the raw offset produces a smooth measurement of the angular drift of wheel odometry relative to compass, which happens to be constant around 1°/m.

- Using compass-based odometry (Duckett, Marsland, and Shapiro, 2002). Compass-based odometry can be used for outdoor environments where compass information can be considered reliable.

- Using large-scale compass-based angular drift compensation (appendix 2, section 4): while compasses are not reliable indoor at a small scale due to metallic objects and magnetic field emitters, integrating the offset between odometric heading and compass values over a few meters or dozens of meters leads to a reliable measurement of odometric drift, as shown on Figure 5.4.

- If the robot reaches the same place twice and unless there is locally a strong magnetic field gradient, the magnetic field is likely to be the same each time. Thus, even though the field itself does not give the geographic north, it still allows computing the relative angular drift accumulated between both traversals of the place. Furthermore, place signatures can be oriented relative to the field, which simplifies signature matching by removing the rotational degree of freedom.

The difference between compass-based odometry and odometry without a compass is sketched on Figure 5.5.

Nowadays, most robots are equipped with a magnetometer or a similar device, so that the "no large-scale angular odometric drift" hypothesis is realistic. There are also hints (Kimchi, Etienne, and Terkel, 2004) that some mammals such as blind mole rats use

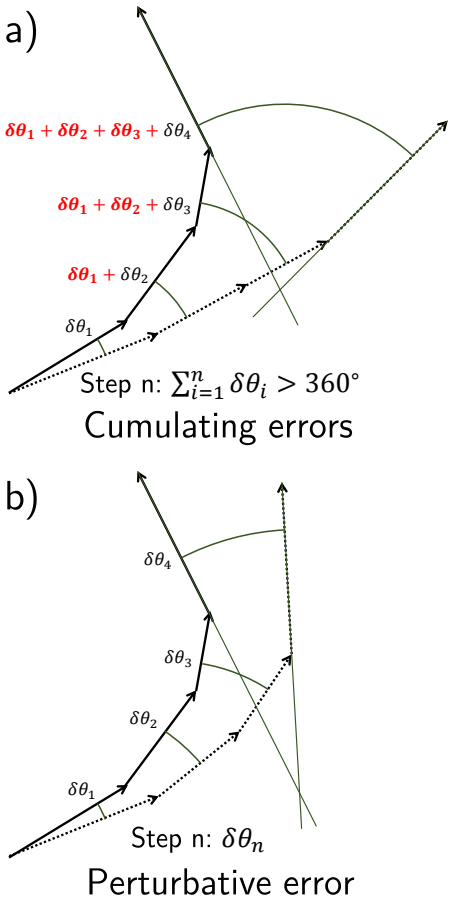



**Figure 5.5:** Odometric drift without (a) and with (b) a compass. Actual and measured trajectory are respectively represented with plain and dashed arrows, with each arrow corresponding to a measurement. Angles are always computed relative to a reference. In case (a), an angle measurement consists in estimating the rotation accumulated since the last angle measurement (taken as reference) and adding it to this last measurement, resulting in an angular drift. On the contrary, in case (b), the reference is fixed (geographical North) so that even though the angle measurement is not completely accurate, it does not drift with successive measurements. The additional angular errors in case (a) are written in bold red. After hundreds or thousands of measurements in case (a), the angular drift may exceed 360°. In case (b), the measured angle relative to geographical North remains close to its actual value, so that the vectorial error on one single odometric measurement (one of the arrows) is infinitesimal.

some kind of compass-based odometry. Without large-scale angular odometric drift, the error $\delta_{\vec{r_i}}$ on a single odometric measurement $\vec{r_i}$ verifies $\delta_{\vec{r_i}} \ll ||\vec{r_i}||$ (perturbative regime). If there was large-scale angular odometric drift, $\delta_{\vec{r_i}}$ may be of the same order of magnitude than $||\vec{r_i}||$.

Without large-scale angular odometric drift and in a 2D world, all transformations stored in edges are translations, coded with a vector in $\mathbb{R}^2$.

### 5.3.3 Bounded uncertainty model

We suppose that all inputs (sensors, commands, ...) of a robot performing SLAM are bounded. Then, the following property shows that all compound measurements obtained as continuous functions of inputs are bounded:

**Property 1** (boundedness propagation)**.** *Let $(m_1, \ldots, m_p) \in \mathbb{R}^p$ be physical quantities, of which measurements $(\widetilde{m_1}, \ldots, \widetilde{m_p}) \in \mathbb{R}^p$ have been taken. Suppose that these measurements' uncertainties are bounded: $\forall i \in 1 \ldots p, \exists \epsilon_i \in \mathbb{R}^+, ||\widetilde{m_i} - m_i|| < \epsilon_i$. Let $f$ be a continuous function defined on the compact $C = [m_1 - \epsilon_1; m_1 + \epsilon_1] \times \ldots \times [m_p - \epsilon_p; m_p + \epsilon_p]$ with values in $\mathbb{R}^q (q \in \mathbb{N}^*)$. Then, $f(C)$ is bounded.*

*Proof.* $f$ is continuous on a compact with values in the separable space $\mathbb{R}^q$, which means its image is compact, so $f(C)$ is bounded. □

We chose a vector (as center) and a single real (as radius) to represent uncertain measurements as *maximum uncertainty Euclidean balls.* [1] The radius is taken so that the uncertainty ball covers the whole support of the distribution, including bias if any. Given the lack of precision of this representation, the value of the actual quantity measured can be *anywhere inside* the ball, without any specific probability density. However and by definition, it cannot be *outside* the ball, which is the critical part of our formulation. Therefore, the following property holds:

**Property 2** (ball intersection)**.** *If a point $P$ belongs to uncertainty balls $B_1, \ldots, B_n$ returned by different measurements $m_1, \ldots, m_n$ (correlated or not), then it must belong to the intersection of these balls: $(\forall i \in 1 \ldots n, P \in B_i) \Rightarrow P \in \bigcap_{i=1}^{n} B_i$*

*Proof.* The point cannot be outside of any of the balls, so it must belong to the intersection of all balls. Formally, $P \notin \bigcap_{i=1}^{n} B_i \Rightarrow \exists i \in 1 \ldots n, P \notin B_i$ is trivial. □

[1] Directionality of uncertainty in SLAM has been questioned by Duckett et al.(2002) who found that there was not much change in their results when using circles instead of ellipses to describe covariances in their model.

We define an error estimate as consistent if it does not underestimate uncertainty of the quantity it describes. Formally:

**Definition 1** (consistent error estimates of a measurement)**.** *Let $\mathcal{M}$ be the set of all possible measurements of a physical quantity $m \in \mathbb{R}^p$ by one or multiple sensors. $\epsilon \in \mathbb{R}^+$ is said to be a consistent error estimate of the measurement of $m$ if $\forall \widetilde{m} \in \mathcal{M}, ||\widetilde{m} - m|| \leq \epsilon$, where $||\bullet||$ denotes the Euclidean norm in $\mathbb{R}^p$.*

If $\forall \widetilde{m} \in \mathcal{M}, ||\widetilde{m} - m||$ has an upper bound $s = Sup_{\widetilde{m} \in \mathcal{M}}\{||\widetilde{m} - m||\}$, then all values $\epsilon \geq s$ are consistent error estimates. Using property 1, the image $f(m_1, \ldots, m_n)$ of a continuous function $f$ of bounded measurements $m_1, \ldots, m_n$ is bounded. Thus, we can define consistent error measurements of $f$:

**Definition 2** (consistent error estimates of a function of measurements)**.** *Let $\{\mathcal{M}_i, i = 1 \ldots n\}$ be the set of all possible measurements of physical quantities $\{m_i \in \mathbb{R}, i = 1 \ldots n\}$ by one or multiple sensors. Suppose that these measurements' uncertainties are bounded: $\forall i \in 1 \ldots n, \exists \epsilon_i \in \mathbb{R}^+, \forall \widetilde{m_i} \in \mathcal{M}_i, ||\widetilde{m_i} - m_i|| < \epsilon_i$. Let $f$ be a continuous function defined on the compact $C = [m_1 - \epsilon_1; m_1 + \epsilon_1] \times \ldots \times [m_p - \epsilon_p; m_p + \epsilon_p]$ with values in $\mathbb{R}^q (q \in \mathbb{N}^*)$. Then $\epsilon_f \in \mathbb{R}^+$ is said to be a consistent error estimate of $f(m_1, \ldots, m_n)$ if $\forall (\widetilde{m_1}, \ldots, \widetilde{m_n}) \in (\mathcal{M}_1, \ldots, \mathcal{M}_n), ||f(\widetilde{m_1}, \ldots, \widetilde{m_n}) - f(m_1, \ldots, m_n)|| \leq \epsilon_f$, where $||\bullet||$ denotes the Euclidean norm in $\mathbb{R}$ and $\mathbb{R}^q$.*

### 5.3.4 Single edge traversal

First, let us study what happens on a single edge $E(n)$ (see figure 5.6). Suppose that $V(n-1) \in \mathcal{W}$ and $V(n) \in \mathcal{W}$ have precise and point-of-view independent world positions $\mathbf{R_{n-1}} = \mathbf{R_{V(n-1)}} \in \mathbb{R}^2$ and $\mathbf{R_n} = \mathbf{R_{V(n)}} \in \mathbb{R}^2$ in some referential $\mathcal{R}_0$ (this referential is not important and cancels out in calculations). We are looking for consistent error estimates of the edge vector $\mathbf{r_{n-1,n}} = \mathbf{R_n} - \mathbf{R_{n-1}}$ from $V(n-1)$ to $V(n)$.

Let $\widetilde{\mathbf{R_n}}$ be a noisy estimator of $\mathbf{R_n}$ in $\mathcal{R}_0$. $\widetilde{\mathbf{R_n}} = \mathbf{R_n} + \omega_{\mathbf{R_n}}$ where $\omega_{\mathbf{R_n}} \in \mathbb{R}^2$ is an additive noise (or error) parameter describing some absolute vertex positioning error (it will cancel out in calculations). Let $\mathbf{d_n}$ be the position of the navigating agent relative to vertex $V(n)$ in $\mathcal{W}$. The navigating agent is thus at $\mathbf{R_n} + \mathbf{d_n}$ in $\mathcal{R}_0$. A noisy estimation of $\mathbf{d_n}$ is $\widetilde{\mathbf{d_n}} = \mathbf{d_n} + \omega_{\mathbf{d_n}}$ with $\omega_{\mathbf{d_n}}$ a *pose uncertainty*. Let $\mathbf{t_n} = \mathbf{R_n} + \mathbf{d_n} - \mathbf{R_{n-1}} - \mathbf{d_{n-1}}$ be the distance traveled in $\mathcal{W}$ by the agent when moving from the vicinity of $V(n-1)$ to the vicinity of $V(n)$. Its noisy estimation is $\widetilde{\mathbf{t_n}} = \mathbf{t_n} + \omega_{\mathbf{t_n}}$ with $\omega_{\mathbf{t_n}}$ a *movement uncertainty*. $\omega_{\mathbf{R_n}}, \omega_{\mathbf{d_n}}$ and $\omega_{\mathbf{t_n}}$ follow some probability distributions not necessarily centered on $\vec{0}$. Finally, let $\widetilde{\mathbf{r}}_{n-1,n}$ be the noisy estimation of the edge's vector $\mathbf{r}_{n-1,n}$. By definition, $\widetilde{\mathbf{r}}_{n-1,n} = \widetilde{\mathbf{t_n}} + \widetilde{\mathbf{d}}_{n-1} - \widetilde{\mathbf{d}_n}$. All notations are sketched on Figure 5.6 for $n = 2$.

The definition of $t_n$ writes:

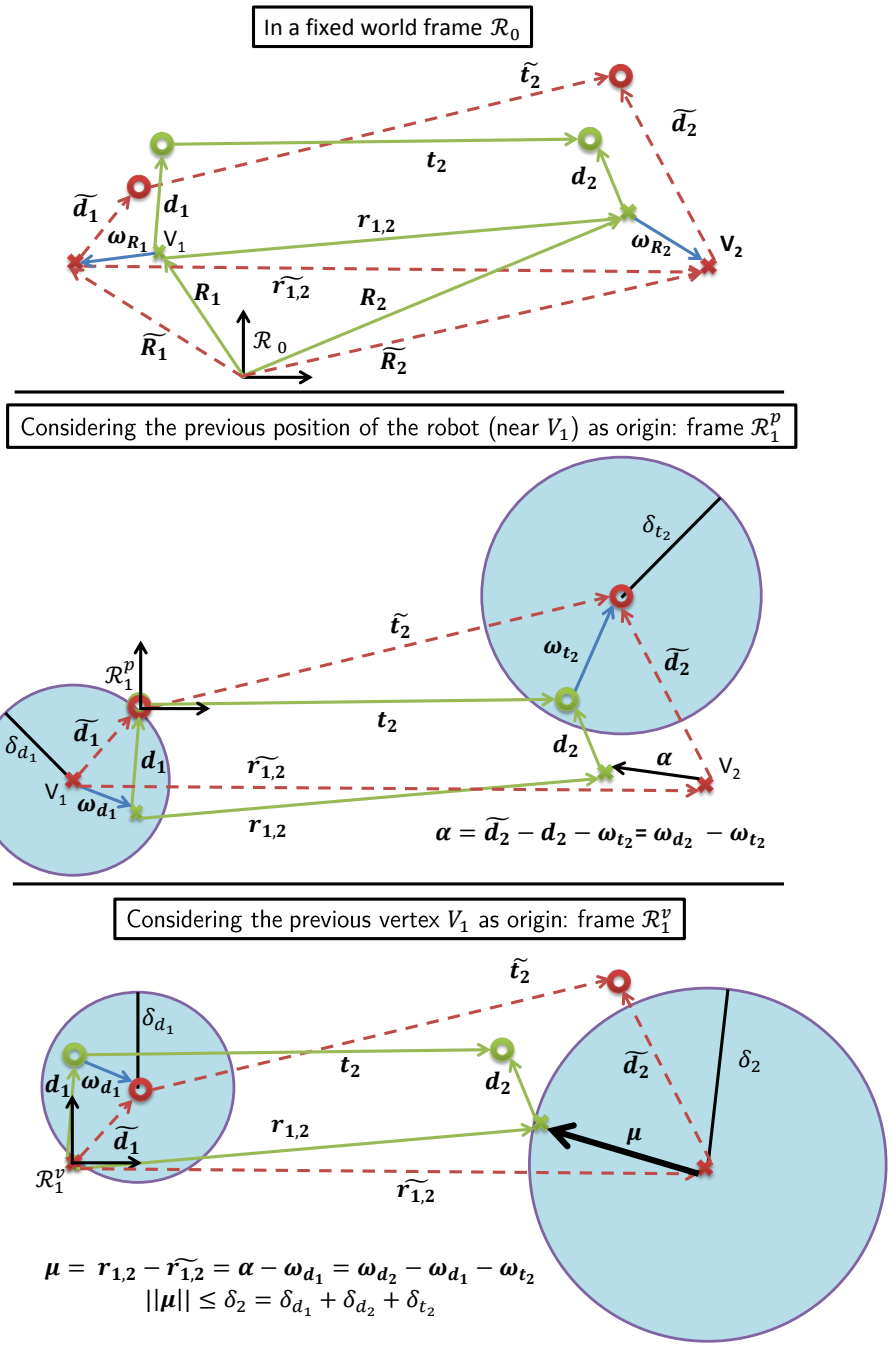



**Figure 5.6:** Uncertainty relations. Vertex positions represented with crosses, robot positions with rings and maximum uncertainties with balls. Translating the dashed red quadrilateral representing measured quantities relative to the green quadrilateral representing world positions (ground truth) allows a graphical derivation of equation 5.3.

$$\mathbf{R_n} + \mathbf{d_n} = \mathbf{R_{n-1}} + \mathbf{d_{n-1}} + \mathbf{t_n} \tag{5.1}$$

This equation expresses the fact that the current position of the agent (in the vicinity of $V(n)$) can be obtained from the previous position (in the vicinity of $V(n-1)$) and the movement from $V(n-1)$ towards $V(n)$. Replacing exact quantities by their estimates, we get:

$$\begin{aligned}\widetilde{\mathbf{R_n}} - \widetilde{\mathbf{R}}_{n-1} &= \widetilde{\mathbf{r}}_{n-1,n} \\ &\quad + (\omega_{\mathbf{R_n}} + \omega_{\mathbf{d_n}}) \\ &\quad - (\omega_{\mathbf{R_{n-1}}} + \omega_{\mathbf{d_{n-1}}}) - \omega_{\mathbf{t_n}}\end{aligned} \tag{5.2}$$

or also going back to the theoretical vertex positions, we get:

$$\begin{aligned}\mathbf{r}_{n-1,n} &= \mathbf{R_n} - \mathbf{R_{n-1}} \\ &= \widetilde{\mathbf{r}}_{n-1,n} + \omega_{\mathbf{d_n}} - \omega_{\mathbf{d_{n-1}}} - \omega_{\mathbf{t_n}}\end{aligned} \tag{5.3}$$

In equation 5.3, the difference between the measured edge $\widetilde{\mathbf{r}}_{n-1,n}$ and its theoretical value $\mathbf{r_{n-1,n}}$ is $\omega_{\mathbf{d_n}} - \omega_{\mathbf{d_{n-1}}} - \omega_{\mathbf{t_n}}$. We call $\omega_{\mathbf{d_n}}$ the *arrival error* on $V(n)$ and $\omega_{\mathbf{d_{n-1}}}$ the *departure error* on $V(n-1)$. Note that the $\omega_{\mathbf{R_\bullet}}$ terms disappeared.

If the support of the distribution of $\omega_{\mathbf{d_n}}$, $\omega_{\mathbf{d_{n-1}}}$ and $\omega_{\mathbf{t_n}}$ is bounded: $||\omega_{\mathbf{d_{n-1}}}|| \leq \delta_{d_{n-1}}$, $||\omega_{\mathbf{d_n}}|| \leq \delta_{d_n}$ and $||\omega_{\mathbf{t_n}}|| \leq \delta_{t_n}$, then:

$$||\mathbf{r}_{n-1,n} - \widetilde{\mathbf{r}}_{n-1,n}|| \leq \delta_{d_{n-1}} + \delta_{d_n} + \delta_{t_n} \tag{5.4}$$

Equation 5.4 expresses $\delta_{d_{n-1}} + \delta_{d_n} + \delta_{t_n}$ as a consistent estimate of the error on the edge vector $\widetilde{\mathbf{r}}_{n-1,n}$ relative to ground truth $\mathbf{r_{n-1,n}} = \mathbf{R_n} - \mathbf{R_{n-1}}$.

The calculation was carried in a context where it is possible to define a precise position $\mathbf{R_v}$ for each vertex $v$ but can be extended to vertices whose definition is more vague by increasing the uncertainty radius $\delta_d$ of uncertain vertices. If the vertex lies for sure within a ball of radius $\rho$ but its exact position cannot be known with more precision, then $\delta_d$ should be replaced by $\delta_d + \rho$.

### 5.3.5 Sequential traversal

Now, imagine that the navigating agent visits a vertex $V(n+1)$ after $V(n)$. We have:

$$\begin{aligned}\mathbf{r}_{n-1,n+1} =& \mathbf{R_{n+1}} - \mathbf{R_{n-1}} \\ =& \widetilde{\mathbf{r}}_{n-1,n+1} \\ &+ \omega_{\mathbf{d_{n+1}}} - \omega_{\mathbf{d_n}} \\ &+ \omega_{\mathbf{d_n}} - \omega_{\mathbf{d_{n-1}}} \\ &- \left(\omega_{\mathbf{t_{n+1}}} + \omega_{\mathbf{t_n}}\right)\end{aligned} \tag{5.5}$$

The departure error $\omega_{\mathbf{d_n}}$ cancels out the arrival error $\omega_{\mathbf{d_n}}$ in equation 5.5. This cancellation of the vertex errors on $V(n)$ is logical since the robot arrived on $V(n)$ with some error but left $V(n)$ immediately from the same position, so that the actual position $\mathbf{d_n}$ of the navigating agent relative to $V(n)$ has no impact ($\widetilde{\mathbf{r_n}}$, $\mathbf{r_n}$, $\widetilde{\mathbf{d_n}}$ and $\mathbf{d_n}$ get canceled out).

Cancellation of the error on traversed vertices $V_i$ occurs if and only if $Card(Visit(V_i)) = 1$. We say that a path has been traversed sequentially if all the vertices on this path except the departure and arrival vertices have been traversed exactly once. Figure 5.7 shows examples of sequential and non-sequential paths. Sequentially traversing a path creates favorable correlations between the $\omega_{\mathbf{d_\bullet}}$ terms which cancel out altogether.

The case of two sequential edge traversals is trivially extended to an arbitrary number of sequential edge traversals, leading to the following theorem expressing consistency of error estimates for sequential traversal:

**Theorem 4** (Sequential traversal)**.** *For the path $V(i) \to V(i+1) \to \ldots \to V(j-1) \to V(j)$ between arbitrary vertices $V(i)$ and $V(j)$ such that $\forall k \in i+1 \ldots j-1, Card(Visit(V(k)) = 1$:*

$$\begin{aligned}\mathbf{r_{i,j}} =& \mathbf{R_j} - \mathbf{R_i} \\ =& \widetilde{\mathbf{r_{i,j}}} + \left(\omega_{\mathbf{d_j}} - \omega_{\mathbf{d_i}} - \sum_{k=i+1}^{j} \omega_{\mathbf{t_k}}\right)\end{aligned} \tag{5.6}$$

*which leads to the consistent error estimate on the compound transformation $\widetilde{\mathbf{r_{i,j}}}$ on the path:*

$$||\mathbf{r_{i,j}} - \widetilde{\mathbf{r_{i,j}}}|| \leq \delta_{d_j} + \delta_{d_i} + \sum_{k=i+1}^{j} \delta_{t_k} \tag{5.7}$$

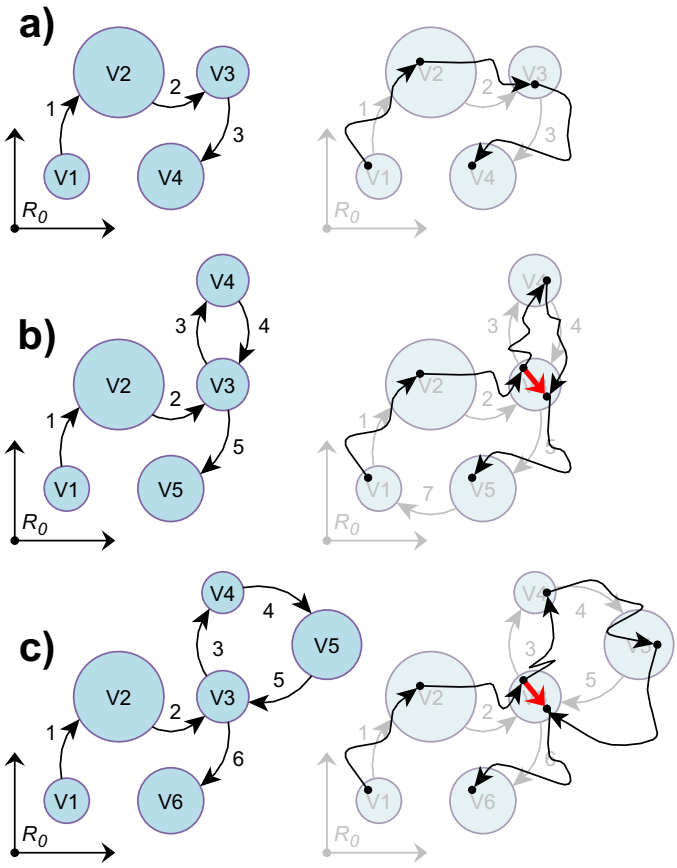



**Figure 5.7:** Sequential (a) and non-sequential (b,c) traversals supposing no movement uncertainty. Model (left) and actual trajectory (right). The radius of circles represents *pose uncertainty*. For case (b), the robot turned back after visiting $V4$. Reasoning within the context of sequential traversal for cases (b) and (c) will cause inconsistencies in uncertainty projection due to the robot detecting different positions for vertex $V3$ both time it is reached. The resulting error, due to *pose uncertainty* is represented with a bold red arrow. When pose uncertainty is taken into account during uncertainty projection, the actual trajectory 3-4-3 (b) or 3-4-5-3 (c) of the robot *can be ignored* and the uncertainties accumulated on the 3-4-3 or 3-4-5-3 loop are *replaced* by twice the *pose uncertainty* on vertex $V_3$, which is the maximum length of the bold red arrow and a consistent error estimate of the uncertainty accumulated on any loop starting and terminating on the vertex.

### 5.3.6 Non-sequential traversal

In the case of non-sequential traversal, no hypothesis can be made on whether the $\omega_{\mathbf{d_\bullet}}$ are correlated, so that a pessimistic model (ensuring consistency of error estimates) will take the hypothesis of defavorable correlation, thus preserving all $\omega_{\mathbf{d_\bullet}}$ terms. An intuitive explanation for the non-cancellation of $\omega_{\mathbf{d_\bullet}}$ terms is the following: when a path is not traversed sequentially, the robot doesn't remember *where* it was when it measured the vertex's coordinates when arriving on it and when leaving it. When the arrival errors $\omega_{\mathbf{a}}$ and departure errors $\omega_{\mathbf{l}}$ on a vertex $V$ are bounded by $\delta_{d_V}$, Figure 5.7 gives an intuitive solution to this issue: the actual trajectory of the robot can be forgotten by considering an uncertainty $2\delta_{d_V}$ when traversing $V$. More generally, we will prove in the following paragraphs that the pessimistic model preserving all $\omega_{\mathbf{d_\bullet}}$ terms leads to consistent error estimates.

The case of non-sequential traversal of a path requires considering not only one single traversal but all traversals of each edge and vertex along the path. Equation 5.3 can be written for each visit of an edge $e$ from vertex $v_1$ to vertex $v_2$ as follows:

$$\begin{aligned} &\forall n \in Visit(e), \mathbf{r_{n-1,n}} = \mathbf{R_n} - \mathbf{R_{n-1}} \\ &= \widetilde{\mathbf{r}}_{n-1,n} + \left(\omega_{\mathbf{d_n}} - \omega_{\mathbf{d_{n-1}}} - \omega_{\mathbf{t_n}}\right) \end{aligned} \tag{5.8}$$

$\mathbf{r_\bullet}$ and $\mathbf{R_\bullet}$ notations represent real world coordinates which do not depend on the measurements, so that

$$\begin{aligned} &\forall (n,p) \in Visit(e)^2, \\ &\mathbf{r_{n-1,n}} = \mathbf{r_{p-1,p}} = \mathbf{r_e}, \\ &\mathbf{R_{n-1}} = \mathbf{R_{p-1}}, \mathbf{R_n} = \mathbf{R_p} \end{aligned} \tag{5.9}$$

Writing the uncertainty projection formula of theorem 4 on a given path is not possible since each traversal yields different measurements. However, with the hypothesis of bounded uncertainty, equation 5.4 still holds for each individual traversal:

$$\begin{aligned} &\forall n \in Visit(e), \\ &||\mathbf{r_e} - \widetilde{\mathbf{r}}_{n-1,n}|| \leq \delta_n = \delta_{d_{n-1}} + \delta_{d_n} + \delta_{t_n} \end{aligned} \tag{5.10}$$

With finite memory, a fusion strategy has to be used to find per-edge (and not per-edge-traversal) values of $\widetilde{\mathbf{r}}_{n-1,n}$, $\delta_{d_{n-1}}$, $\delta_{d_n}$ and $\delta_{t_n}$ which we respectively call $\widetilde{\mathbf{r_e}}$, $\delta_{d_{eo}}$ (for edge origin), $\delta_{d_{ed}}$ (for edge destination) and $\delta_{t_e}$. These values should be updated each time the edge gets traversed. $\delta_e = \delta_{d_{eo}} + \delta_{d_{ed}} + \delta_{t_e}$ must remain a consistent error estimate (definition 2), which means:

$$||\mathbf{r_e} - \widetilde{\mathbf{r_e}}|| \leq \delta_e = \delta_{d_{eo}} + \delta_{d_{ed}} + \delta_{t_e} \tag{5.11}$$

Since $\delta_e$ is a consistent error estimate at all times, $r_e$ must lie within a ball $B_{old}(\delta_e, \widetilde{\mathbf{r_e}})$ of radius $\delta_e$ around $\widetilde{\mathbf{r_e}}$. Similarly, for each traversal at the $n^{th}$ time step, $r_e$ must lie within a ball $B_{cur}(\delta_n, \widetilde{\mathbf{r}}_{n-1,n})$ (Equation 5.10). Property 2 thus implies that $r_e \in B_{old} \cap B_{cur}$. The tightest upper bound in uncertainty respecting this constraint is represented on Figure 5.8. The drawings of the figure show how to compute new values $\delta_{new}, \widetilde{\mathbf{r}}_{new}$ of $\delta_e, \widetilde{\mathbf{r_e}}$:

$$\begin{aligned}\delta_{new} = \min\{\delta \in \mathbb{R}^+ | \exists \widetilde{\mathbf{r}}_{new} \in \mathbb{R}^2, \\ B_{old}(\delta_e, \widetilde{\mathbf{r_e}}) \cap B_{cur}(\delta_n, \widetilde{\mathbf{r}}_{n-1,n}) \subset B_{new}(\delta, \widetilde{\mathbf{r}}_{new})\}\end{aligned} \tag{5.12}$$

From equation 5.12 and as visible on Figure 5.8, $B_{old}(\delta_e, \widetilde{\mathbf{r_e}}) \cap B_{cur}(\delta_n, \widetilde{\mathbf{r}}_{n-1,n}) \neq \varnothing \Rightarrow \delta_{new} \leq \min(\delta_n, \delta_e)$. This necessary decrease of $\delta_{new}$ when error estimates are consistent can be seen as a form of metrical convergence of the map (measurement errors are progressively forgotten), even though $\delta_{new}$ may never reach 0.

Without the sequential traversal hypothesis, keeping one value of $\delta_t$ and two values of $\delta_d$ just to describe $\delta_{new}$ is not necessary: one parameter such as $\delta_t$ is sufficient. However, if the constraint extraction uncertainty and the place extraction uncertainty need to remain separated for other calculations, we suggest to update $\delta_{d_{eo}}$ through $\delta_{d_{eo}} \leftarrow \min(\delta_{d_{eo}}, \delta_{d_{n-1}})$ and $\delta_{d_{ed}}$ through $\delta_{d_{ed}} \leftarrow \min(\delta_{d_{ed}}, \delta_{d_n})$. After $\delta_{d_{eo}}$ and $\delta_{d_{ed}}$ have been updated, $\delta_{t_e}$ takes the value $\max(\delta_{new} - \delta_{d_{eo}} - \delta_{d_{ed}}, 0)$. Thus, error estimates are consistent and the non-probabilistic nature of the measurements returned by the *place extraction system* is accounted for.

Now that consistent error estimates are enforced on each individual edge (equation 5.11), we need to check that they are enforced on any path from vertex $v_1 \in \mathcal{V}$ to vertex $v_2 \in \mathcal{V}$. Let $\mathbf{R_{v_1}}$ be the position of $v_1$ ($\forall n \in Visit(v_1), \mathbf{R_{v_1}} = \mathbf{R_n}$). Let $\mathbf{R_{v_2}}$ be the position of $v_2$ ($\forall n \in Visit(v_2), \mathbf{R_{v_2}} = \mathbf{R_n}$). If the edges $e_i$ form a path from $v_1$ to $v_2$, the vector from $v_1$ to $v_2$ in $\mathcal{W}$ writes:

$$\mathbf{R_{v_2}} - \mathbf{R_{v_1}} = \Delta\mathbf{R} = \sum_{e_i} \mathbf{r_{e_i}} \tag{5.13}$$

Then, using the triangular inequality:

$$\begin{aligned}&||\Delta\mathbf{R} - \sum_{e_i} \widetilde{\mathbf{r_{e_i}}}|| \\ =&||\sum_{e_i} \mathbf{r_{e_i}} - \sum_{e_i} \widetilde{\mathbf{r_{e_i}}}|| \\ \leq&\sum_{e_i} ||\mathbf{r_{e_i}} - \widetilde{\mathbf{r_{e_i}}}||\end{aligned} \tag{5.14}$$

Using equation 5.11 which gives a consistent error estimate for each individual edge, we can derive the following theorem expressing consistency of the error estimates on a path without the sequential traversal hypothesis:

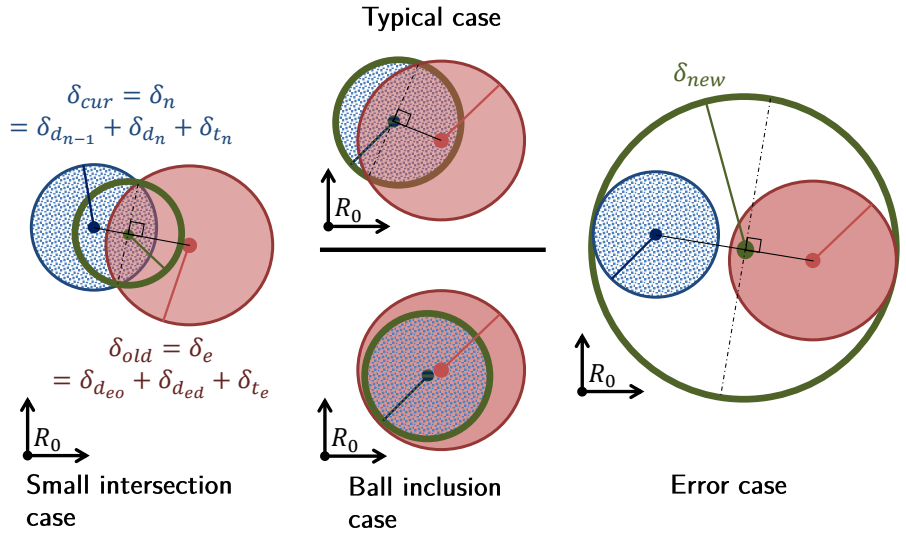



**Figure 5.8:** Uncertainty update when traversing an edge for the $n^{th}$ time. Old uncertainty ball $B_{old}$ (plain, red), newly measured ball $B_{cur}$ (textured, blue). The new uncertainty ball $B_{new}$ (bold green circle) is the smallest ball containing the intersection of $B_{old}$ and $B_{cur}$. The error case (right) represents a situation that could occur if at least one error estimate was not consistent.

**Theorem 5** (Non-sequential traversal)**.** *If the edges $e_i$ form a path from $v_1$ to $v_2$, then*

$$||(\mathbf{R_{v_2}} - \mathbf{R_{v_1}}) - \sum_{e_i} \widetilde{\mathbf{r}_{\mathbf{e_i}}}|| \leq \sum_{e_i} \delta_{e_i} \tag{5.15}$$

The intuitive idea that loops can be ignored when projecting uncertainty along a path is expressed in the following corollary:

**Corollary 2** (ignoring loops)**.** *In theorem 5, the tightest consistent error estimates are always obtained by considering loop-free paths.*

*Proof.* Let $\mathcal{P}$ be the (possibly infinite) set of paths from $v_a \in \mathcal{V}$ to $v_b \in \mathcal{V}$. Let $P_1 \in \mathcal{P}$ be one specific path which may contain loops. Let $P_2 \in \mathcal{P}$ be the loop-free path obtained by removing all loops from $P_1$. For instance, on Figure 5.7, $v_a = V1, v_b = V6, P_1 = V1 - V2 - V3 - V4 - V5 - V3 - V6$ and $P_2 = V1 - V2 - V3 - V6$. Then, since $\forall e_i \in \mathcal{E}, \delta_{e_i} \geq 0$ and $e_i \in P_2 \Rightarrow e_i \in P_1$, $\sum_{e_i \in P_2} \delta_{e_i} \leq \sum_{e_i \in P_1} \delta_{e_i}$.

□

## 5.4 Loop closure

In the previous section, we derived two uncertainty projection formulae (theorems 4 and 5) to project pose and movement uncertainty from one vertex of the graph to another along a path. In this section, we develop an approach for robust *loop closure* based on *uncertainty projection* and *active disambiguation of place hypotheses*. With well chosen parameters (subsection 5.4.4), our approach is guaranteed (theorem 7) to handle *structural ambiguity* so that the produced map will be topologically correct. Within this section, labels ([$l$]) refer to Figure 5.10 which describes how loop closure is implemented.

### 5.4.1 Generation and pruning of loop closure hypotheses

Each time the robot explores an edge and discovers a new vertex $v$ ([0]), the map is scanned for loop closure candidates $v_i \in \mathcal{V}$ having the same number of outgoing edges as $v$. Indeed, it is possible that $\exists i | v_i = v$. In the worst case, all vertices on the map may be loop closure candidates. Similarly to (Bosse et al., 2004), we use Dijkstra's algorithm to project uncertainty and edge vectors from each $v_i$ to $v$ along the path of lowest uncertainty ([1]). Some invalid loop closure candidates $v_i | v_i \neq v$ are then pruned using the following theorem checking whether summed vectors and projected uncertainty between $v$ and $v_i$ are compatible:

**Theorem 6** (necessary condition for loop closure)**.** *Let $(v_1, v_2) \in \mathcal{V}^2$ be two vertices of the map. Let $\mathcal{P}$ be the set of all paths from $v_1$ to $v_2$ on the graph, and $\delta_i$ be the uncertainty accumulated on the edges forming one of these paths $P_i \in \mathcal{P}$ according to theorems 4 or 5. Let $\Delta\mathbf{R_i}$ be the sum of vectors along $P_i$. Then, $(\exists P_i \in \mathcal{P} | d_i = ||\Delta\mathbf{R_i}|| > \delta_i) \Rightarrow \mathbf{R_{v_1}} \neq \mathbf{R_{v_2}}$.*

*Proof.* The proposition to prove is strictly equivalent to $\mathbf{R_{v_1}} = \mathbf{R_{v_2}} \Rightarrow \forall P_i \in \mathcal{P}, d_i \leq \delta_i$. If $\mathbf{R_{v_1}} = \mathbf{R_{v_2}}$, theorems 4 and 5 both simplify to $d_i \leq \delta_i$. It is easy to know which paths or parts of paths were traversed sequentially (and thus whether theorem 4 or 5 should be used for each path or part of path) because all vertices on sequential parts have been traversed exactly once, and parts containing only vertices traversed exactly once are necessarily sequential. □

Once invalid candidates have been removed, the set of remaining candidates is further pruned using the minimum distance between two vertices, $d_{min}$: if $d_{min} \geq d_i + \delta_i$ then $v = v_i$. Even with this refinement sketched on Figure 5.9, pruning is not sufficient to perform robust loop closure. Indeed:

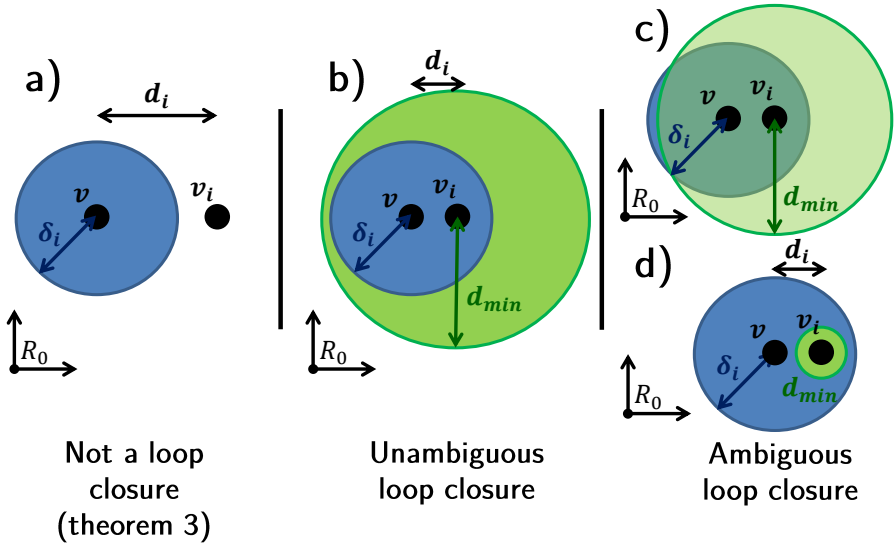



**Figure 5.9:** Four possible cases when looking for loop closure candidates. (a): $d_i > \delta_i$, (b): $d_{min} \geq d_i + \delta_i$ and two ambiguous cases (c), (d). In cases (b), (c) and (d), loop closure candidates are valid according to theorem 6.

- Pruning is done according to only one path: that of smallest uncertainty. It should be done using all paths.
- The pruning condition expressed in theorem 6 is necessary but not sufficient.
- $d_{min} \geq d_i + \delta_i$ is not verified in most cases.

Consequently, a *disambiguation strategy* is required in order to ascertain whether $v = v_i$ for each remaining candidate $v_i$ ([2], [3]). When the *kidnapped robot problem* has to be solved, the uncertainty projection phase is skipped (uncertainty is considered infinite) and the robot starts the *disambiguation strategy* directly.

### 5.4.2 Disambiguation of hypotheses

A disambiguation strategy consists in updating the likelihoods of a set of loop closure hypotheses according to evidence gathered during traversal of the environment. In this subsection, we explain how the robot actively looks for evidence, what this evidence is and how it is collected and memorized.

### An active disambiguation strategy

We implement disambiguation by having the robot choose one hypothesis $v = v_i$ and try to follow a loop-free path which it would be able to follow if the hypothesis was correct. On this path (which we term *disambiguation path*), evidence is gathered (see subsubsection 5.4.2 for details). Figure 5.11 gives an example of disambiguation paths. If the hypothesis is correct, there should not be any loops on the path, so if a loop is found, then the hypothesis is incorrect. Thus, branching of hypotheses (generation of new hypotheses during disambiguation) is impossible. The disambiguation strategy ends when enough evidence has been gathered to validate one hypothesis ([6]) or discard all hypotheses ([10]). If a hypothesis is validated ([6]), the map is corrected accordingly and navigation continues normally (without the need for backtracking to the vertex where ambiguity was initially detected). On the contrary, if the hypothesis is invalidated ([9]), the robot backtracks ([11]) to the ambiguous vertex and choses another hypothesis. If there are no hypotheses left ([10]), the current location has never been reached before and a new vertex is created. While following the disambiguation path of one hypothesis, the robot may test the compatibility of other hypotheses ([7]) with incoming evidence and discard ([8]) or accept ([6]) them if necessary.

Computation of a disambiguation path ([4]) can be done with a simple graph traversal or planning algorithm such as Dijkstra, as done on Figure 5.11. However, if the robot is navigating towards a specific goal, the disambiguation path may be chosen to bring the robot closer to the goal, explaining the arrow linking the "find disambiguation path" ([4]) and the "graph planning algorithm" ([12]) blocks of Figure 5.10. The number of edges in the longest disambiguation path in a specific environment corresponds to the time horizon $\Delta$ after which the actual trajectory of the robot gets forgotten. Koenig et al. (2006) provide an efficient method to choose disambiguation paths in order to solve the *kidnapped robot problem*. This method can also be used to choose disambiguation paths for a regular loop closure.

Not being able to backtrack is an error which can occur in a dynamic environment (a door has just been closed behind the robot) or when one of the edges on the disambiguation path is one way only (the disambiguation path determination algorithm tries as much as possible to select only two-way edges). If such an error occurs, vertices found since the first ambiguous vertex are written down on the map ([13]) (possibly duplicating vertices which may be fused together later) and navigation continues normally.

During the first few steps of topological mapping after the robot has been switched on, the map may not contain enough vertices and edges to find a long enough disambiguation path. In this border case, once the end of the disambiguation path has been reached, the limited evidence available is used to guess whether the loop closure hypothesis is valid or not. If an erroneous guess is made, the resulting error may be corrected during a later traversal.

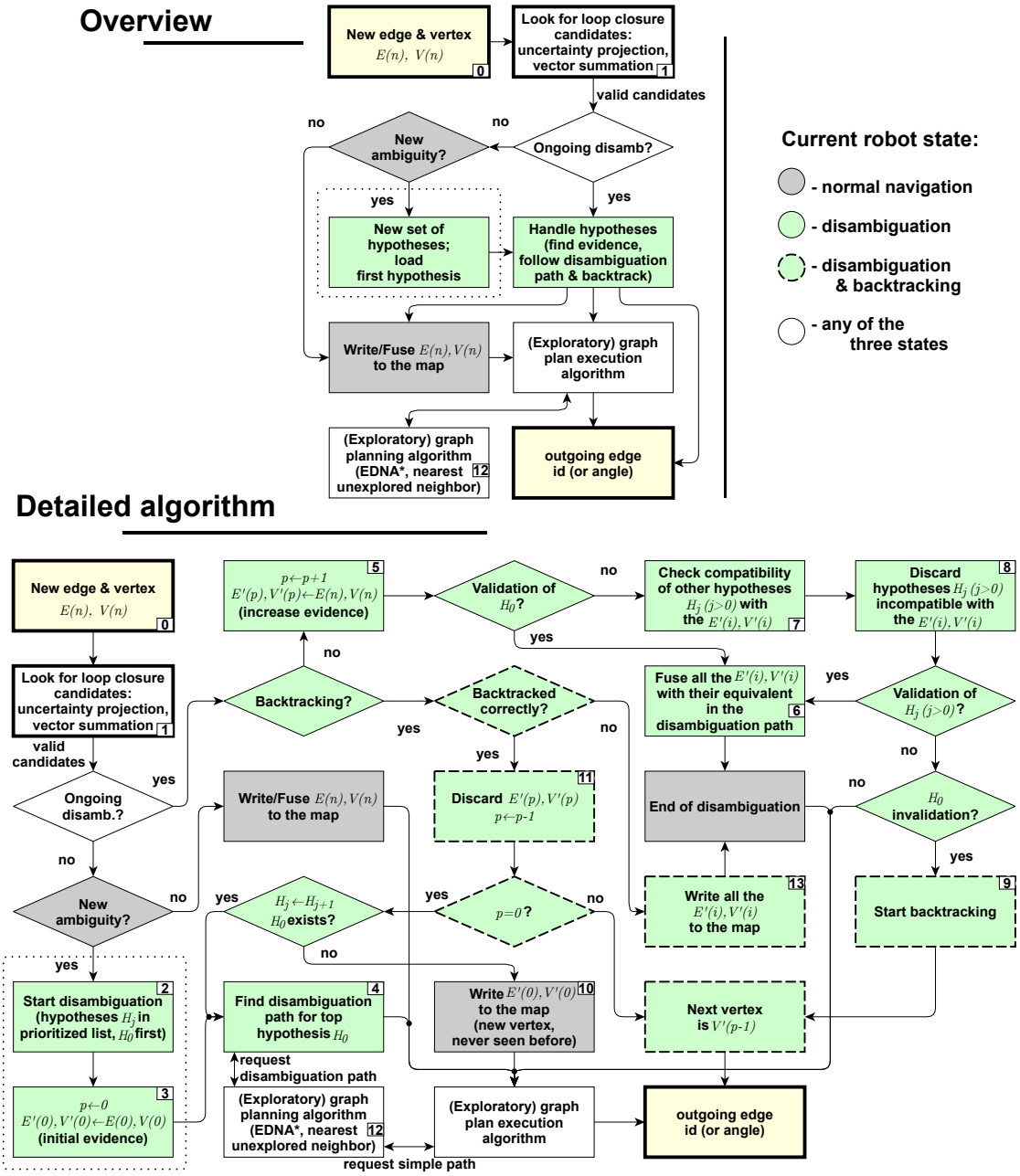



**Figure 5.10:** Implementation of hypothesis handling. Hypotheses handling decides whether or not a new loop was found in the environment and updates the map accordingly. As the loop closure hypothesis may be ambiguous, a disambiguation strategy (in green) may be triggered to try to find evidence (in)validating a hypothesis.

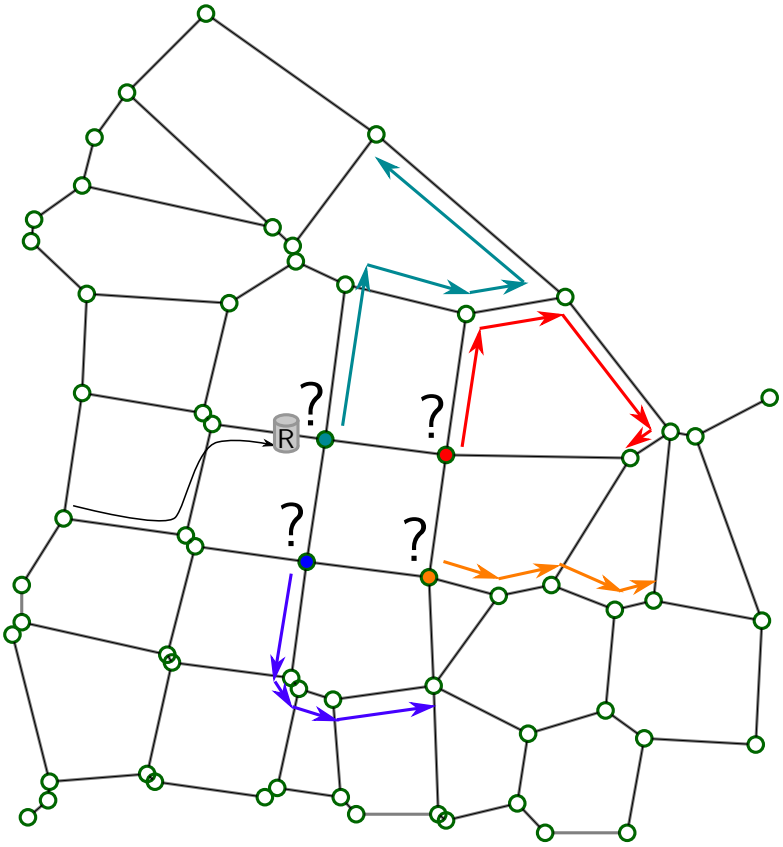



**Figure 5.11:** Example of disambiguation paths. The robot has four place hypotheses and has to use active disambiguation. A disambiguation path for each hypothesis is marked with arrows. It may not be necessary to travel along all four paths since a hypothesis may be eliminated ([8]) or accepted ([6]) while disambiguating another hypothesis. In this thesis, disambiguation paths are chosen as random loop-free paths containing a fixed number of edges (here, 4). This strategy does not necessarily discriminate hypotheses optimally.

A highly ambiguous environment like a regular grid may require more traversals for disambiguation. On the contrary, if very accurate information about a vertex $v_i$ is available (such as a high quality snapshot taken during a previous traversal), the disambiguation procedure is facilitated.

### Accumulating evidence

While traversing the environment during disambiguation, evidence is accumulated ([5]) on whether the loop closure hypothesis is correct or not. In general, the following pieces of evidence can be used:

- local topology (Kuipers et al., 2004), such as the number and angles of outgoing edges on each vertex reached (if a vertex is reached with a topology different than forecast, the hypothesis is immediately invalidated),
- edge metric, that is checking whether maximum uncertainty balls corresponding to two edges with the same supposed origin do intersect (not intersecting balls cause the hypothesis to be immediately invalidated thanks to the *consistent error model* developed in section 5.3),
- vertex signatures, such as 360°snapshots, lists of nearby features or sonar scans (Duckett, Marsland, and Shapiro, 2002),
- edge signatures, such as snapshots, occupancy grids or a list of features seen while traversing the edge and
- global positioning such as a GPS position or triangulation of wireless beacons.

Evidence is stored as a log-odd $l = log(\frac{p}{1-p})$ with $p$ the probability that the hypothesis is correct. Log-odds are a preferred representation in problems where neither validation nor invalidation of a hypothesis is privileged, such as in occupancy grid approaches (Elfes, 1987; Thrun, 2001). They allow easy integration of new evidence by summation. Let 1 denote the fact that the hypothesis is correct and 0 denote the fact that the hypothesis is incorrect. In our framework, new evidence related to measurement $m_i$ is described by a probability $P(m_i|1)$ and $P(m_i|0)$.

A simple calculation analogous to that carried by Thrun et al. (2001) for occupancy grids leads to:

$$
\begin{aligned}
l &= log(\frac{P(1|m_1,\ldots,m_n)}{P(0|m_1,\ldots,m_n)}) \\
&= \sum_{i=1}^{n} log(\frac{P(m_i|m_1,\ldots,m_{i-1},1)}{P(m_i|m_1,\ldots,m_{i-1},0)})
\end{aligned} \tag{5.16}
$$

where the priors $P(0)$ and $P(1)$ have been set to $1/2$ since there is no reason to privilege a hypothesis or another initially.

A hypothesis is validated as soon as $P(1|m_1,\ldots,m_n) = \frac{1}{e^{-l}+1}$ exceeds a certain threshold $p_h$. It is invalidated if $P(1|m_1,\ldots,m_n) < p_l$. Perfect hypothesis disambiguation requires two criteria to be met: first, the high threshold $p_h$ must never be underestimated, otherwise an incorrect hypothesis could be validated, thus discarding the correct hypothesis. Second, the low threshold $p_l$ must never be overestimated, otherwise a correct hypothesis could be discarded.

Equation 5.16 can be simplified supposing that measurements are conditionally independent given the hypothesis: $\forall i \leq n, P(m_i|m1,\ldots,m_{i-1},1) = P(m_i|1)$. This hypothesis is similar to the *static world assumption* of Thrun (2001) which states that "[measurements] are conditionally independent given knowledge of the map" and is valid in a static environment with a non-changing map (Thrun, 2001). The simplified log-odd equation writes:

$$
\begin{aligned}
l^s &= log(\frac{P(1|m_1,\ldots,m_n)}{P(0|m_1,\ldots,m_n)}) \\
&= \sum_{i=1}^{n} log(\frac{P(m_i|1)}{1-P(m_i|1)})
\end{aligned} \tag{5.17}
$$

where the "s" stands for "simplified".

In order to compensate the effects of this last formula which may be incorrect, especially in non-static environments, we overestimate $p_h^s$ and underestimate $p_l^s$. In a given (finite) environment, there always exist $p_l^s$ and $p_h^s$, so that $\frac{1}{e^{-l^s}+1} < p_l^s \Rightarrow \frac{1}{e^{-l}+1} < p_l$ and $\frac{1}{e^{-l^s}+1} > p_h^s \Rightarrow \frac{1}{e^{-l}+1} > p_h$.

### 5.4.3 Theoretical validation of topological correctness

Loop closure errors in a topological context were classified by Dudek et al. (1993) in three categories:

- *Type 1: old looks new* (not recognizing an existing vertex),

- *Type 2: mis-correspondence* (recognizing the current vertex as being the already visited vertex $v_j$ even though it is the already visited vertex $v_i$) and

- *Type 3: new looks old* (thinking that the current vertex is already on the map when it is not).

Using theorem 6 and the active disambiguation strategy, we can find conditions under which our framework is topologically correct (absence of loop closure errors). These are expressed in the following theorem:

**Theorem 7** (localization provability)**.** *The SLAM framework described in this article is topologically correct provided that:*

1. *$\mathcal{W}$ can be abstracted as a static directed graph of places and paths, with the robot moving along paths from one place to another,*

2. *none of the uncertainties $\delta_t$ and $\delta_d$ were underestimated (they are consistent error estimates),*

3. *$d_{min}$ was underestimated,*

4. *$p_h$ was overestimated and $p_l$ was underestimated and*

5. *the map allows computing long enough disambiguation paths, where $p_h$ and $p_l$ can be reached in finite time for each loop closure hypothesis,*

*Proof.* Suppose that our framework makes a mistake by thinking the robot is somewhere on the graph while it is actually elsewhere (wrong loop closure). Then, the topology of the graph does not match that of the environment the robot is trying to navigate in. In order for the robot to make a mistake, it is necessary that either

- a loop closure was not detected (Type 1), or

- a loop closure was incorrectly detected (Type 2 or 3).

Type 1 errors: It is impossible for a loop closure not to be detected since the uncertainty projection model always overestimates uncertainty (error estimates are consistent in theorems 4 and 5) and the threshold $p_l$ always underestimates hypotheses' plausibility.

Type 2 and type 3 errors: If a loop was incorrectly detected, then an incorrect hypothesis has been validated because either $d_{min}$ is too high (see Figure 5.9, case (b)), or because the validation threshold $p_h$ was reached. If $d_{min}$ is underestimated and $p_h$ is overestimated, no incorrect hypothesis can be validated.

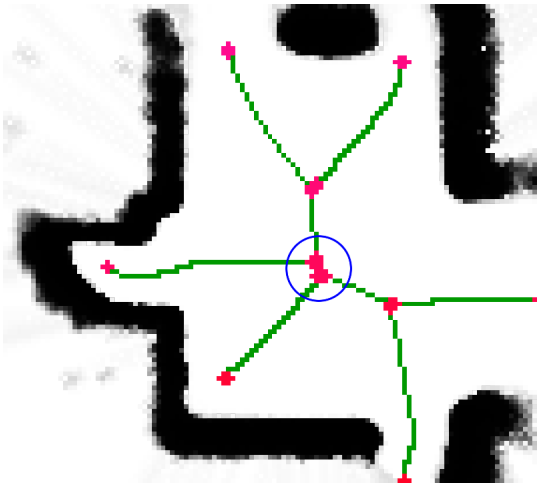


**Figure 5.12:** Vertex positioning inaccuracy (*pose uncertainty*) when drawing the Generalized Voronoï Graph on an occupancy grid (occupation probability in shades of grey, edges in green, vertices in red): which position should be given to the center vertex?

Finally, if $\mathcal{W}$ can be abstracted as a static directed graph, backtracking when a hypothesis is invalidated always succeeds and traversal of already known places during normal navigation or disambiguation always succeeds.

Thus, under the hypotheses, the loop closure process always succeeds, and the robot never gets lost. □

Hypothesis (1) of theorem 7 may seem unrealistic. It can however be replaced by the following three constraints:

1. the world $\mathcal{W}$ is static,

2. places of of $\mathcal{W}$ and their topological characteristics (number of outgoing paths) are detected identically each time the place is traversed ("perfect vertex extraction") and

3. navigation along a path from one place to another always succeeds.

### 5.4.4 Parameters of our approach

Our model is driven by only five parameters or families of parameters: $d_{min}$ to facilitate pruning of loop closure hypotheses, $p_h$ and $p_l$ for evidence acquisition and $\delta_t$ and $\delta_d$ (arrival and departure) for each edge. We can devise a few guidelines on how to choose these parameters:

- $d_{min}$ is an intrinsic property of the place extraction system whose meaning is: "how close can two places be". For instance, robustly detecting intersections using the GVG computed from an occupancy grid (see chapter 4) requires a clustering step, so that vertices cannot be closer than twice the clustering radius (four pixels on Figure 5.12). If unsure, $d_{min}$ can be set to zero, which does not prevent the approach from being topologically correct but increases the amount of ambiguous loop closures (see Figure 5.9). However, $d_{min}$ should never be overestimated according to theorem 7.

- $\delta_t$ and $\delta_d$ are properties of the sensors and algorithms used to perform constraint and place extraction respectively. For instance, if movements are computed from odometry alone, then $\delta_{t_e}$ is the maximum odometry error associated to the traversal of edge $e$. $\delta_t$ and $\delta_d$ must be consistent error estimates in order for the produced map to be topologically correct according to theorem 7. $\delta_t$ and $\delta_d$ can be set after running field experiments with actual sensors in real environments.

- Finally, the two probabilistic thresholds $p_l$ and $p_h$ are application-, dataset- and sensor-dependent. We see these thresholds as a conceptual guarantee given to the robot by its user: "As long as you wait until these thresholds are reached, I can guarantee that you will not get lost in the environment you are currently traversing". The probability densities related to individual measurements as well as values for the thresholds can be learned by supervised or automatic methods. A rich representation such as a high quality RGB-D snapshot is likely to provide much better disambiguation than a sonar reading. As setting $p_l$ and $p_h$ is somewhat ad-hoc, we carry experiments with more or less ad-hoc values and show that topologically correct SLAM is still achievable. Chapter 6, subsection 6.2.1 shows a simplified case where $p_h$ and $p_l$ are directly related to the number of edges to be traversed to achieve disambiguation.

## 5.5 Building a global map

The map constructed by our SLAM framework is made of a graph where only relative positioning, described by edge measurements, is relevant for SLAM, planning and navigation. However, in order to interact with humans, a global map within a single

reference frame may be beneficial, if not required. In addition, global coordinates are required (see chapter 7) to decrease the time complexity of our SLAM framework from $\mathcal{O}(N.log(N))$ to $\mathcal{O}(log(N))$ with $N = Card(\mathcal{V})$ the number of vertices on the map at a given time.

### 5.5.1 Spring-mass optimization: principle

We use spring-mass optimization in order to get a global map within a single reference frame from the graph of relative coordinate frames obtained by SLAM. The idea of spring-mass optimization was introduced by Lu and Millios (1997) and later referred to as *back-end* in the context of graph SLAMs. Spring-mass optimization is notably used in (Duckett, Marsland, and Shapiro, 2000; Duckett, Marsland, and Shapiro, 2002; Filliat, 2001; Golfarelli, Maio, and Rizzi, 1998) and (Lu and Milios, 1997; Thrun and Montemerlo, 2006) with spring constants replaced by concentration matrices. The graph is considered as an out-of-equilibrium network of springs and masses. Springs are associated to edges and the relaxed length of a spring is the associated edge's (measured) vector. Masses are associated to vertices. Spring-mass optimization sets or updates the global coordinates of vertices in order to minimize the energy $\mathcal{H}$ defined as follows:

$$\begin{aligned}\mathcal{H} &= \sum_{e\in\mathcal{E}} \frac{1}{2} K_e ||(T(e) - S(e)) - e||^2 \\ &= \sum_{v_i\in\mathcal{V}} \sum_{v_j\in\mathcal{V}} \sum_{\substack{e\in\mathcal{E} \\ S(e)=v_i \\ T(e)=v_j}} \frac{1}{2} K_e ||(v_j - v_i) - e||^2 \end{aligned} \tag{5.18}$$

where each vertex is identified with its global coordinates and the $K_e$ are stiffness constants which can be set to 1 for instance. $S(e)$ and $T(e)$ are respectively the origin and destination of edge $e$, as defined in section 5.3. Optimal vertex coordinates obtained after spring-mass optimization are denoted with a star $(O^*(e), D^*(e), \ldots)$. Spring-mass optimization does not change edge vectors. Thus, spring-mass optimization does neither affect SLAM, nor planning or navigation. Spring-Mass optimization however reduces *global metrical uncertainty* in the sense of Kuipers (2004). An example of spring-mass optimization is displayed on Figure 5.13.

Note that spring-mass optimization corrects vertex positions found through simple odometric integration (for each edge $e$, $T(e) = S(e) + e$). Thus, comparing the relative positions $T^*(e) - S^*(e)$ of vertices after spring-mass optimization to the edge $e$ is a way to find (and correct) biases in the odometric system.

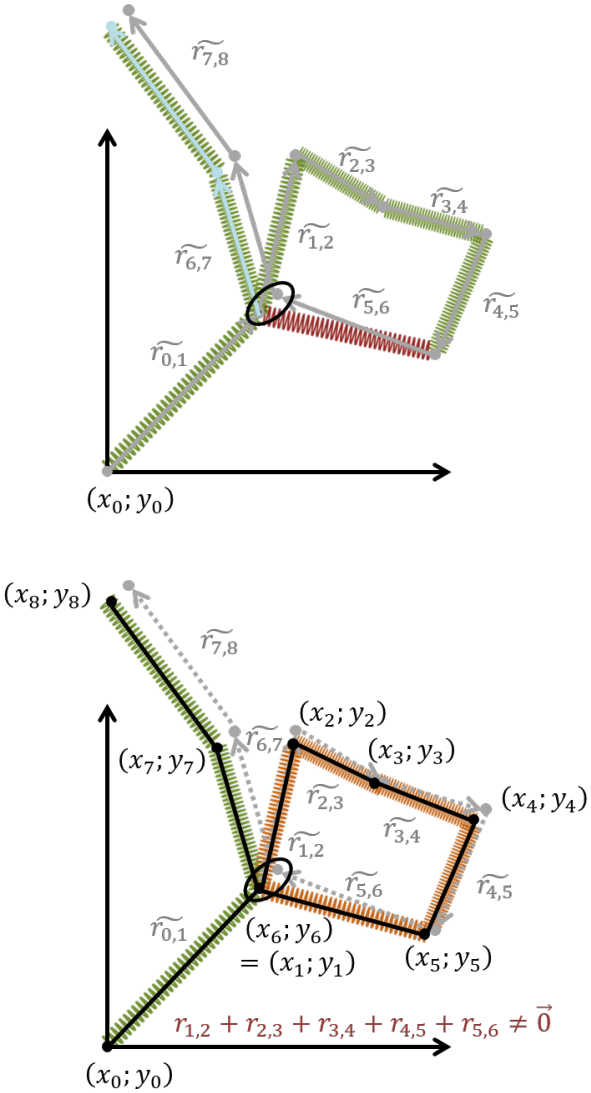



**Figure 5.13:** Spring-mass optimization deducing the most likely vertex positions from measured edge vectors. Edge vectors in gray, final position of vertices in black. (Top) Originally, the red spring contains the energy added by the loop closure constraint. (Bottom) After spring-mass optimization, this energy is spread amongst multiple springs shown in orange.

### 5.5.2 Solving the spring-mass equation

Equation 5.18 is quadratic, so that finding a set of positions $\{v_i^* \in \mathcal{V}\}$ which minimizes $\mathcal{H}$ comes down to solving a linear equation system:

$$\forall u \in \mathcal{V},$$
$$\sum_{v \in \mathcal{V}} \sum_{\substack{e \in \mathcal{E} \\ S(e)=u \\ T(e)=v}} K_e((v-u)-e) + \sum_{\substack{e \in \mathcal{E} \\ S(e)=v \\ T(e)=u}} K_e((v-u)+e) = 0 \tag{5.19}$$

Equation 5.18 only describes differences between the position of adjacent vertices of the graph. Consequently, it is invariant to a global translation of all vertices. The underlying phenomenon is probably the same that causes the absolute uncertainty not to decrease in probabilistic SLAM while the relative uncertainty does, as noted by Durrant-Whyte et al. (1996). It is however possible to get a unique solution by fixing an absolute position. For instance, $v_1 = 0$ for some vertex $v_1 \in \mathcal{V}$. Alternatively, $\sum_{v \in \mathcal{V}} v = 0$ sets the coordinates of the isobarycenter of all vertices to 0. Both conditions are mutually exclusive, and each one can be implemented using Lagrange multipliers.

For $v_1 = 0$, the equation set is:

$$\forall u \neq v_1 \in \mathcal{V},$$
$$\sum_{v \neq v_1 \in \mathcal{V}} \sum_{\substack{e \in \mathcal{E} \\ S(e)=u \\ T(e)=v}} K_e((v-u)-e) + \sum_{\substack{e \in \mathcal{E} \\ S(e)=v \\ T(e)=u}} K_e((v-u)+e) = 0 \tag{5.20}$$
$$\sum_{\substack{e \in \mathcal{E} \\ S(e)=u \\ T(e)=v_1}} K_e((v_1-u)-e) + \sum_{\substack{e \in \mathcal{E} \\ S(e)=v_1 \\ T(e)=u}} K_e((v_1-u)+e) + \lambda = 0 \tag{5.21}$$
$$\sum_{v \neq v_1 \in \mathcal{V}} \sum_{\substack{e \in \mathcal{E} \\ S(e)=v_1 \\ T(e)=v}} K_e((v-v_1)-e) + \sum_{\substack{e \in \mathcal{E} \\ S(e)=v \\ T(e)=v_1}} K_e((v-v_1)+e) + \lambda = 0 \tag{5.22}$$
$$v_1 = 0 \tag{5.23}$$

For $\sum_{v\in\mathcal{V}} v = 0$, the equation set is:

$$\forall u \in \mathcal{V},$$

$$\sum_{v\in\mathcal{V}} \sum_{\substack{e\in\mathcal{E}\\S(e)=u\\T(e)=v}} K_e((v-u)-e) + \sum_{\substack{e\in\mathcal{E}\\S(e)=v\\T(e)=u}} K_e((v-u)+e) + \lambda = 0 \tag{5.24}$$

$$\sum_{v\in\mathcal{V}} v = 0 \tag{5.25}$$

In both cases, solving the equation system comes down to finding the solution of a linear system with $n = 2(Card(\mathcal{V}) + 1)$ unknowns. This can be done for instance using $LU$ decomposition, running in $\mathcal{O}(\frac{2}{3}n^3)$ (Trefethen and David, 1997). Since there are only a few edges on each vertex, the matrix describing the system is sparse, so that a smart LU solver including a permutation matrix may require much less than $\mathcal{O}(\frac{2}{3}n^3)$ running time.

### 5.5.3 Our implementation

We chose not to solve the equation system directly. Instead, we use relaxation as defined in (Duckett, Marsland, and Shapiro, 2000) to solve iteratively and locally. Relaxation consists for each vertex $u \in \mathcal{V}$ in minimizing the local energy:

$$\mathcal{H}_u = \sum_{v\in\mathcal{V}} \sum_{\substack{e\in\mathcal{E}\\S(e)=u\\T(e)=v}} \frac{1}{2}K_e||(v-u)-e||^2 + \sum_{\substack{e\in\mathcal{E}\\S(e)=v\\T(e)=u}} \frac{1}{2}K_e||(v-u)+e||^2 \tag{5.26}$$

which comes down to finding the coordinates of $u$ that verify:

$$\sum_{v\in\mathcal{V}} \sum_{\substack{e\in\mathcal{E}\\S(e)=u\\T(e)=v}} K_e((v-u)-e) + \sum_{\substack{e\in\mathcal{E}\\S(e)=v\\T(e)=u}} K_e((v-u)+e) = 0 \tag{5.27}$$

that is:

$$u = \frac{\sum_{\substack{e\in\mathcal{E}\\S(e)=u\\T(e)=v}} K_e(v-e) + \sum_{\substack{e\in\mathcal{E}\\S(e)=v\\T(e)=u}} K_e(v+e)}{\sum_{\substack{e\in\mathcal{E}\\S(e)=u,T(e)=v\\or\ S(e)=v,T(e)=u}} K_e} \tag{5.28}$$

considering all vertices but $u$ as fixed. Or as Duckett (2000) puts it, "pick each vertex in turn, and move it to where its neighbors think it should be".

Vertices $u$ are considered one after another (the process can be parallelized), and each vertex may be updated multiple times. It is also possible to choose which vertices should be updated randomly. Proof of convergence of the relaxation algorithm is given in (Duckett, Marsland, and Shapiro, 2000).

The global translation invariance of the equations is taken care of either by not allowing one vertex $v_1 \in \mathcal{V}$ to move (guaranteeing $v_1 = 0$) or by translating all vertices after relaxation. For instance, when relaxation is finished, $b = \frac{1}{Card(\mathcal{V})} \sum_{v \in \mathcal{V}} v$ can be computed as the isobarycenter of all vertex positions. Then, $\forall v \in \mathcal{V}, v \leftarrow v - b$ to ensure $\sum_{v \in \mathcal{V}} v = 0$.

### 5.5.4 Local relaxation

According to Thrun et al. (2000) and Filliat (2001), the amount of energy initially stored within a spring decreases with the distance from the loop closure point. We thus suggest to run a few steps of the relaxation algorithm of Duckett et al. (2000) to perform spring-mass optimization in the vicinity of the vertex where a loop closure was detected each time a loop is found and added to the map (Figure 5.14). Spring-mass optimization can run as a background task with a low priority since it is neither needed by navigation nor by SLAM.

The idea behind local relaxation in the vicinity of a loop closure point is the following: when the robot travels in uncharted space along edges $e$ and add them to $\mathcal{G}$, the optimal position of each new vertex $T(e)$ is $S(e) + e$ since no other information as to where $T(e)$ should be located is available. Thus, $\mathcal{H}_{T(e)} = 0$. However, when a loop closure occurs, the position $S(e) + e$ of the edge's end vertex $T(e)$ may not correspond to the coordinates of the vertex $T(e)$ already in $\mathcal{G}$. This disparity in positions results in energy $J = J(1) = \mathcal{H}_{T(e)} > 0$ being associated to $e$. If $\mathcal{G}$ was a physical spring-mass network (such as the lattice of a crystal), this energy would relax to neighboring "springs" and "masses".

Quantitatively, during the first relaxation step, the energy gets distributed to the first-neighbor springs (edges). If there are four neighboring edges, the additional energy per first-neighbor-spring will roughly be $J(2) = J/(4+1)$ or less. During the second step, the energy gets distributed to the second neighbors. If there are sixteen second neighbors, the additional energy per third-neighbor-spring will roughly be $J(3) = J/(16 + 4 + 1)$ or less, etc. In a regular 2D grid of springs, a (single-way) edge has $\mathcal{O}(n^2)$ n-nearest neighbors (edges closer than $n$ edges away), so that the energy per n-nearest neighbor $J(n)$ decreases as $\mathcal{O}(J/n^2)$. In a less regular network of springs, the decrease is at least

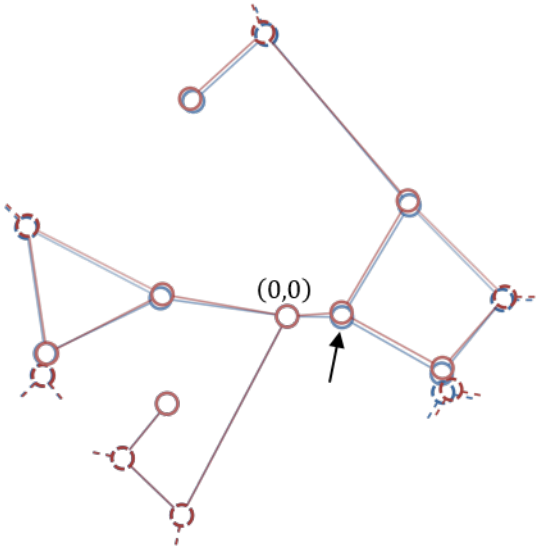



**Figure 5.14:** Local relaxation after a loop closure on the vertex marked with a black arrow. (blue) map $\mathcal{G}$ before spring-mass optimization. (red) map $\mathcal{G}$ after spring-mass optimization. The origin of the map (first vertex encountered) remained at $(0, 0)$.

in $J/n$ (1D array of edges). Consequently, in an infinite network of springs, $\forall \epsilon_J > 0, \exists n_{max} \in \mathbb{N}^* | J(n_{max}) < \epsilon_J$. Thus, it is not necessary to run spring-mass optimization on the whole graph, but only on edges up to the $n_{max}$-nearest neighbor of the loop closure point depending on the required precision.

As a side note, each edge traversal leading to an already known vertex could be considered as a loop closure, even though the edge itself was already known (retraversing an already known environment). Indeed, retraversing a known edge $e$ leads to updating its uncertainty parameters $\delta_d, \delta_t$ and its vector $e$ (non-sequential traversal). Thus, energy is created within the spring-mass network that should be relaxed. However, we don't run the relaxation algorithm each time for performance reasons. Indeed, retraversing a known edge does not create a lot of energy in the network, at least compared to the energy created by coming back to a known place after traversing uncharted space. Consequently, not running relaxation each time does not greatly degrade vertex positions.

### 5.5.5 Rigging

We create dense maps of the environment on top of spring-mass optimized global maps through *rigging*. Rigging is a word from the computer graphics community which refers to the process of attaching bones to the skin of a model in order to deform the model

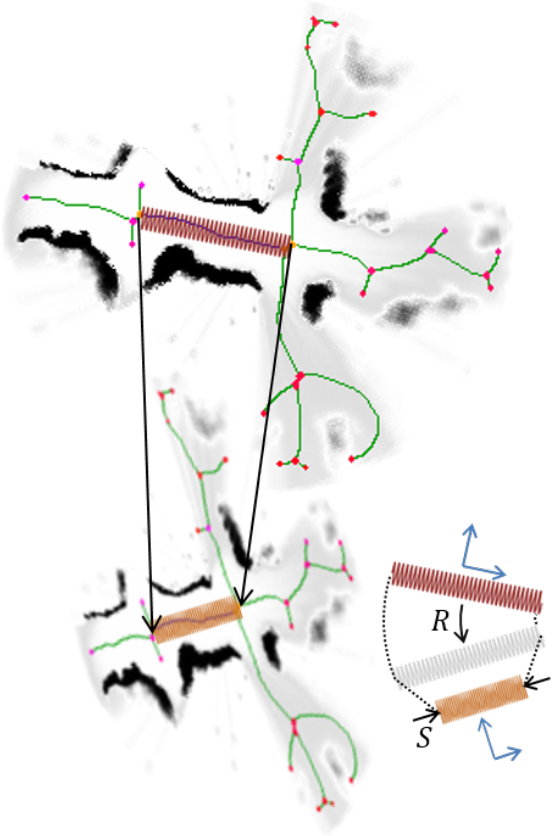



**Figure 5.15:** Rigging deforms sensed data (here, a local occupancy grid with its local Generalized Voronoï Graph) in the vicinity of an edge using a rotation (R) and an axial scaling (S) according to the graph. Initial occupancy grid (top) and deformed grid (bottom).

by moving the bones. In the context of SLAM, sensed data is deformed according to the graph. Even though the name "rigging" was not given to the process, this idea was introduced by Nieto et al. (2004). In the work of Nieto et al., deformation takes into account multiple nearby edges and vertices. We decided to deform edges independently of each other. Indeed, measurements are always related to one single edge and uncertainty is always reset to 0 when reaching a vertex after traversing an edge, so that conceptually, there is no reason for measurements along one edge to affect another edge. An example of rigging is given on Figure 5.15

Formally, suppose that a feature $\phi$ of the environment was sensed at position $\vec{\rho}$ relative to $S(e)$ while traversing edge $e \in \mathcal{E}$. Define a linear function $f^{l*}_{\mathcal{EV}} * (e) = T^*(e) - S^*(e)$ where $T^*(e)$ and $S^*(e)$ are optimal coordinates obtained after spring-mass optimization. Then, $\phi$ should be located at position $f^{l*}_{\mathcal{EV}}(\vec{\rho})$.

In order to deform edges, we use a composition of a rotation (to align the edge vector to the vertex position difference) and an anisotropic scaling (to match the edge vector's length to the distance between both vertices). The direction orthogonal to the edge vector is left unchanged. The complete transformation represented on Figure 5.15 composes (in order): a projection in a referential whose axes are along- and perpendicular to the edge vector, the rotation, the (anisotropic) scaling and a final projection to the usual coordinate axes. In 2D, the transformation writes:

$$f_{\mathcal{EV}}^{l*} = \begin{pmatrix} \Delta\vec{P}_u \cdot \vec{u_x} & -\Delta\vec{P}_u \cdot \vec{u_y} \\ \Delta\vec{P}_u \cdot \vec{u_y} & \Delta\vec{P}_u \cdot \vec{u_x} \end{pmatrix} \cdot \begin{pmatrix} \frac{||\Delta\vec{P}||}{||\vec{e}||} & 0 \\ 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} cos(\theta) & -sin(\theta) \\ sin(\theta) & cos(\theta) \end{pmatrix} \cdot \begin{pmatrix} \vec{e_u} \cdot \vec{u_x} & \vec{e_u} \cdot \vec{u_y} \\ -\vec{e_u} \cdot \vec{u_y} & \vec{e_u} \cdot \vec{u_x} \end{pmatrix} \tag{5.29}$$

where $(\vec{u_x}, \vec{u_y}, \vec{u_z})$ is an orthonormal basis, $\vec{e}$ is the edge vector, $\vec{e_u} = \frac{\vec{e}}{||\vec{e}||}$, $\Delta\vec{P} = T^*(e) - S^*(e)$, $\Delta\vec{P}_u = \frac{\Delta\vec{P}}{||\Delta\vec{P}||}$, $cos(\theta) = \vec{e_u} \cdot \Delta\vec{P}_u$ and $sin(\theta) = (\vec{e_u} \times \Delta\vec{P}_u) \cdot \vec{u_z}$

(5.30)

It is easy to check that this function transforms the edge vector $e$ to $\Delta\vec{P} = T^*(e) - S^*(e)$.

Figure 5.16 shows an example of global map generated using spring-mass optimization and rigging during a data collection run on a pioneer robot equipped with a Kinect camera and MEMS compass.

## 5.6 Conclusion on the SLAM framework

In this chapter, we introduced a new online hybrid metrical/topological SLAM framework designed for PNSLAM in huge environments (hundreds of loops). This framework models *physical places* and *paths* between places as a *directed graph*, introducing *single-way paths*. The duality between edges and vertices in the graph is used in two ways:

1. *Movement uncertainty* is associated to edge measurements and *pose uncertainty* is asssociated to vertex detections. Modeling pose uncertainty is necessary to decorrelate the map from the actual trajectory of the robot by allowing loops to be ignored in uncertainty projection. This decorrelation makes it possible to forget old movements of the robot and perform uncertainty projection (theorems 4 and 5).

2. Edges are associated to a measured vector $\tilde{r}_e$ and to measured uncertainties $\delta_t$ and $\delta_d$. They describe *relative* transformations from one vertex to another, each vertex

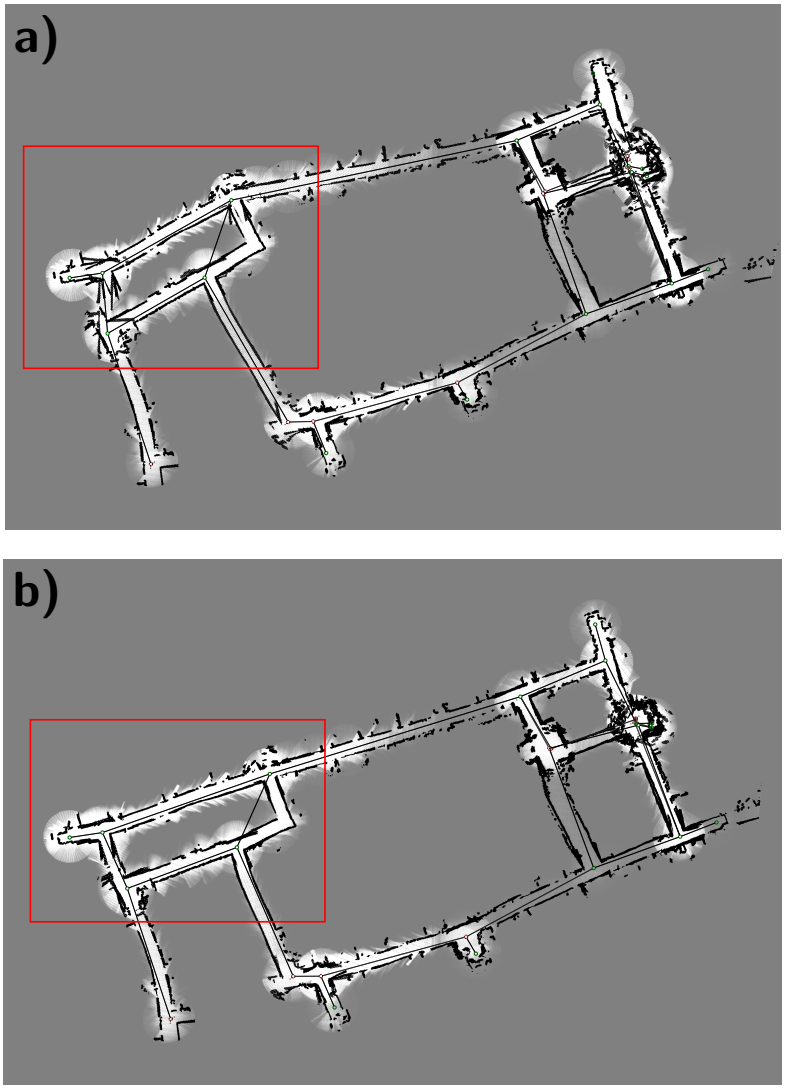



**Figure 5.16:** Generating a global map from $\mathcal{G}$ and edge-local occupancy grids using spring-mass optimization and rigging. (a) before and (b) after spring-mass optimization. While spring-mass optimization rectifies the graph, rigging transposes this rectification to edge-local occupancy grid. Spring-mass optimization was very efficient in the zone within the red frame.

defining its own coordinate frame. On the other side, *absolute* vertex positions referenced within a single coordinate frame can be obtained using spring-mass optimization. Vertex positions are not required by path planning and navigation algorithms.

We used *bounded uncertainties* and *active disambiguation of place hypotheses* to prove that our approach can achieve *topological correctness* (theorem 7), *i.e.* it can produce a map free of *structural ambiguity*. The integration of our SLAM framework with path planning and navigation algorithms allows unsupervised navigation in initially uncharted environments as well as other PNSLAM missions such as "go to a target whose approximate position is known" or "look for a treasure". The hypothesis testing approach allows solving the *kidnapped robot problem* as a loop closure with infinite initial metrical uncertainty.

We took the hypothesis of compass-based odometry, which implies that no rotational drift was taken into account. We are confident that taking into account rotational drift within the framework described in this chapter is possible by considering all rotations not relative to an absolute referential, but to the last edge visited. In order to simplify comparison of vertex signatures of $v_1 \in \mathcal{V}$ and $v_2 \in \mathcal{V}$, we can get an initial rotation estimate by matching the angles of the outgoing edges of $v_1$ and $v_2$. The spring-mass model does not need to be modified to take into account a rotational drift.

The capacity of our approach to handle dynamics of the environment was not assessed theoretically but we demonstrate successful mapping in environments with moderate dynamics in chapter 6. Dynamics that do not disrupt place detection are not seen by the algorithm. Conversely, if places are extracted directly from the topological skeleton, a passer-by or slight changes in the positioning of objects in a room may change the topology of the skeleton. We can devise two approaches to mitigate these errors: first, it is possible to detect objects, especially moving objects, and subtract them from the occupancy grid before skeleton extraction. Such a filtered occupancy grid would only reflect the immutable part of an environment, similarly to the place cells in the brain of mammals (see chapter 2). The second possible approach, used in chapter 6, would be to constantly change the map according to the newly detected topology. However, using this second approach, a single passer-by blocking a corridor may be sufficient to part the map in two disjunct components. We believe that there is a limit on the spatial and temporal amount of dynamics the second approach can handle without heavy modifications of the algorithms. For instance, the second approach would not work for a robot touring visitors around a crowded Museum (there would be incessant changes of topology due to moving people) but would probably succeed for unsupervised everyday navigation inside an office building (with occasional people movements) or autonomous navigation in a countryside or forest environment.

# 6 Assembling PNSLAM components - experiments

*"In the original 'Star Wars' movie, there is a small toaster-sized and shaped robot on the Death Star that guides Stormtroopers to where they need to go. I always liked that robot because I could imagine how to build it - and it served a real purpose."*
**Colin Angle**

In this chapter, we glue together the components described in the previous chapters: the *Exploratory Planner / Exploratory Digraph Navigation approach*, *local navigation and topology extraction* and the *SLAM framework*.

In chapter 5, we showed that with well chosen parameters and assuming perfectly robust vertex detection and navigation, our SLAM framework would always produce topologically correct mapping and localization. However, when working with real world data, it is often not possible or not efficient in terms of energy spent navigating (notably during disambiguation phases) to enforce all the highly conservative hypotheses on $\delta_t, \delta_d, p_h$ and $p_l$ required for theorem 7. Neither is it possible to extract vertices with 100% robustness (there may be parasitic vertices, some legitimate vertices may not be detected and edges starting on a vertex may not be detected correctly, see chapter 4). For this reason, we run simulations and experiments in real environments where values of $\delta_t$, $\delta_d$, $p_h$ and $p_l$ may be erroneous, where vertices are extracted from sensor data and where the robot follows with more or less success the commands it is given (desired direction of movement and speed). Figure 5.3 shows the general architecture of our test bench.

## 6.1 Finding an experimental protocol

### 6.1.1 Difficulties in comparing to state of the art

We chose to test our SLAM framework on an exploration mission, which is a variant of PNSLAM. Exploration is the PNSLAM problem corresponding to most existing SLAM experiments whose aim is to reconstruct an environment. Comparison of our approach to state of the art SLAM frameworks is however not straightforward.

First, our framework is oriented towards autonomous robot mapping and navigation, so that the map should indicate traversable and occupied space instead of simply the position of points of interest.

Second, state of the art datasets such as Bosse et al.'s MIT Killian court (Bosse et al., 2004), Duckett et al.'s Örebro university (Duckett, Marsland, and Shapiro, 2002), Thrun's Museum (Thrun et al., 1998) or Werner et al.'s Carnegie Mellon Wean hall (Werner et al., 2009) only exhibit a handful of loops, so evaluation of topological correctness using the number of erroneous loops on the map as done in (Werner et al., 2009) is not a reliable measurement for these datasets. Thrun and Montemerlo (2006) studied environments with more loops, but topological correctness required factoring in GPS for at least one of their dataset (*Gate's Computer Science Building, Stanford University's main campus*). Pinies et al. (2009), Liang et al. (2013) and later graph SLAM approaches present results for the *Manhattan* and *City10000* datasets which have tens or even hundreds of loops. However, these datasets are regular grids with some untraversable squares, which makes them easy to map since the robot always moves an integer number of grid units in the $x$ and $y$ directions between each intersection.

Third, a topologically correct SLAM algorithm should provide the current topological position of the robot, *i.e.* determine on which edge or vertex of the map the robot is currently located. In other words, for each new infinitesimal movement, the navigating robot should update its attachment to one edge or vertex, which is not a straightforward process. One way to avoid this difficulty is to have the robot explicitly track and follow one edge while being extremely careful not to change edge until a vertex is reached. This is not possible with existing measurement datasets, where the robot is not even guaranteed to traverse an edge entirely from the vicinity of its origin vertex to the vicinity of its destination vertex. Additionally, if an edge is not traversed completely (from the detection range of its origin vertex to the detection range of its destination vertex), collection of edge vectors and uncertainties is not possible, and thus, the map cannot be built. Traversal of complete edges is not guaranteed in state of the art measurement datasets.

Fourth, our disambiguation strategy requires the robot to be able to freely choose its

path. In order to use existing measurement datasets, we could implement another disambiguation strategy where loop closure is deferred until there is enough evidence, even though there may never be, at least in a finite time horizon (Bosse et al., 2004). If we change our disambiguation strategy, the rigorous threshold-based approach cannot be applied. Paradoxically, we could still map state of the art datasets perfectly, even without a disambiguation strategy. The reason for this is that since there are only a handful of loops, there always exists a threshold in similarity between places such that places more similar than the threshold are the same and places less similar are not (this can be observed using for example occupancy grid matching of local sensor data for each place). A challenging dataset would exhibit extremely similar places and tens to hundreds of loops.

Finally and as detailed in subsection 6.1.4, we could not find consensus metrics in literature to compare SLAM algorithms in a sensor-agnostic way.

For these five reasons, our simulation protocol described in the next subsection is based not on sensor dumps describing odometry and sensor readings acquired beforehand but on floorplans of environments (as black and white bitmaps). Simulations use the floorplan to emulate distance sensors (sonars, laser range finders, stereographic cameras, Kinect, ...) and odometric readings in real time. Distance readings are obtained by casting a ray in the floorplan. The dimensioning unit for space is the pixel. The robot keeps mapping an environment until no vertex has unexplored edges on the produced map. New metrics are introduced in subsection 6.1.4 to assess the topological and metrical correctness of the map produced by a SLAM approach.

### 6.1.2 Experimental protocol

We use the Generalized Voronoï Graph (GVG), computed on a local robot-centered scrolling occupancy grid, to extract the local topology of the environment, namely edges (as points where two obstacles of the environment are equidistant) and vertices (as points where at least three edges meet) (Beeson, Jong, and Kuipers, 2005; Choset and Nagatani, 2001; Kuipers et al., 2004). "Stopper" vertices are also placed at the end of edges terminating on a wall (dead ends) and edges whose destination is not visible on the local occupancy grid (because it is too far). The implementation is described in chapter 4. GVG extraction is sensitive to noise in the occupancy grid, causing vertices to be detected differently (different number of edges) and sometimes not to be detected at all depending on the past trajectory and sensor values. The situation of Figure 6.1 will likely cause errors. The GVG may be replaced by the Extended GVG (Beeson, Jong, and Kuipers, 2005) (see chapter 4) for environments with large open areas. The local occupancy grid only describes obstacles within a bounded range of the current position of the robot and construction of the grid does not take into account structural ambiguities due for instance to loops in the environment (these are handled at the graph

level).

The robot always tracks and follows a single edge at a time while keeping some movement freedom around the edge (edge attachment is comparable to a ski-tow towing a skier, as described in chapter 4), which differs from the works of Choset et al. (2001) where the robot is constrained to stay equidistant to two obstacles at all times. The robot always goes from the vicinity of the vertex where the edge starts to the vicinity of the vertex where the edge ends.

We use local occupancy grids around a vertex to produce (possibly ambiguous) *place signatures*. The dimension of the grids used are $200 \times 200$ or $300 \times 300$ pixels depending on the scale of the features in the traversed environment. In addition to the vertex-local occupancy grids, we also store edge-local occupancy grids. We could use them as *edge signatures* but we did not in order to save computing power (matching occupancy grids is compute-intensive and edge occupancy grids largely overlap vertex occupancy grids). However, we use these edge occupancy grids together with a *rigging* approach (chapter 5, section 5.5) in order to produce visually satisfying dense maps of the environments. Edge-local grids are about twice as large in both dimensions as vertex-local grids.

Due to sensor dynamics, it happened during mapping that wrong vertices or edges were detected (for example, two vertices with three outgoing edges each were fused into a single vertex with four outgoing edges for half of the traversals, as shown on Figure 6.1). To correct errors due to sensor dynamics, inconsistent edges and vertices were deleted when retraversing the same place and detecting a different topology. An equivalent process in *Atlas* would be map fusion and obsolescence, which is not described in (Bosse et al., 2004). Additionally, when re-traversing an already mapped environment, if a vertex with $n \in \mathbb{N}^*$ edges was expected and a vertex with $m \in \mathbb{N}^*, m < n$ edges is found instead, the occupancy grid is examined near the current vertex to check whether multiple neighboring vertices with a few edges each could be fused into a single vertex with $m$ edges, in which case the fused vertex with $m$ edges is returned instead of the initial $n-$edges vertex. For instance, on Figure 6.1, if vertex 1&2 was expected with $n = 4$ edges and only vertex 1 was found with $m = 3$ edges, examining the occupancy grid would lead to vertices 1 and 2 being fused together and vertex 1&2 being reported correctly.

Figure 6.2 shows a close-up view of the sensor profile and the associated local occupancy grid and extracted GVG.

The code (in C) runs faster than real time on an average laptop computer, using up to one 2.5Ghz processor core (two threads actively carrying computations and five threads idle most of the time) to run:

- a sensor and actuator simulation (for simulations) or communication with actual

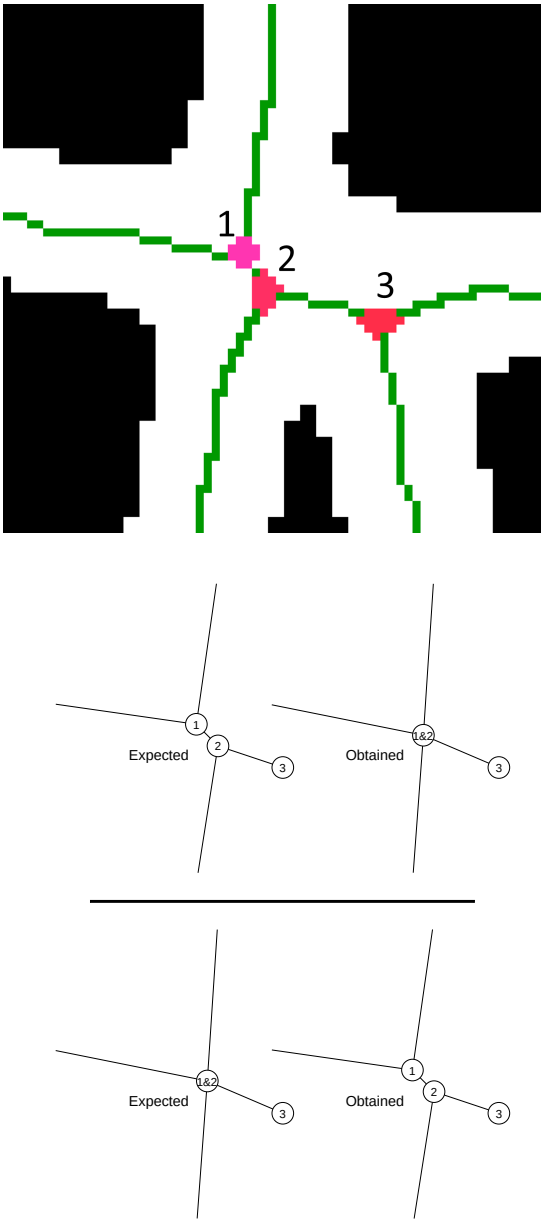



**Figure 6.1:** Vertices 1 and 2 may alternatively be detected as one single vertex or as two distinct vertices depending on sensor noise and past trajectory of the robot. Vertex 3 on the contrary will probably be well detected independently of the past trajectory, even in the presence of noise. The two bottom sketches represent two possible detection errors.

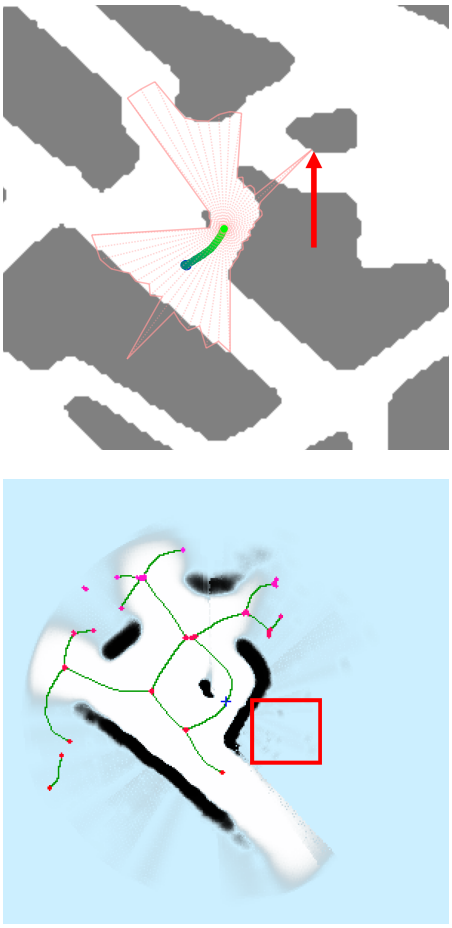


**Figure 6.2:** SLAM detail: noisy sensor profile and past trajectory superimposed on actual floorplan(top) and local occupancy grid with local GVG computed from simulated noisy sensor data (bottom). On the top image, one of the sensors pointing right had its value multiplied by a random number in $[1; 4]$ and produced a completely erroneous value (red arrow). On the bottom image, erroneous readings translate to patches of hot pixels which are barely visible (red frame). These patches are however obvious on Figure 6.12 where they create "fur" along the external border of the map.

sensors and actuators (for robot experiments),

- occupancy grid updates,
- GVG extraction,
- occupancy grid navigation,
- uncertainty projection,
- disambiguation (including occupancy grid matching),
- graph planning and plan execution,
- spring-mass optimization and
- exports (including rigging to get a dense map).

The computational effort of the algorithms described in chapter 5 is negligible in comparison to GVG extraction (chapter 4), occupancy grid matching and data export to files. A more detailed analysis of running times and complexities of the different components can be found in chapter 7.

### Simulations

For simulations, errors on odometric measurements are reproduced by transforming each infinitesimal odometric measurement $\vec{r_o}$:

- adding a strongly anisotropic translational noise $\alpha.(|r_o^x|, |r_o^y|)$ with $\alpha$ uniformly distributed between 0.015 and 0.03
- rotating $\vec{r_o}$ by a random value $\rho$ uniformly distributed between 0° and 10° (up to 5° compass noise with simulated 5° bias)

The 5° rotational bias is inspired by the work of Borenstein and Feng (1996) on measuring and correcting odometric errors. The 1.5 to 3% translational noise is inspired by wheel encoders with 64 binary values, supposing one angular sector gets skipped per wheel rotation. The $\delta_t^{\vec{r_o}}$ associated to each infinitesimal vector measurement $\vec{r_o}$ is $\max(\alpha)||\vec{r_o}|| + \max(\rho).||\vec{r_o}||(1 + \max(\alpha)))$, which is a consistent error estimate.

Simulations use a ring of 64 range sensors. Each range reading $r_m$ is added a random value $\epsilon_r$ with $\epsilon_r$ uniformly distributed in $[-0.08r_m; 0.08r_m]$ with the aim to approach

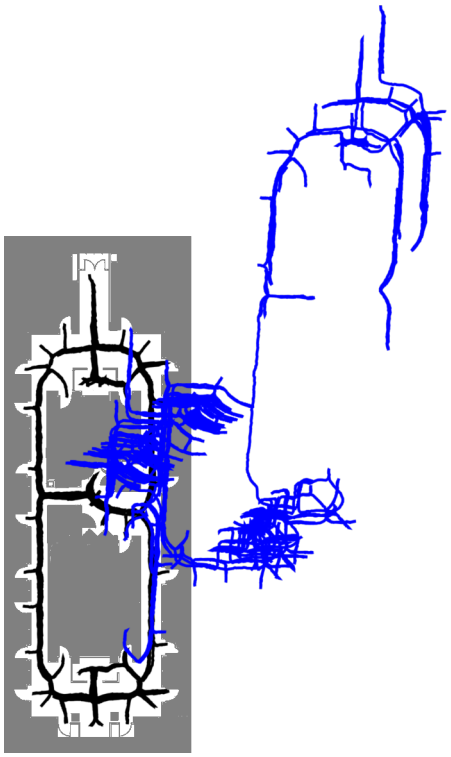


**Figure 6.3:** Killian court SLAM: actual simulated trajectory (black) superposed on the floorplan and trajectory deduced from simulated raw odometry and compass (blue).

the results of Drumheller (1987) on mobile robots with sonar sensors. This uniform noise up to 8% is higher than the typical error of a Kinect sensor which is about 1% or lower (Khoshelham and Elberink, 2012). The precision of the sensor readings is also limited by pixelization (the maximum precision is always 1 pixel of the floorplan) which typically results in 1 to 10% of additional errors (depending on resolution of the image and distance of the robot to an obstacle). Finally, 2% of the distance readings get multiplied by a random number in $[1; 4]$ to reproduce invalid values returned by sonar sensors when multiple reflections of the emitted ray occur (Drumheller, 1987). Figure 6.2 shows a typical sensor profile.

The simulated error model's parameters were tuned to approximately reproduce state of the art measurement datasets of Bosse et al., Thrun et al. and Duckett et al. with the hypothesis of compass-based odometry (Duckett, Marsland, and Shapiro, 2002). An example of odometry on Killian court is shown on figure 6.3. This result visually resembles Duckett's results with compass-based odometry (2002).

Simulation is carried using a custom version of *Stage* (Gerkey, Vaughan, and Howard,

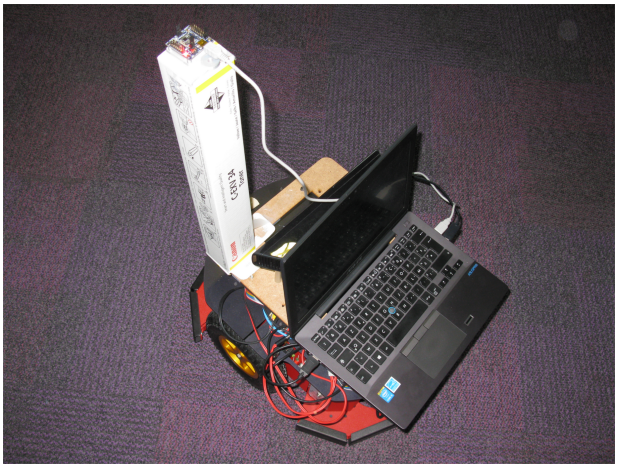


**Figure 6.4:** The pioneer 3 robot equipped with a Kinect 1 and MEMS magnetometer used for experiments. The magnetometer is mounted on top of a long cardboard box to avoid magnetic field disturbances due to metallic parts of the robot.

2003) as simulator.

### Robot experiments

Robot experiments are carried out using a Pioneer 3 robot, a MEMS magnetometer (LSM303DLHC 3D accelerometer and magnetometer) and a Kinect 1 camera ($320 \times 240$ infrared depth sensing, field of view $57 \times 43°$, range 50cm to 5m). The Robot Operating System (ROS/indigo) is used for communications with a laptop running Ubuntu (version 14.04) or a board such as the Tegra K1 onboard the robot. This board is responsible for running the algorithms. Figure 6.4 shows a picture of the robot during a data collection run.

From the depth image of the Kinect, a 1D depth profile is extracted. Since the horizontal field of view of the Kinect is far from 360°, each time the robot reaches a GVG vertex, it rotates around itself to reconstruct a 360° signature of the environment and to confirm the local topology of the environment. One interesting side-effect of the rotation is that if the vertex is due to a temporary obstruction (typically, a passer-by), there is a chance that the obstruction disappears before the end of the rotation. If this happens, the erroneous vertex disappears from the occupancy grid and navigation continues as if there was no vertex.

The use of an occupancy grid as intermediary representation allows a wide range of depth sensors to be used and even fused together, including infrared, sonar, lidar and

stereographic cameras. Sensors with a non-negligible vertical field of view such as cameras have the advantage of providing depth information over a wide vertical range, which is required to correctly detect obstacles whose shape is not constant on the vertical axis such as chairs or desks.

#### Parameters for SLAM

Parameters of our approach (chapter 5, subsection 5.4.4) are chosen as follows:

$d_{min}$ is set to the minimum distance between distinct GVG vertices detected from an occupancy grid, that is 4 pixels.

Since the occupancy grid integrates range readings over time, the precision of vertex extraction from an occupancy grid is better than that without an occupancy grid (directly from the sensor profile, as done in (Choset and Nagatani, 2001)). Since it is hard to estimate the improvement of vertex detection precision allowed by the integration effect, we arbitrarily set $\delta_d = (d_{min} - 1)/2 = 1.5$px for each vertex. This value is higher than the unavoidable 1px error due to discretization and is about the average radius of a single vertex, as shown for instance on Figure 6.1. It is not guaranteed that $\delta_d$ is a consistent error estimate.

$\delta_t$ is chosen according to the odometric system and always overestimates odometric uncertainty. For simulations, $\delta_{t_e} = 0.2||\tilde{\mathbf{r}_\mathbf{e}}||$ whereas for robot experiments, $\delta_{t_e} = 0.25||\tilde{\mathbf{r}_\mathbf{e}}||$ for each edge $e$ traversed whose odometric measurement is $\tilde{\mathbf{r}_\mathbf{e}}$.

$p_h$ and $p_l$ are set using a trial and error approach. We kept the same values of the parameters for all simulations in order to make mapping more challenging and prove robustness of the approach. For robot experiments, the relative weights of local topology, signature matching and other probabilistic measurements were tuned for the sensors used.

### 6.1.3 Datasets

In the second part of the SLAM survey by Bailey and Durrant-Whyte (2006), the authors wrote: "*The challenge now is to demonstrate SLAM solutions to large problems where robotics can truly contribute: driving hundreds of kilometers under a forest canopy or mapping a whole city without recourse to global positioning system (GPS) [...]*". We thus decided to try and map a whole city. A quick image search on the internet gave us a map of the historic center of Cuzco in Peru, which we turned into a black and white bitmap (Figure 6.5) and used exactly like the floorplan of the environments mentioned

above. The Cuzco dataset is interesting because it exhibits both repetitive structures (to lure the disambiguation algorithm into finding erroneous loops) and very long paths without intersections (to create loops with large odometric errors in a context where odometric errors arise from each movement). While the issue of very long paths without intersections causes metrical inaccuracies, we observed it to be less of a concern than the repetitive structures in the process of obtaining a topologically correct map. The issue of repetitive structures was also reported by, for instance, Bosse et al. (2004).

There is also a less visible but more problematic issue due to four-way intersections in ambiguous zones such as that on Figure 6.1. Vertices with more than three edges are likely to lead to incorrect place recognition: an expected vertex is not found anymore because its amount of outgoing edges is different than forecast. Incorrect place recognition will cause navigation as well as backtracking errors (impossibility to come back to a known place when an hypothesis gets invalidated during disambiguation). The top-right part of our map of Cuzco (Figure 6.5) has an array of vertices with four outgoing edges, which is a challenging situation for our SLAM algorithms since it combines repetitive structures and ambiguously defined vertices.

Cuzco exhibits 184 loops (Figure 6.5), which makes it an interesting benchmark for topology-based techniques.

In addition to the Cuzco dataset, we also show simulations in three environments used in SLAM litterature: Bosse et al.'s MIT Killian court (Bosse et al., 2004), Duckett et al.'s Örebro university (Duckett, Marsland, and Shapiro, 2002) and Thrun's Museum (Thrun et al., 1998).

We use a pioneer robot to run indoor experiments in the second floor of one of the Nano-Innov buildings on the Saclay plateau, France (Figure 6.6), a typical office building. A total of five topological loops are present in this environment, which is not much compared to *Cuzco* but we could not have access to buildings containing more loops.

### 6.1.4 Metrics

#### Issues of existing metrics

Despite having been explored by Thrun et al. (1998) and Werner et al. (2009), measurements of topological correctness of a SLAM algorithm do not seem to be widely used in SLAM literature. Werner et al. used the number of places/loops in the map versus in the environment as an estimation of topological correctness. Such a metric does not make sense in environments where some parts may not be explored at all (reducing the number of vertices on the map relative to the environment) while other parts may

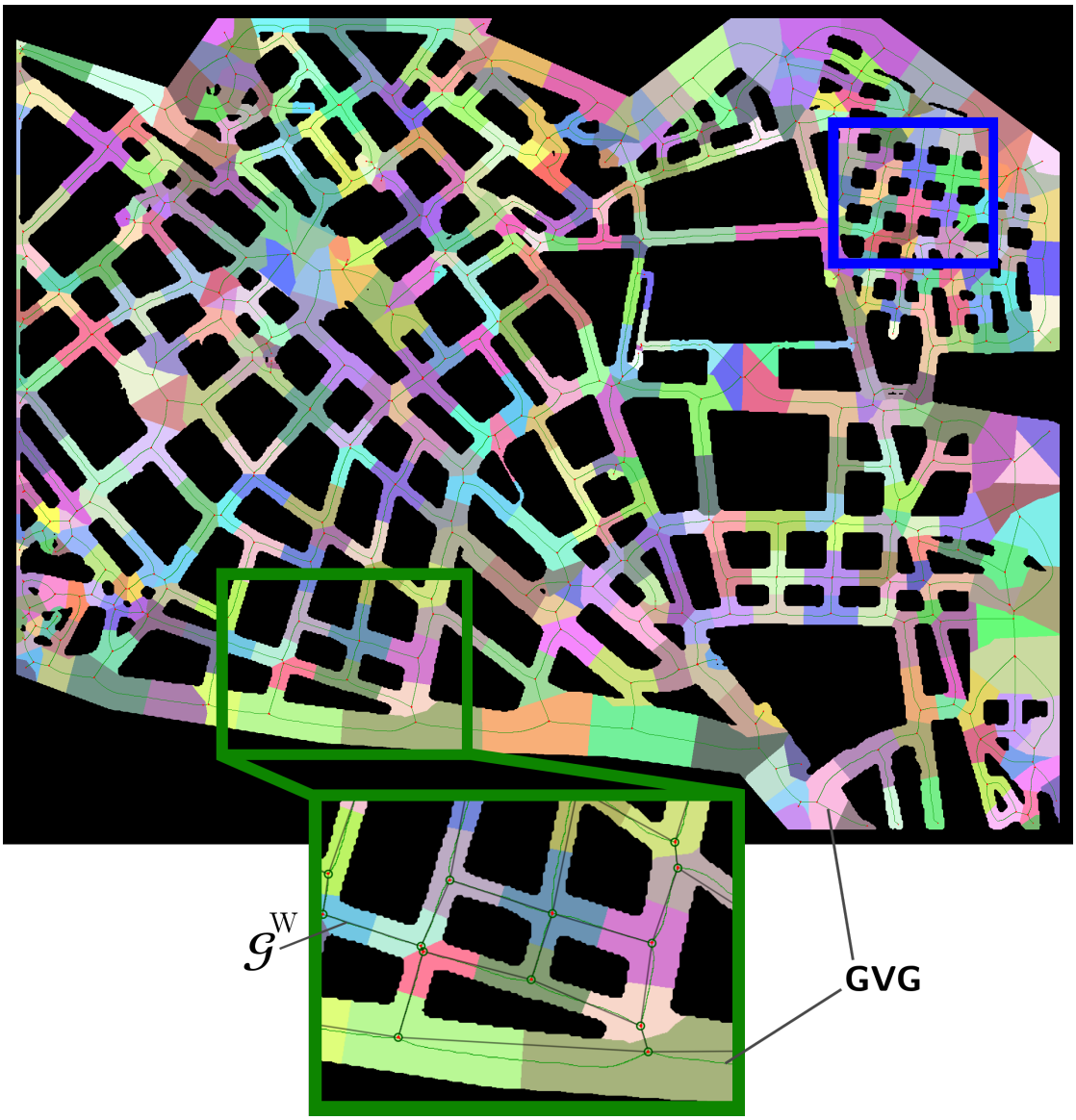



**Figure 6.5:** The map of Cuzco used in simulations. Note the repetitive structure of four-way intersections of the top-right corner (blue rectangle). In addition to the floorplan, this image shows a Generalized Voronoï Diagram of the environment used to partition the environment into cells. The partitioning method is explained in subsection 6.1.4.


**Figure 6.6:** Robot runs took place on the second floor of one of the Nano-Innov buildings. (Image: wikimedia commons/Lionel Allorge, 2014, https://commons.wikimedia.org/wiki/File:B%C3%A2timent_Nano-Innov_sur_le_plateau_de_Saclay_le_23_juillet_2014.jpg)

produce duplicate vertices on the map or lead to completely wrong edges. Both errors may eventually compensate each other, erroneously stating that the map is topologically correct.

We could not find consensus metrics in literature for robust comparison of metrical correctness of SLAM algorithms either. Gutmann and Konolige (1999) for example use the average error accumulated between adjacent poses of the robot, which heavily depends on the sensors, and as such cannot be used to compare SLAM approaches (the error should somehow get normalized by a parameter describing the precision of sensors). Dissanayake et al. (2001) express the absolute errors in meters, which not only is sensor-dependent but may also disadvantage hybrid metrical/topological frameworks based on relative positioning. Similarly, while the RMS error used for instance in (Sunderhauf and Protzel, 2013) is useful for comparing the maps produced by various algorithms using the same sensor dumps, it does not make sense with different sensors or a different trajectory of the robot within the environment (let alone different environments).

### Proposed set of metrics

Giving the lack of robust and sensor-agnostic metrics in SLAM, we propose the following set of four metrics to assess topological and metrical correctness of a map produced by a SLAM framework:

- the normalized **residual energy** $\eta_r$ of the spring-mass model after spring-mass optimization,
- a **graph-based** evaluation $f_t$ of topological correctness,
- a **navigation-based** evaluation $f_{nm}, f_c, f_l$ of topological and metrical correctness and
- the **average relative RMS distance and angle errors** $\eta_l, \eta_\theta$ after spring-mass optimization relative to ground truth.

**Residual energy**: Let $S^*(e)$ and $T^*(e)$ be the optimal positions of the start- and end-vertices $S(e), T(e)$ of edge $e \in \mathcal{E}$ obtained from edge measurements using a spring-mass model. Spring-mass optimization minimizes (chapter 5 section 5.5) an energy function $\mathcal{H} = \sum_{e\in\mathcal{E}} ||(T(e) - S(e)) - \tilde{r}_e||^2$. $\mathcal{H}_r = \frac{1}{Card(\mathcal{E})} \sum_{e\in\mathcal{E}} ||(T^*(e) - S^*(e)) - \tilde{r}_e||^2$ is the *residual energy* of the spring-mass model after spring-mass optimization, homogeneous to a square length. $\eta_r$ is then defined as $\eta_r = \sqrt{\mathcal{H}_r} / \left( \frac{\sum_{e\in\mathcal{E}} ||T(e)-S(e)||}{Card(\mathcal{E})} \right)$. The residual energy score is intrinsic to the robot in that it can be computed without ground truth. It reveals how much difference between measurements (edge vectors) and optimized vertex

positions is present in the final globally coherent map. It depends on the probability distribution of sensor measurements. In graph SLAMS, $\eta_r$ corresponds to the remaining error after running the back-end algorithm (Grisetti et al., 2010).

In order to assess topological correctness of the maps produced by our approach, we need to compute a graph abstraction $\mathcal{G}_\mathcal{W}$ of the simulated environment/floorplan $\mathcal{W}$ with which the map can be compared. Thus, prior to running a simulation, the GVG (or Extended GVG (Beeson, Jong, and Kuipers, 2005)) of the simulated environment $\mathcal{W}$ is computed and used as $\mathcal{G}_\mathcal{W}$. The environment is then partitioned into places $C_i$ around GVG vertices $w_i$ and GVG edges $p_{ij}$ (starting on $w_i$ and ending on $w_j$) the following way: take a position $R$ in the environment. Then, $R \in C_i \iff \exists j | \forall (k,l) \neq (i,j), SD(R, p_{kl}) > SD(R, p_{ij})$ and $||R - w_i|| < ||R - w_j||$ where $SD(R,p)$ is the length of the shortest path from $R$ to $p$ in the environment. For any physical places and paths (not necessarily GVG vertices and edges), $SD$ can be computed using algorithms (Lazy) Multi Theta* or MHydra1/2 (appendix 2). Figure 6.5 shows a partitioned floorplan with tags represented by random colors. During simulation, when the robot detects a place $w_i \in \mathcal{G}_\mathcal{W}$, it reports it as $v_i \in \mathcal{V}$ and appends to $v_i$ the tag $C_i$ corresponding to $w_i$.

**Graph-based** evaluation of topological correctness: this score evaluates the topological differences between the produced map $\mathcal{G}$ and the ground truth network of places $\mathcal{G}_\mathcal{W}$. It is computed the following way (Figure 6.7): first, both $\mathcal{G}_\mathcal{W}$ and $\mathcal{G}$ are simplified by removing vertices with exactly two edges. The situation of Figure 6.1 may cause the floorplan and the map to differ locally because of subtle differences in the way the GVG is computed for ground truth and during a simulation or experiment. This difference should not be considered when assessing topological correctness of the produced map. Thus, the map is manually modified by fusing or breaking up vertices when necessary. Then, for each tag $C_i$, a list $L^\mathcal{G}_{C_i}$ of tags that can be accessed with a single hop (traversing exactly one edge) of a vertex tagged $C_i$ on $\mathcal{G}$ is computed. If no vertex in $\mathcal{G}$ is tagged $C_i$, $L^\mathcal{G}_{C_i} = \varnothing$ A similar list is constructed using $\mathcal{G}_\mathcal{W}$ instead of $\mathcal{G}$: $L^{\mathcal{G}_\mathcal{W}}_{C_i}$. Let $\mathcal{V}_1$ be the set of tags $C_i$ associated exclusively to vertices with exactly one outgoing edge in $\mathcal{G}$. Let $\mathcal{W}_1$ be the set of tags $C_i$ associated exclusively to vertices with exactly one outgoing edge in $\mathcal{G}_\mathcal{W}$. Let $\cap_i = L^\mathcal{G}_{C_i} \cap L^{\mathcal{G}_\mathcal{W}}_{C_i}$ and $\cup_i = L^\mathcal{G}_{C_i} \cup L^{\mathcal{G}_\mathcal{W}}_{C_i}$. Let $n_c^{C_i} = Card(\cap_i)$ be the amount of correct local matches around tag $C_i$. Let $n_t^{C_i} = n_c^{C_i} + \frac{1}{2} Card\{C_j \in \cup_i \setminus \cap_i | C_j \notin (\mathcal{V}_1 \cup \mathcal{W}_1)\}$ be the average number of possible local matches around tag $C_i$. We define the topological correctness of the map $\mathcal{G}$ relative to ground truth $\mathcal{G}_\mathcal{W}$ as $f_t = \frac{\sum_{C_i} n_c^{C_i}}{\sum_{C_i} n_t^{C_i}}$. Note that the definition of $n_t$ does not count mismatches on vertices with exactly one outgoing edge. Indeed, these may correspond to shallow dead ends and weak meet points (Choset and Nagatani, 2001) which may or may not be detected depending on sensor noise. $f_t = 100\%$ implies that the (simplified) map $\mathcal{G}$ is homeomorphic to a subset of the (simplified) ground truth graph $\mathcal{G}_\mathcal{W}$.

**Navigation-based** evaluation of topological and metrical correctness (Figure 6.8): this

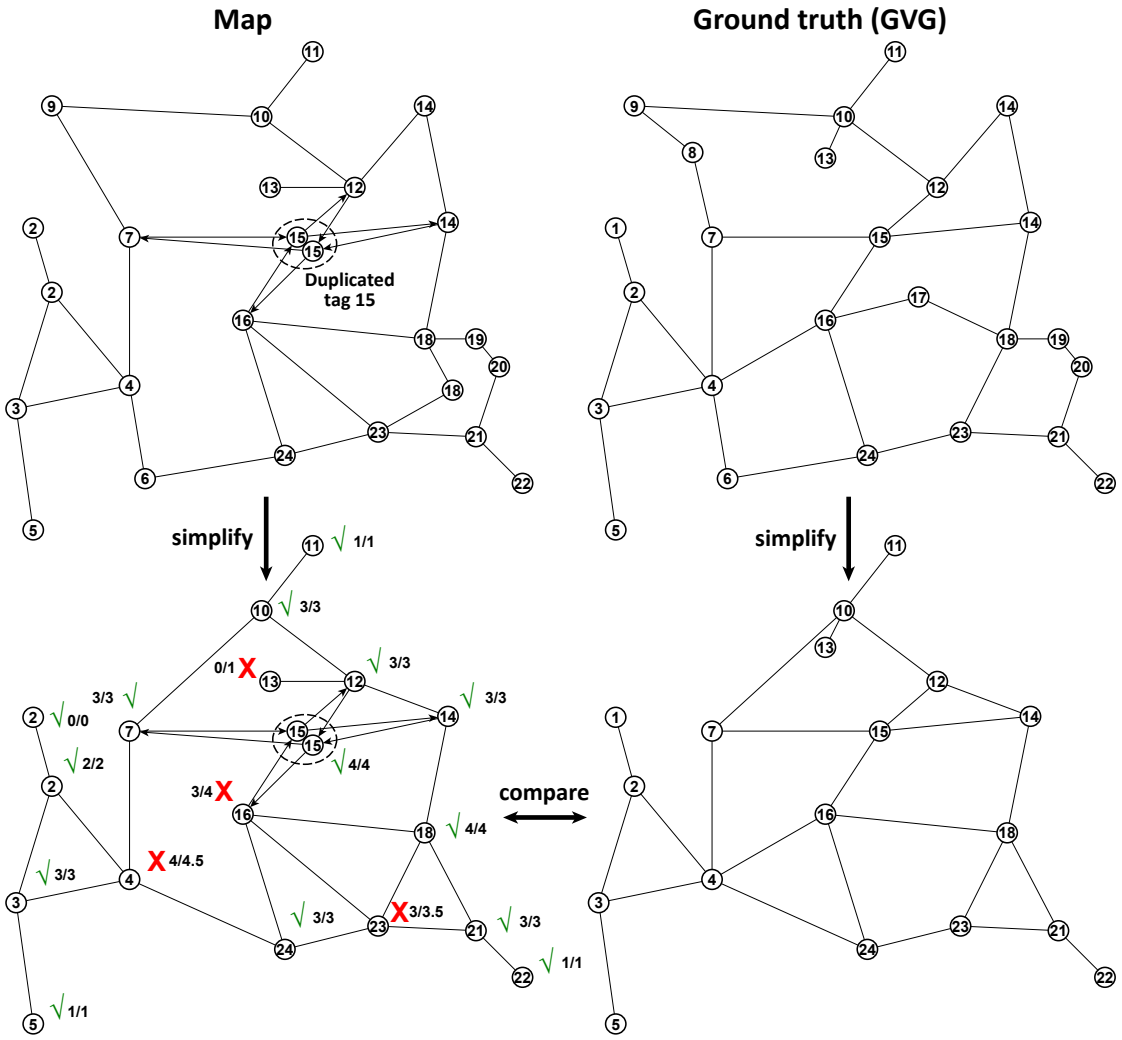



**Figure 6.7: Graph-based** evaluation of topological correctness: after vertices with two edges have been removed, the set of tags $\{C_j\}$ accessible with a single hop from a given tag $C_i$ is compared between simplified map and simplified ground truth. For each tag $C_i$, $n_c^{C_i}$ and $n_t^{C_i}$ are computed. Both are reported on the sketch as $n_c^{C_i}/n_t^{C_i}$. Here, $f_t = 44/47 = 94\%$

score is computed using the raw (not simplified) map $\mathcal{G}$. It consists in planning shortest paths on $\mathcal{G}$ using $A*$ (Hart, Nilsson, and Raphael, 1968) between random places in $\mathcal{V}$, navigating along these paths and marking the paths traversed in $\mathcal{G}_\mathcal{W}$ as a sequence of tags $C_i$ (Figure 6.8 [A]). Then, the trajectory in $\mathcal{G}_\mathcal{W}$ is compared to that obtained by running $A*$ from the same origin to the same destination but directly on the set of places and paths in $\mathcal{G}_\mathcal{W}$ (Figure 6.8 [B]). Both can be different in terms of sequence of places traversed (topology) and in terms of path length (metric). If the sequence of traversed places (or place tags $C_i$) is different in both cases, then the ratio of both path lengths is an evaluation of the suboptimality of the map for navigation tasks. Since path lengths are always estimated on $\mathcal{W}$, this ratio is largely independent from the probability distribution of sensor measurements. If the series of place traversed is the same, the map is considered to provide topologically correct navigation for the specific path chosen. Using this method, it is possible to compute the fraction of paths $f_c \in [0; 1]$ for which the map provides topologically correct navigation and the average added path length $f_l$ for all paths, with or without topological errors. A path with topologically correct navigation has an individual added length of 0. Places reachable by the robot but not present on the map are counted separately as a "fraction of the environment not mapped" $f_{nm} \in [0; 1]$, ignoring shallow dead ends. Navigation-based evaluation of topological and metrical correctness is not to be confused with the "path distance" metric developed by Konolige et al. (2011). Indeed, this metric describes discrepancies between trajectories obtained by path planning using a graph and path planning using a dense map, while the discrepancies we describe are between using the map and using ground truth for navigation. Navigation-based evaluation of topological and metrical correctness can be done without physical traversal of the environment (since each time an edge gets traversed, the map potentially changes) using environment tags $C_i$.

Evaluation of topological correctness with the graph-based and navigation-based scores is more accurate and robust than simply counting erroneous loops. Indeed, both metrics do not overly penalize the case where a place was detected differently from different points of view (leading to single-way edges and vertex duplication as sketched on Figure 6.7) and the situation of Figure 6.1. Still, incorrect edges decrease $f_t$ and $f_c$ and increase $f_l$. In general, $f_t \neq f_c$ since $f_c$ uses shortest paths, which require the metrical properties (edge vectors) of the map to be accurate enough. We expect $f_t$ to be higher than $f_c$ since $f_c$ is weak to cases where there are multiple paths of nearly equal lengths.

Finally, $f_c$, $f_l$ and $f_{nm}$ take into account the actual use of the map for a navigation task. It may happen for the map to be completely wrong from the point of view of an external observer ($f_t \simeq 0$) but to provide perfect or nearly perfect navigation capacities ($f_l \simeq 0$), which is the the primary interest of an autonomous robot.

**Average relative RMS distance and angle errors**: $\eta_l$ and $\eta_\theta$ assess metrical correctness of a map relative to ground truth. The map is first scaled uniformly to compensate the scale bias that can't be corrected by SLAM alone (scale-invariance). A set of $n \in \mathbb{N}^* \gg Card(\mathcal{V})$ tag pairs $(C_1^i, C_2^i)$ are then sampled with the constraint that for

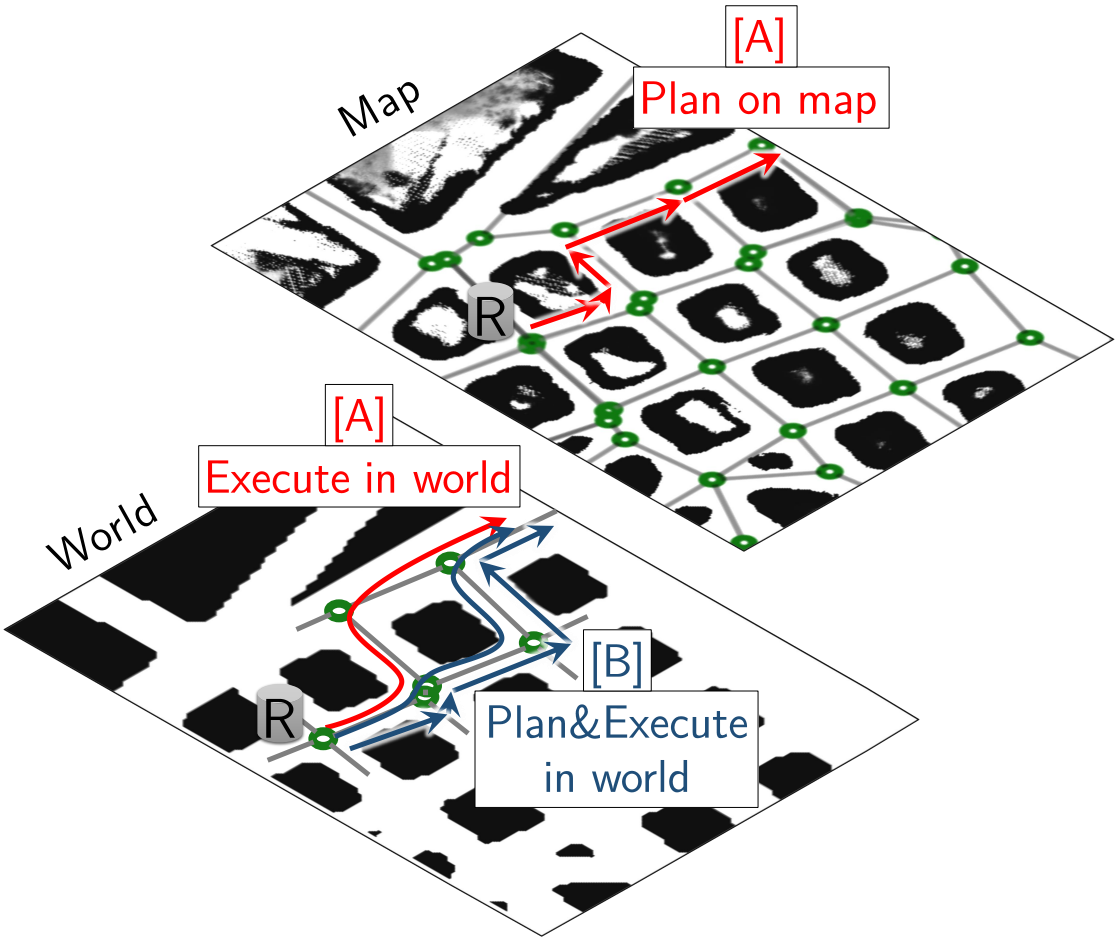



**Figure 6.8: Navigation-based** evaluation of topological correctness. [A]: planning on $\mathcal{G}$ and executing the trajectory in $\mathcal{W}$. [B]: planning on $\mathcal{G}_{\mathcal{W}}$ and executing the trajectory in $\mathcal{W}$. Here, [A] and [B] use different trajectories in $\mathcal{W}$, which reduces $f_c$. The path length difference between both cases is described by $f_l$.

each $C_j^i$, there must exist exactly one vertex $V_j^i$ in $\mathcal{V}$ and one vertex $W_j^i$ in $\mathcal{W}$ that have tag $C_j^i$. $\eta_l$ and $\eta_\theta$ are defined as:

$$\eta_l = \frac{\sqrt{\overline{(\Delta L)^2} - \overline{\Delta L}^2}}{\overline{L}} \tag{6.1}$$

$$\eta_\theta = \sqrt{\overline{(\Delta \theta)^2} - \overline{\Delta \theta}^2} \tag{6.2}$$

with:

$$\begin{aligned}
\overline{L} &= \frac{\sum_{i=1}^{n} L_i}{n}, L_i = \frac{||\mathbf{V_1^i V_2^i}|| + ||\mathbf{W_1^i W_2^i}||}{2} \\
\overline{\Delta L} &= \frac{\sum_{i=1}^{n} (||\mathbf{V_1^i V_2^i}|| - ||\mathbf{W_1^i W_2^i}||) L_i}{\sum_{i=1}^{n} L_i} \\
\overline{(\Delta L)^2} &= \frac{\sum_{i=1}^{n} (||\mathbf{V_1^i V_2^i}|| - ||\mathbf{W_1^i W_2^i}||)^2 L_i}{\sum_{i=1}^{n} L_i} \\
\overline{\Delta \theta} &= \frac{\sum_{i=1}^{n} arg(\mathbf{V_1^i V_2^i}, \mathbf{W_1^i W_2^i}) L_i}{\sum_{i=1}^{n} L_i} \\
\overline{(\Delta \theta)^2} &= \frac{\sum_{i=1}^{n} arg(\mathbf{V_1^i V_2^i}, \mathbf{W_1^i W_2^i})^2 L_i}{\sum_{i=1}^{n} L_i}
\end{aligned}$$

where vertices are identified with their spring-mass optimized position and inter-vertex distances $L_i$ are used as a weighting factor to limit noise on short distances and favor large-scale corrections. $\eta_l$ is also normalized by the average length $\overline{L}$ to produce an adimensional metric.

## 6.2 Exploration missions: simulations and experiments

### 6.2.1 Simulations supposing perfect place extraction and navigation

We first abstracted away the difficulties of navigation and topology extraction and made the hypothesis that places were always detected within a predefined range of their actual position. Each time a place was detected, the number of paths from this place to neighboring places was also detected without error. The robot was always able to travel along a detected path. These assumptions are necessary for theorem 7 to be used. Edge vectors nevertheless contain realistic pose and movement uncertainties as described in subsection 6.1.2. Results for the simplified experiments described in this subsection are reported in table 6.1.

The main purpose of this first simplified test is to validate "experimentally" theorem 7 as well as to test the disambiguation strategy without having to cope with place detection problems. We used a "closest-boundary-vertex-first" exploration strategy which consists in always targeting the nearest vertex with at least one unexplored outgoing edge.

In order to test the uncertainty projection model without the disambiguation strategy on the Cuzco dataset, we first had an oracle validating and invalidating loop closure hypotheses (thus, $p_h$ and $p_l$ were not used). With the oracle, the SLAM algorithms always produced a topologically correct map, which was expected due to theorem 7.

Then, we made the oracle random (50% of the time declaring places as being the same, 50% as being different). Mapping failed badly in all cases with the algorithm infinitely adding new vertices, thus confirming that a disambiguation strategy is necessary.

Next, we made the oracle slightly imperfect: it had a 1/10 chance to make a mistake. Each edge traversed during disambiguation brought $p = 0.9$ of evidence if it was correct and $p = 0.1$ if it was wrong. As such, 3 traversals were necessary to reach 0.99 certainty and 6 traversals to reach 0.99999 certainty.

With $p_h = 0.99999$ and $p_l = 0.00001$, topological correctness was obtained in almost all cases (mapping terminated and the final map was homeomorphic to the Generalized Voronoï graph of ground truth). There were a few cases where the map was locally incorrect due to an ambiguity that could not be resolved properly. For instance, the situation of Figure 6.9 caused local inaccuracies in the map. These kind of inaccuracies were always observed in very tiny portions of the graph. Another cause of inaccuracies is when mapping starts, the map does not contain enough edges and vertices to provide disambiguation and reach $p_h$ or $p_l$. We did not observe cases where a non-negligible part of the map got duplicated or where portions of the environment could not be mapped at all, although this situation is theoretically possible.

We finally set $p_h$ to 0.99 and $p_l$ to 0.01. Mapping gained a factor $1.8 - 2$ in speed, corresponding to twice less edges being visited during disambiguation. This result was expected since the algorithm responsible for choosing a *disambiguation path* (Dijkstra) did not try and avoid distracting the robot from its trajectory. $f_t$ and $f_c$ showed little to no change compared to the previous case, probably because the Cuzco dataset does not present arrays of more than three identical places in a row.

### 6.2.2 Simulations supposing perfect place extraction but realistic navigation

We then removed the constraint that the robot was always able to travel along a detected path. As a result, the robot sometimes did not obey orders and followed an edge it was not asked to, or changed edges without signaling a vertex. However, when a place was

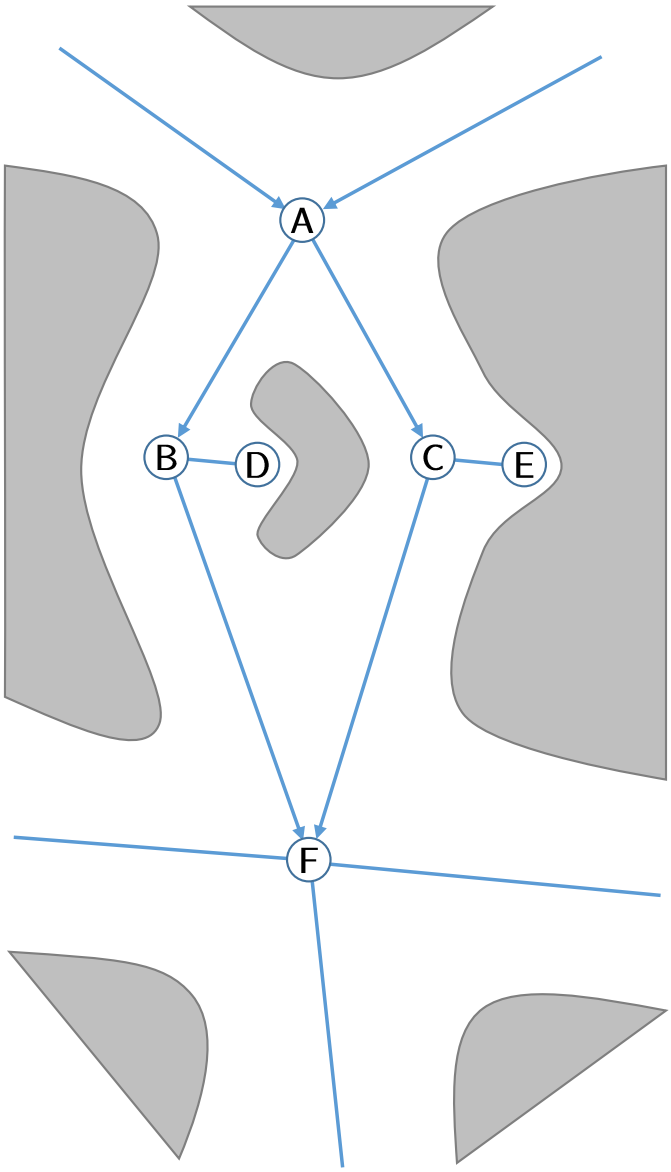



**Figure 6.9:** Situation where disambiguation is likely to fail: whether B and C are the same or different vertices will not be ascertained by further movements in the environment. More precise sensors may however clear this ambiguity.

| **disambiguation by** | $p_l, p_h$ | **map construction** | **traveled length** |
|---|---|---|---|
| oracle (100%) | - | t.c. | shortest |
| none (random oracle) | - | failure | - |
| oracle (90%) | $0.00001, 0.99999$ | almost always t.c. | long |
| oracle (90%) | $0.01, 0.99$ | almost always t.c. | medium |

**Table 6.1:** Simplified SLAM simulations with perfect navigation and place extraction. *t.c.* stands for *topologically correct*.

detected, its local topology was reported correctly. We disabled the oracle and replaced it with the actual evidence-based disambiguation strategy. $p_h$ and $p_l$ were set by trial and error. This test is intended to show the maximum obtainable values of the metrics given an imperfect navigation system and ad-hoc $p_h$ and $p_l$.

The obtained map (spring-mass optimized and rigged) is reproduced on Figure 6.10 and the metrics of subsection 6.1.4 are computed and displayed in table 6.2. The whole map on Figure 6.10 seems to be rotated clockwise relative to the floorplan. We did another test with a higher translational noise (between 0.025 and 0.05 instead of 0.015 and 0.03) which showed about the same rotation, thus demonstrating that rotation is essentially due to the 5° rotational bias and that the translational bias was mostly corrected by spring-mass optimization.

Despite its visual aspect and as shown in table 6.2, this map is not topologically correct ($f_t \neq 100\%$) due to subtle navigation errors (the navigation system fails to robustly follow an edge due to the discrepancies between floating point representations and pixelized representations). Navigation errors create single-way edges and duplicate vertices on the map.

### 6.2.3 Realistic simulations

We defined the place- and constraint extractors as black-box systems whose output was used by our framework, so that *in theory*, we should not cope with place detection errors. However, there is no perfect place extraction system, which means that we should use a realistic place extractor in order to prove compatibility of our framework with real-world data.

For this subsection, vertices and edges are extracted from noisy occupancy grids which are also used for navigation. In this setup, ground truth is only used in three ways:

- to emulate sonar rays (as described in subsection 6.1.2),


**Figure 6.10:** *Simulation: perfect place extraction - realistic navigation.* Final map of the Cuzco dataset obtained by our algorithms with perfect simulated (unrealistic) vertex detection but realistic navigation (subsection 6.2.2). The red circle indicates a local glitch caused by a one pixel offset in topology extraction (chapter 4).

- to simulate odometric measurements (as described in subsection 6.1.2) and
- to tag places for automatic computation of metrics (tags are not used for SLAM).

Using the same $p_h$ and $p_l$ as in paragraph 6.2.2, disambiguation was achieved by traversing up to five edges (depending on datasets and locations within datasets).

Figures 6.11, 6.12, 6.13 and 6.14 show outputs of our algorithms for the various datasets. The implementation of local occupancy grids with wrap-around coordinates creates erroneous replicas of map portions outside the physical boundaries of the map. These artifacts are only present on the rigged output, not on the graph describing places and paths.

### 6.2.4 Robot experiments

Robot experiments are carried similarly to simulations. The robot (Figure 6.4) is switched on and left alone in the environment (Nano-Innov, Figure 6.6) until mapping is complete (no vertex has unexplored edges). An approximate floorplan of the environment is displayed on figure 6.15. On this floorplan, some obstacles such as chairs, sofas and tables have been represented as circles and ellipses in addition to the walls of the building represented with rectangles. The output of the algorithms for one of the mapping runs is displayed on Figure 6.16. For this 1.5km run, the cumulated odometric error amounts to 102m and the cumulated angular drift to about 1540° (Figure 6.17).

### 6.2.5 Results and discussion

Results for the datasets are displayed in table 6.2.

Globally, the produced dense maps look visually similar to the floorplans used for simulations, despite local deformations (notably visible on the middle part of the museum dataset, Figure 6.13). Eye inspection of the produced graphs does not reveal severe topological errors but local glitches can be seen, notably on the Cuzco (realistic) dataset (Figure 6.14) where some vertices have been duplicated. This apparent duplication is mostly caused by place detection errors, due to the situation of Figure 6.1, to dead ends sometimes being considered too shallow to have an edge leading to them or to discretization errors in the navigation system leading to edges not being followed correctly. Duplication causes the number of loops to increase relative to ground truth.

Mapping of the Cuzco (unrealistic) dataset (Figure 6.10) is the only experiment reported in Table 6.2 which was carried enforcing perfect vertex extraction. It is the reference sim-

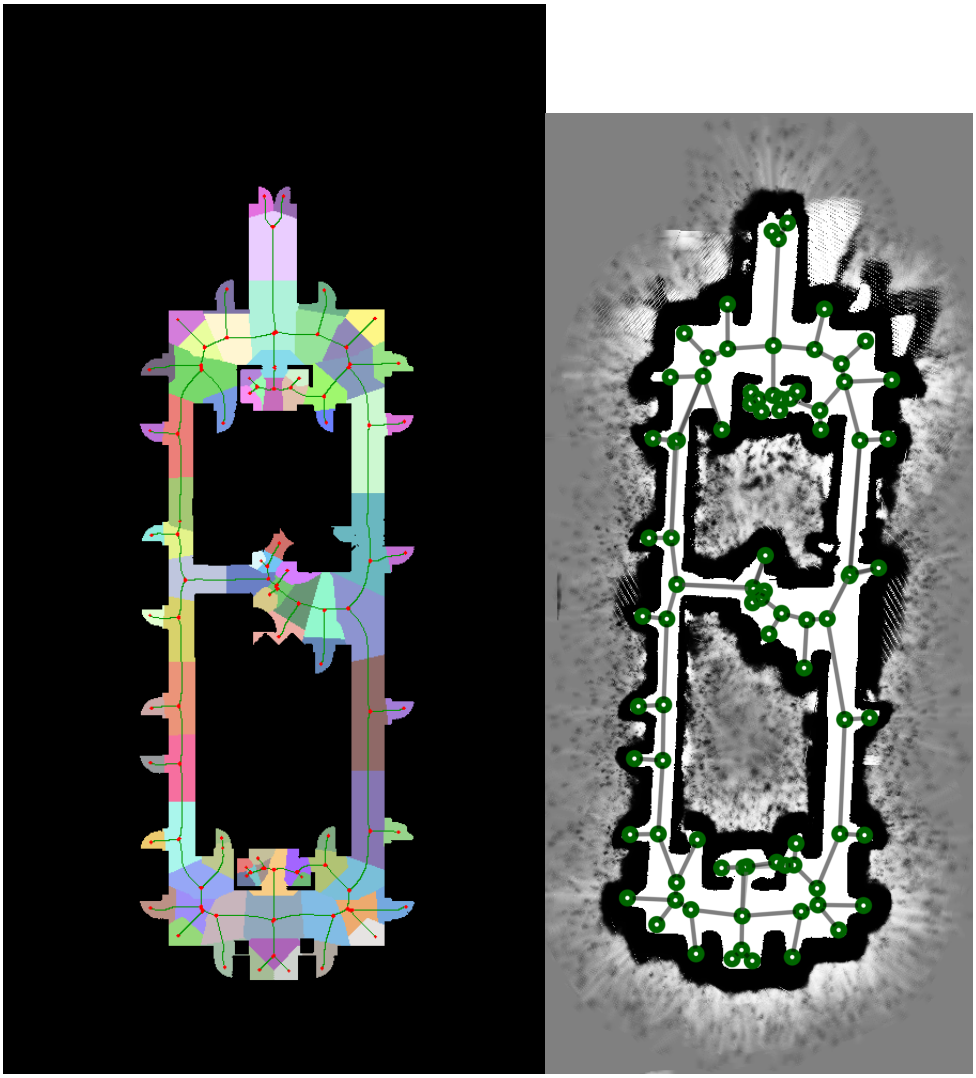


**Figure 6.11:** Floorplan used for Killian court simulations on the left and map obtained by our algorithms (the dense map is obtained by superposing edge-local occupancy grids inside a global occupancy grid through rigging with the graph) on the right. For this dataset, 300 pixels correspond to 48 feet.

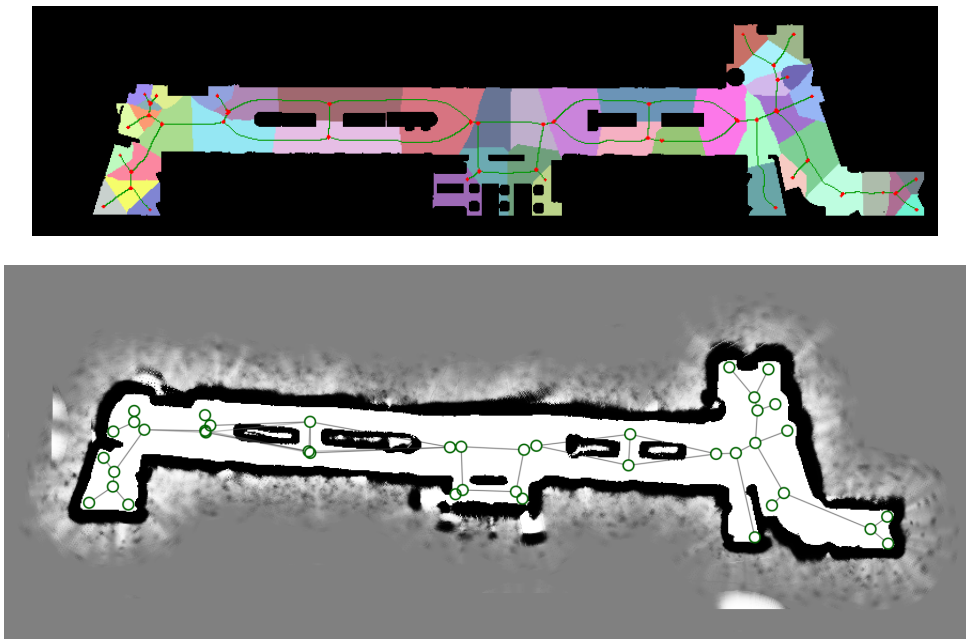


**Figure 6.12:** Floorplan used for Örebro simulations on top and map obtained by our algorithms at the bottom (the dense map is obtained by superposing edge-local occupancy grids inside a global occupancy grid through rigging with the graph).

| Dataset | size ($px^2$) | n. loops $\mathcal{G}_\mathcal{W}/\mathcal{G}$ | traveled d. ($px$) | odom. er. ($px$) | $\eta_r$ (%) | $\eta_l$ (%) | $\eta_\theta$ (°) | $f_t$ (%) | $f_{nm}$ (%) | $f_c$ (%) | $f_l$ (%) |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Killian | 350x900 | 4/4 | 24169 | 447 | 3.5 | 1.0 | 0.5 | 97.1 | 0 | 92 | 1.10 |
| Örebro | 848x215 | 5/5 | 11533 | 250 | 4.8 | 0.7 | 1.0 | 99.4 | 0 | 78 | 0.16 |
| M. (large) | 950x331 | 4/4 | 12825 | 250 | 4.0 | 0.6 | 0.6 | 96.7 | 0 | 98 | 0.03 |
| '' (small) | 950x331 | 5/11 | 53694 | 1092 | 7.4 | 1.3 | 0.8 | 97.5 | 0 | 92 | 1.20 |
| Cuzco u. | 2.4x1.9k | 184/186 | 282752 | 5932 | 3.0 | 0.2 | 0.2 | 99.7 | 0 | 95 | 0.02 |
| '' r. | 2.4x1.9k | 184/207 | 381616 | 7895 | 4.8 | 1.4 | 0.8 | 98.2 | 0 | 82 | 0.33 |
| Nano-Innov | 726x1494 | 5/5 | ~30000 | ~2000 | 10.7 | 1.4 | 1.6 | 100 | 0 | 90 | 0.05 |

**Table 6.2:** Simulation and experiment results. *M.* stands for *Museum*, *u.* for *unrealistic* (subsection 6.2.2) and *r.* for *realistic* (subsection 6.2.3). The exact size of the Cuzco dataset is 2397x1881 pixels. For the Nano-Innov dataset, 20px $\simeq$ 1m.

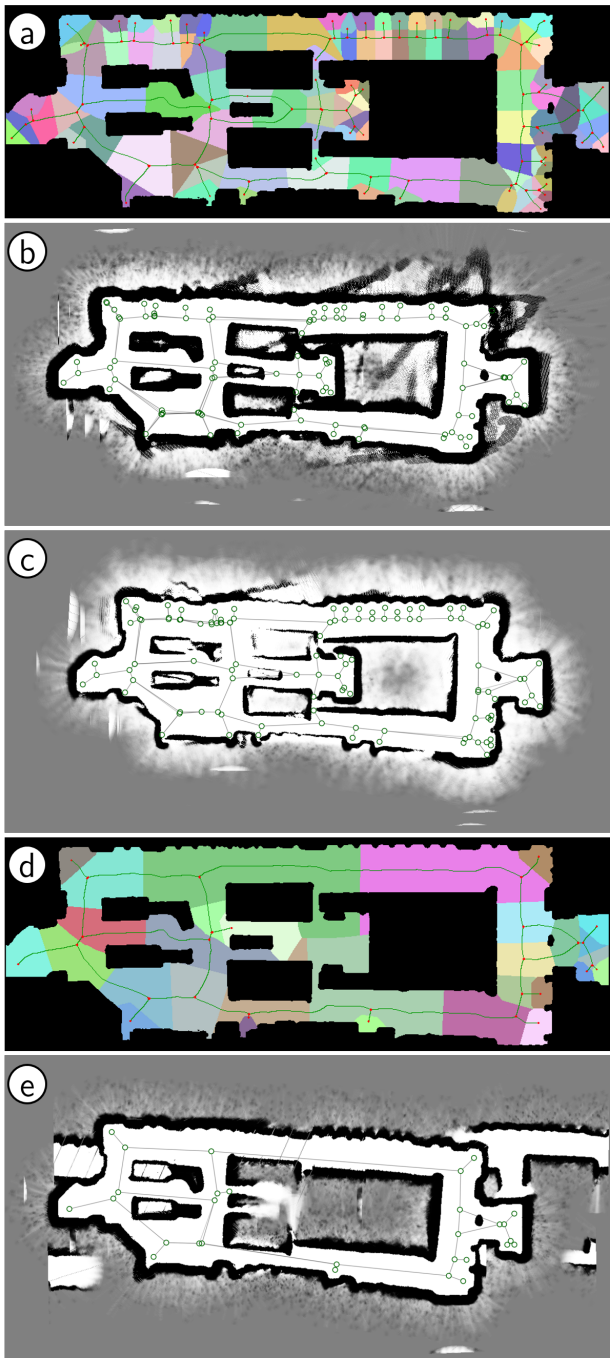



**Figure 6.13:** Simulations on the Museum dataset. (a) floorplan with GVG for a small (4px) robot. (b), (c) simulations with a small robot and different starting places. (d) floorplan with GVG for a large (8px) robot ignoring small places and narrow paths. (e) simulation with a large robot.

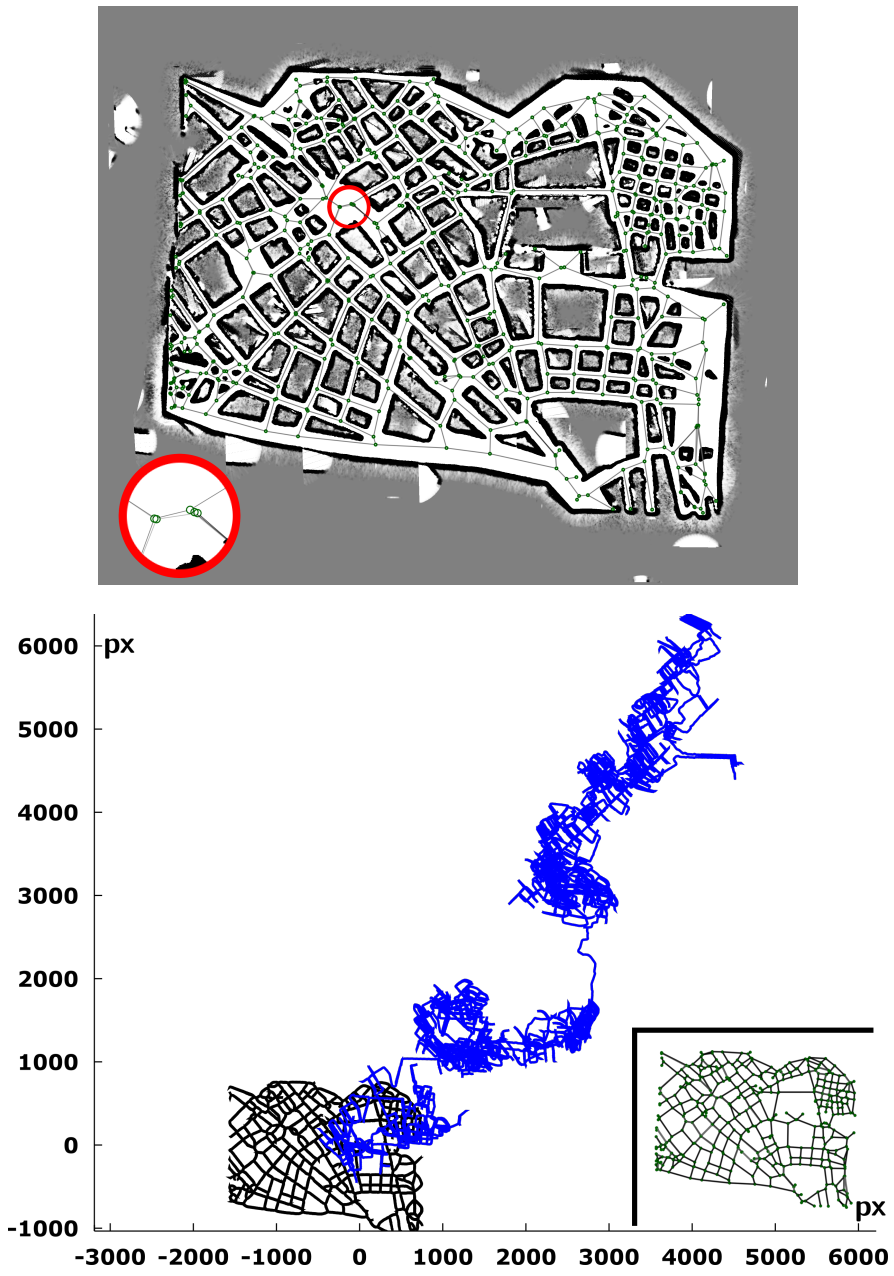



**Figure 6.14:** Final map of the Cuzco dataset obtained by our algorithms with realistic navigation and topology extraction (top) and trajectory followed by the robot (bottom) with actual position in black (bottom left) and odometry in blue. The graph of the top figure is reproduced in the lower right corner of the bottom figure for comparison. The robot started at (0;0). The translational bias of our sensor model creates a constant odometric drift to the top-right corner clearly visible on the figure. Sensor noise led to vertex duplication such as that shown with a red circle.

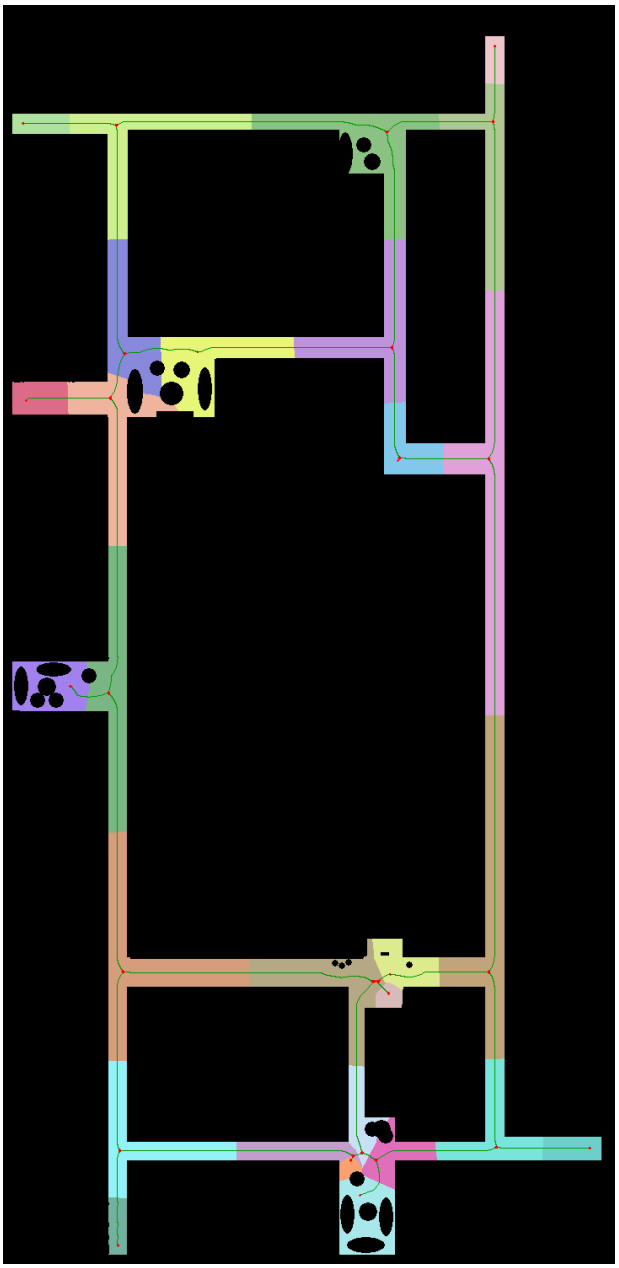


**Figure 6.15:** Floorplan used to compute metrics for the mapping runs in Nano-Innov. The topology described by this floorplan does not exatcly reflect that seen by the robot due to movable objects such as chairs, sofas or tables.

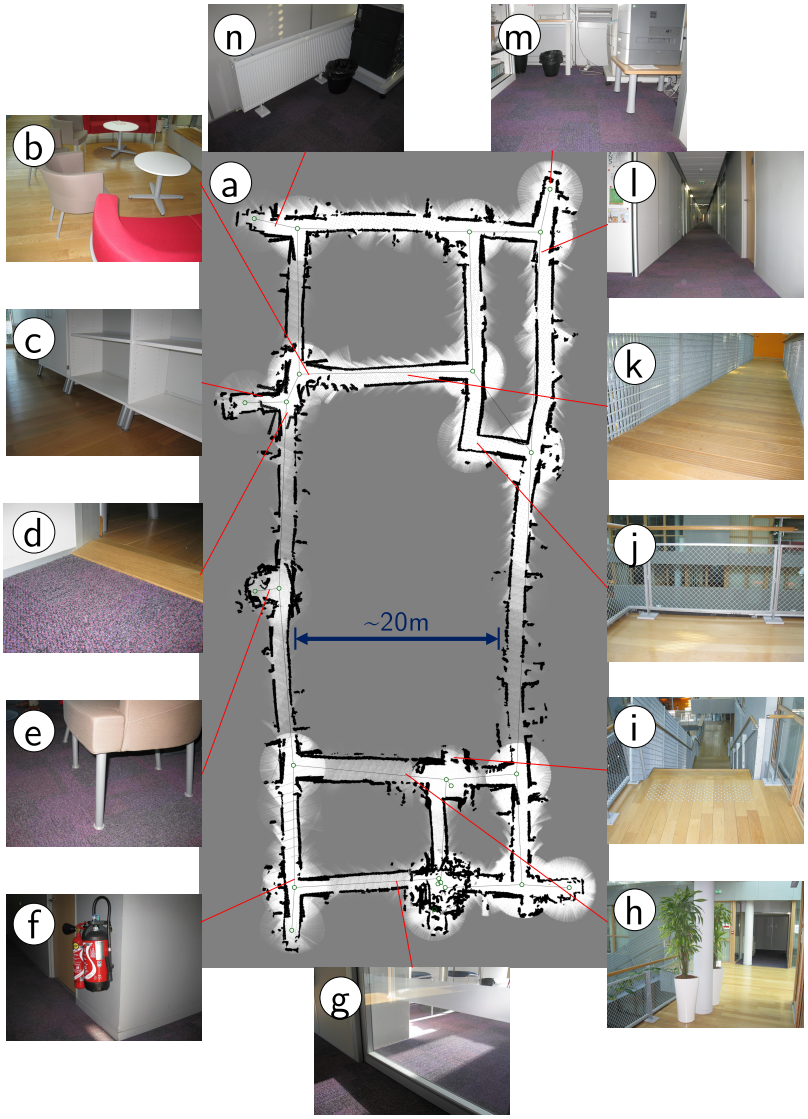



**Figure 6.16:** (a) Final map of the Nano-Innov dataset obtained by our algorithms using a pioneer robot, a Kinect 1 and a MEMS magnetometer. Various difficulties are present in the environment such as: (b,c,e,f,m) - obstacle whose size vary on the vertical axis, (d) - small steps and different floor coatings, (k) - uneven wooden floor, (h,i,j,k) - fence walls barely detected by the Kinect 1, (g) - glass doors barely seen by the Kinect 1, (b,h,n) - round-shaped obstacles, (n,l,h) - varying visible and infrared illumination, (d,i,j,n) - huge metal objects modifying the magnetic field. During experiments, stairways (i) were blocked by placing a cardboard box in front of them.

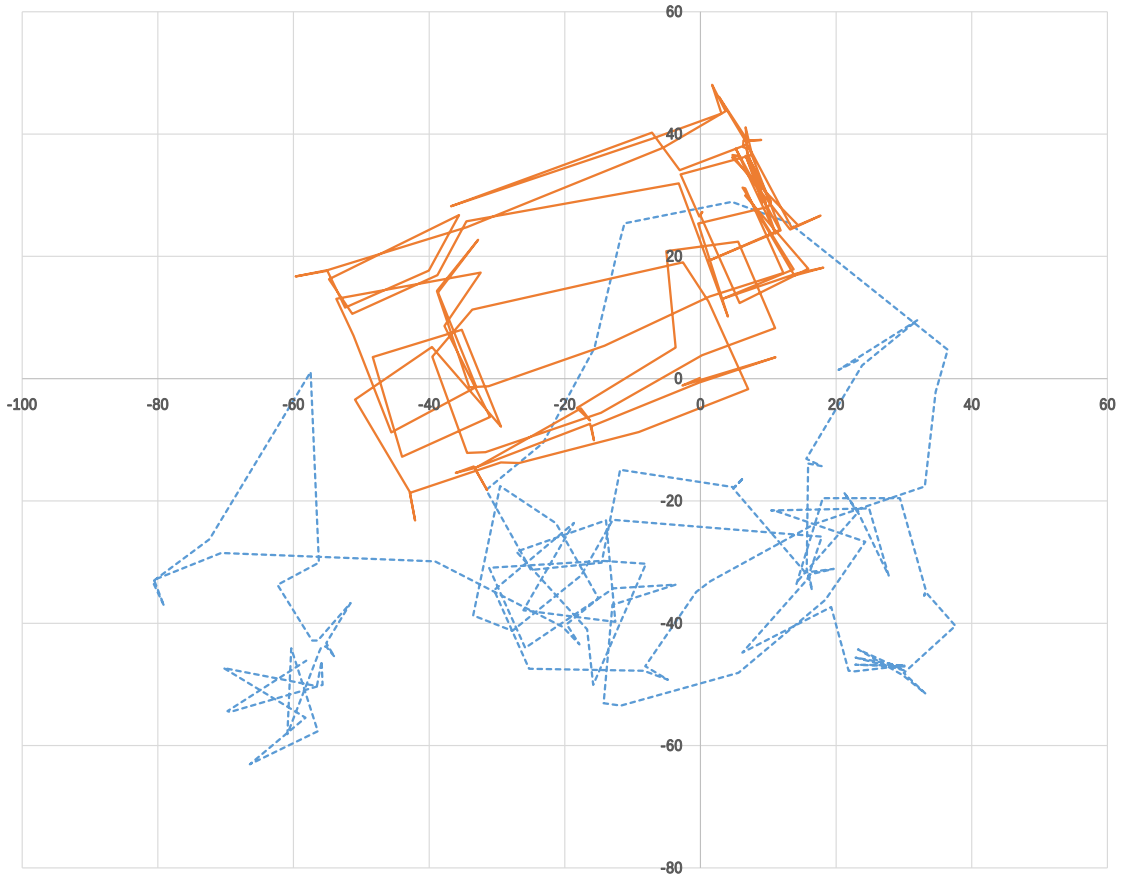



**Figure 6.17:** Coarse odometry (measured edge vectors) without (dashed blue line) and with (plain orange line) Large-Scale Angular Drift Compensation (LSADC), as described in chapter 5, section 5.3 and appendix 2. The first vertex of the map is at position $(0;0)$. Before reaching this vertex, the robot had already traveled around 50m in order to initialize the LSADC algorithm. Without LSADC, the trajectory is completely unrecognizable, almost random. With LSADC, the shape of Nano-Innov becomes recognizable despite an irreducible odometric drift that will only be corrected by the topological SLAM (chapter 5).

ulation to which other results can be compared. We can read metrics for this simulation the following way:

1. The topology of the map is very similar to the actual topology of the environment ($f_t > 90\%$). $f_t = 99.7\%$ is the ultimate limit that can be reached with the current navigation system and definition of metrics. Indeed, inspection of the logs produced during simulation reveals that the only errors are navigation errors ($p_h$ and $p_l$ are respectively high enough and low enough for perfect disambiguation).

2. Using the map to navigate is equivalent to using a GPS in $f_c = 95\%$ of all navigation attempts. As expected, $f_c < f_t$. $f_c = 95\%$ is the ultimate limit that can be reached with the current sensors and navigation system.

3. Even when navigation is not optimal, it is close to optimal, with an average additional traveled distance of $f_l = 0.02\%$.

4. The average error on lengths in the map is $\eta_l = 0.2\%$ which is an order of magnitude less than the constant anisotropic translational noise (2.25%) of the simulated odometric sensor.

5. The RMS error on angles in the map is $\eta_\theta = 0.2°$which is an order of magnitude less than the constant angular bias of 5°of the simulated compass. $\eta_\theta$ is a RMS error and thus does not count the average angular error, which happens to be 4.7°, that is almost exactly the simulated bias. Thus, the whole map is rotated uniformly according to sensor bias but once this effect is factored out, angle measurements are very accurate.

6. In average, $\eta_r = 3.0\%$ of disparity could not be absorbed by spring-mass optimization. $\eta_r$ is of the same order of magnitude as the constant odometric bias (2.25%). The values of $\eta_r$, $\eta_l$ and $\eta_\theta$ show that metrical errors in the map do not keep growing with the distance from the starting point of the run (while the raw odometric error does increase, as shown on Figure 6.14). This was expected since edge vectors and their associated pose and movement uncertainties are expressed relative to the edge's origin vertex and not relative to a single global origin. Even with an infinite number of edge traversals, $\eta_r$, $\eta_l$ and $\eta_\theta$ would probably not converge to 0 due to the odometric and compass biases.

On the Museum dataset, the 8-pixel robot was too big to enter the little "alcoves" on the top part of the map, as well as the middle space (which as a consequence was not mapped). The map produced with a bigger robot is coarser than that produced with a smaller robot. $f_c$ is higher and $f_l$ is lower for the larger robot, since it is easier to achieve topological correctness with a coarser representation.

For all runs, navigation from any vertex of the map to any other vertex was always

possible. Given that $f_{nm} = 0$ for all runs, this implies that the map can be used to navigate from any place to any other place in the environment. Additionally, the average added path length using the maps produced by our SLAM framework does not exceed $f_l = 1.20\%$ of the path length obtained when using a perfect (ground truth) map, which validates the practical use of our SLAM framework in a robot navigation context. For the Nano-Innov experiment, $f_l$ is as low as 0.05%, which means that the robot built a map that is almost optimal for path planning.

Given the visual similarity between ground truth and the produced maps, we could have expected higher values of $f_c$, exceeding 90% in all cases. In fact, $f_c$ is lower than 90% for the Örebro (Figure 6.12) and Cuzco-realistic (Figure 6.14) datasets because they exhibit multiple paths of similar lengths which get swapped (the length difference between the path of optimal length and that of second optimal length is negligible, so that the map may list the path of second optimal length as being shorter due to sensor noise). Since the actual additional path length $f_l$ remains under 1% for these datasets, the impact of switching paths on navigation is negligible. As expected, the "Museum - large robot" run has the highest $f_c$, because it is topologically the simplest and the map can be used to traverse each path in both directions. On this dataset, $f_c \neq 100\%$ only because the environment contains paths of nearly equal length.

$\eta_r$ is up to twice higher for the Museum - small robot (Figure 6.13, bottom) run than for any other simulation. This is probably because our SLAM algorithms actually confused two of the little alcoves in the top right of the map (there is an array of at least 8 of these) and performed an incorrect loop closure during one traversal. This structural error must have been corrected during a later traversal since the final map does not show such an incorrect loop. This temporary structural error occurred since $p_h$ and $p_l$ were not taken respectively high enough and low enough. The higher value of $f_l$ for the Museum - small robot dataset is likely caused by bad tags being attributed to some places during the incorrect loop closure. The higher value of $\eta_r$ for the Nano-Innov robot experiment reveals that less precise odometric sensors were used for experiments compared to simulations.

Finally, the total distance traveled in order to produce the maps in our framework is comparable to state of the art approaches. For instance, our simulated robot traveled 1.18 km for the Killian dataset (Figure 6.11), for which Bosse et al. (2004) achieved mapping after the robot traveled approximately 2.2km. For the Nano-Innov dataset, the robot traveled 1.5km. For the realistic Cuzco dataset (Figure 6.14), each edge was traversed in average 3.34 times in each direction. Since we use single-way edges, each dual-way edge has to be traversed at least once in each direction. Fiddling with $p_h$ and $p_l$ allows to trade off topological map accuracy ($f_t$, $f_c$, $f_{nm}$) for mapping speed.

## 6.3 Other PNSLAM missions

As mentioned in chapter 3, *exploration* of an environment is a form of goal-directed navigation where the goal is not reachable. In this section, we show results of a robot carrying the more generic PNSLAM mission *go to approximate destination* or *find treasure around specified position*. This mission is a generalization of:

- find a treasure (position unknown),
- explore the environment (or find a non-existing treasure) and
- go to a precise destination known by its coordinates (find a treasure at some known position).

Figure 6.18 shows a typical *find treasure around specified position* mission carried in the same conditions as the exploration missions described in the previous paragraphs, using EDNA* with a low risk factor ($\alpha = 1$, see chapter 3) which promotes exploration. The *find treasure around specified position* mission can be viewed as an exploration mission around the position where the treasure should be. For figures 6.19 and 6.20, each time the robot completed a treasure hunt (a,b,c,f) , it was given the approximate position of a new treasure to hunt. These figures show the strong integration of planning, navigation and SLAM. Indeed, the robot explores the environment when it thinks there is a benefit in exploration and navigates on known parts otherwise, as described in chapter 3. Loop closures (d,e) are performed when the current location is ambiguous. The SLAM framework (chapter 5) tolerates errors from the navigation system (chapter 4) while EDNA* (chapter 3) decides which way to go, occasionally making a detour to close a loop. As far as we know, it is the first time that such an integration between planning, navigation and large-scale SLAM is demonstrated.

## 6.4 Conclusion

In this chapter, we demonstrated that a real robot using the approach described in:

1. chapter 3 for (exploratory) path planning,
2. chapter 4 for local navigation and topology extraction and
3. chapter 5 for SLAM

was able to move from any place to any other place of an environment reachable using

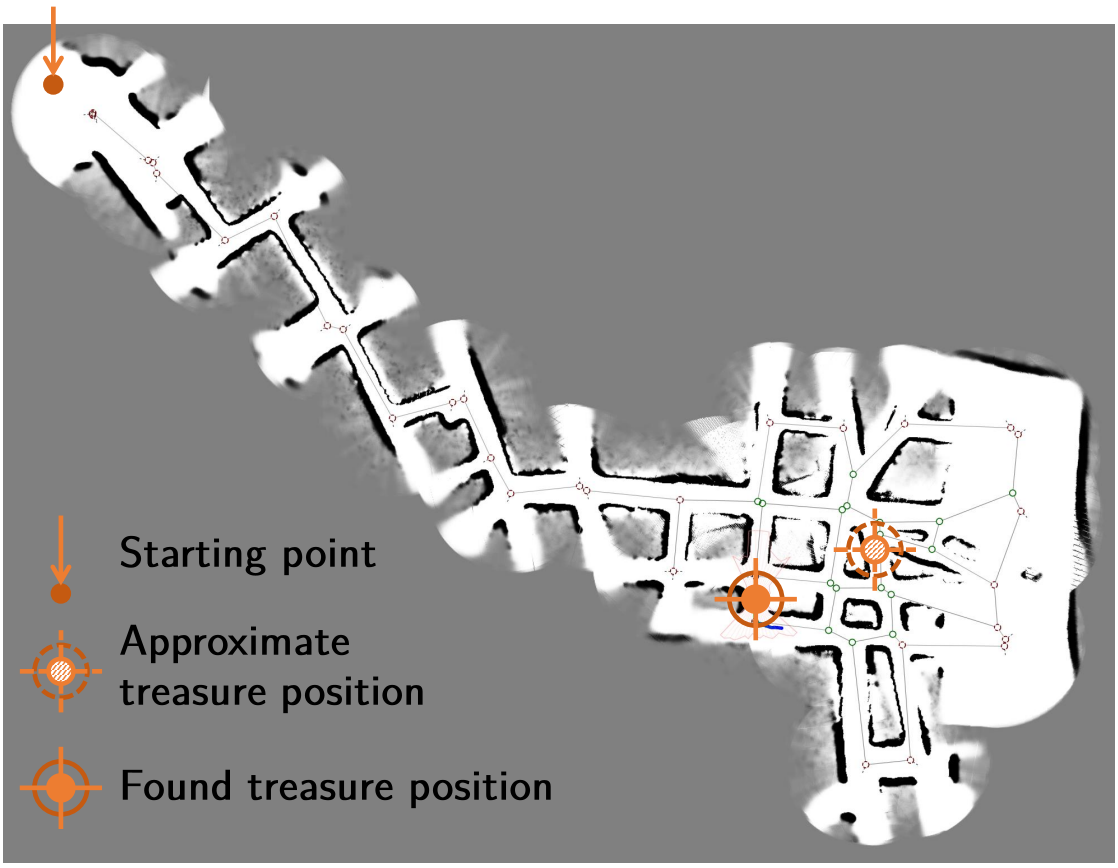



**Figure 6.18:** Finding a treasure in *Cuzco* around a position given as hint. The robot first goes directly towards the hint position then searches around it for the treasure. Only the necessary part of the environment is explored and mapped, which is one of the main characteristics of PNSLAM.

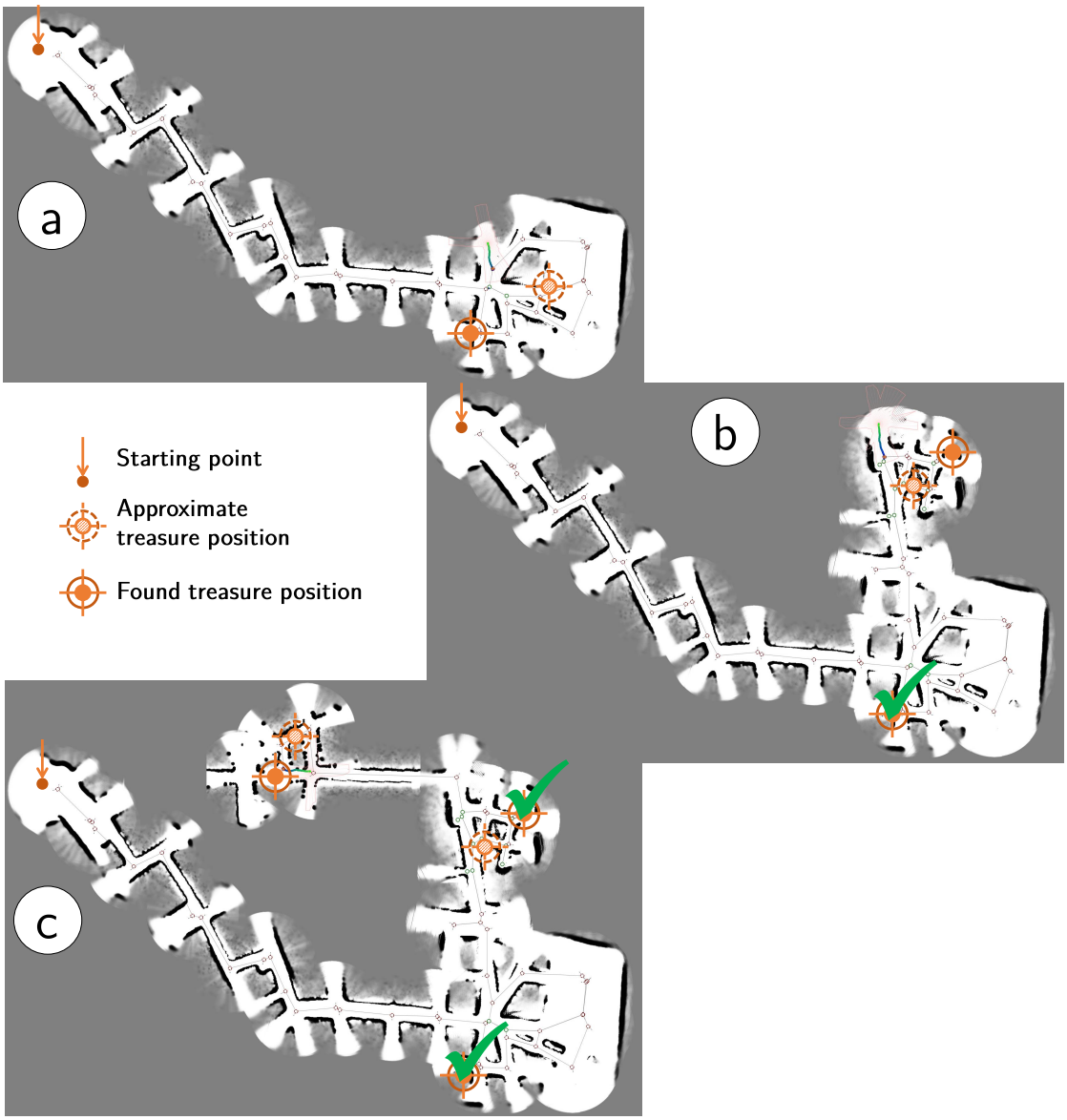



**Figure 6.19:** Multiple treasure hunts 1/2 (explanations in section 6.3).


**Figure 6.20:** Multiple treasure hunts 2/2 (explanations in section 6.3).

2D movements (the environment does not need to be 2D and can notably include bridges and tunnels). We showed using simulations that environments with tens or hundreds of loops would not pose a problem to the robot. Tasks that can be performed by a robot whose control system is implemented as described in this thesis include:

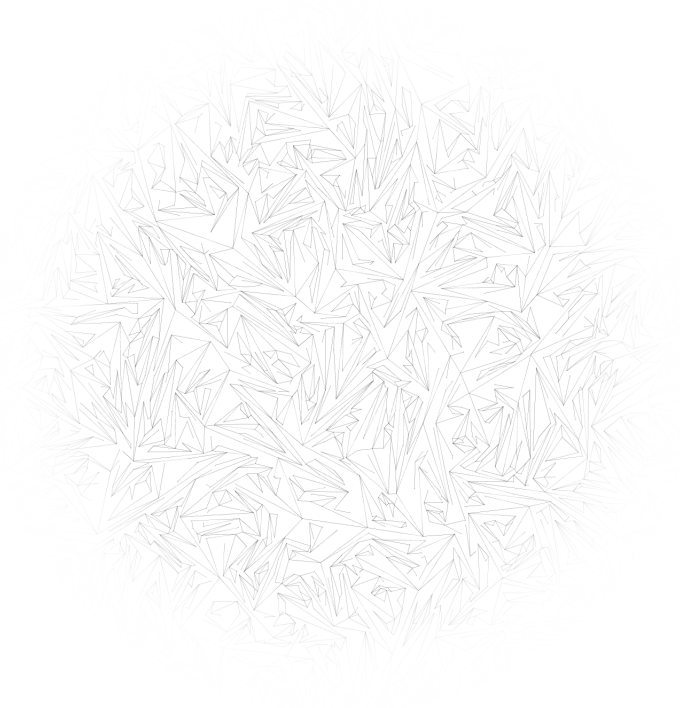
- going to a semantic destination, such as a specific place,
- going to an exact or approximate position,
- exploring an environment and
- finding a treasure.

These tasks cover the spectrum of PNSLAM tasks described in chapter 2.

We introduced four metrics to assess the topological and metrical correctness of a map and its usability for a navigation task. We used these metrics to evaluate our SLAM technique in various environment, simulated and real. We observed that our framework enables a robot to navigate from any reachable point in the environment to any other using only odometry, a compass and local distance measurements provided for instance by a ring of sonar sensors or a Kinect camera. The obtained maps were visually equivalent or better than the state of the art on benchmark datasets. We proposed a new dataset for simulations, *Cuzco*, with 184 loops, far more challenging than the existing benchmarks. We showed that our SLAM is visually correct and close to topological correctness even in this case. Using our approach, the mobile robot performing the mapping task was able to autonomously navigate in the environment with path lengths within 1.20% of the theoretical optimum in simulated environments. Experiments using a Pioneer 3 robot in an office building showed that the PNSLAM approach could build maps allowing the robot to navigate in the environment with path lengths within 0.05% of their theoretical optimum in average.

# Part III

# Resource management and lifelong operation



Huge graphs raise memory and computing power issues.

# 7 Handling finiteness of computing power

*"Fancy algorithms are slow when N is small, and N is usually small."*
**Rob Pike**

In this chapter, we study both theoretical time complexities and running times of (exploratory) planning (chapter 3), navigation and topology extraction (chapter 4) and SLAM (chapter 5). We narrow down the performance bottleneck of PNSLAM to the SLAM component and show that the theoretical time complexity of our SLAM framework (chapter 5) can be reduced to $\mathcal{O}(1)$ in the number of edges or vertices on the map at a given time. Sublinear complexity is critical when operating in very large environments for an unlimited period of time (lifelong operation). The constant time behavior is confirmed experimentally.

## 7.1 PNSLAM computational loads

In order to implement Lifelong Exploratory Navigation, we have to check whether PNSLAM can be used to navigate in huge environments with potentially thousands of loops. In this section, we give a theoretical and practical analysis on compute loads and running time of Planning, Navigation and SLAM as implemented in chapters 3, 4 and 5.

### 7.1.1 Planning (chapter 3)

We showed in chapter 3 that the effective algorithmic complexity of A* and EDNA* on the graphs used throughout Part II should lie between $\mathcal{O}(||OF||)$ and $\mathcal{O}(||OF||^2)$, where $O$ and $F$ are respectively the origin and destination of the A*/EDNA* run.

Importantly, this effective complexity does not depend on the total size of the graph, only on the minimum length of the trajectory to compute. As long as the length of all trajectories that should be computed is bounded, which is always true in real-life navigation scenarios, the complexity of A* and EDNA* does not raise issues in Lifelong Exploratory Navigation.

In practice, even on the *Cuzco* dataset (Figure 6.5) or on the 10 000 vertex graphs used in chapters 3 and 8, the running time of EDNA* on $\mathcal{G}$ is negligible compared to the timescales of sensors and actuators, no matter $O$ and $F$. Indeed, each vertex expansion step of EDNA* translates to only a handful of assembly instructions.

### 7.1.2 Navigation (chapter 4)

Theoretically speaking, navigation runs in constant complexity in the total size of the environment. Indeed, all occupancy grids used are of fixed size and computations carried on these grids do not vary with the total size of the environment. Thus, navigation will never be a hindrance when transitioning from simple missions to lifelong operation.

In practice, navigation is the component that uses the most processing power. The most costly part is topology extraction, whose running time is plotted on Figure 4.10. For experiments reported in chapter 6, topology extraction on floating-point occupancy grids typically takes a dozen of milliseconds to complete. Including sensor delays, the sensing-and-command loop induces a maximum delay of around 50ms using a naive single-threaded $C$ implementation. This delay could be reduced using a parallel and/or hardware implementation of topology extraction if faster response times are needed. However, a 50ms delay is still much lower than the typical minimum response time of a human being, which is around 200ms (Eckner, Kutcher, and Richardson, 2010).

### 7.1.3 SLAM (chapter 5)

The theoretical algorithmic complexity of our SLAM approach is the same as that of the ATLAS framework (Bosse et al., 2004), that is $\mathcal{O}(N.log(N))$ where $N$ is the number of vertices on the map at a given time. The $\mathcal{O}(N.log(N))$ process is the search for candidate vertices performed during loop closure. The search for candidates returns up to $N$ candidates, each of which carrying a signature that should be matched against the current vertex's own signature. While *finding* vertices using Dijkstra's algorithm is an $\mathcal{O}(N.log(N))$ process, the worst case $\mathcal{O}(N)$ signature matching step is by far the most time-consuming part of the SLAM framework, with running times amounting to multiple seconds on Cuzco (Figure 6.5) for a single-threaded implementation. The robot remains motionless while performing signature matching. Thus, it has to stay motionless

for multiple seconds, which is not admissible in a real-life situation and is an incentive towards trying to reduce the number of signatures being matched.

Spring-mass optimization is an optional step whose complexity depends on the implementation. Using matrix inversion, the complexity of spring-mass optimization lies between $\mathcal{O}(N^2)$ and $\mathcal{O}(N^3)$. In chapter 5, section 5.5, we explained that it was possible to use a relaxation algorithm to perform spring-mass optimization locally in the vicinity of a loop closure point. The complexity of local spring-mass optimization needs to be ascertained.

## 7.2 Achieving sub-linear complexity in SLAM

In order to reduce the computing power necessary for the SLAM framework, we propose to reduce the complexity of the whole SLAM framework to constant in the number $N$ of vertices on the map.

### 7.2.1 Sub-linear loop closure in SLAM

To our knowledge, while linear complexity SLAM algorithms can be found (Liang, Shoudong, and Dissanayake, 2013), no existing SLAM algorithm is able to achieve an algorithmic complexity strictly lower than $\mathcal{O}(N)$, where $N$ is the number of features, landmarks or places that can be mapped in the traversed environment. FastSLAM (Montemerlo et al., 2002) achieves a $\mathcal{O}(log(N))$ complexity assuming perfect landmark identification, a hypothesis which corresponds to assuming perfect loop closure in a hybrid metrical/topological approach. No linear time landmark identification procedure is described in (Montemerlo et al., 2002). In our approach, the theoretically hard and computationally expensive problem is precisely loop closure. In order to perform logarithmic complexity landmark identification, Montemerlo et al. (2002) propose to reference landmarks inside a kd-tree. A kd-tree (quadtree in 2D) can indeed be used to address vertices using their spring-mass optimized coordinates (Figure 7.1). Addressing vertices using their coordinates is useful to give the robot a mission but it is not necessary for (PN)SLAM.

### 7.2.2 Spring-mass optimization in logarithmic and linear complexity

Local spring-mass optimization operates within a bounded range of a loop closure point ($n_{max}$ nearest neighbors), so that with a bounded degree of each vertex, the number

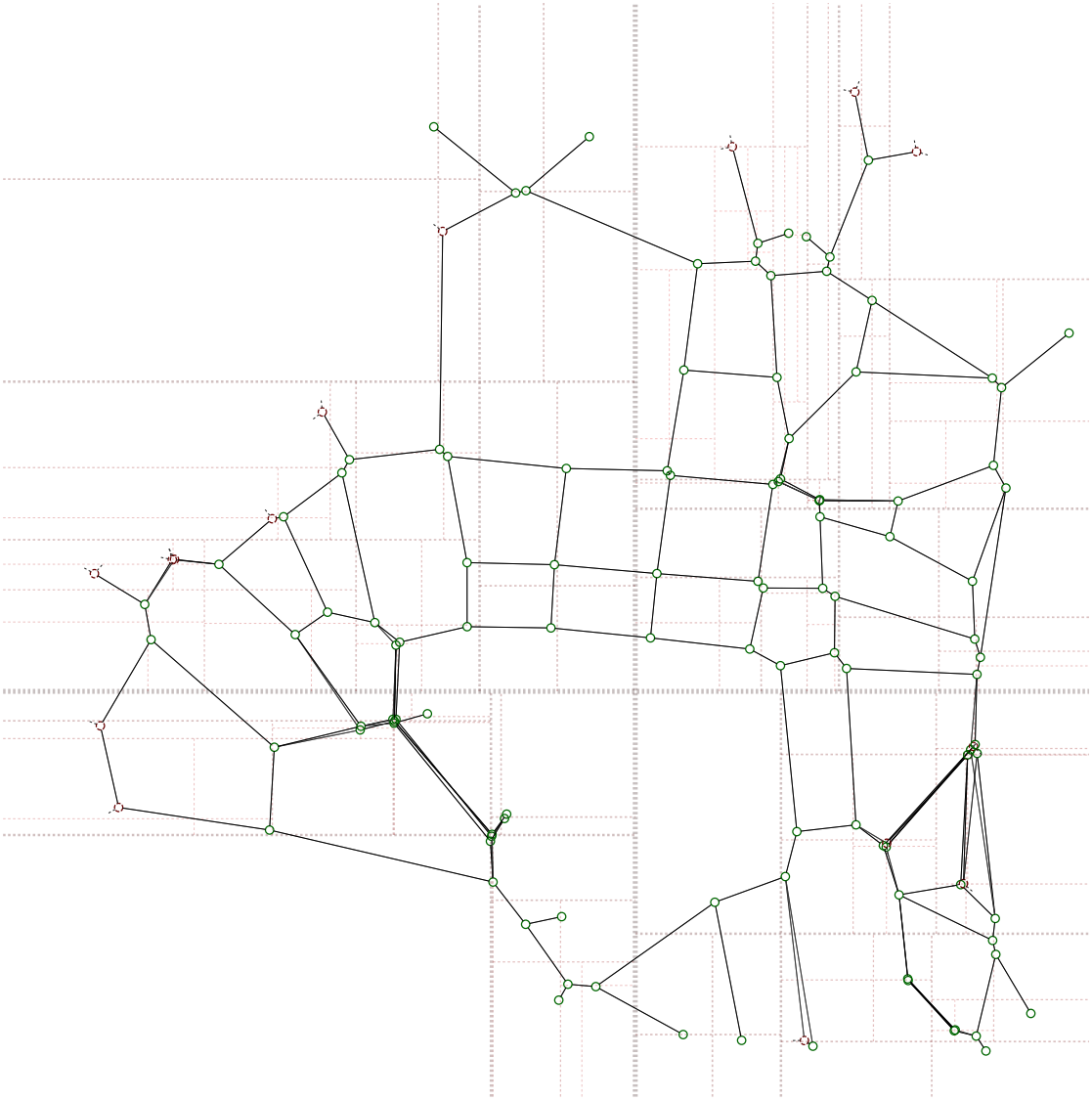


**Figure 7.1:** A quadtree (dashed lines) allows accessing vertices in $\mathcal{O}(log(N))$ time complexity using their spring-mass optimized coordinates, where $N$ is the number of vertices currently on the graph. Our quadtree implementation is asymmetrical, which is necessary since the size of the environment and its extension in all four directions from the origin is not known during mapping.

of vertices moved is polynomial in $n_{max}$, likely between $\mathcal{O}(n_{max})$ and $\mathcal{O}(n_{max}^2)$. This number does not depend on $N$.

If vertices need to be referenced inside a quadtree using their spring-mass-optimized position, each local spring-mass optimization step removes from the quadtree each vertex whose position needs to be updated. At the end of the optimization step, vertices get reintegrated into the quadtree. Since individual insertions and deletions from the tree can be performed in $\mathcal{O}(log(N))$, the total complexity of a single local spring-mass optimization should lie between $\mathcal{O}(n_{max}log(N))$ and $\mathcal{O}(n_{max}^2log(N))$.

### 7.2.3 Reducing the number of candidates for vertex signature matching

Theorem 8 demonstrates that it is possible under some mild hypotheses to give a bound independent of $N$ on the number of candidate vertices to be considered for any loop closure at vertex $O$.

**Theorem 8.** *Suppose that the following conditions are met:*

1. *The range $U = U(O)$ in which to look for candidate vertices $F_i$ which could be vertex $O$, is known and independent of the total size of the graph and*

2. *the minimum distance between two vertices, $d_{min}$, is strictly positive*

*Then, we can find a bound independent of $N$ on the number of candidate vertices $F_i$.*

*Proof.* The maximum surface density of vertices is $\rho = \frac{1}{d_{min}^2\sqrt{5}/2}$ using an optimal compacity hexagonal 2D packing. We are looking for candidate vertices $F_i$ within range $U$ of $O$, that is $||F_i - O|| \leq U$. There is thus a maximum of $\mathcal{O}(\pi U^2 \rho)$ candidate vertices $F_i$ in range $U$ of $O$. □

While hypothesis 2 of theorem 8 is easy to achieve, hypothesis 1 is somewhat restrictive. It combines two constraints:

1. The maximum uncertainty accumulated along a path before a loop closure is bounded, and a consistent estimate of this bound is known, and

2. the distortion created by spring-mass optimization is bounded.

It is possible to bound the distortion created by spring-mass optimization by constraining movements in the relaxation algorithm. When finding the optimal position of a single

vertex $u$, it is sufficient to ensure that:

$$\forall e \in \mathcal{E} | S(e) = u \text{ or } T(e) = u, ||T(e) - S(e) - e|| \leq \delta_e \tag{7.1}$$

where $\delta_e$ is the uncertainty associated to edge $e$ ($\delta_e = \delta_{eo} + \delta_{ed} + \delta_{t_e}$, see chapter 5). Note that the constraints can be implemented with Lagrange multipliers when relaxation is not used. Since all error estimates in our model are consistent, it is always possible to find a spring-mass solution verifying the constraints (world positions $\mathbf{R}$ and edge vectors $\mathbf{r}$ do verify all constraints, so there is at least one solution).

This constraint is represented on Figure 7.2. When it is enforced, the following equation holds:

$$\forall (u, v) \in \mathcal{V}^2, \forall P_{u \to v}, ||v - u|| \leq \delta_{P_{u \to v}} \tag{7.2}$$

where $P_{u \to v}$ is a path from $u$ to $v$ on $\mathcal{G}$ and $\delta_{P_{u \to v}}$ is the uncertainty accumulated along $P_{u \to v}$ using the non-sequential formula. Note that using the non-sequential formula becomes mandatory even in cases where the sequential formula could be used.

Ensuring that the maximum accumulated uncertainty along a path before a loop closure is bounded is not always possible since it notably depends on which (exploratory) planner and navigation algorithm the robot uses to traverse uncharted space. For instance, in a planar environment, if the robot follows walls and always turns left in uncharted space, each new loop discovered corresponds to an obstacle being circled by the robot. Consequently, the maximum uncertainty along a loop is that accumulated by circling the largest obstacle in the environment in a context where uncertainty is proportional to traveled length. If the size of the largest obstacle is known, the maximum accumulated uncertainty can be computed. Thus, theorem 7 is preserved.

With more complex exploratory planners and navigation algorithms (such as EDNA* with an arbitrary risk heuristic), it may not be possible to determine a maximum uncertainty along any loop traversed by the robot before returning to a known place. While setting a highly overestimated maximum uncertainty value will almost always work for practical situations, theorem 7 does not hold anymore since loop closure candidates may be skipped by Dijkstra uncertainty projection when the uncertainty goes over the (highly overestimated) value.

### 7.2.4 Multiple target Dijkstra with bounded complexity

In this subsection, we show that under certain conditions, Dijkstra's algorithm has a bounded complexity $\mathcal{O}(U^4/d_{min}^4)$ (theorem 9) with $U$ a spatial range independent of $N$. We use this result to project uncertainty with a complexity independent of $N$.

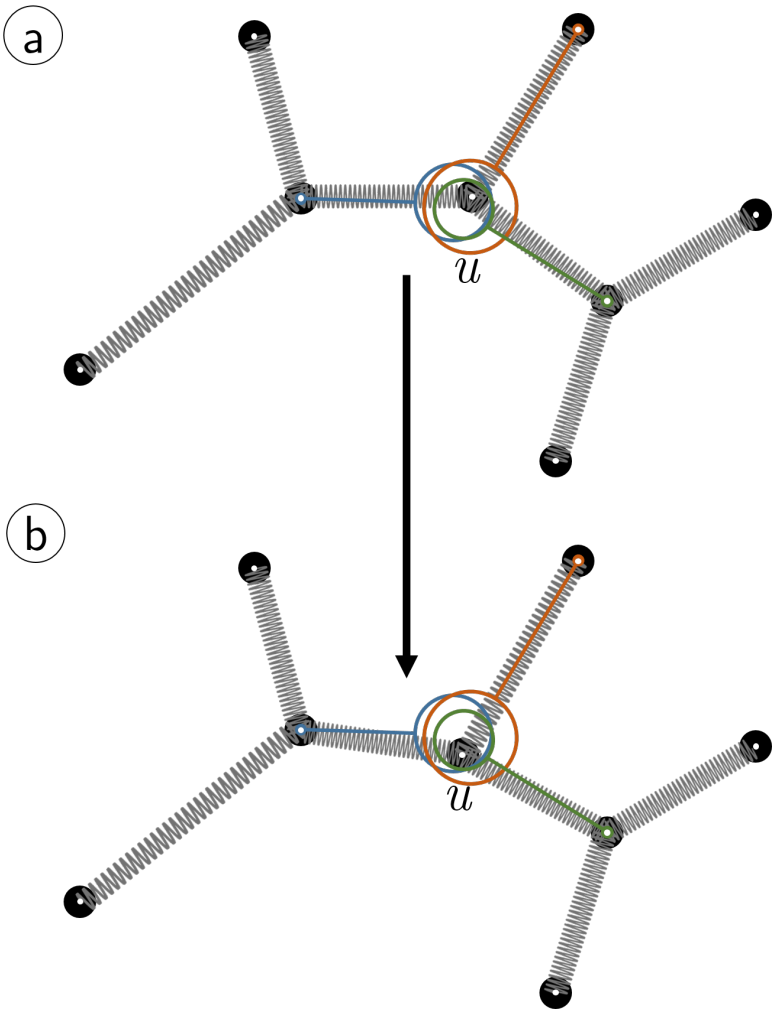



**Figure 7.2:** Constraining spring-mass optimization on vertex $u$ so that vertex position differences remain consistent with edge measurements and pose and movement uncertainties defined in section 5.3 (represented with circles). It is necessary that the vertex position remains within the three uncertainty balls for hypothesis 1 of theorems 8 and 9.

Dijkstra's algorithm is a graph traversal or pathfinding algorithm able to find the shortest, or one of the shortest (if there are multiple paths of equal length $D(O, F_1)$) path on a graph from vertex $O \in \mathcal{V}$ (for "origin") to at least one destination vertex $F_1 \in \mathcal{V}$. The algorithm has a forward phase, where target vertices are reached from an origin vertex, and a backward phase where a path is computed from the target vertices to the origin vertex. Instead of simply looking for a path, we project uncertainty (forward phase) and sum edge vectors (backward phase) along said path.

**Theorem 9.** *Suppose that the following conditions are met:*

1. *The range $U = U(O)$ in which to look for candidate vertices $F_i$ which could be vertex $O$, is known and independent of the total size of the graph,*

2. *the degree (number of outgoing edges) of each vertex is bounded,*

3. *the minimum uncertainty projected between any pair of vertices is bounded relative to the Euclidean distance between these vertices, and*

4. *the minimum distance between two vertices, $d_{min}$, is strictly positive*

*Then, we can devise a Dijkstra variant to perform uncertainty projection and edge vector summation from all candidates $F_i$ to $O$ in $\mathcal{O}(U^4/d_{min}^4)$ time complexity.*

*Proof.* Let $||V_2 - V_1||$ denote the Euclidean distance between vertices $V_1 \in \mathcal{V}$ and $V_2 \in \mathcal{V}$ in $\mathbb{R}^2$. Let $D(V_1, V_2, P)$ denote the maximum uncertainty projected from $V_2$ to $V_1$ using graph edges on a specific path $P$, using equation 5.7 on parts that were sequentially traversed and equation 5.15 otherwise. The maximum uncertainty of a single edge $e$ is $\delta_e$. $D(V_1, V_2) = min_P(D(V_1, V_2, P))$ is the minimum projected uncertainty on any path from $V_2$ to $V_1$.

The forward phase of Dijkstra's algorithm works by expanding vertices $X$ in order of increasing heuristic $h = D(O, X)$, where expanding a vertex means adding its immediate neighbors $Y_i$ to a heap along with their heuristic $D(O, Y_i)$. Dijkstra can be implemented as A* (see Algorithm 11) with $\mathcal{H} = 0$. The time complexity of the algorithm for a graph containing $n_v$ vertices and $n_e$ edges is $\mathcal{O}((n_v + n_e)log(n_v))$ with a heap implementation. If the degree of each vertex is bounded, which is true for our SLAM application, this complexity comes down to $\mathcal{O}(n_v log(n_v))$.

Now, suppose that the minimum uncertainty projected between any pair of vertices is bounded relative to the Euclidean distance: $\exists k \in \mathbb{R}^{*+}, \forall (X, Y) \in \mathcal{V}^2, D(X, Y) < k.||Y - X||$. We call $k$ the maximum suboptimality ratio. $k$ describes the ratio between worst-case on-graph distance (here, uncertainty) and Euclidean distance. In finite environments and as long as a path between $X$ and $Y$ exists, we can always find such a

$k$:

$$k \leq k_{max} = \frac{\sum_{(X,Y)\in\mathcal{V}^2, Y \text{ nb. } X} D(X,Y)}{\min_{(X\neq Y)\in\mathcal{V}^2} ||Y-X||} \tag{7.3}$$

where $k_{max}$ is finite since $\min_{(X\neq Y)\in\mathcal{V}^2} ||Y-X|| > d_{min}$:

$$k_{max} < \frac{\sum_{(X,Y)\in\mathcal{V}^2, Y \text{ nb. } X} D(X,Y)}{d_{min}} \tag{7.4}$$

where nb. means "neighbor of". $\sum_{(X,Y)\in\mathcal{V}^2, Y \text{ nb. } X} D(X,Y)$ is necessarily longer than any path returned by Dijkstra (since Dijkstra can only return a path where each edge is traversed at most once). Conversely, $\min_{(X\neq Y)\in\mathcal{V}^2} ||Y-X||$ is necessarily shorter than any Euclidean distance between distinct vertices of the graph. Thus, $k_{max}$ describes the worst case detour using Dijkstra on a given graph.

We are looking for candidate vertices $F_i$ within range $U$ of O, that is $||F_i - O|| \leq U$. If $k$ exists, Dijkstra expansion will have reached all the $F_i$ and found $D(O,F_i)$ at least when $h \geq k.U$ since $\forall X \in \mathcal{V}, ||X-O|| \leq U \Rightarrow D(O,X) < k.||X-O|| <= k.U$.

If vertices represent physical places and $d_{min} > 0$, the maximum surface density of vertices is $\rho = \frac{1}{d_{min}^2 \sqrt{5}/2}$ using an optimal compacity hexagonal 2D packing. Then there are at most around $\pi k^2 U^2 \rho$ vertices in range $k.U$ around $O$. Since we can stop the algorithm as soon as $h \geq k.U$, Dijkstra's algorithm practically runs on a subgraph containing a maximum of around $\pi k^2 U^2 \rho$ vertices, so that the time complexity falls down to $\mathcal{O}(\pi k^2 U^2 \rho . log(\pi k^2 U^2 \rho)) = \mathcal{O}(U^2/d_{min}^2 log(U^2/d_{min}^2))$.

At the end of the forward phase, all vertices $X$ reached by the algorithm are associated with their projected uncertainty to $O$. In the backward phase and for each of the candidate vertices $F_i$ (for which a path of minimal projected uncertainty is known from the forward phase), edge vectors are summed on the path of minimal projected uncertainty. This path cannot contain loops, since these would cause an increase in projected uncertainty and the path would not be that of minimal uncertainty (corollary 2). Thus, each vertex in the $U$ range around $O$ can only be traversed once by the path. As a consequence, the path contains a maximum of $\mathcal{O}(\pi k^2 U^2 \rho)$ vertices so that edge vector summation runs in $\mathcal{O}(U^2/d_{min}^2)$.

Since there is a maximum of around $\mathcal{O}(\pi U^2 \rho)$ candidate vertices $F_i$, the backward phase can compute vector sums from each $F_i$ to $O$ in $\mathcal{O}(U^4/d_{min}^4)$ complexity. □

Note that hypotheses 2 and 4 are always true for SLAM. Hypothesis 3 is always true in practice, since $\delta_e < ||\vec{r_e}|| + \nu$ for an arbitrary edge $e \in \mathcal{E}$ in the vast majority of cases (where $\nu$ is some highly overestimated vertex detection error). Finally, hypothesis 1 was discussed in the previous subsection.

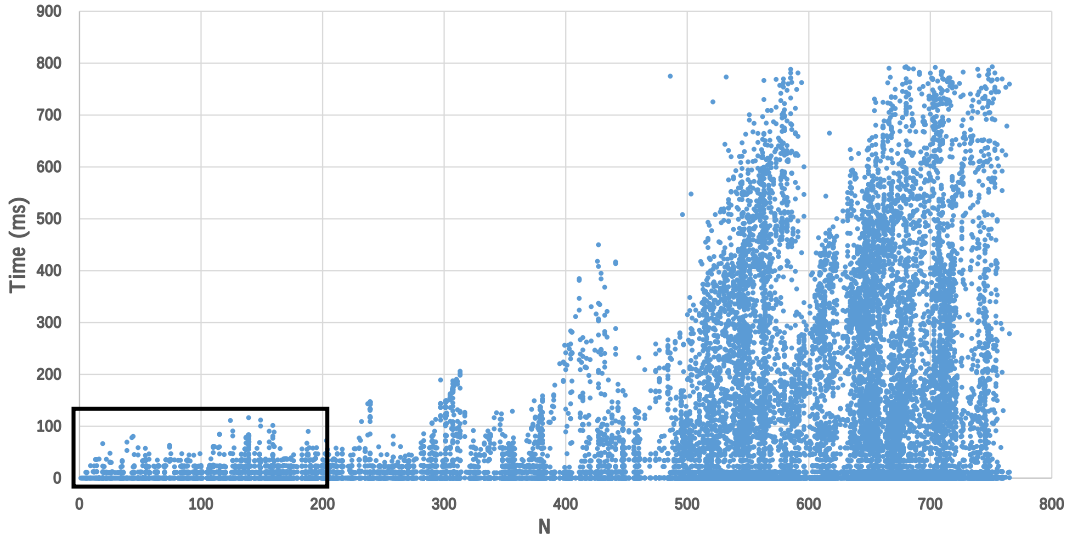



**Figure 7.3:** The running time of our SLAM framework for each new vertex reached during a mapping experiment as a function of the number $N$ of vertices currently on the map, using a single 2.7GHz thread to carry computations. Starting around $N = 550$ vertices, the maximum running time stabilizes around 800ms for each vertex reached. This plateau corresponds to the vertex density reaching its maximum as described in subsection 7.2.3. Even in an extremely large environment, the running time with the current code implementation should thus not exceed 800 ms. Figure 7.4 shows a zoom on the black rectangle.

### 7.2.5 Experimental running times

We logged the running time of the loop closure part of our SLAM framework (chapter 5, section 5.4) during a mapping experiment on the Cuzco dataset (see chapter 6). Loop closure includes the expensive signature matching processes. Figure 7.3 displays the observed running time as a function of $N$, the number of vertices on the map at a given time. A zoom on the bottom-left part of the Figure, displayed on Figure 7.4, shows that the running time of loop closure only takes quantified values. In fact, each horizontal line of Figure 7.4 corresponds to a number of vertices being considered for disambiguation: $0, 1, 2, 3, \ldots$. This quantification means that the running time of loop closure is dominated by signature matching running $0, 1, 2, 3, \ldots$ times. We showed in subsection 7.2.3 that in an environment where vertices are at least $d_{min}$ apart, the number of vertices considered as loop closure candidates could not exceed a fixed threshold $\mathcal{O}(\frac{\pi\mathcal{U}(\mathcal{O})^{\in}}{\rho})$. On Figure 7.3, the vertex density in the graph increases until it reaches its maximum around $N = 550$ and does not change anymore. Consequently, past $N = 550$, the running time of loop closure becomes constant in $N$.

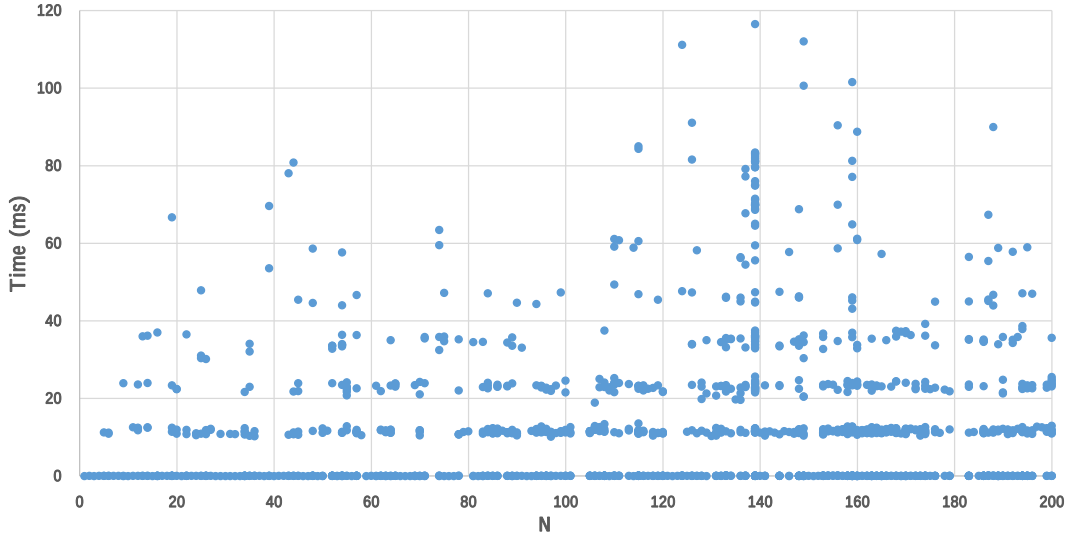



**Figure 7.4:** The running time of our SLAM framework for each new vertex reached during a mapping experiment as a function of the number $N$ of vertices currently on the map, using a single 2.7GHz thread to carry computations. This Figure is a zoom on the bottom-left corner of Figure 7.3. Running times form horizontal lines, each of which corresponding to a number of vertices considered for disambiguation $(0, 1, 2, 3, \ldots)$.

## 7.3 Conclusion

In this chapter, we studied the time complexity and practical running time of algorithms used in the previous chapters. Time complexities do not necessarily translate into running times due to the constants hidden in complexities. Thus, our approach was twofold: we showed that running times achieved *in practice* allowed operation in the real world amongst people, animals and vehicles and demonstrated that all components of the PNSLAM framework can run in $\mathcal{O}(1)$ time complexity as a function of the number $N$ of places in the environment. Constant time operation may become critical in huge environments and for *Lifelong Exploratory Navigation*. If addressing vertices by their position is required, storing the vertices in a quadtree using their spring-mass optimized positions is possible. In a quadtree, retrievals, insertions and deletions can be performed in $\mathcal{O}(log(N))$ time complexity. Addressing vertices using their coordinates may be required to share the map with a human for instance.

While all PNSLAM algorithms run in constant time complexity and *decently fast in practice*, the memory cost of the approach is still at least linear in the number of places on the map. The point of the next chapter is to *compress* the memory representation (graph) in order to control the memory usage of the approach.

# 8 Handling finiteness of memory

*"Nothing fixes a thing so intensely in the memory as the wish to forget it."*
**Michel de Montaigne**

In order for the memory of the robot not to grow unbounded and thus to allow for lifelong operation, it is necessary to compress the map stored in memory. The aim is to achieve a high compression ratio with minimal navigation overhead. The compression methods can be optimized depending on the robot's habits (favorite trajectories and places).

## 8.1 Introduction

Suppose that a *navigating agent* wants to go from its current position $O$ to a given location $F$ in a physical environment $\mathcal{W}$. In order to perform the task, a PNSLAM approach is used. An (exploratory) planner such as EDNA* (chapter 3) computes a trajectory from $O$ to $F$ in the environment using an abstract representation (or *map*) $\mathcal{G}$ of $\mathcal{W}$. Then, the navigating agent follows the computed trajectory in order to reach $F$ while updating the map using SLAM. We showed in section 1.2.1 that it was not possible in general to reach $F$ without using a map, that is without *memorizing* the environment. This chapter only discusses environments represented as directed graphs. However, dense maps such as grids of pixels (including occupancy grids as described in chapter 4) can be treated as graphs whose vertices are pixels and whose edges describe traversability between neighboring pixels. Let $\mathcal{V}$ and $\mathcal{E}$ be the set of vertices and edges of $\mathcal{G}$ respectively.

One of the characteristics of SLAM algorithms operating on graphs such as ours (chapter 5) is the amount of data used to describe one vertex (also called place) or edge (also called path) of $\mathcal{G}$: place and path signatures may consist in 360° visual snapshots (Beranger

and Herve, 1996; Korrapati and Mezouar, 2014), sonar signatures (Duckett, Marsland, and Shapiro, 2002), local occupancy grids (Beeson, Modayil, and Kuipers, 2010) or other local representations with a non-negligible memory cost. While for short runs in small environments, the memory of a robot may not saturate, longer runs in potentially huge environments (a whole city for instance) may lead to memory saturation even when place signatures are compressed. Maintaining a high-bandwidth stable connexion with an external database server is not an option for mobile robots in most environments. Memory saturation raises the issue of how to perform lossy compression (or pruning) of the map $\mathcal{G}$ with minimum impact on navigation performances.

This chapter proposes lossy compression techniques as well as compatible planning algorithms operating on compressed data. These approaches are evaluated in terms of memory gains and path length overheads. The rest of this chapter goes as follows: section 8.2 reviews existing planning and compression approaches within the fields of robot navigation and packet routing. Section 8.3 describes our benchmark methodology. Section 8.4 lists the existing and new algorithms that will be compared against each other. Section 8.5 describes the datasets used and exposes and explains experimental results. Section 8.7 explains how to choose an approach based on the results of section 8.5. Finally, section 8.8 concludes on the approach.

## 8.2 Planning, navigation and compression: existing approaches

The problem of a navigating agent moving from an origin to a destination vertex can be approached from two different points of view: when the navigating agent is responsible for computing its path on a passive graph structure (as in robot navigation or video game path planning) and when the graph structure is responsible for routing the passive navigating agent towards its goal (as in wireless geographic packet routing). Both situations have their own abundant literature.

### 8.2.1 Robots: path planning and A*

For mobile robot navigation, the problem of finding an itinerary in $\mathcal{W}$ using $\mathcal{G}$ is referred to as *path planning* and dealt with using mostly variants of A* Hart, Nilsson, and Raphael, 1968. There are numerous variants of A* (see the review by Ferguson et al. (2005)) adapted to various scenarios, such as when physical movements are required to compute a path (physical A* (Felner et al., 2004)).

Lossy compression of a map of an environment stored as a graph means that some vertices and edges of the graph have to be forgotten. However, even with an incomplete map, the robot should still be able to navigate from any already visited place to any

other visited place. Of course, the path computed on an incomplete map may not be optimal in terms of length and may even require the robot to reconstruct parts of the map that were compressed away using SLAM while moving in the environment. Navigation using a map where some parts are missing because they were compressed away is no different from navigation using a map where some parts are missing because they have not been explored yet. We called navigating using an incomplete graph representation to plan paths *Exploratory Digraph Navigation* in chapter 3. In Exploratory Digraph Navigation, the robot only has a map of a subset of $\mathcal{W}$, the *currently known graph* (Felner et al., 2004). Dynamic path planners such as D* (Stentz, 1995) or D* Lite (Koenig and Likhachev, 2005) are limited to situations where the set of possible edges and vertices of the graph is known in advance, the only unknown being whether they are traversable or not. Examples of such graphs are grids with each vertex connected to its 4- or 8- nearest neighbors when space is traversable. The general case of navigation in an arbitrary digraph with unknown zones is handled by EDNA* (chapter 3). D* (Lite) and EDNA* are guaranteed to reach their destination $F$ but the amount of memory required may be proportional to the size of the environment, typically $\mathcal{O}(Card(\mathcal{E}))$. When $\mathcal{G}$ is an arbitrary digraph, it is not possible in general to reach $F$ without using a map at all, that is without *memorizing* the environment.

In addition to A* variants, some simpler algorithms such as variants of the "Bug" (Lumelsky and Stepanov, 1987) maze solving algorithm are also used for robot planning. These are reviewed in a technical report by Rao et al. (1993) available online. Bug uses greedy navigation in free space and clockwise circumventing of obstacles. Bug variants are limited to strictly planar environments with only two-way paths. However, in such environments, they are guaranteed to reach $F$ without even a map, whence their "memory-less" or "state-less" denomination. Note that Bug variants are in fact *not able* to use a map.

Even though graph compression techniques are used in the Simultaneous Localization and Mapping community (see (Carlevaris-Bianco, Kaess, and Eustice, 2014)) to reduce the computational burden of SLAM algorithms, we are not aware of previous studies on lossy compression of a map stored as a graph for robot navigation purposes.

### 8.2.2 Wireless routing: greedy forwarding and face routing

On the other side, routing algorithms defer path planning to the graph vertices (routers), which are equipped with a mechanism to find where they have to forward packets they receive. Since routers may have to deal with huge number of packets, their routing strategy needs to be efficient in terms of computing power, even though the obtained path is not optimal in terms of length or traversal effort. The naive solution to the routing problem would be to have each router maintain a routing table indicating for each packet destination $F$ the edge to be taken as next step. This would require at least

$\mathcal{O}(Card(\mathcal{V}))$ memory for each of the $Card(\mathcal{V})$ routers. In a network with millions of routers, the cost of maintaining such routing tables would become intractable. It is thus necessary to prune routers from routing tables while still guaranteeing packet delivery. This is done for instance by adopting a hierarchical view of routing where routing is done first between top-level entities (such as internet service providers) and then within entities. One should note that defining a hierarchy in the vertices of a graph is not trivial and is done manually as far as the internet is concerned. Maps used for robot navigation typically do not have such a hierarchy between places.

One additional constraint of router networks is that only minimum information can be stored within the packet being routed, which calls for stateless routing algorithms. The simplest stateless routing algorithm is Greedy forwarding, where a router only knows its immediate neighbors and forwards packets to them in order to minimize the Euclidean distance to $F$. Greedy forwarding guarantees delivery of any packet to any destination if and only if $\mathcal{E}$ is a superset of the non-degenerate edges of the Delaunay triangulation of $\mathcal{V}$ (Ghaffari, Hariri, and Shirmohammadi, 2010). Greedy forwarding was extended by Bose et al. (2001) and Karp et al. (2000). The resulting face routing algorithm, Greedy Perimeter Stateless Routing (GPSR), defines a recovery phase to exit local minima of the Euclidean distance to destination. During a recovery phase, routing is done along the faces of the graph (similarly to the Bug algorithms). Delivery of packets with GPSR is only guaranteed when $\mathcal{G}$ is planar or when the condition for Greedy forwarding is met, in which case the recovery algorithm is never triggered. Like Bug variants in robotics, greedy forwarding and face routing take decisions locally, without considering large-scale trajectory optimization.

Lam et al. (2013) use another recovery strategy to cope with greedy forwarding getting stuck in local minima of the Euclidean distance to $F$: if a non-degenerate Delaunay edge is missing from $\mathcal{E}$ (preventing greedy routing to be used (Ghaffari, Hariri, and Shirmohammadi, 2010)), a "virtual" edge made of a series of edges in $\mathcal{E}$ is stored locally as replacement for the missing Delaunay edge. This virtual edge can be computed for instance by Dijkstra's algorithm or by A* (Hart, Nilsson, and Raphael, 1968). The work of Lam et al. is valid not only in 2D but also at least in 3− and 4 dimensions according to the authors. We can devise a straightforward extension of the work of Lam et al. which consists in creating virtual edges not only to replace missing Delaunay edges but also to improve navigation performances.

## 8.3 Methodology

In this chapter, we study how to compress $\mathcal{G}$ and how to use the compressed graph $\mathcal{G}^C \subset \mathcal{G}$ to plan paths and navigate in $\mathcal{W}$ using (exploratory) planning and navigation algorithms, further abridged "PN". For simplicity, the SLAM problem is not considered,

which is equivalent to assigning to each vertex $v \in \mathcal{V}$ a unique identifier automatically recognized upon traversal. Moreover, the world $\mathcal{W}$ is supposed to be static.

Metrics collected are of two types: navigation performances on $\mathcal{G}^C$ and memory usage. We split memory usage in three categories: *static*, *dynamic* and *execution* memory. *Static* memory is related to the smallest subgraph $\mathcal{G}^C$ of $\mathcal{G}$ that allows navigation from any place $O$ to any other place $F$ in $\mathcal{W}$ using the PN approach considered. Given an $(O, F)$ pair, *dynamic* memory describes what should be added to $\mathcal{G}^C$ in order to navigate from $O$ to $F$ using the PN approach. Finally, *execution* memory describes the maximum amount of memory required to execute the exploratory planning algorithm. Memory necessary for navigation and SLAM is not considered since it does not vary with the size of the environment $\mathcal{G}$ and is in practice negligible compared to memory used to store the map $\mathcal{G}^C$ or $\mathcal{G}$ even with small maps. Memory necessary for graph compression is not considered either since it is negligible in comparison to memory necessary to store $\mathcal{G}$ (memory for compression is comparable to execution memory, which is negligible compared to static and dynamic memory, see subsection 8.5.1).

### 8.3.1 Additional notations

Let $\mathcal{G}$ be an undirected graph representing paths and places of a physical environment $\mathcal{W}$. Let $\mathcal{V}$ and $\mathcal{E}$ be the set of vertices and edges of $\mathcal{G}$ respectively. We suppose that each vertex $v \in \mathcal{V}$ has a position $R_v \in \mathbb{R}^2$. Each edge $e \in \mathcal{E}$ from vertex $S(e) \in \mathcal{V}$ to vertex $T(e) \in \mathcal{V}$ is associated to a traversal cost $c(e) \in [||R_{T(e)} - R_{S(e)}||; +\infty[$ and each vertex $v \in \mathcal{V}$ is associated to a traversal cost $c(v) \in \mathbb{R}^+$. Let $S(v) = \{e \in \mathcal{E}, S(e) = v\}$ and $T(v) = \{e \in \mathcal{E}, T(e) = v\}$. The definition intervals of $c(e)$ and $c(v)$ are chosen so that the Euclidean distance underestimates path lengths, computed as the sum of traversal costs along a path. Thus, the Euclidean distance is said to be an *admissible heuristic* for heuristic-guided algorithms such as A* (Hart, Nilsson, and Raphael, 1968). Within this chapter, we only consider path planning to a target vertex $F \in \mathcal{V}$ whose position $R_F$ is known. We suppose that the out-degree of any vertex $v \in \mathcal{V}$ is bounded, that is $\exists b \in \mathbb{N}, \forall v \in \mathcal{V}, Card(S(v)) \leq b$.

### 8.3.2 Benchmark protocol overview

The process of obtaining $\mathcal{G}^C$ from $\mathcal{G}$ and collecting metrics consists in two phases: a *preparatory phase* and a *data collection phase*. The preparatory phase contains all steps that can be executed only once on $\mathcal{G}$, no matter the considered navigation problems and parameters. The data collection phase is a parametric study of navigation and memory metrics. The process is represented on Figure 8.1.

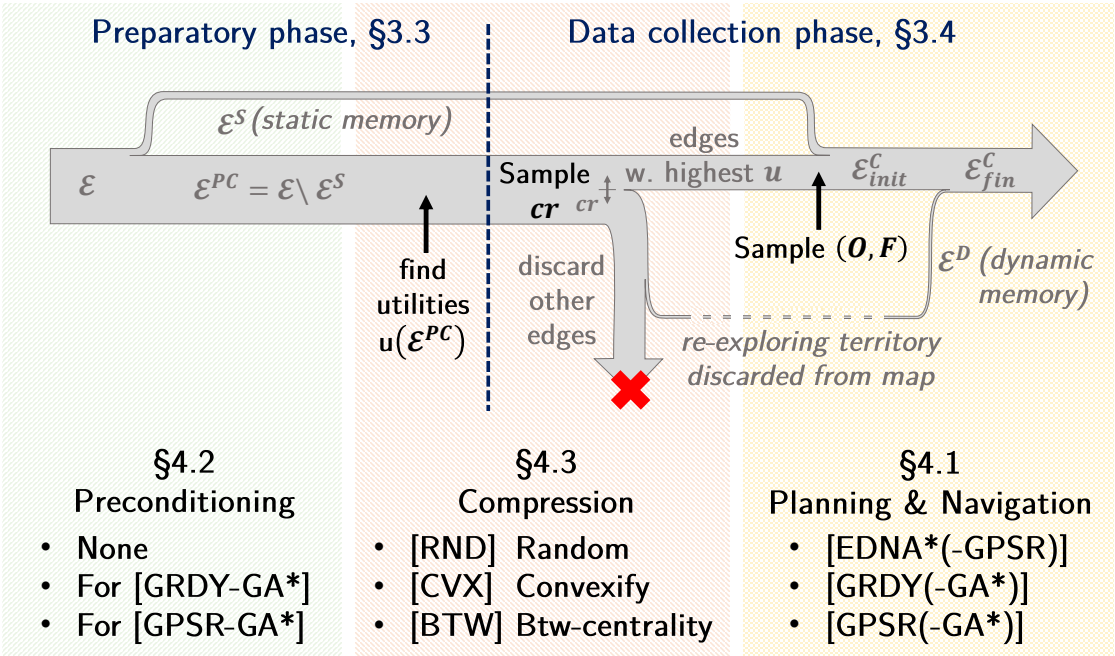



**Figure 8.1:** Graph compression for navigation: methodology. During the preparatory phase, preconditioning determines edges that are necessary for navigation with a given PN algorithm. These edges count as static memory and limit the compression ratio to $cr^*$. Then, a compression algorithm assigns a utility $u(e)$ to each remaining edge $e \in \mathcal{E}^{PC}$. During the data collection phase, the graph $\mathcal{G}$ is compressed according to edge utilities and a target compression ratio $cr < cr^*$ and the PN algorithm is used to navigate from a vertex $O \in \mathcal{G}$ to a vertex $F \in \mathcal{G}$, possibly adding edges in $\mathcal{E}^D$ to the compressed graph $\mathcal{G}^C$. These new edges count as dynamic memory.

### 8.3.3 Preparatory phase

Suppose that we want to study a PN algorithm such as greedy navigation. This algorithm will not necessarily operate on any $\mathcal{G}^C \subset \mathcal{G}$. Thus, a first step is to determine a set of edges $\mathcal{E}^S \subset \mathcal{E}$ such that the PN algorithm is able to lead the navigating agent from any $O \in \mathcal{V}$ to any $F \in \mathcal{V}$ reachable from $O$ using only edges in $\mathcal{E}^S$ and exploration of areas missing in $\mathcal{G}^C$. We call this step *preconditioning*. Depending on the PN algorithms used, this preconditioning phase may mark from $Card(\mathcal{E}^S) = 0$ to $Card(\mathcal{E}^S) = Card(\mathcal{E})$ edges as necessary for navigation. We call $Card(\mathcal{E}^S)$ *static memory*. Static memory can be expressed as a *static memory ratio* $sm^*(PN) = \frac{Card(\mathcal{E}^S)}{Card(\mathcal{E})}$. The maximum compression ratio of the map is $cr^* = \frac{1}{sm^*(PN)}$. It should be noted that *static memory* is required no matter the mission of the navigating agent. For instance, if the agent successively goes to multiple destinations $F_i$ instead of a single one, static memory will not change.

All edges in $\mathcal{E}^{PC} = \mathcal{E} \backslash \mathcal{E}^S$ are not essential for navigation and can be compressed away. Thus, as a second step, a *compression* algorithm is used to associate a cardinal utility $u(e) \in [0; 1[$ to each edge in $\mathcal{E}^{PC}$. Since edges in $\mathcal{E}^S$ have to be preserved in $\mathcal{G}^C$ whatsoever, their cardinal utility does not need to be computed. Preconditioning the graph and finding cardinal utilities are the two steps of the *preparatory phase*.

### 8.3.4 Data collection phase

The data collection phase consists in multiple data collection runs with different origins $O$ and destinations $F$ as well as different map compression ratios $cr$.

A single data collection run consists in the following steps (Figure 8.1): first, a target compression ratio $cr \in ]1; cr^*[$ is set. From $cr$, we compute $N_r$, the number of edges to remove as $N_r = Card(\mathcal{E}) \left(1 - \frac{1}{cr}\right)$. The $N_r$ edges with the lowest utility are found using the quickselect algorithm (also called Hoare's selection algorithm (Hoare, 1961)) on the array of utilities of all edges in $\mathcal{E}^{PC}$. $\mathcal{G}^C$ is created at this point. The initial set of vertices of $\mathcal{G}^C$ is $\mathcal{V}^C = \mathcal{V}$ and its initial set of edges $\mathcal{E}^C = \mathcal{E}^C_{init}$. $\mathcal{E}^C_{init}$ is $\mathcal{E}$ with the $N_r$ least useful edges removed. Vertices with zero remaining incoming edges are removed from $\mathcal{V}^C$. $Card(\mathcal{E}^C_{init})$ is the memory used at the beginning of the data collection run. Note that $cr = \frac{Card(\mathcal{E})}{Card(\mathcal{E}^C_{init})}$.

Second, an origin $O \in \mathcal{V}$ and a destination $F \in \mathcal{V}, F \neq O$ are chosen randomly. A run of A* on $\mathcal{G}$ returns one of the shortest paths in terms of traversal cost from $O$ to $F$. Its length is $l^*$, computed as the sum of traversal costs of the $n^*$ edges and $n^* + 1$ vertices traversed during the trip from $O$ to $F$. This initial A* run expands $s^*$ vertices, where $s$ stands for "search space" and is a measurement of the *execution memory* required by

A* on $\mathcal{G}$.

Third, the *PN* algorithm is used to reach $F$ from $O$. As there may not exist a path from $O$ to $F$ using only the edges of $\mathcal{G}^C$, this algorithm may have to resort to exploration of zones compressed away from the map (see chapter 3). Let $l$ be the length traveled by the navigating agent, computed as the sum of traversal costs of the $n$ edges and $n+1$ vertices traversed during the trip from $O$ to $F$. Let $s$ be the maximum amount of memory ("search space") used by the planning algorithm. Let $\mathcal{E}^D$ be the set of edges added to $\mathcal{E}^C$ on the path from $O$ to $F$, where $dm = Card(\mathcal{E}^D)$ is the *dynamic memory* of the run. The final set of edges in $\mathcal{G}^C$ is $\mathcal{E}^C_{fin} = \mathcal{E}^C_{init} \cup \mathcal{E}^D$ Contrary to static memory which is required no matter the itinerary of the navigating agent, dynamic memory is unique to one data collection run and is erased after each run. If runs consisted of multiple successive choices of $O$ and $F$, dynamic memory could also be erased between each choice.

## 8.4 Algorithms on the test bench

### 8.4.1 Planning and navigation algorithms

We chose to test five combinations of (exploratory) planning and navigation (PN) algorithms:

(A) [GPSR] Greedy Perimeter Stateless Routing (GPSR),

(B) [EDNA*] EDNA*,

(C) [EDNA*-GPSR] EDNA* with a dynamic memory limit and GPSR when the dynamic memory budget has been spent,

(D) [GRDY-GA*] Greedy navigation with greedy-A* (called gA* in the following paragraphs) recovery, which reproduces the work of (Lam and Qian, 2013) within a navigation context and

(E) [GPSR-GA*] GPSR with gA* recovery which is a new and more elaborate version of Greedy navigation with gA* recovery.

Each combination of algorithms has advantages and drawbacks. Worst case memory complexities of all five algorithms are reported in table 8.1.

**(A) [GPSR]** GPSR (Bose et al., 2001; Karp and Kung, 2000) alone is restricted to

planar graphs but ensures infinite compression ratios and an $\mathcal{O}(1)$ worst case complexity for both dynamic and execution memory. GPSR cannot take advantage of the existing map $\mathcal{G}^C$ which leads to poor performances with low compression ratios compared to algorithms using $\mathcal{G}^C$. We do not test Greedy alone, since almost no graph supports greedy navigation, and graphs supporting it also support GPSR since its recovery phase is never triggered.

**(B) [EDNA*]** EDNA* (chapter 3) also allows infinite compression ratios but takes advantage of $\mathcal{G}^C$, which should lead to shorter paths than GPSR on average with low compression ratios. However, while EDNA* can operate on arbitrary graphs (even directed graphs), it may require up to $\mathcal{O}(Card(\mathcal{E}))$ units of dynamic memory and $\mathcal{O}(Card(\mathcal{V}))$ units of execution memory to reach the goal vertex $F$. We tested various values of the risk heuristic for EDNA*. In average, shorter paths were obtained with higher risk overestimations. As in (Mayran de Chamisso, Soulier, and Aupetit, 2015), we take $\mathcal{R}(O, F, Z \in \mathcal{V}^C) = D(O, Z) + \alpha\,(1 - \beta cos(\theta))\,||F - Z||$ with $\theta$ the angle between an edge going out from $Z$ and the vector from $Z$ to $F$. $\alpha >> 1$ ensures high risk overestimation. $\beta$ is chosen according to (Mayran de Chamisso, Soulier, and Aupetit, 2015), that is equal to 0.25 for arbitrary digraphs and equal to 0.125 for grid-graphs.

**(C) [EDNA*-GPSR]** Using EDNA* until a predefined dynamic memory budget is reached and GPSR then ensures that dynamic memory remains bounded while taking advantage of $\mathcal{G}^C$. However, this technique is limited to planar graphs due to the use of GPSR.

**(D, E) [GRDY/GPSR-GA*]** We introduced gA* (algorithm 3, page 57) as a variant of EDNA* whose principle is to compute the risk heuristic $\mathcal{R}$ on all vertices expanded, not just on boundary vertices. Greedy navigation or GPSR with gA* recovery (algorithms 5 and 6) are not limited respectively to environments where greedy navigation between any two vertices is possible and to planar graphs. However, both require a computationally expensive preconditioning phase which limits the compression ratio $cr$ (due to static memory consumption). While greedy and GPSR with gA* recovery use static memory and up to $\mathcal{O}(Card(\mathcal{V}))$ units of execution memory, their dynamic memory consumption is $\mathcal{O}(1)$.

### 8.4.2 Preconditioning algorithms

We propose two preconditioning algorithms:

(A) Preconditioning for [GRDY-GA*] (Algorithm 7) and

(B) Preconditioning for [GPSR-GA*] (Algorithm 8).

**Algorithm 5** [GRDY-GA*] Greedy navigation with gA* recovery

**Input:** $\mathcal{G}^C = \{\mathcal{V}^C, \mathcal{E}^C\}, \mathcal{E}^S, O, F$
$curv \leftarrow O$
**while** $curv \neq F$ **do**
  $e^* \leftarrow argmin_{e \in S(curv)}(||F - T(e)||)$
  run gA* from $curv$ with destination $F$
  returns path $\mathcal{P}$ to vertex $v'$
  **if** $||F - v'|| < ||F - T(e^*)||$ **then**
    follow path $\mathcal{P}$ to $v'$
    $curv \leftarrow v'$
  **else**
    $curv \leftarrow T(e^*)$

**Algorithm 6** [GPSR-GA*] GPSR navigation with gA* recovery

**Input:** $\mathcal{G}^C = \{\mathcal{V}^C, \mathcal{E}^C\}, \mathcal{E}^S, O, F$
$curv \leftarrow O, mindist \leftarrow ||F - O||$
**while** $curv \neq F$ **do**
  $e^* \leftarrow argmin_{e \in S(curv)}(||F - T(e)||)$
  run gA* from $curv$ with destination $F$
  returns path $\mathcal{P}$ to vertex $v'$
  **if** $||F - v'|| < min(||F - T(e^*)||, mindist)$ **then**
    follow path $\mathcal{P}$ to $v'$
    $curv \leftarrow v', mindist \leftarrow ||F - v'||$
    OnObstacle $\leftarrow$ false
  **else**
    **if** OnObstacle **then**
      **if** $||T(e^*) - v'|| <$ mindist **then**//back to greedy
        $curv \leftarrow T(e^*), mindist \leftarrow ||F - curv||$
        OnObstacle $\leftarrow$ false
      **else**
        $e^* \leftarrow$ circle obstacle clockwise, only using $e^* \in S(curv) \backslash \mathcal{E}^{NP}$
        Change face if necessary, following (Karp and Kung, 2000)
    **else**
      **if** $||v' - T(e^*)|| < ||v' - curv||$ **then**
        $curv \leftarrow T(e^*), mindist \leftarrow ||F - curv||$
      **else**
        OnObstacle $\leftarrow$ true
        $e^* \leftarrow$ circle obstacle clockwise, only using $e^* \in S(curv) \backslash \mathcal{E}^{NP}$

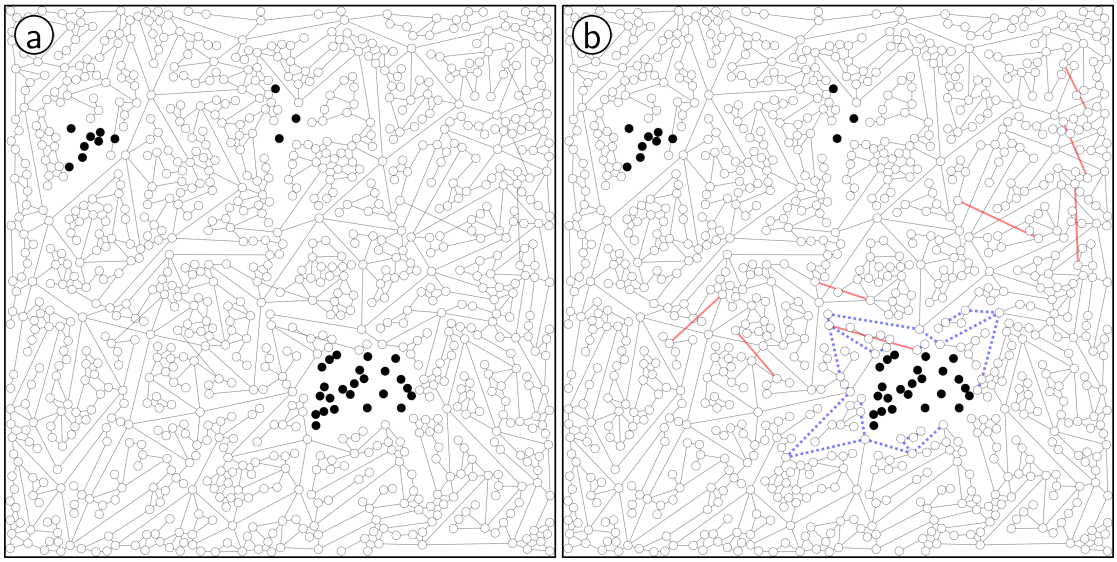



**Figure 8.2:** Preconditioning for [GPSR-GA*]. (a) Graph before preconditioning. (b) During preconditioning, edges preventing the graph from being planar are found (bold red) and stored separately as $\mathcal{E}^{NP}$. Then, an attempt is made to traverse each edge $\epsilon$ of the Delaunay triangulation of $\mathcal{V}$. When traversing $\epsilon$ is not possible using GPSR on $\mathcal{G}$, a sequence of edges found using A* on $\mathcal{G}$ between the extremities of $\epsilon$ is remembered (bold dashed blue) as a replacement for $\epsilon$. Edges preventing the graph from being planar and replacement trajectories for missing Delaunay edges form static memory $\mathcal{E}^{S}$.

Both preconditioning algorithms are based on constructing the Delaunay triangulation of $\mathcal{G}$ and trying from each vertex $v$ to reach each one of its Delaunay neighbors $v'$ using Greedy **(A)** or GPSR **(B)**. Let $l^*$ be the minimal traversal cost from $v$ to $v'$ according to A* on $\mathcal{G}$. A threshold $T_s$ is used to determine whether a path to a Delaunay neighbor is short enough. If the neighbor cannot be reached without traveling more than $T_s.l^*$, the A* path from $v$ to $v'$ is stored, contributing to static memory. This path is intended to be used by gA* during the data collection phase if necessary. An example of preconditioning is given on Figure 8.2.

Delaunay triangulation is carried using the open source TRIANGLE library by Jonathan Shewchuk (1996; 2002).

### 8.4.3 Compression algorithms

We propose three compression algorithms to assign a cardinal utility $u(e)$ to each edge $e \in \mathcal{E}$:

**Algorithm 7** Preconditioning for [GRDY-GA*]

**Input:** $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$, $T_s$
$\mathcal{E}^S \leftarrow \varnothing$
compute Delaunay triangulation $DLN$ of $\mathcal{V}$
**for all** $v \in \mathcal{V}$ **do**
| **for all** $v' \in \mathcal{V}$ neighbor of $v$ in DLN **do**
| | $curv \leftarrow v$
| | $\mathcal{P} \leftarrow$ A* path from $v$ to $v'$ on $\mathcal{G}$
| | $dist \leftarrow 0, dist^* \leftarrow cost(\mathcal{P})$
| | **while** $curv \neq v'$ **do**
| | | **if** $dist > T_s.dist^*$ **then**
| | | | **break**
| | | $e^* \leftarrow argmin_{e \in S(curv)}(||v' - T(e)||)$
| | | **if** $||v' - T(e^*)|| < ||v' - curv||$ **then**
| | | | $dist \leftarrow dist + c(e^*) + c(curv)$
| | | | $curv \leftarrow T(e^*)$
| | | **else**
| | | | **break**
| | **if** $curv \neq v'$ **then**
| | | $\mathcal{E}^S \leftarrow \mathcal{E}^S \cup \{e \in \mathcal{P}\}$

(A) [RND] Random utility,

(B) [CVX] *Convexify* and

(C) [BTW] Betweenness-centrality

**(A) [RND]** The Random utility compression method assigns random utilities to edges. Assigning random utilities to edges is the simplest method and simulates a system where memory locations are randomly overwritten when memory saturation occurs.

**(B) [CVX]** The idea of the *convexify* algorithm (algorithm 9) is that if all obstacles were close to circles, Greedy navigation would always succeed (it is not sufficient for obstacles to be convex). Additionally, it is intuitively more useful to remember zones that may lead to navigation detours (obstacles) than free space where Greedy routing will find optimal routes. The *convexify* algorithm tries to make all obstacles support greedy navigation by repeatedly smoothing them.

We developed this algorithm for grid-like graphs where it is equivalent to an iterated local sum filter, represented with convolution matrix $\begin{pmatrix} 1 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \end{pmatrix}$. Figure 8.3 shows outputs

**Algorithm 8** Preconditioning for [GPSR-GA*]

**Input:** $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$, $T_s$
  $x \leftarrow \varnothing$ //$x$ = edges crossing each other
  **for all** $(e, e') \in \mathcal{E}^2, e' \neq e$ **do**
    **if** $e$ crosses $e'$ **then**
      $x \leftarrow x \cup \{ee', e'e\}$
  $\mathcal{E}^{NP} \leftarrow \varnothing$ //$\mathcal{E}^{NP}$ = edges breaking planarity
  **while** $x \neq \varnothing$ **do** //are there crossings?
    $e \leftarrow argmax(f : e \in \mathcal{E} \mapsto Card\{a \in \mathcal{E} | ea \in x\})$
    $\mathcal{E}^{NP} \leftarrow \mathcal{E}^{NP} \cup e, x \leftarrow x \backslash \{ab, a = e \text{ or } b = e\}$
  $\mathcal{E}^S \leftarrow \mathcal{E}^{NP}$
  **compute** Delaunay triangulation $DLN$ of $\mathcal{V}$
  **for all** $v \in \mathcal{V}$ **do**
    **for all** $v' \in \mathcal{V}$ neighbor of $v$ in DLN **do**
      $curv \leftarrow v, \mathcal{P} \leftarrow$ A* path from $v$ to $v'$ on $\mathcal{G}$
      $dist \leftarrow 0, dist^* \leftarrow cost(\mathcal{P})$, OnObstacle $\leftarrow$ false
      **while** $curv \neq v'$ **do**
        **if** $dist > T_s.dist^*$ **then break**
          **break**
        $e^* \leftarrow argmin_{e \in S(curv) \backslash \mathcal{E}^{NP}}(||v' - T(e)||)$
        **if** $e^* = \varnothing$ **then**
          **break**
        **if** OnObstacle **then**
          **if** $||T(e^*) - v'||$ < ContactDistance **then**
            OnObstacle $\leftarrow$ false
            $curv \leftarrow T(e^*)$ //greedy
          **else**
            $e^* \leftarrow$ circle obstacle clockwise, only using $e^* \in S(curv) \backslash \mathcal{E}^{NP}$
            Change face if necessary, following (Karp and Kung, 2000)
        **else**
          **if** $||v' - T(e^*)|| < ||v' - curv||$ **then**
            $curv \leftarrow T(e^*)$ //greedy
          **else**
            OnObstacle $\leftarrow$ true
            ContactDistance $\leftarrow ||curv - v'||$
            $e^* \leftarrow$ circle obstacle clockwise, only using $e^* \in S(curv) \backslash \mathcal{E}^{NP}$
        $dist \leftarrow dist + c(e^*) + c(curv)$
      **if** $curv \neq v'$ **then**
        $\mathcal{E}^S \leftarrow \mathcal{E}^S \cup \{e \in \mathcal{P}\}$

**Algorithm 9** [CVX] *Convexify* compression

**Input:** $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$
  **for all** $v \in \mathcal{V}$ **do**
  | $C(v) \leftarrow 1$//counter assigned to each vertex
  **for** count from 1 to huge_value **do**
  | $maxcnt \leftarrow 0$
  | **for all** $v \in \mathcal{V}$ **do**//counter gets sum of neighbors
  | | $C_n(v) \leftarrow \sum_{x \in \mathcal{V}, \exists e \in \mathcal{E}, (S(e),T(e))=(v,x) | (x,v)} C(x)$
  | | $maxcnt \leftarrow \max(maxcnt, C_n(v))$
  | **for all** $v \in \mathcal{V}$ **do**//normalize C to avoid overflow
  | | $C(v) \leftarrow \frac{C_n(v)}{maxcnt}$
  **for all** $e \in \mathcal{E}$ **do**
  | $u(e) \leftarrow 1 - \frac{1}{2}\left(C_{S(e)} + C_{T(e)}\right)$

of the algorithm for a grid-graph.

**(C) [BTW]** The betweenness-centrality compression algorithm (algorithm 10) assigns each edge a cardinal utility $u(e)$ proportional to the fraction of optimal trajectories between random vertices which include this edge. Because of repeated A* calculations to compute optimal trajectories, betweenness-centrality compression is more compute intensive than any of the two other compression techniques. Figure 8.4 shows outputs of the algorithm for a nearly planar graph.

**Algorithm 10** [BTW] Betweenness-centrality compression

**Input:** $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$
  **for all** $e \in \mathcal{E}$ **do**
  | $C(e) \leftarrow 0$//counter assigned to each edge
  $maxcnt \leftarrow 0$
  **for** count from 1 to huge_value **do**
  | sample random $(O, F) \in \mathcal{V}^2$
  | **for all** $e \in \mathcal{E}$ on A* path from $O$ to $F$ **do**
  | | $C(e) \leftarrow C(e) + 1$
  | | $maxcnt \leftarrow \max(maxcnt, C(e))$
  **for all** $e \in \mathcal{E}$ **do**
  | $u(e) \leftarrow \frac{C(e)}{maxcnt}$

Note that the utility of an edge using betweenness-centrality compression is higher when the edge is much used for navigation. This differs from *convexify* for which edges not often taken (such as edges in a dead end) are assigned a higher utility. Betweenness-centrality-based compression can be described as "remembering the highways but forgetting smaller roads" and should intuitively be beneficial for large-scale planning. Conversely, *convex-*

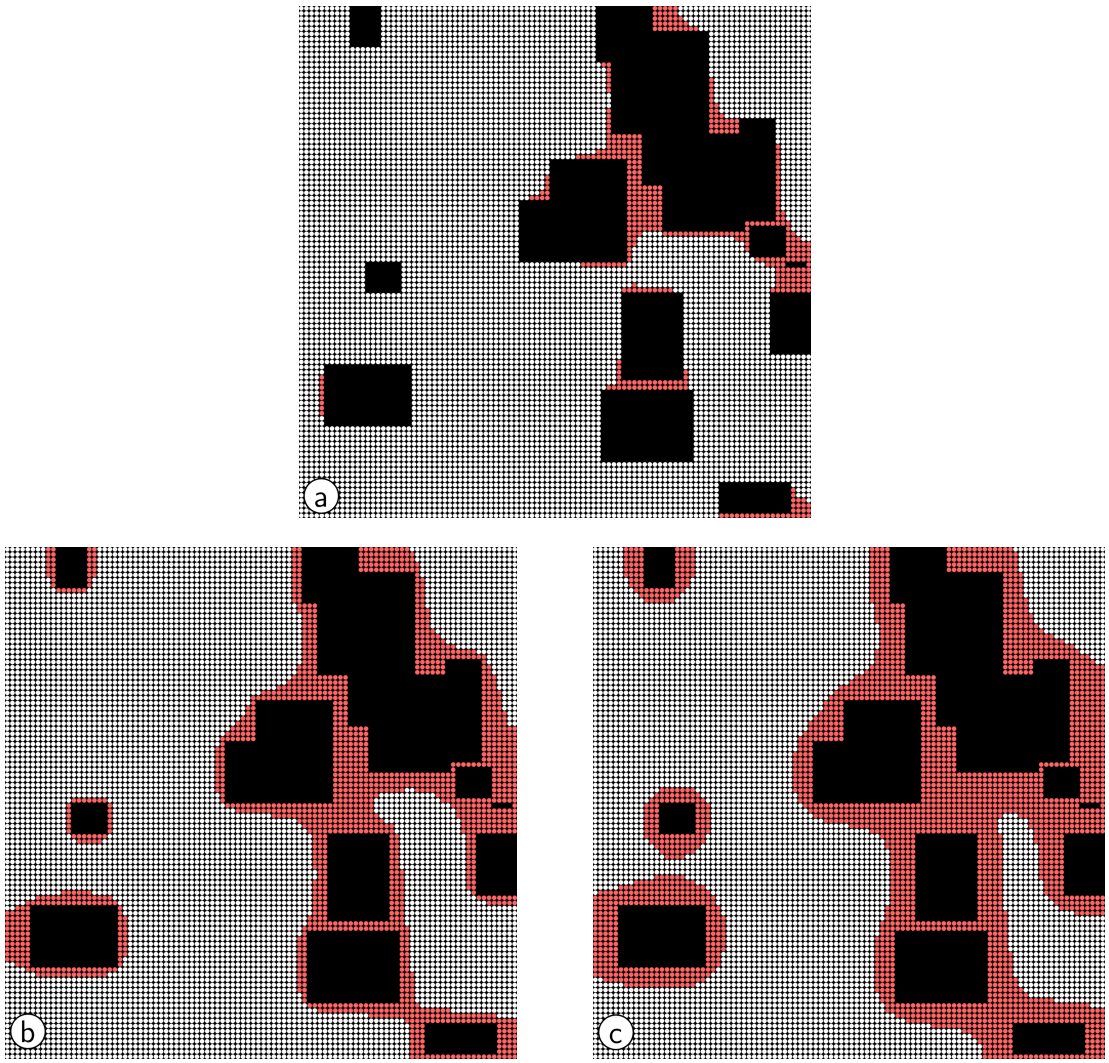



**Figure 8.3:** The *convexify* algorithm assigns a utility to each edge so that keeping edges with the highest utilities will make obstacles convex and fill small gaps. The three images show the output of *convexify* with three different utility thresholds: 0.1 (a), 0.3 (b) and 0.5 (c). Obstacles are represented in black, edges/vertices to keep in red and edges/vertices to remove in white. Here, the graph is 8-way connected.

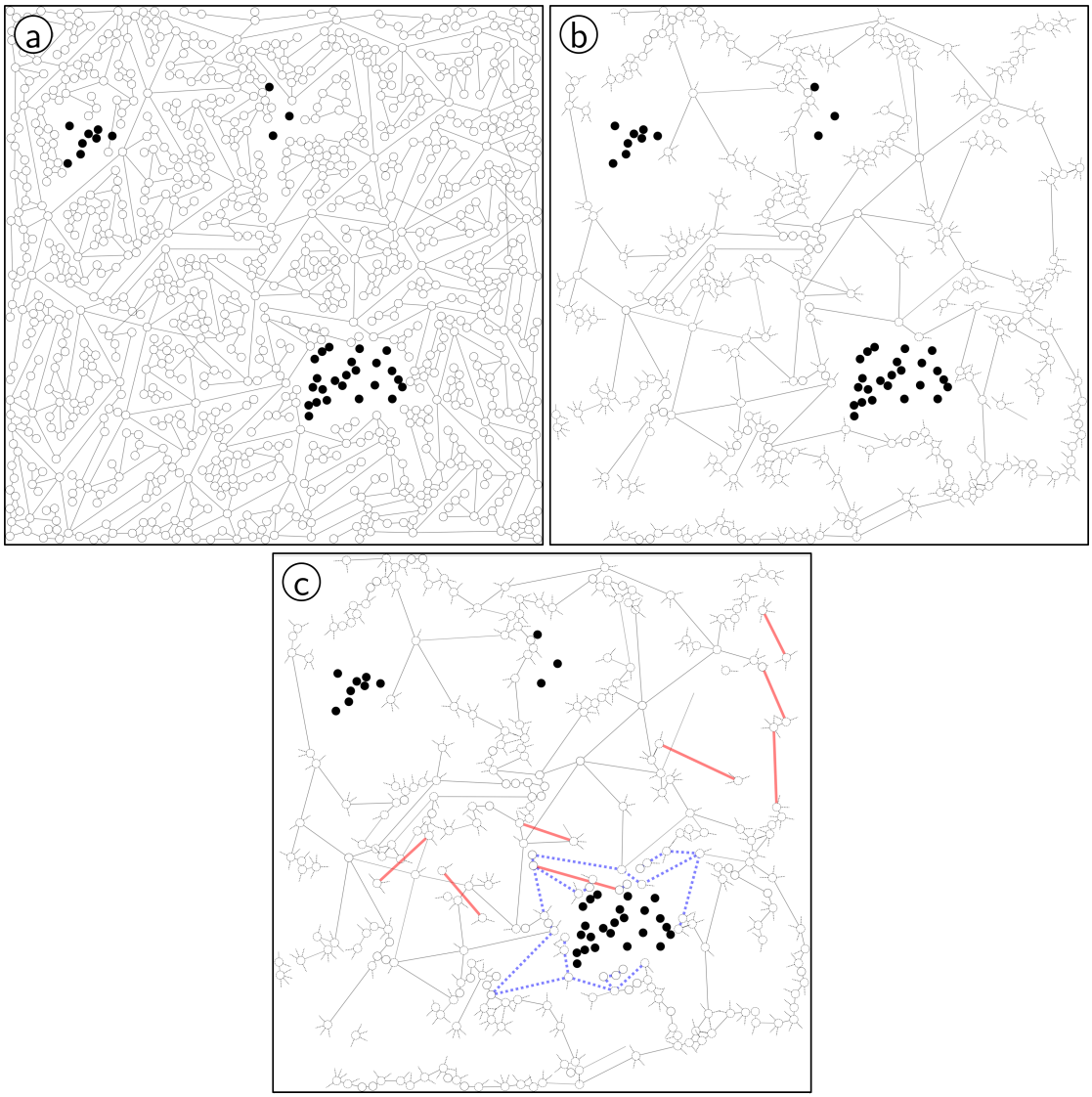



**Figure 8.4:** The betweenness-centrality [BTW] algorithm assigns each edge a utility proportional to the number of optimal trajectories between random vertices which happen to pass through it. (a) Graph before compression. (b) Compression using betweenness-centrality, without preconditioning. (c) Compression using betweenness-centrality with the same utility threshold as (b), using preconditioning for [GPSR-GA*]. Edges marked by preconditioning are shown in bold (plain and dashed) lines.

*ify*-based compression can be described as "only remembering smaller roads because we can assume there is always a highway within $x$ miles of a given point". With similar compression ratios, we expect betweenness-centrality to lead to shorter paths on average when traversal costs are not proportional to the Euclidean distance. Conversely, *convexify* may lead to shorter paths on average on graphs exhibiting regularity properties (grid-graphs) and whose edge traversal costs are equal to the Euclidean distance.

## 8.5 Experiments

### 8.5.1 Performance metrics

We compute normalized performance metrics as follows:

- *navigation suboptimality*: $m_n = \frac{l}{l^*}$ describes the added navigational cost using a compressed map $\mathcal{G}^C$ instead of the uncompressed one $\mathcal{G}$.
- *static memory consumption*: $cr^*(PN) = \frac{Card(\mathcal{E})}{Card(\mathcal{E}^S)}$ describes the maximum compression ratio or equivalently the (minimum) amount of static memory required by the PN algorithm. For PN algorithms without preconditioning, $cr^*$ is infinite.
- *dynamic memory consumption*: $m_{dm} = \frac{\mathcal{E}^D}{n^*}$ describes the amount of dynamic memory used by the PN algorithm. Normalization by $n^*$ comes from the experimentally-confirmed intuition that the amount of dynamic memory is roughly proportional to the number of edges in a minimum-cost path.
- *execution memory consumption*: $m_{em} = \frac{s}{s^*}$ describes the maximum amount of memory ("search space") used during a single run of the planning algorithm relative to A* on $\mathcal{G}$. It is only necessary to consider a single run of the planning algorithm since individual planning steps don't have a memory of past planning steps with the PN approaches tested. Normalization by $s^*$ comes from the fact that A* is optimal in terms of search space $s^*$ amongst equally-informed algorithms (Hart, Nilsson, and Raphael, 1968).

In a situation where much memory is associated to vertices in $\mathcal{G}^C$, execution memory is negligible compared to static and dynamic memory (the constant hidden in the memory complexities of PN algorithms is much higher for static and dynamic memory). Moreover, static memory is computed for the whole graph $\mathcal{G}$ while dynamic memory is used for a single trajectory. Thus, static memory is (by far) the dominant memory consumption term. In a situation where not much memory is associated to vertices in $\mathcal{G}^C$, execution memory may become prominent.

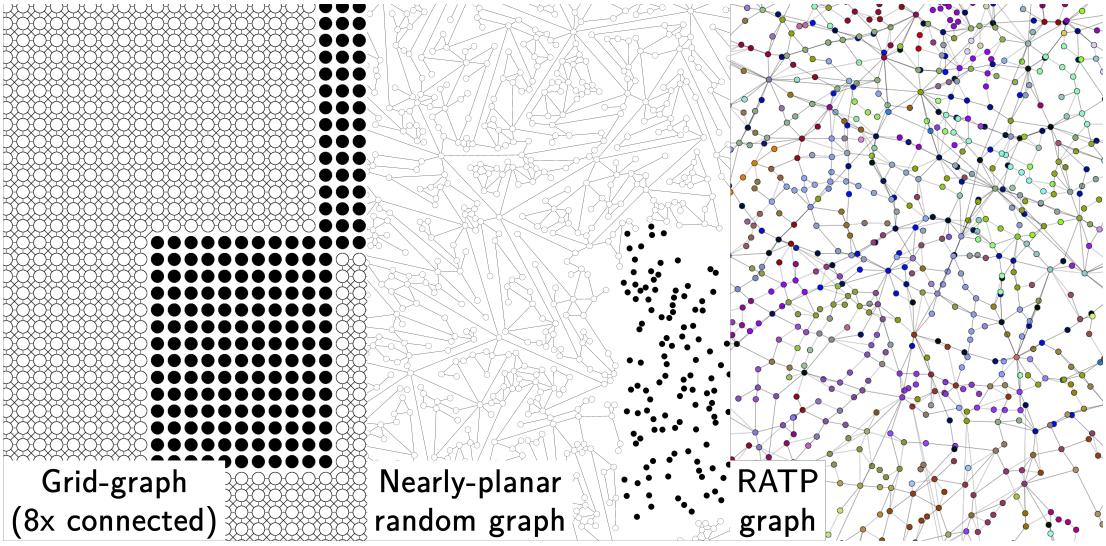



**Figure 8.5:** Overview of the three datasets used within this chapter.

### 8.5.2 Datasets

We use three different datasets represented on Figure 8.5:

- [Grid] 4-way- (to enforce planarity) or 8-way-connected grid-graphs with randomly generated obstacles,
- [Random planar] random planar graphs, made nearly-planar by adding a few random edges and
- [RATP] the network of all buses, metros, trains and trams operated by French transport agency *RATP* in the Paris region (`http://data.ratp.fr/`).

[Grid]-graphs are generated by punching rectangular possibly-overlapping holes in a square $100 \times 100$ grid of 4- or 8-way connected vertices. Holes are punched until one fourth of the initial edges of the grid fall within- or intersect a hole (see Figure 8.3). 100 random edges up to 25 units in length are then added if the experimental protocol does not require the graph to be planar. Hole punching and edge adding are tried again until the graph has a single component. The Euclidean distance is used as heuristic for A* variants.

[Random planar] graphs are generated with 10000 vertices and 15000 dual-way edges within a $100 \times 100$ square. They then undergo the same hole-punching phase as grid-

graphs, with a threshold at one twentieth of the initial edges. 100 random edges up to 15 units in length are then added to make the graph nearly-planar. Hole punching and edge adding are tried again until the graph has a single component. The Euclidean distance is used as heuristic for A* variants. The random nearly-planar graphs intend to mimic road networks, which are mostly planar but locally exhibit non-planar features such as bridges or tunnels.

The [RATP] dataset (Figure 8.6) shows stops (as vertices) and trips or transfers between stops (as edges), with each edge of the graph being associated to a timing. The resulting graph is not planar. In order to transform timings into edge traversal costs compatible with A* variants using the Euclidean distance, we intended to find the speed $v_{max}$ of the fastest transport and scale all timings in the graph by $v_{max}$ in order to obtain a distance overestimating the Euclidean distance between adjacent stops in all cases. Since individual timing measurements are only given with a precision of one minute, this gave us $v_{max} > 1400$km/h, which is obviously wrong. Thus, we manually set $v_{max} = 90$km/h which seems reasonable for any city transportation. When speed values higher than $v_{max}$ where found, they were reduced to $v_{max}$. We removed single-way edges from the graph, which had the effect of splitting it into disconnected components. We only kept the main component of the graph, which contains 4502 vertices and 6183 edges. Note that the resulting graph is still not planar.

### 8.5.3 Experiments carried

Table 8.1 describes experiments carried on each dataset. Each experiment tests a different combination of compatible preconditioning, compression and PN algorithms. The aim of the experiments is to find out the navigation performances and memory usage patterns of each combination of algorithms. Each experiment is conducted on all datasets it is compatible with.

### 8.5.4 Results and discussion

Within this subsection, capital letters within brackets refer to Figures 8.7, 8.8 and 8.9.

#### Compression ratios and static memory

First of all, for grid-graphs, $cr = 16\times$ for experiment 7 [A] and only $3.4\times$ for experiment 10 [B], while for nearly-planar random graphs, $cr = 53\times$ for experiment 10 [C] and only $1.2\times$ for experiment 7 [D]. This is due to grid-graphs being 8-way connected, and as such not planar initially. So, GPSR preconditioning does not work well for these graphs.

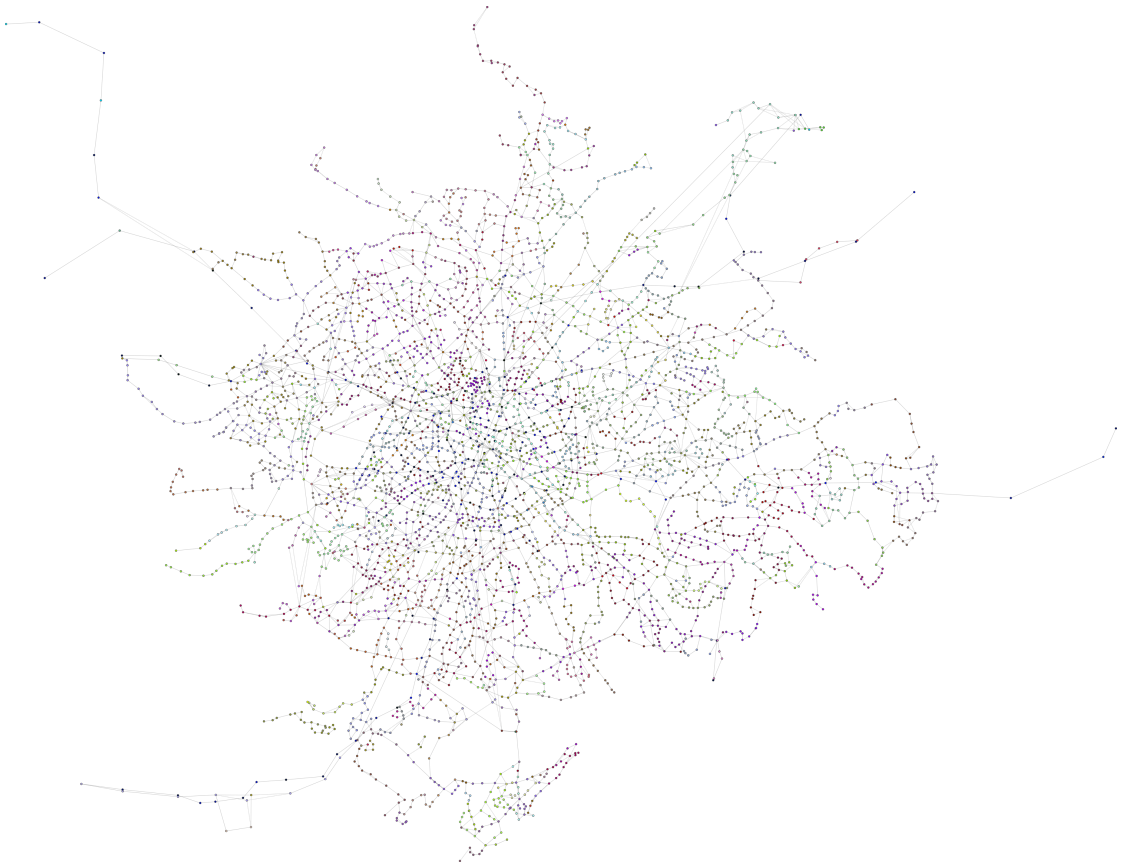


**Figure 8.6:** The RATP dataset, showing most stops and trips of buses, trains, metros and trams within the network operated by RATP as well as transfers between nearby stops. UTM coordinates of stops are used for display. Raw data from `http://data.ratp.fr/`.

| exp. nb. | exp. name | precond ($T_s$) | compress. algorithm | PN algorithm | restricted to |
|---|---|---|---|---|---|
| 1 | GPSR | - | any | GPSR | planar graphs |
| 2 | CVX-EDNA*-GPSR | - | *convexify* | EDNA*+GPSR | planar graphs |
| 3 | BTW-EDNA*-GPSR | - | btw-cntrlty | EDNA*+GPSR | planar graphs |
| 4 | RND-EDNA* | - | random | EDNA* | no restriction |
| 5 | CVX-EDNA* | - | *convexify* | EDNA* | no restriction |
| 6 | BTW-EDNA* | - | btw-cntrlty | EDNA* | no restriction |
| 7 | $\infty$-BTW-GRDY-GA* | Greedy ($\infty$) | btw-cntrlty | Greedy + gA* | no restriction |
| 8 | 2-BTW-GPSR-GA* | GPSR (2) | btw-cntrlty | GPSR + gA* | no restriction |
| 9 | 5-BTW-GPSR-GA* | GPSR (5) | btw-cntrlty | GPSR + gA* | no restriction |
| 10 | $\infty$-BTW-GPSR-GA* | GPSR ($\infty$) | btw-cntrlty | GPSR + gA* | no restriction |

| exp. | symb. | name | max. sm. | max. dm. | max. em. |
|---|---|---|---|---|---|
| 1 |  | GPSR | 0 | 0 | 0 |
| 2 |  | CVX-EDNA*-GPSR | 0 | tunable | $\mathcal{O}(s^*)+$ |
| 3 |  | BTW-EDNA*-GPSR | 0 | tunable | $\mathcal{O}(s^*)+$ |
| 4 |  | RND-EDNA* | 0 | $\mathcal{O}(n^*)+$ | $\mathcal{O}(s^*)+$ |
| 5 |  | CVX-EDNA* | 0 | $\mathcal{O}(n^*)+$ | $\mathcal{O}(s^*)+$ |
| 6 |  | BTW-EDNA* | 0 | $\mathcal{O}(n^*)+$ | $\mathcal{O}(s^*)+$ |
| 7 |  | $\infty$-BTW-GRDY-GA* | $Card(\mathcal{E})$ | 0 | $\mathcal{O}(s^*)+$ |
| 8 |  | 2-BTW-GPSR-GA* | $Card(\mathcal{E})$ | 0 | $\mathcal{O}(s^*)+$ |
| 9 |  | 5-BTW-GPSR-GA* | $Card(\mathcal{E})$ | 0 | $\mathcal{O}(s^*)+$ |
| 10 |  | $\infty$-BTW-GPSR-GA* | $Card(\mathcal{E})$ | 0 | $\mathcal{O}(s^*)+$ |

**Table 8.1:** Experiments carried in this chapter and the theoretical limitations of the algorithms tested. Note that static memory in experiment 10 is 0 if the graph is planar. As betweenness-centrality-based compression leads to lower navigational and memory costs on average, we do not present results for *convexify* and random compression for each PN combination. Experiments 4 to 10 are carried on each dataset. Experiments 1 to 3 are not carried on the RATP dataset since the RATP graph is not planar. For experiments 2 and 3, we observed that $m_{dm} = 0$ led to shorter paths on average. Symbols refer to Figures 8.7, 8.8 and 8.9.

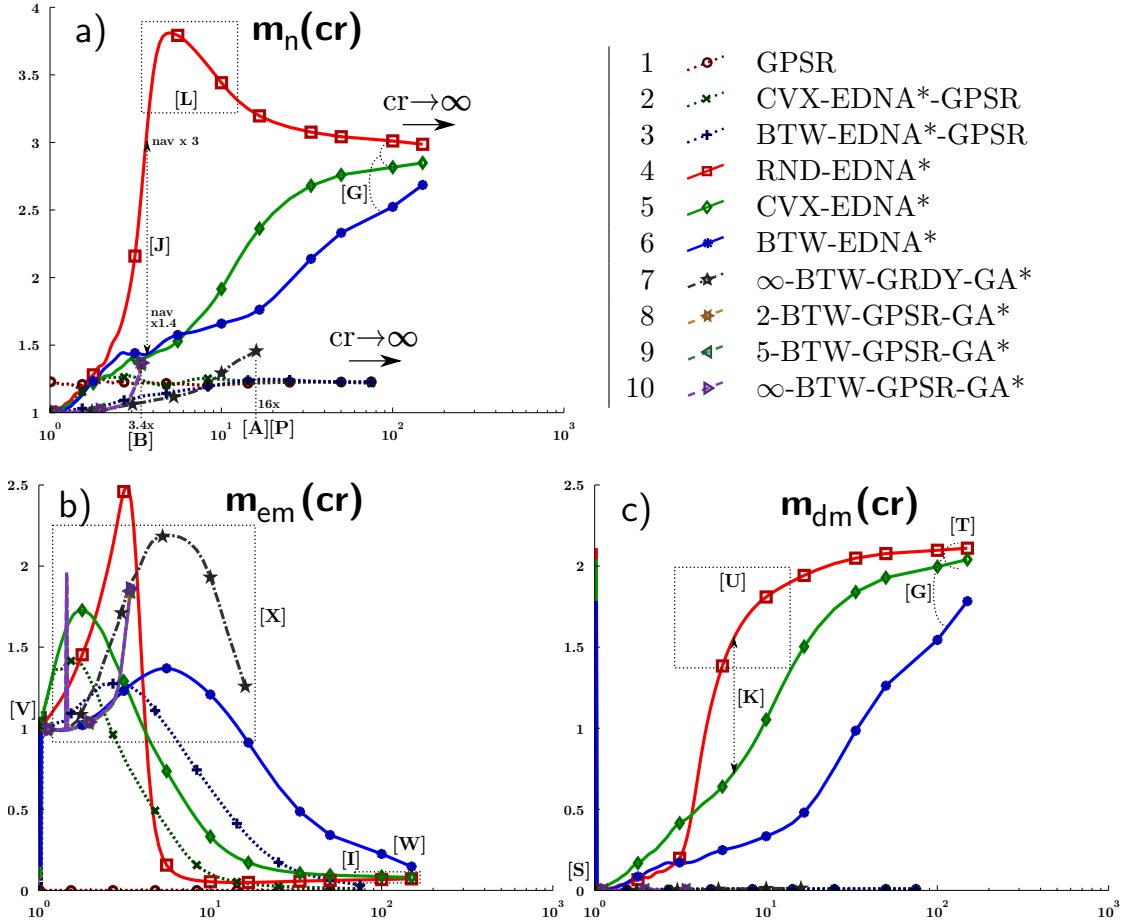



**Figure 8.7:** [GRID] Experiments on the grid-graphs dataset, with line types and colors referring to Table 8.1. (a) Navigation performance $m_n$ loss as a function of the compression rate $cr$. (b) Usage of execution memory $m_{em}$ as a function of the compression rate $cr$. (c) Usage of dynamic memory $m_{dm}$ as a function of the compression rate $cr$. Experiments 8 to 10 can hardly be distinguished from each other on the figure.

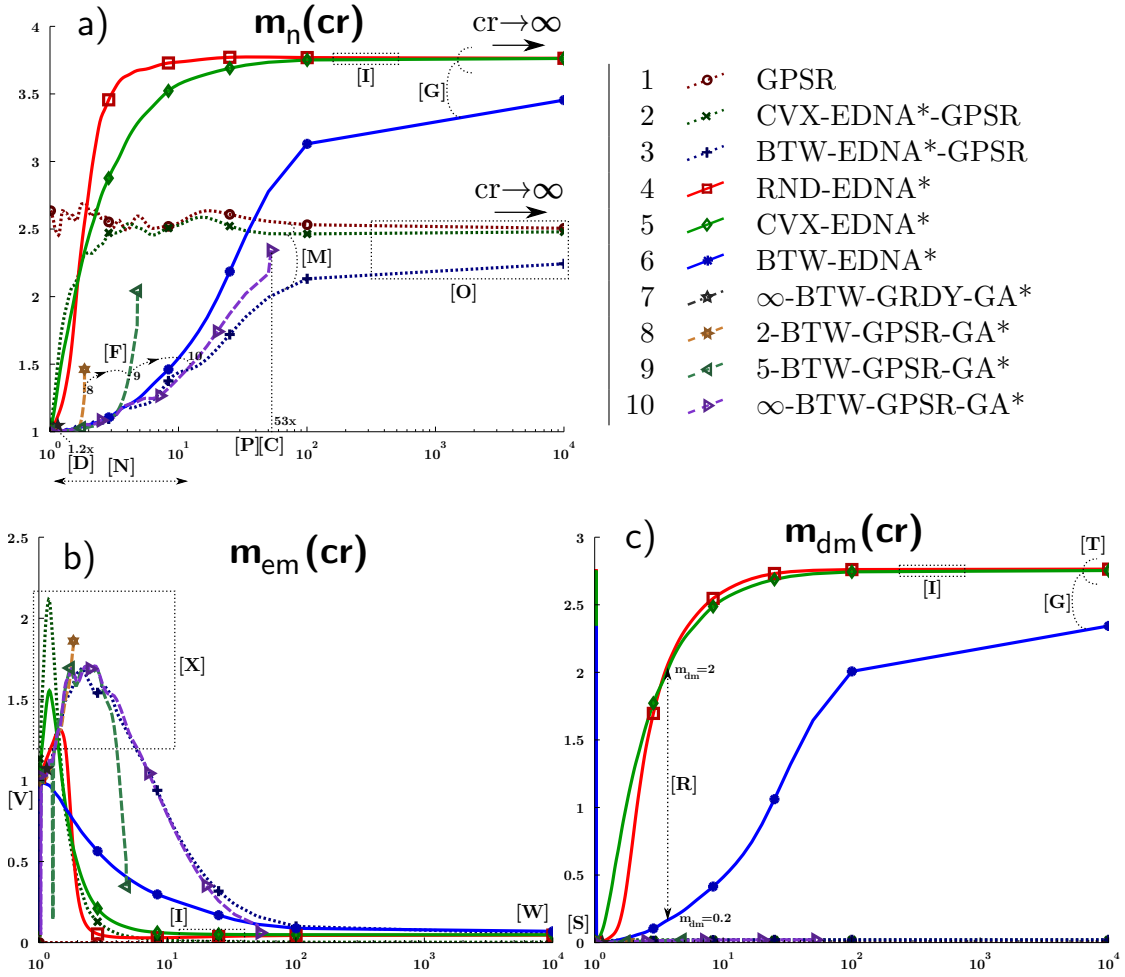



**Figure 8.8:** [RANDOM PLANAR] Experiments on the nearly-planar random graphs dataset, with line types and colors referring to Table 8.1. (a) Navigation performance $m_n$ loss as a function of the compression rate $cr$. (b) Usage of execution memory $m_{em}$ as a function of the compression rate $cr$. (c) Usage of dynamic memory $m_{dm}$ as a function of the compression rate $cr$.

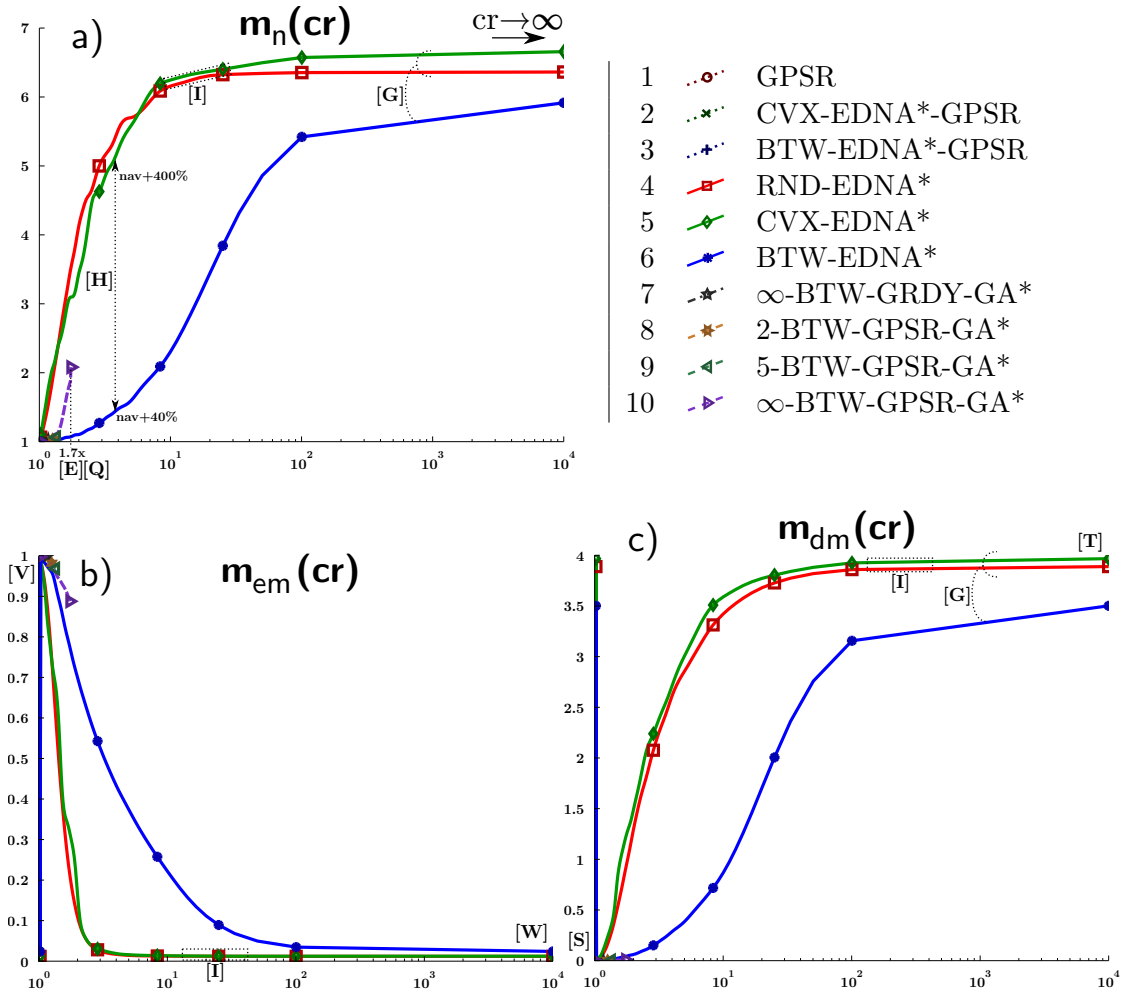



**Figure 8.9:** [RATP] Experiments on the RATP dataset, with line types and colors referring to Table 8.1. (a) Navigation performance $m_n$ loss as a function of the compression rate $cr$. (b) Usage of execution memory $m_{em}$ as a function of the compression rate $cr$. (c) Usage of dynamic memory $m_{dm}$ as a function of the compression rate $cr$. Experiments 7 to 10 are almost invisible on the figure since their maximum compression ratios $cr^*$ are respectively $1.2\times, 1.3\times, 1.4\times$ and $1.7\times$.

However, this simple 8-way connected base is favorable for Greedy preconditioning. On the contrary, nearly-planar random graphs are almost planar, so that GPSR preconditioning "only" has to find and mark as non-planar a few edges to make the graph planar and allow for GPSR to be used.

The RATP dataset is not regular like a grid-graph but also far from planar, so that both preconditioning techniques perform poorly, limiting the compression ratio to a maximum of 1.7×, obtained for experiment 10 [E]. Thus, Greedy-gA* with Greedy preconditioning or GPSR-gA* with GPSR preconditioning are not viable options for graph compression and navigation on compressed data, leaving EDNA* as the only viable PN approach for the RATP dataset and probably for most far-from-planar graphs with complicated topology.

### Navigation performances

Experiments 8 and 9 are clearly superseded by experiment 10 in terms of navigation performances [F] on all datasets these experiments are run on. Experiments 4 and 5 are superseded by experiment 6 in terms of navigation performances and dynamic memory [G], showing that betweenness-centrality is the best compression technique. On the RATP dataset, using betweenness-centrality to compress the graph can lead to more than ten times lower navigational cost overheads than random or *convexify* compression [H]. For nearly-planar random graphs and the RATP dataset, random- and convexify-based compression perform similarly in terms of navigation performances, dynamic and execution memory [I]. For grid-graphs, convexify leads to paths up to twice shorter than random [J], which also reflects in dynamic memory use [K]. We could not find a convincing explanation for the bump around $cr = 6$ of $m_n$ using random compression and EDNA* on grid-graphs [L], even though we suspect that this bump is a side-effect of the size of the holes punched into the graph to create obstacles.

For experiments 1 to 3, betweenness-centrality leads to shorter paths on all datasets, with convexify coming second [M]. GPSR without EDNA* (experiment 1) exhibits navigational performances independent of $cr$, which was expected since GPSR does not take advantage of $\mathcal{G}^C$. For $cr \to 1$, EDNA* dominates the behavior of experiments 2 and 3 [N], leading to performances similar to that of A* on $\mathcal{G}$, and to $m_n \to 1$. For $cr \to \infty$, experiments 2 and 3 are dominated by the behavior of GPSR, so that experiments 1, 2 and 3 exhibit to the same navigational performances in the high-$cr$ zone [O].

For experiments 6, 7 and 10, the following rule of thumb can be used: if $cr^*$ goes over 4 or 5 for experiment 7 or 10, the experiment achieving the highest compression ratio gives the best navigation performances when $cr < cr^*$ [P]. If the compression ratio is under 4 or 5, only EDNA* (experiment 6) is a viable solution [Q].

### Dynamic memory

The behavior of the $m_{dm}$ curves for experiments 4 to 6 is mostly similar to that of the $m_n$ curves, with betweenness-centrality leading to $m_{dm}$ values up to ten times lower than both other compression algorithms [R]. With $cr = 1$, $m_{dm} = 0$ [S] while with $cr \to \infty$, $m_{dm}$ asymptotically tends to a constant equal to 4 or less [T]. So, the amount of dynamic memory used is of the order of magnitude of $4n^*$ or less. The $m_{dm}$ curve using random compression and EDNA* on grid-graphs [U] does not show the bump [L] that was visible on the $m_n$ curve.

It is not possible to limit both $m_{dm}$ and $sm^*$: experiments 4 to 6 use $sm^* = 0$ so that even though $m_{dm} \sim 4$ is an acceptable average, extreme cases show individual $m_{dm}$ higher than 200. Reciprocally, experiments 7 to 10 use $m_{dm} = 0$ but let $sm^*$ vary.

Experiments 2 and 3 were attempted with various $m_{dm}$ limits. However, shorter paths were obtained on average with $m_{dm} = 0$. $m_{dm} = 0$ means that only a single run of EDNA* is allowed at the beginning of the planning and navigation phase (each subsequent execution of EDNA* would require one unit of dynamic memory). We believe that this is due to GPSR being more efficient than the exploration phase of EDNA*. Indeed, EDNA* does not take advantage of $\mathcal{G}$ being planar, which GPSR does.

### Execution memory

Since one unit of execution memory is much less than one unit of static or dynamic memory, execution memory consumption is only given for reference.

Execution memory is computed relative to A* on $\mathcal{G}$. Thus, with $cr \to 1$, $\mathcal{G}^C = \mathcal{G}$ and $m_{em} \to 1$ in experiments 2 to 10 [V]. Conversely, with $cr \to \infty$, $\mathcal{G}^C \to \varnothing$ initially (before dynamic memory comes in), so that $m_{em} \to 0$ [W]. Even with dynamic memory, $m_{em}$ remains bounded by the amount of dynamic memory $m_{dm}$. In the intermediate $cr$ regime, the behavior of $m_{em}$ depends on the dataset. While, in average, $m_{em}$ remains of the order of magnitude of 1 or less, curves of the grid- and random graphs datasets show a bump between $cr = 1$ and $cr = 10$ [X] which does not appear in the $m_{dm}$ and $m_n$ curves. We think that this phenomenon is due to the structure of these datasets, for instance to the size of the rectangular possibly overlapping holes that were punched into them. Even within the bumps, $m_{em}$ does not go beyond 2.5 in average, even though extreme cases show individual $m_{em}$ as high as 40, with $s \sim Card(\mathcal{V})$.

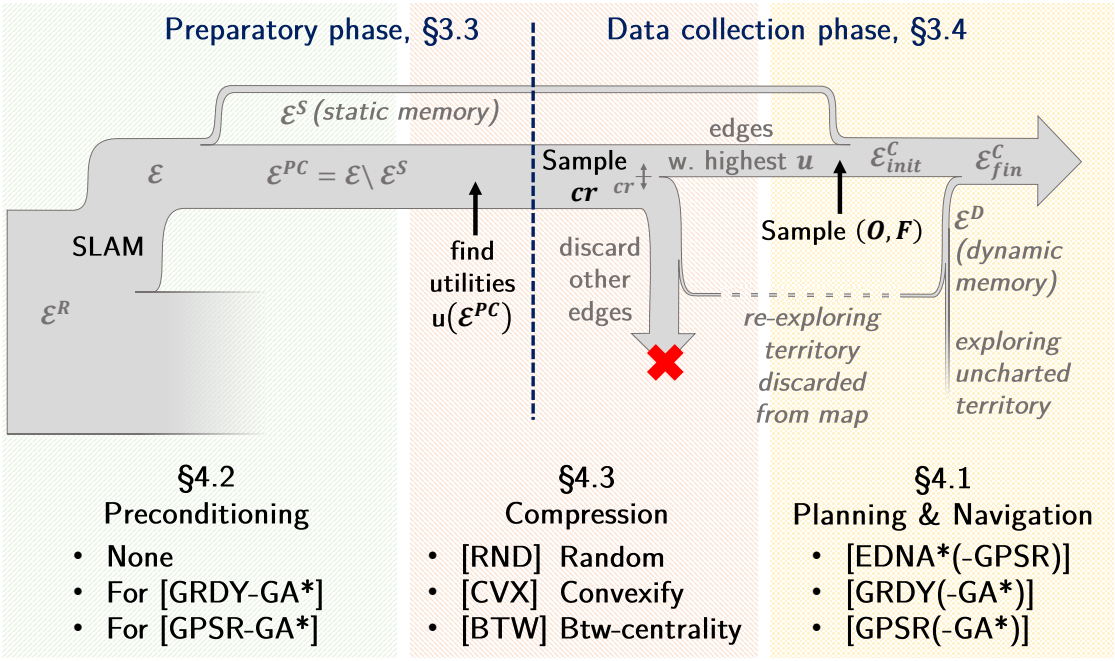



**Figure 8.10:** Graph compression for LEN. In a LEN context, the currently known graph $\mathcal{G}$ is not $\mathcal{G}^R$. Indeed a (potentially large) part of the environment may not have been explored and mapped when compression is applied. Moreover, the robot may explore new areas of the world after compression, so that dynamic memory may contain edges that did not belong to $\mathcal{G}$ before compression.

## 8.6 Graph compression in a robotics context

### 8.6.1 Graph compression in a Lifelong Exploratory Navigation context

So far, this chapter described graph compression independently from other components (SLAM, navigation). We applied compression to a graph $\mathcal{G}$ which describes the whole environment $\mathcal{W}$. Thus, $\mathcal{G}$ is equivalent to the $\mathcal{G}^R$ graph used in chapter 3. Within a Lifelong Exploratory Navigation context, the diagram of Figure 8.1 needs to be extended, which is done on Figure 8.10.

Navigation techniques requiring preconditioning such as GPSR with gA* are not well suited to LEN since their (static) memory usage depends on the total size of the map. Moreover, we do not see a simple way to run preconditioning on an already incomplete graph $\mathcal{G}$ instead of $\mathcal{G}^R$. Thus, pure EDNA*-based techniques should be preferred for LEN in non-planar environments. In planar environments, EDNA* can be assisted by GPSR. EDNA* is able to lead the robot from any origin to any destination no matter the current content of $\mathcal{G}$ or $\mathcal{G}^C$. Moreover, while the *dynamic memory* usage of EDNA*

does not have a theoretical bound lower than $Card(\mathcal{E})$, it is low in practical situations.

As an additional issue, we did not explicitly state *when* graph compression should be used. The general idea, represented on Figure 2.5, page 26, is to run graph compression on the currently stored (incomplete) graph $\mathcal{G}$ in parallel to the rest of the algorithms. We propose to call graph compression in two situations:

- When memory usage reaches an alert threshold, which should be less than the amount of total memory available (typically, 90%).
- When the robot has nothing more important to do, or when the robot is in sleep mode. In this perspective, map compression and optimization (chapter 5) can be viewed as a "dream" phase, where the brain of the robot simplifies its data structures and consolidates its knowledge, similarly to what happens in animals and humans.

In a LEN perspective, the betweenness-centrality compression algorithm can be replaced by an attendance and/or aging metric, where edges more often and more recently visited by the robot are assigned a higher utility, causing edges never traversed or not traversed for a long time to be compressed away. We did not test such a strategy in this chapter since it would require real mission scenarios (some vertices of the graph should be defined as interest points and the robot should be assigned somewhat repetitive missions between these interest points, etc.) and data collection would become more complex and not reproducible. Betweenness-centrality considers all vertices of the graph as of equal interest to the robot.

In chapter 7, we explained that the algorithmic complexity of EDNA* (chapter 3) depended on the distance between $O$ and $F$ and on the number and shape of obstacles on the way. If compression is applied, the execution memory of EDNA* decreases. However, since the execution memory metric actually describes the search space of EDNA*, a decrease in execution memory translates to a decrease in computation time and algorithmic complexity of EDNA*. Within a Lifelong Exploratory Navigation context, this property is a hint that EDNA* should be able to operate in arbitrarily large environments, even though adversely-constructed environments may still require too much memory and computing power for a robot or for any other navigating agent.

### 8.6.2 Orders of magnitude

Using the components described in chapters 3, 4 and 5, we can give order of magnitudes for the memory usage patterns.

Within our implementation, execution memory is essentially free, with measured values

limited to a maximum of a few dozens of kilobytes, even when working on graphs with more than 1000 vertices. One unit of execution memory amounts to a few bytes.

Navigation (chapter 4) operates with bounded memory, typically up to a few megabytes. Navigation memory comprises one local occupancy grid around $300 \times 300$ pixels (scrolling occupancy grid), one around $600 \times 600$ pixels (edge-local occupancy grid) and a few additional vector and scalar grids as well as small data structures such as lists and vectors to perform topological skeleton extraction.

The memory usage of the SLAM framework (chapter 5) is essentially due to the use of a disambiguation strategy, which typically stores up to 5 or 6 edges and vertices. Each edge/vertex couple used in the disambiguation strategy counts as one unit of dynamic memory.

The main memory usage of our framework comes from vertex-local and edge-local occupancy grids stored in the map $\mathcal{G}$. Vertex-local grids weigh around 400 kilobytes each while edge-local grids weigh around 1.4 megabytes. Thus, one unit of static or dynamic memory amounts to about 2 megabytes. The final map for the Cuzco dataset presented in chapter 6 weighs more than 3 gigabytes.

## 8.7 Guidelines for choosing an approach

From the experiments carried in this chapter, we can deduce the following criteria to choose a compression and navigation approach:

- In all cases, compression based on betweenness centrality, attendance or an aging metric should be preferred.

- Within a LEN context, if the graph is not planar, it is necessary to use an Exploratory Planning algorithm such as EDNA*. Even if the graph is planar, EDNA* can be used but GPSR-assisted EDNA* (experiments 2 and 3) performs better than pure EDNA*.

- Outside a LEN context, techniques using preconditioning can be used on undirected graphs if they achieve a high enough compression ratio. Greedy with gA* recovery should be used on graphs close to Delaunay graphs and GPSR with gA* on nearly-planar graphs. The maximum compression ratio achievable with both algorithms depends on how close the graph is to a Delaunay graph (for greedy with gA* recovery) or to a planar graph (for GPSR with gA* recovery). For graphs far from planar and from a Delaunay graph or for directed graphs, the use of EDNA* alone is recommended.

## 8.8 Conclusion

In this chapter, we demonstrated that with adequate preconditioning, compression and planning techniques, navigation performances would degrade smoothly with memory restrictions imposed on the system. We devised planning algorithms whose behavior in situations without memory constraints is optimal (they fall back to A*) and which are either able to navigate without any map initially but using dynamic memory or to navigate with a minimal map ("static memory") but without dynamic memory. We see this static/dynamic memory requirement as a hint that giving a bound on the amount of memory required to navigate between any two vertices of any graph $\mathcal{G}$ is not possible. Navigation requires either a map or the ability to build one. Using GPSR with gA* recovery, a maximum compression ratio of 53× was achieved on the nearly-planar random graph dataset, with paths 2.3 times the theoretical optimum on average. Using EDNA*, navigation without an initial map (infinite compression) led to paths from 3 to 6.5 times longer than if a map was available, using a limited amount of dynamic memory. On the real-world RATP dataset, EDNA* performed best with a doubling of navigational cost around 7.7× compression and an asymptotic 6.5× optimal navigational cost which is quite good given that our implementation of the risk heuristic of EDNA* chooses paths in free space according to geometrical hints (distance, angles), not considering actual costs. For instance, the risk heuristic $\mathcal{R}$ does not use the fact that trains are usually faster than buses. We removed single-way edges from the RATP dataset in order to compare planning algorithms but EDNA* could operate directly on directed graph.

We did not study algorithms with limited execution memory such as finite-horizon planners like gA* with $K \geq 1$ alone, since these are not guaranteed to lead a navigating agent to its destination with one of the preconditioning techniques exposed. Moreover, for any finite-horizon planner, we can always find $\mathcal{G}, O \in \mathcal{V}^C, F \in \mathcal{V}^C$ such that reaching $F$ from $O$ exceeds the horizon of the planner.

Experiments 1 to 3 showed that GPSR outperformed EDNA* during exploratory phases on planar graph. This suggests an improvement of the exploration phase of EDNA* on planar graphs, replacing the Greedy exploration process of EDNA* by GPSR.

We used results from the network literature (Lam and Qian, 2013) to perform Greedy-preconditioning of a graph. We improved greedy preconditioning by introducing GPSR-preconditioning, allowing a 34 times reduction of static memory on the nearly-planar random graphs dataset. Back-porting this new algorithm for use within networks may lead to a drastic reduction of the size of routing tables in nearly-planar router layouts.

Finally, within a Lifelong Exploratory Navigation context, compressing the map reduces memory requirements and the running time of EDNA* while increasing path lengths. Consequently, a compromise between memory, computing power and navigation performances has to be found for Lifelong operation.

# 9 Conclusion

This chapter contains a summary of which modifications had to be introduced to planning, navigation and SLAM in order to implement PNSLAM. With the help of a new memory management unit, Lifelong Exploratoy Navigation can be achieved. This chapter also lists a few improvements that could be implemented into the proposed approach.

## 9.1 Summary of our contributions

### 9.1.1 Modifications introduced for PNSLAM and LEN

In order to implement PNSLAM and LEN, we had to introduce a few conceptual modifications to the usual paradigms of planning, navigation and SLAM.

#### Exploratory Planning and Exploratory Digraph Navigation (chapter 3)

We propose to modify the planning component to take into account space not mapped yet. The EDNA* algorithm (Mayran de Chamisso, Soulier, and Aupetit, 2015) is an *exploratory planner* based on A* (Hart, Nilsson, and Raphael, 1968) which balances exploitation of the map and exploration of uncharted space through a *risk heuristic*. Both exploitation and exploration require a navigation component in order for the robot to physically move in the environment, and a SLAM component in order to update the map according to the environment sensed by the moving robot. We termed *Exploratory Digraph Navigation* the PNSLAM problem viewed from the perspective of planning. Exploratory Planning and Exploratory Digraph Navigation are new paradigms which extend beyond the EDNA* algorithm used to first demonstrate them.

### Local navigation and topology (chapter 4)

EDNA* is a graph-based path planning algorithm, which means that it returns an itinerary made of places and paths. Places and paths are detected locally in the environment every few milliseconds using the topological skeleton (Mayran de Chamisso, Soulier, and Aupetit, 2016). We propose to use places and paths to build a global map of the environment through a hybrid metrical/topological SLAM approach.

We propose to transform sparse graph navigation commands issued by an EDN approach to dense actuator commands using a ski-tow approach, where the robot is loosely attached to the graph by a tow. Avoidance of static and dynamic obstacles is guaranteed when using the Vectorial Euclidean Distance Map, a grid of vectors pointing from each pixel of free space of an occupancy grid to the closest occupied pixel of the grid. The Vectorial euclidean Distance Map is conveniently obtained as a byproduct of topological skeleton extraction.

### Simultaneous Localization and Mapping (chapter 5)

In order to allow path planning, we implemented a SLAM framework maintaining a single robot position at any time. The approximate current position is derived relative to a last known position through odometric integration. Odometric integration is understood as any process obtaining a trajectory through integration, thus including visual odometry for instance. The recent trajectory and the existing map are matched against each other to perform map update, with ambiguous situations resulting in an active disambiguation phase. During active disambiguation, the robot moves in order to find evidence that it is at a specific place in the environment. If active disambiguation succeeds, the place hypothesis is fused into the map, and the robot is virtually relocated on the map.

### Realistic simulations and experiments (chapter 6)

We introduced new metrics based on *comparing the topology of the map and the mapped environment* and *using the map for navigation* to evaluate the quality of a map produced by a sparse topological SLAM such as the one described in chapter 5. These metrics make more sense from the point of view of a mobile robot than metrics describing geometrical correctness relative to ground truth.

Simulations of planning, navigation and SLAM were run with realistic sensor models in multiple environments including one with 184 loops. Indoor robot experiments using the same PNSLAM code were run. Both simulations and experiments produced a map that could be used for navigation and covered the whole environment. Moreover, paths found

on the maps were always the shortest possible or marginally longer than the shortest possible.

Typical PNSLAM missions such as *find a treasure around specified position* could also be successfully carried.

#### Resource management (chapters 7 and 8)

In order to reduce the computing power required for SLAM, we managed to give bounds on the search space to be considered when trying to find if the current place was already visited. This was done thanks to the uncertainty model and the approximate metrical characteristics of the SLAM framework. With a bounded search space, constant time complexity (in the number of places or paths on the map) can be achieved. Experimentally-collected running times show that *in practice*, the running time of the whole PNSLAM framework is constant as a function of the size of the environment.

We introduced graph (lossy) compression for navigation (chapter 8) in order to try and control the amount of memory used by the system. While it is not possible to give a minimal bound on the amount of memory necessary to guarantee that the robot will reach its goal, we found there was a tradeof between map compression and navigation performances. Our view on the problem is that thanks to the proposed compression approaches, memory will not be an issue *in practice* even though adversely constructed datasets would cause memory saturation.

### 9.1.2 Additional contributions

In addition to the properties necessary for PNSLAM and LEN, our SLAM framework (chapter 5) can model one-way paths and point-of-view dependence of place detection (to some extent). A limited amount of dynamic noise (such as passers-by) is also tolerated. The framework separates uncertainty of place detection (*pose* uncertainty) from odometric uncertainty (*movement* uncertainty), which we think is one of the key aspects to consider for large-scale SLAM. Finally, the kidnapped robot problem is solved elegantly as a generalized loop closure problem.

Our topology extraction approach (chapter 4) provides an elegant way to switch from the usual topological skeleton to Beeson et al.'s Extended GVG (2005) with almost no computational overhead. The extended GVG allows navigation in environments locally wider than the range of the distance sensors of the robot.

## 9.2 Future work

We introduced the Lifelong Exploratory Navigation paradigm and a minimal set of algorithms allowing a robot to follow this paradigm. There is room for improvement for these algorithms:

- *Better place signatures*: Detection of places in the environment can be made more robust by adding visual data in vertex signatures. As explained in chapter 2, multi-sensory place signatures are probably used by animals and humans.

- *Improving the support for dynamic environments*: It may be possible to improve hypothesis handling (chapter 5) to better cope with dynamic environments. It may be necessary to allow the graph structure to store the probability for each individual edge to be blocked.

- *Detecting and filtering moving obstacles*: It should be possible to detect and remove moving objects from an occupancy grid. Care should however be taken since moving objects can be confused with occluded zones suddenly becoming visible. Thus, it may be necessary to consider multiple frames before deciding whether an occupied pixel belongs to a dynamic obstacle or not. If moving objects are detected correctly, the dynamic obstacle avoidance behavior can be improved by predicting the trajectory of the obstacle.

- *GVG-assisted occupancy grid matching*: In order to make matching of two occupancy grids more robust without considering a rotational degree of freedom, it is possible to match the topological skeletons extracted from both grids as a first step. Matching the skeletons gives an accurate estimate of the translation and rotation between both grids. The matching process may be implemented similarly to edge remapping (chapter 4, algorithm 4. Furthermore, it may be possible to consider skeletons, and not grids, as vertex signatures.

- *Maneuver unit for non-holonomic robots*: Within chapters 4 and 6, a holonomic robot is used (it can almost turn in place). The maneuver unit (Figure 2.5) required for such a robot is minimalistic. Control of non-holonomic robots is an open research subject, but our approach can provide a local occupancy grid with topological skeleton and Vectorial Euclidean Distance Map (chapter 4), which is a solid base for non-holonomic trajectory planning.

- *Hardware acceleration*: Multiple parts of our algorithms can easily be parallelized, such as topological skeleton extraction and vertex/edge signature matching. Computation of the Vectorial Euclidean Distance Map in chapter 4 and occupancy grid matching could even potentially run on an ASIC or FPGA.

- *3D navigation*: In most cases, maintaining a full 3D map is useless because most environments are either mostly 2D (indoor environments) or 3D but with the possibility to get around obstacles by flying higher (outdoor environments). If a 3D map is needed, EDNA* as well as our SLAM approach remain valid but topology extraction should be performed on voxels (3D pixels) instead of pixels. Our vision of indoor 3D mapping is to have a 2D map describing floor topology used to perform loop closure, and a 3D SLAM used to reconstruct the 3D environment without having to cope with the loop closure problem.

- *EDNA* risk heuristic and exploration strategy*: We proved in chapter 3 that with a perfectly informed risk heuristic and exploration strategy, EDNA* would always find optimal paths in an environment, no matter the initial state of the map of this environment. Consequently, integrating relevant information into the risk heuristic and exploration strategy should be the main concern when trying to reduce the length of paths produced by EDNA*. Within this thesis, we tested EDNA* with a simple risk heuristic based solely on estimated distances and angles and a simple greedy exploration strategy. We can devise strategies to improve both the risk heuristic and exploration strategy. For instance, both could take advantage of semantic clues (say, signs along a road or the fact that it is only possible to cross a highway or a river at a bridge, incurring a huge detour in most cases). The exploration strategy could take advantage of characteristics of the environment, such as planarity through face routing (see chapter 8).

- *EDNA* risk heuristic and disambiguation*: When considering the PNSLAM problem from the point of view of *planning*, EDNA*'s risk heuristic may also be used as a way for SLAM to suggest a trajectory, for instance when disambiguating between paths. It would thus be possible to balance map consistency (represented by quick disambiguation) and navigation efficiency (represented by goal-directed navigation).

- *Really useful missions*: PNSLAM is able to accomplish missions such as "find a treasure". However, we did not describe how the robot recognized it just found the treasure (a simple colored blob detector was used for testing). Moreover, even though it finds the treasure, our PNSLAM approach does not describe how to interact with the treasure. All in all, the integration of our model with higher-level AI still requires testing.

- *Dynamic and progressive compression*: We did not test graph compression within a Lifelong Exploratory Navigation perspective. It may be preferable to run small compression steps often during normal operation or on the contrary to wait until memory saturation and perform compression during a "dream" phase. The time that the robot spends charging or being motionless for whatever reason can be used to run compression with minimal impact on the robot's behavior.

- *Data compression*: In addition to graph compression, we could reduce the precision of vertex or edge signatures (here, occupancy grids) not much used by the robot, using downscaling for instance. Signatures may also be compressed using image Run-Length-Encoding or image compression techniques.

- *Fleet of robots*: The ideas described in this thesis allow Lifelong Exploratory Navigation for a single robot to be implemented. Robot fleets or swarms are not considered. Handling multiple robots is a challenge since each robot is a dynamic obstacle for the other robots. However, multiple robots can also share information, leading to faster operation of the swarm. Swarms raise the issue of communication between robots.

- *Other navigating agents than robots*: Some components we introduced or improved relative to the state of the art are not specific to robot navigation. For instance, topology extraction (chapter 4) may be used in medical imagery and other fields where efficient image processing is needed. Exploratory planning (chapter 3) and memory compression for navigation (chapter 8) may find uses in GPS devices, packet routing or semantic database management.

# Index

# List of Figures

# List of Tables

# Appendices

# A characterization of environments supporting greedy navigation

This appendix contains a characterization of environments supporting greedy navigation.

## 1 Greedy navigation: definition and properties

Greedy navigation is a simple method, which consists in the robot always trying to minimize its distance to destination. This is done iteratively by moving straight towards the goal, following the boundary of obstacles in the way if any. Locally, if multiple movements lead to the same distance to destination, one movement is chosen randomly. The robot is not allowed to stay in place until the destination is reached. Thus, even though the environment does not locally allow a movement bringing the robot closer to its destination, the robot choses a movement leading to the minimal increase of distance to destination.

**Definition 3** (Formal definition of greedy navigation)**.** *Let $E$ be the (connected) set of free space accessible to the robot. $E$ is a subset of $\mathbb{R}^2$ or $\mathbb{R}^3$ depending on whether the robot navigates in 2D or 3D. Let $O$ be the current position of the robot. The destination of the robot is $F \in E$. Let $\epsilon \in \mathbb{R}^{*+}$ be the planning horizon of the robot: from $O$, the robot plans its move up to $O' \mid ||OO'|| < \epsilon$. Let $D : \begin{vmatrix} E \mapsto \mathbb{R}^+ \\ X \mapsto ||XF|| \end{vmatrix}$ Greedy navigation is defined by $O' = argmin(D(X), X \in E, ||OX|| < \epsilon)$. If $||OF|| < \epsilon$, greedy succeeds at reaching $F$. If multiple $O'$ are valid, the one closest to $O$ is chosen. If no $O'$ can be found, the robot does not move.*

The *argmin* behavior of greedy causes paths to be straight lines in free space.

**Definition 4** (Greedy-reachability from a point)**.** *A position $F \in E$ (for "finish") which can be reached in a finite number of planning steps using Greedy from $O \in E$ is called greedy-reachable from $O$.*

**Definition 5** (Greedy-reachability)**.** *A position $F \in E$ (for "finish") which can be reached in a finite number of planning steps using Greedy from any $O \in E$ is called greedy-reachable.*

**Definition 6** (Greedy-supporting environment)**.** *$E$ is said to be a greedy-supporting environment (abridged G-environment) if and only if all its reachable positions are greedy-reachable. Environments which do not support greedy navigation from any reachable position to any other reachable position are called non-G-environments.*

**Property 3** (Greedy-reachability characterization)**.** *Greedy-reachability of $F \in E$ in non-pathological environments is equivalent to: $\forall O \in E, O \neq F, \exists O' \in E|\ ||OO'|| < \epsilon, ||O'F|| < ||OF||$. Thus, $E$ is a G-environment if and only if $\forall F \in E, \forall O \in E, O \neq F, \exists O' \in E|\ ||OO'|| < \epsilon, ||O'F|| < ||OF||$.*

*Proof.* First of all, we have to dismiss a subtle issue related to Zenon's paradox: it is not because the robot can move towards $F$ from any $O \in E$ that $F$ will be reached. Indeed, the robot may only reduce $D$ by an infinitesimal amount at each planning step, as shown on Figure 1. This situation only occurs with weird adversely-designed obstacles such as fractal obstacles where if $O' = argmin(D(X), X \in E, ||OX|| < \epsilon)$, $O' \neq F$ and $O \neq O'$, there is no inferior bound on $\frac{D(O)-D(O')}{\epsilon}$. For non-pathological cases, $F$ is greedy-reachable from $O = O_0$ if and only if at each planning step, $O_{n-1} \neq O_n = argmin(D(X), X \in E, ||O_{n-1}X|| < \epsilon$.

The rest of the proof is pretty straightforward.

Suppose that $\forall O \in E, O \neq F, \exists O' \in E|\ ||OO'|| < \epsilon, ||O'F|| < ||OF||$. Then, if $O'' = argmin(D(X), X \in E, ||OX|| < \epsilon)$, we have $||O''F|| < ||O'F|| < ||OF||$. Thus, from any position $O \neq F$, the greedy algorithm will bring the robot to $O''$ which is *strictly* closer to $F$ than $O$. By iterating the algorithm, the distance to $F$ decreases monotonically until $F$ is eventually reached in non-pathological cases. $F$ is thus greedy-reachable.

Reciprocally, suppose that $F$ is greedy-reachable and that $\exists O \in E, O \neq F, \forall O' \in E|\ ||OO'|| < \epsilon, ||O'F|| \geq ||OF||$. Then, the robot does not move according to greedy. Thus, $F$ is never reached and is not greedy-reachable. So, $F$ is greedy-reachable if and only if $\forall O \in E, O \neq F, \exists O' \in E|\ ||OO'|| < \epsilon, ||O'F|| < ||OF||$.

□

Knowing whether $F$ is greedy-reachable from $O$ can be made easier using the following theorems 10 and 11:

**Theorem 10** (reduction to boundaries: $O$)**.** *Let $O \in E$ such that $O$ is not on an internal or external boundary of $E$. Let $F \in E, F \neq O$ and $]OF)$ be the half-line*

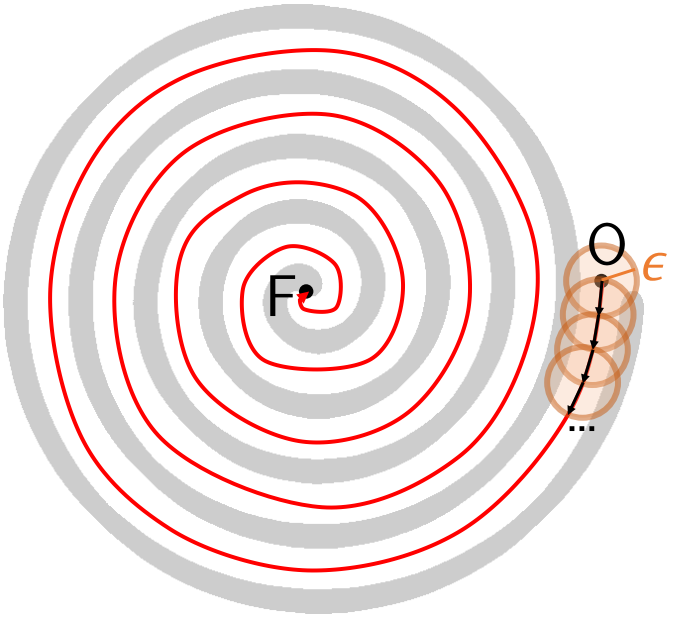



**Figure 1:** An adversely-designed obstacle such as a spiral of infinite length may pose theoretical issues for greedy navigation. For non-pathological obstacles, $F$ is greedy-reachable from $O$ if and only if at each planning step, $O \neq O' = argmin(D(X), X \in E, ||OX|| < \epsilon)$.

*starting at $O$ and going towards $F$.* $\forall M \in ]OF), \exists t \in ]0; +\infty[, M = O + t.OF$. *Let* $t_1 = \sup\{t \in ]0; +\infty[, \forall 0 < u < t, O + u.OF$ *is traversable*$\}$. $M_1 = M(t_1) = O + t_1.OF$ *is the first point of the* $]OF)$ *half-line that belongs to either the boundary of an internal obstacle of $E$ or the external boundary of $E$. Then, $F$ is greedy-reachable from $O$ if and only if $F$ is greedy-reachable from $M_1$.*

*Proof.* Suppose that $F$ is greedy-reachable from $O$. Suppose that $M_1 \in ]OF]$. Paths generated by greedy are straight lines towards $F$ in free space and thus from $O$ to $M_1$. Since $F$ is greedy-reachable from $O$, it is greedy-reachable from any point of the greedy trajectory from $O$ to $F$ including $M_1$. If $M_1 \notin ]OF]$, $[OM_1]$ is entirely located in free space with $F \in [OM_1]$, so $F$ is greedy-reachable from $M_1$.

Reciprocally, suppose that $F$ is greedy-reachable from $M_1$. Suppose that $M_1 \in ]OF]$. Then, greedy from $O$ towards $F$ and from $M_1$ towards $F$ are equivalent since the trajectory using greedy from $O$ towards $F$ is a straight line until $M_1$. If $M_1 \notin ]OF]$, then the $[OF]$ segment is entirely located in traversable space and greedy navigation from $O$ towards $F$ succeeds. □

**Theorem 11** (closer is more restrictive)**.** *Let* $O \in E$. *Let* $F \in E, F \neq O$. $\forall M \in ]OF], \exists t \in ]0; 1], M = O + t.OF$. *Let* $M_2 = M(t_2), t_2 \in ]0; 1]$. *Then, if a greedy move*

*towards $M(t_2)$ is possible from $O$, then a greedy move towards $F$ is possible from $O$.*

**Corollary 3** (reduction to boundaries: F)**.** *It is sufficient to consider greedy-reachability of points $F$ on either the boundary of internal obstacles or the external boundary of $E$.*

*Proof.* First, it is sufficient to consider $O \in E$ on the boundary $S$ of an internal obstacle according to theorem 10. Since $O, M_2$ and $F$ are aligned in this order, $||OF - M_2F|| = ||OF|| - ||M_2F||$. Suppose that a greedy move towards $M(t_2)$ is possible at $O$: $\exists O' \in E|\ ||OO'|| < \epsilon, ||O'M_2|| < ||OM_2||$. $||O'M_2|| < ||OM_2|| \Leftrightarrow ||O'F - M_2F|| < ||OF - M_2F|| \Rightarrow ||O'F|| - ||M_2F|| < ||OF|| - ||M_2F|| \Rightarrow ||O'F|| < ||OF||$, so a greedy move towards $F$ is possible from $O$. Note that we use the property: $\forall (a,b) \in (\mathbb{R}^2)^2, ||a-b|| > |\ ||a|| - ||b||\ |$.

The corollary is proved the following way: first, suppose that $]OF]$ crosses an (internal or external) boundary. Let $M_2 = M(t_2), t_2 = \sup\{t \in ]0;1], O + t.OF$ is not traversable$\}$, which is the last point belonging to an obstacle on the $]OF]$ segment. Since $M_2 \in ]OF]$, if a greedy move towards $M_2$ is possible from $O$, then a greedy move towards $F$ is possible from $O$ according to theorem 11. If $]OF]$ does not cross an (internal or external) boundary, $]OF)$ does. Let $M_2 = M(t_2), t_2 = \inf\{t \in ]1; +\infty[, O + t.OF$ is not traversable $\}$. $[OM_2]$ is entirely contained in traversable space, so if $M_2$ is greedy-reachable from $O$, then $F \in [OM_2]$ is greedy-reachable from $O$. □

When determining whether greedy navigation is possible in an environment, theorems 10 and 11 and corollary 3 can be used to reduce the search to the boundary of obstacles, as shown on Figure 2.

## 2 A characterization of environments supporting greedy navigation

While property 3 provides a rigorous definition of $G$-environments, it is not obvious from the property what $G$-environments look like. In this section, we give a more precise characterization of obstacles allowing greedy navigation. More precisely, we derive a differential description of obstacle boundaries locally allowing greedy navigation.

Let $B_E$ be the external boundary of $E$. The coordinate system is centered on $F$. In the following, we suppose that the planning horizon $\epsilon$ of the robot is infinitesimal and that $E$ is two dimensional.

Suppose that there is an obstacle within $E$. The contour of this obstacle is a $\mathcal{C}^\infty$ curve $S$ parametrized by $t \in [0;1[$. $\forall M \in S, \exists t \in [0;1[, M = (x(t), y(t))$ with $\forall t_0 \in$

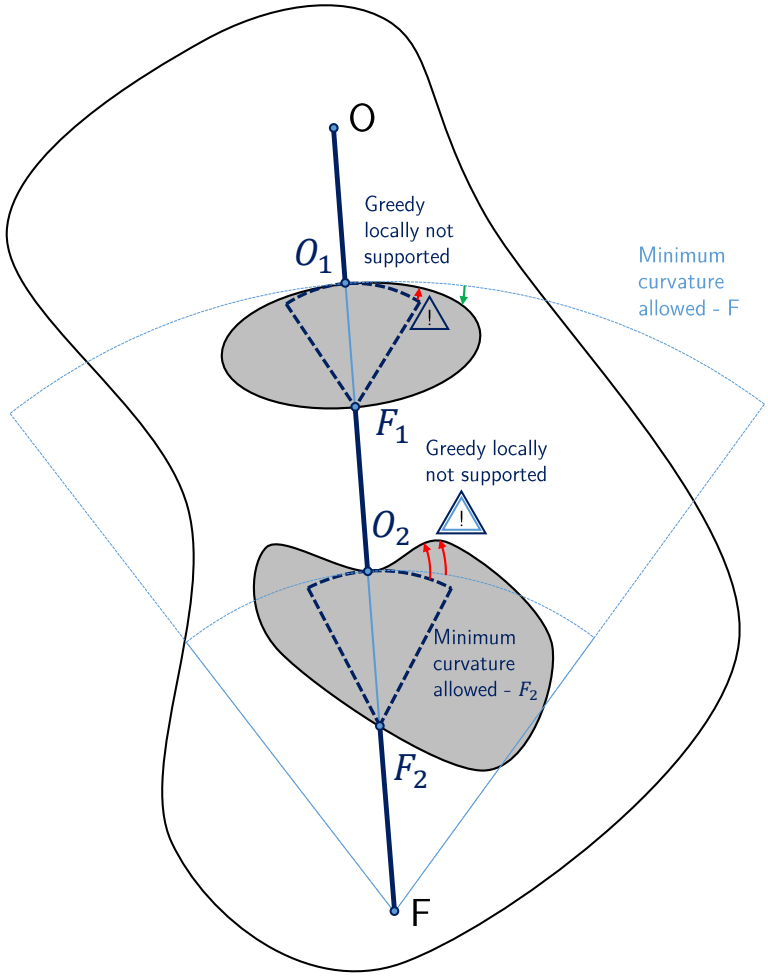



**Figure 2:** It is sufficient to study the trajectories from $O_1$ to $F_1$ and from $O_2$ to $F_2$ to prove greedy-reachability of $F$ from $O$ according to theorems 10, 11 and corollary 3.

$[0;1[, (\frac{dx}{dt}\Big|_{t=t_0}, \frac{dy}{dt}\Big|_{t=t_0}) \neq (0,0)$. Parametrization is chosen so that $(y'(t), -x'(t))$ points outwards.

It is sufficient to study trajectories where $O \in S$ according to theorem 10. So, $\exists t_O \in [0;1[, O = (x(t_O), y(t_O))$. Obstacles will only cause problems on trajectories coming in contact with them, that is when $(y'(t), -x'(t)).(x(t), y(t)) > 0$.

Let $f : \left| \begin{array}{c} [0;1[ \mapsto \mathbb{R}^+ \\ t \mapsto D((x(t), y(t)) \end{array} \right.$. So, $||OF|| = f(t_O)$. Let $n = \min(p \in \{1; +\infty\}, \frac{d^p f}{dt^p}\Big|_{t=t_O} \neq 0)$. Depending on $n$, multiple cases arise:

- If $n = \infty$ ($f$ is constant), the robot stays at $O$ and greedy navigation fails.
- If $n$ is odd:
  - If $\frac{d^n f}{dt^n}\Big|_{t=t_O} > 0$, then $\exists \delta \in \mathbb{R}^{*+}, \forall t \in [0;1[, (t_O - t) < \delta \Rightarrow f(t) < f(t_O)$. Thus, there is a point $O' = (x(t), y(t))$ in any planning horizon $\epsilon < ||(x(t_O), y(t_O))(x(t_O - \delta), y(t_O - \delta))||$ so that $||O'F|| < ||OF||$. Thus, greedy navigation is locally possible from $O$ to $F$ according to property 3.
  - If $\frac{d^n f}{dt^n}\Big|_{t=t_O} < 0$, then $\exists \delta \in \mathbb{R}^{*+}, \forall t \in [0;1[, (t - t_O) < \delta \Rightarrow f(t) < f(t_O)$. Thus, there is a point $O' = (x(t), y(t))$ in any planning horizon $\epsilon < ||(x(t_O), y(t_O))(x(t_O + \delta), y(t_O + \delta))||$ so that $||O'F|| < ||OF||$. Thus, greedy navigation is locally possible from $O$ to $F$ according to property 3.
- If $n$ is even:
  - If $\frac{d^n f}{dt^n}\Big|_{t=t_O} > 0$, then $O$ is a local minimum of the distance to $F$ and $\exists \delta \in \mathbb{R}^{*+}, \forall t \in [0;1[, |t_O - t| < \delta \Rightarrow f(t) > f(t_O)$ and the robot will remain at $O$, never reaching $F$.
  - If $\frac{d^n f}{dt^n}\Big|_{t=t_O} < 0$, then $O$ is a local maximum of the distance to $F$ and $\exists \delta \in \mathbb{R}^{*+}, \forall t \in [0;1[, |t_O - t| < \delta \Rightarrow f(t) < f(t_O)$. Thus, there is a point $O' = (x(t), y(t))$ in any planning horizon $\epsilon < min(||(x(t_O), y(t_O))(x(t_O - \delta), y(t_O - \delta))||, ||(x(t_O), y(t_O))(x(t_O + \delta), y(t_O + \delta))||)$ so that $||O'F|| < ||OF||$. Thus, greedy navigation is locally possible from $O$ to $F$ according to property 3.

Derivatives of $f : t \mapsto \sqrt{x(t)^2 + y(t)^2}$ are hard to express due to the square root. Let us prove that we can ignore it with the following lemma:

**Lemma 1** (studying variations of compound functions)**.** *If $g$ and $h$ are two $C^{n \geq 1}$ functions with values in $\mathbb{R}$, with $h$ defined on $I_h \subset \mathbb{R}$ and $g$ defined and strictly increasing*

*on* $h(I_h)(\forall u \in h(I_h), \left.\frac{dg(t)}{dt}\right|_{t=u} > 0)$*, then* $\forall k \in \{1, ..., n\}, \left.\frac{d^k g\circ h(t)}{dt^k}\right|_{t=t_O} = 0 \Leftrightarrow \forall k \in \{1, ..., n\}, \left.\frac{d^k h(t)}{dt^k}\right|_{t=t_O} = 0$ *and if* $\forall k \in \{1, ..., n-1\}, \left.\frac{d^k h(t)}{dt^k}\right|_{t=t_O} = 0$*, then* $\left.\frac{d^n g\circ h(t)}{dt^n}\right|_{t=t_O} > 0 \Leftrightarrow \left.\frac{d^n h(t)}{dt^n}\right|_{t=t_O} > 0$ *and* $\left.\frac{d^n g\circ h(t)}{dt^n}\right|_{t=t_O} < 0 \Leftrightarrow \left.\frac{d^n h(t)}{dt^n}\right|_{t=t_O} < 0$

*Proof.* For simplicity, we write $\left.\frac{d^n \phi}{dt^n}\right|_{t=u} = \phi^{(n)}(u)$ for any function $\phi$. Now, using induction:

$$f^{(1)}(t_O) = h^{(1)}(t_O) g^{(1)}(h(t_O)) \tag{1}$$

so that:

$$\begin{aligned} &f^{(1)}(t_O) = 0 \Leftrightarrow h^{(1)}(t_O) = 0 \\ &f^{(1)}(t_O) > 0 \Leftrightarrow h^{(1)}(t_O) > 0 \end{aligned} \tag{2}$$

given that $\forall u \in h(I_h), g^{(1)}(u) > 0$. Similarly,

$$f^{(2)}(t_O) = h^{(1)}(t_O)^2 g^{(2)}(h(t_O)) + h^{(2)}(t_O) g^{(1)}(h(t_O)) \tag{3}$$

If $f^{(1)}(t_O) = f^{(2)}(t_O) = 0$, then $h^{(1)}(t_O) = 0$. Moreover, $\forall u \in h(I_h), g^{(1)}(u) > 0$. Thus,

$$\begin{aligned} &f^{(2)}(t_O) = 0 \Leftrightarrow h^{(2)}(t_O) = 0 \\ &f^{(2)}(t_O) > 0 \Leftrightarrow h^{(2)}(t_O) > 0 \end{aligned} \tag{4}$$

Just to see what's going on, let's derive $f$ once again:

$$f^{(2)}(t_O) = h^{(1)}(t_O)^3 g^{(3)}(h(t_O)) + 3h^{(1)}(t_O)^2 h^{(2)}(t_O) g^{(2)}(h(t_O)) + h^{(3)}(t_O) g^{(1)}(h(t_O)) \tag{5}$$

If $f^{(1)}(t_O) = f^{(2)}(t_O) = f^{(3)}(t_O) = 0$, then $h^{(1)}(t_O) = h^{(2)}(t_O) = 0$. Moreover, $\forall u \in h(I_h), g^{(1)}(u) > 0$. Thus,

$$\begin{aligned} &f^{(3)}(t_O) = 0 \Leftrightarrow h^{(3)}(t_O) = 0 \\ &f^{(3)}(t_O) > 0 \Leftrightarrow h^{(3)}(t_O) > 0 \end{aligned} \tag{6}$$

We see that each new derivative $f^{(k)}$ of $f$ creates terms that cancel at $t_O$ as well as one single term in $h^{(k)}(t_O) g^{(1)}(h(t_O))$. Since $\forall u \in h(I_h), g^{(1)}(u) > 0$,

$$\begin{aligned} &f^{(k)}(t_O) = 0 \Leftrightarrow h^{(k)}(t_O) = 0 \\ &f^{(k)}(t_O) > 0 \Leftrightarrow h^{(k)}(t_O) > 0 \end{aligned} \tag{7}$$

which proves the lemma.

□

Now, Let $h : \left| \begin{array}{c} [0;1[ \mapsto \mathbb{R}^{*+} \\ t \mapsto x(t)^2 + y(t)^2 \end{array} \right.$. Let $g : \left| \begin{array}{c} \mathbb{R}^{*+} \mapsto \mathbb{R}^{*+} \\ t \mapsto \sqrt{t} \end{array} \right.$. $f = g \circ h$. $g$ is strictly increasing, that is $\forall u \in \mathbb{R}^{*+}$, $\left.\frac{dg}{dt}\right|_{t=u} > 0$. Thus, lemma 1 can be used and studying the variations of $f$ comes down to studying that of $h$, which leads us to the following property:

**Theorem 12** (obstacles supporting greedy navigation)**.** *Let* $n(t_O) = min(p \in \{1; +\infty\}, h^{(p)}(t = t_O) \neq 0)$. *Obstacles for which greedy navigation succeeds towards destination* $F$*, chosen as origin of the coordinate axes, are those for which:* $\forall t_O \in [0;1[$*, either:*

- $(OF)$ *does not cross the obstacle, that is* $(y'(t), -x'(t)).(x(t), y(t)) < 0$ *or*
- $n(t_O)$ *is odd or*
- $n(t_O)$ *is even and* $h^{(n)}(t = t_O) < 0$*, where* $h^{(n)}(t)$ *is easily expressed as* $\sum_{p=0}^{n} \binom{n}{p} x^{(p)}(t)x^{(n-p)}(t) + y^{(p)}(t)y^{(n-p)}(t)$

*A single obstacle supports greedy navigation if and only if the above characterization is true for every* $O, F \in S^2$.

Theorem 12 can be used to derive the following property:

**Theorem 13** (obstacles supporting greedy navigation: refinement)**.** *In order to find points* $O$ *of* $S$ *where the obstacle locally does not support greedy navigation, it is only necessary to consider* $OF$ *perpendicular to the local tangent vector to* $S$ *at* $O$*. Furthermore, only the point* $F \neq O$ *belonging to the closest internal or external boundary of* $E$ *needs to be considered.*

*Proof.* $h^{(1)}(t) = ((x(t), y(t)).(\frac{dx}{dt}, \frac{dy}{dt}))$. Since $S$ is $C^\infty$, $(\frac{dx}{dt}, \frac{dy}{dt}) \neq (0, 0)$. Additionally, $O \neq F$, so $((x(t), y(t)) \neq (0, 0)$. Consequently, $h^{(1)}(t) = 0$ if and only if $(x(t), y(t))$ and $(\frac{dx}{dt}, \frac{dy}{dt})$ are orthogonal, that is $OF$ is perpendicular to the tangent vector to $S$ at $O$. Furthermore, only the point $F \neq O$ belonging to the closest internal or external boundary of $E$ needs to be considered according to corollary 3. Figure 3 shows a sketch of the construction. □

Theorems 12 and 13 imply that internal obstacles allowing greedy navigation towards any destination are necessarily convex. As can be seen on Figures 4 and 2, convexity of internal obstacles is however not sufficient for environments to support greedy navigation. Conversely, the external envelope of the environment does not need to be convex (Figure 4). A geometrical interpretation of theorem 12 is that greedy navigation gives a condition on *local curvature* of an obstacle, which is more restrictive the closer $F$ is to $O$, as shown on Figures 3 and 2.

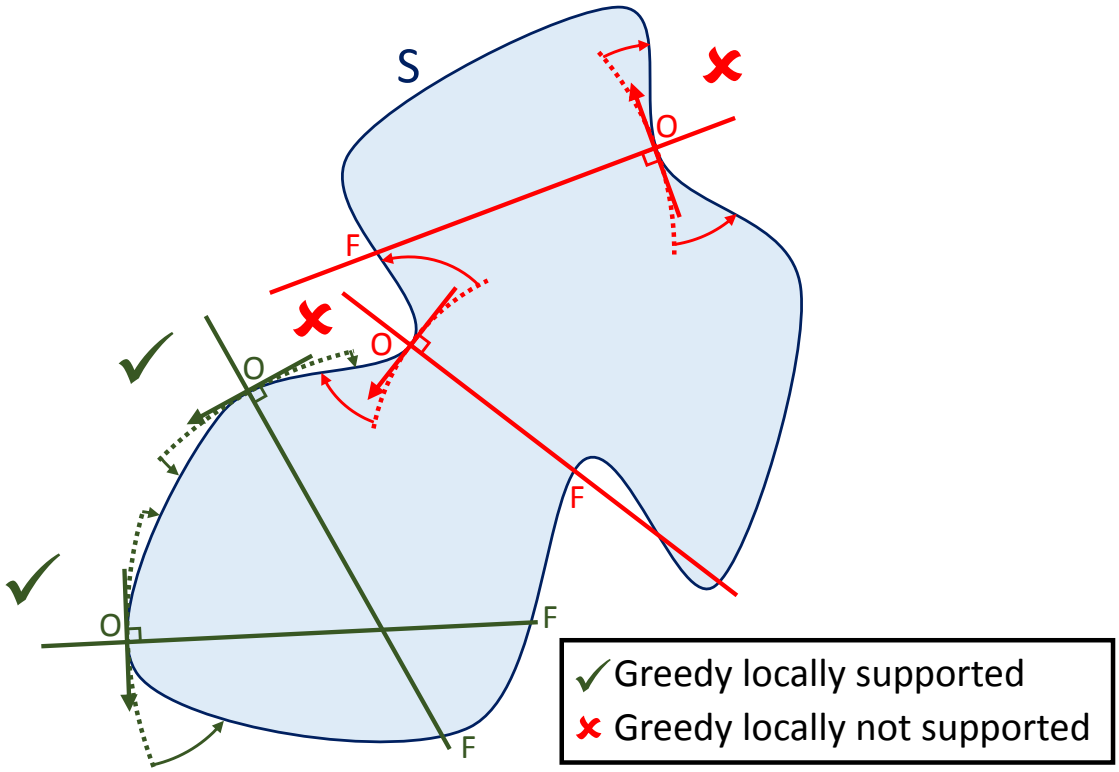



**Figure 3:** Finding whether obstacles locally support greedy navigation at $O$ is done by comparing a circle centered on $F$ and the local boundary at $O$. Corollary 3 and theorem 13 reduce the search for $F$ to the intersection of the obstacle boundary and a half-line orthogonal to the local tangent. Theorem 12 then gives a differential criterion to characterize obstacles locally supporting greedy navigation.

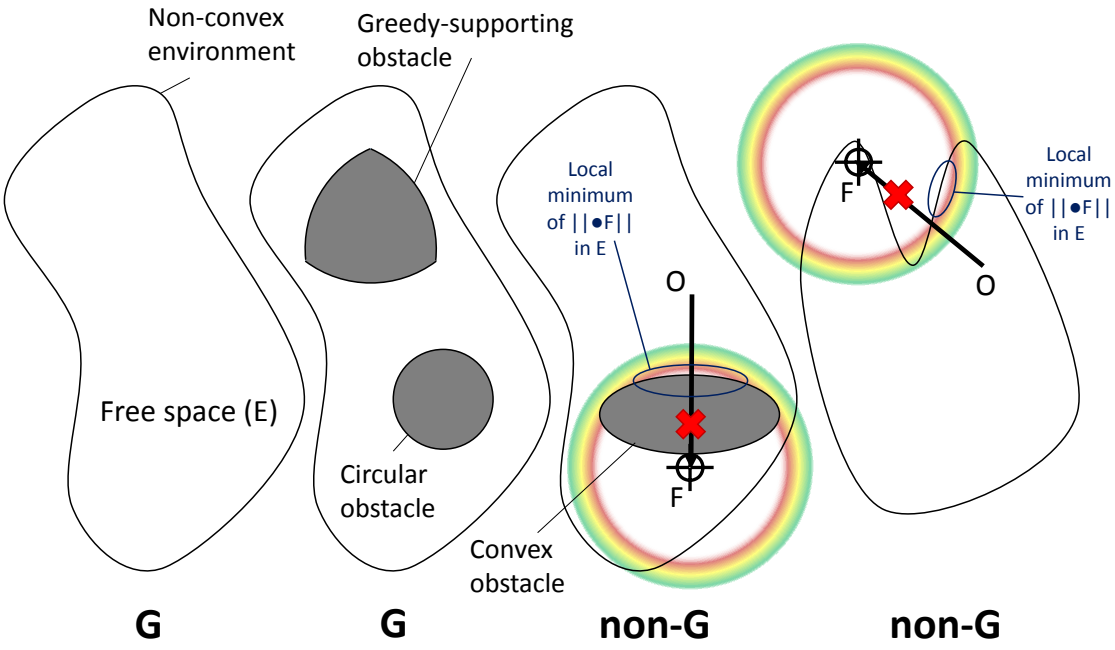



**Figure 4:** Some environments supporting greedy navigation and not supporting it. Supporting greedy navigation is not equivalent to convexity.

Finally, in order to prove that $E$ is a $G$-environment, it is necessary and sufficient to prove that all points $O$ of $E$ on the boundary of obstacles verify theorems 12 and 13. Both theorems are quite restrictive, so that most real-life environments are non-$G$.

# Algorithms

In this appendix, we describe a few algorithms that were written throughout this thesis.

## 1 EDN-(Lazy) Theta* to extend chapter 3

Theta* (Nash et al., 2007) and Lazy Theta* (Nash, Koenig, and Tovey, 2010) are A* variants computing smoother trajectories (Figure 3.12) than A* on graphs representing physical environments with traversable and untraversable space.

This section uses notations defined in chapter 3. Let $g(x \in \mathcal{V}) = D(O_n, X)$. The pseudocodes presented in this section are adapted from `http://aigamedev.com` (last consulted 08/23/2016).

The Theta* algorithm is best derived by starting with A* (Algorithm 11) and tweaking it. The actual Theta* algorithm (Algorithm 12) has its changes relative to A* highlited in red.

Then, Lazy Theta* is derived from Theta* (Algorithm 13, with modifications relative to Theta* highlighted in red).

From Theta*, EDN-Theta* is easily derived (Algorithm 14). The only modifications from Theta* to EDN-Theta* are highlighted in red.

Similarly, we can derive EDN-Lazy Theta* (algorithm(15) as a simple modification of Lazy Theta*. Changes relative to Lazy Theta* are highlighted in red.

## 2 Drawing multiple Voronoï cells for chapter 6, section 6.1.4

The purpose of the algorithms exposed in this section is, given a set of seeds (as vertices in a graph or pixels in a grid), to find for each accessible vertex/pixel to which seed it

**Algorithm 11** A* base

```
procedure Main(O, F, 𝒢, ℋ : X → ℋ(X, F, 𝒢))
    open ← ∅, closed ← ∅, g(O) ← 0, parent(O) ← O
    open.insert(O, g(O) + ℋ(O, F))
    while open ≠ ∅ do
        s ← open.Pop()
        if s = F then
            return path to F
        closed ← closed ∪{s}
        for all s′ ∈ neighbr_vis(s) do
            if s′ ∉ closed then
                if s′ ∉ open then
                    g(s′) ← ∞
                    parent(s′) ← NULL
                UpdateVertex(s, s′)
    return failure (no path found)
procedure UpdateVertex(s,s')
    g_old ← g(s′)
    ComputeCost(s, s′)
    if g(s′) < g_old then
        if s′ ∈ open then
            open.Remove(s′)
        open.Insert(s′, g(s′) + ℋ(s′, F))
procedure ComputeCost(s,s')
    /* Path 1*/
    if g(s) + c(s, s′) < g(s′) then
        parent(s′) ← s
        g(s′) ← g(s) + c(s, s′)
```

**Algorithm 12** Theta*

```
procedure Main(O, F, 𝒢, ℋ : X → ℋ(X, F, 𝒢))
    open ← ∅, closed ← ∅, g(O) ← 0, parent(O) ← O
    open.insert(O, g(O) + ℋ(O, F))
    while open ≠ ∅ do
        s ← open.Pop()
        if s = F then
            return path to F
        closed ← closed ∪{s}
        for all s′ ∈ neighbr_vis(s) do
            if s′ ∉ closed then
                if s′ ∉ open then
                    g(s′) ← ∞
                    parent(s′) ← NULL
                UpdateVertex(s, s′)
    return failure (no path found)
procedure UpdateVertex(s,s')
    g_old ← g(s′)
    ComputeCost(s, s′)
    if g(s′) < g_old then
        if s′ ∈ open then
            open.Remove(s′)
        open.Insert(s′, g(s′) + ℋ(s′, F))
procedure ComputeCost(s,s')
    if lineofsight(parent(s),s′) then
        /* Path 2*/
        if g(parent(s)) + c(parent(s),s′) < g(s′) then
            parent(s′) ← parent(s)
            g(s′) ← g(parent(s)) +c(parent(s), s′)
    else
        /* Path 1*/
        if g(s) + c(s, s′) < g(s′) then
            parent(s′) ← s
            g(s′) ← g(s) + c(s, s′)
```

**Algorithm 13** Lazy Theta*

```
procedure Main(O, F, G, H : X → H(X, F, G))
    open ← ∅, closed ← ∅, g(O) ← 0, parent(O) ← O
    open.insert(O, g(O) + H(O, F))
    while open ≠ ∅ do
        s ← open.Pop()
        SetVertex(s)
        if s = F then
            return path to F
        closed ← closed ∪{s}
        for all s' ∈ neighbr_vis(s) do
            if s' ∉ closed then
                if s' ∉ open then
                    g(s') ← ∞
                    parent(s') ← NULL
                UpdateVertex(s, s')
    return failure (no path found)
procedure UpdateVertex(s,s')
    g_old ← g(s')
    ComputeCost(s, s')
    if g(s') < g_old then
        if s' ∈ open then
            open.Remove(s')
        open.Insert(s', g(s') + H(s', F))
procedure ComputeCost(s,s')
    /* Path 2*/
    if g(parent(s)) + c(parent(s),s') < g(s') then
        parent(s') ← parent(s)
        g(s') ← g(parent(s)) +c(parent(s), s')

procedure SetVertex(s)
    if not lineofsight(parent(s), s) then
        /* Path 1*/
        parent(s) ← argmin_{s'∈neighbor_vis(s)∩closed}(g(s') + c(s', s))
        g(s) ← min_{s'∈neighbor_vis(s)∩closed}(g(s') + c(s', s))
```

**Algorithm 14** EDN-Theta*

```
procedure MAIN(O, F, G, H : X → H(X, F, G), R : Z → R(O, F, Z, G))
    open ← ∅, closed ← ∅, g(O) ← 0, parent(O) ← O
    open.insert(O, g(O) + H(O, F))
    dest ← NULL
    best_distance ← ∞
    while open ≠ ∅ do
        s ← open.Pop()
        if δ(s) > best_distance then
            return path to dest
        if s = F then
            return path to F
        if s ∈ B then
            if R(O, F, s) ≤ best_distance then
                dest ← s; best_distance ← R(O, F, s)
        closed ← closed ∪{s}
        for all s' ∈ neighbr_vis(s) do
            if s' ∉ closed then
                if s' ∉ open then
                    g(s') ← ∞
                    parent(s') ← NULL
                UpdateVertex(s, s')
    return failure (no path found)
procedure UPDATEVERTEX(s,s')
    g_old ← g(s')
    ComputeCost(s, s')
    if g(s') < g_old then
        if s' ∈ open then
            open.Remove(s')
        open.Insert(s', g(s') + H(s', F))
procedure COMPUTECOST(s,s')
    if lineofsight(parent(s),s') then
        /* Path 2*/
        if g(parent(s)) + c(parent(s),s') < g(s') then
            parent(s') ← parent(s)
            g(s') ← g(parent(s)) +c(parent(s), s')
    else
        /* Path 1*/
        if g(s) + c(s, s') < g(s') then
            parent(s') ← s
            g(s') ← g(s) + c(s, s')
```

**Algorithm 15** EDN-Lazy Theta*

**procedure** MAIN($O$, $F$, $\mathcal{G}$, $\mathcal{H} : X \to \mathcal{H}(X, F, \mathcal{G})$, $\mathcal{R} : Z \to \mathcal{R}(O, F, Z, \mathcal{G})$)
- open $\leftarrow \varnothing$, closed $\leftarrow \varnothing$, $g(O) \leftarrow 0$, parent$(O) \leftarrow O$
- open.insert$(O, g(O) + \mathcal{H}(O, F))$
- dest $\leftarrow$ NULL
- best_distance $\leftarrow \infty$
- **while** open $\neq \varnothing$ **do**
  - $s \leftarrow$ open.Pop()
  - SetVertex($s$)
  - **if** $\delta(s) >$ best_distance **then**
    - **return** path to dest
  - **if** $s = F$ **then**
    - **return** path to $F$
  - **if** $s \in \mathcal{B}$ **then**
    - **if** $\mathcal{R}(O, F, s) \leq$ best_distance **then**
      - dest $\leftarrow$ s; best_distance $\leftarrow \mathcal{R}(O, F, s)$
  - closed $\leftarrow$ closed $\cup\{s\}$
  - **for all** $s' \in neighbr_{vis}(s)$ **do**
    - **if** $s' \notin$ closed **then**
      - **if** $s' \notin$ open **then**
        - $g(s') \leftarrow \infty$
        - parent$(s') \leftarrow$ NULL
      - UpdateVertex$(s, s')$
- **return** failure (no path found)

**procedure** UPDATEVERTEX(s,s')
- $g_{old} \leftarrow g(s')$
- ComputeCost$(s, s')$
- **if** $g(s') < g_{old}$ **then**
  - **if** $s' \in$ open **then**
    - open.Remove$(s')$
  - open.Insert$(s', g(s') + \mathcal{H}(s', F))$

**procedure** COMPUTECOST(s,s')
- /* Path 2*/
- **if** $g$(parent$(s)) + c$(parent$(s)$,$s') < g(s')$ **then**
  - parent$(s') \leftarrow$ parent$(s)$
  - $g(s') \leftarrow g$(parent$(s)) + c$(parent$(s)$, $s')$

**procedure** SETVERTEX(s)
- **if not** lineofsight(parent$(s)$, $s$) **then**
  - /* Path 1*/
  - parent$(s) \leftarrow argmin_{s' \in neighbor_{vis}(s) \cap closed}(g(s') + c(s', s))$
  - $g(s) \leftarrow min_{s' \in neighbor_{vis}(s) \cap closed}(g(s') + c(s', s))$

is the closest. We propose four algorithms to perform this task: MHydra1, MHydra2, Multi Theta* and Lazy Multi Theta*.

## 2.1 MHydra

MHydra algorithms operate on a grid of pixels, which in our implementation is a 2D grid, though 3D grids do not pose problems. The principle of MHydra algorithms is to run multiple (whence the "M" of MHydra) iterations of the Vectorial Euclidean Distance Map algorithm of Danielsson (Danielsson, 1980). More precisely, $k$ iterations are necessary, where $k$ is the maximum number of segments from one free pixel to the seed closest to it. The complexity of MHydra algorithms is thus $kmn$ where $m \times n$ is the dimension of the 2D grid. We call *hop pixel* points where the optimal trajectory from a free pixel to its closest seed is not a straight line (SL). Iteration $i$ of an MHydra algorithm finds all hop pixels $i$ hops away from their seed, knowing that a pixel $i$ hops away from a seed may actually be $i + j, j > 0$ hops away from another closer seed. MHydra algorithms stop when no hop pixel was found during an iteration.

If you wonder where the "Hydra" part of MHydra comes from, there are two explanations: first, the algorithm of Danielsson propagates vectors somewhat like water around the seed. Second, an Hydra is a mythological creature with multiple heads, and the algorithm splits the propagation wavefront at each intersection. MHydra1 (Algorithm 16, Figure 1) and MHydra2 (Algorithm 17, Figure 2) are two different implementations of the same method. MHydra2 does not use a temporary buffer and is faster than MHydra1. However, Mhydra1 is slightly more accurate (up to 1 pixel more accurate) since it considers more hop pixels.

## 2.2 Multi Theta*

Multi Theta* (Algorithm 18, Figure 3) is a variant of Theta* (Nash et al., 2007) and Lazy Multi Theta* (Algorithm 19, Figure 4) is a variant of Lazy Theta*. The principle of both algorithms is very simple: (Lazy) Theta* reaches one destination from one origin. (Lazy) Multi Theta* reach multiple destinations from multiple origins. The difference between Multi-algorithms and the original algorithm they are derived from are highlighted in red in algorithms 18 and 19. Essentially, the guiding heuristic $\mathcal{H}$ is set to 0, references to a destination $F$ are removed and multiple seeds $O$ are used as origin. The algorithms stop when the "open" set is empty (it is empty if no path can be improved).

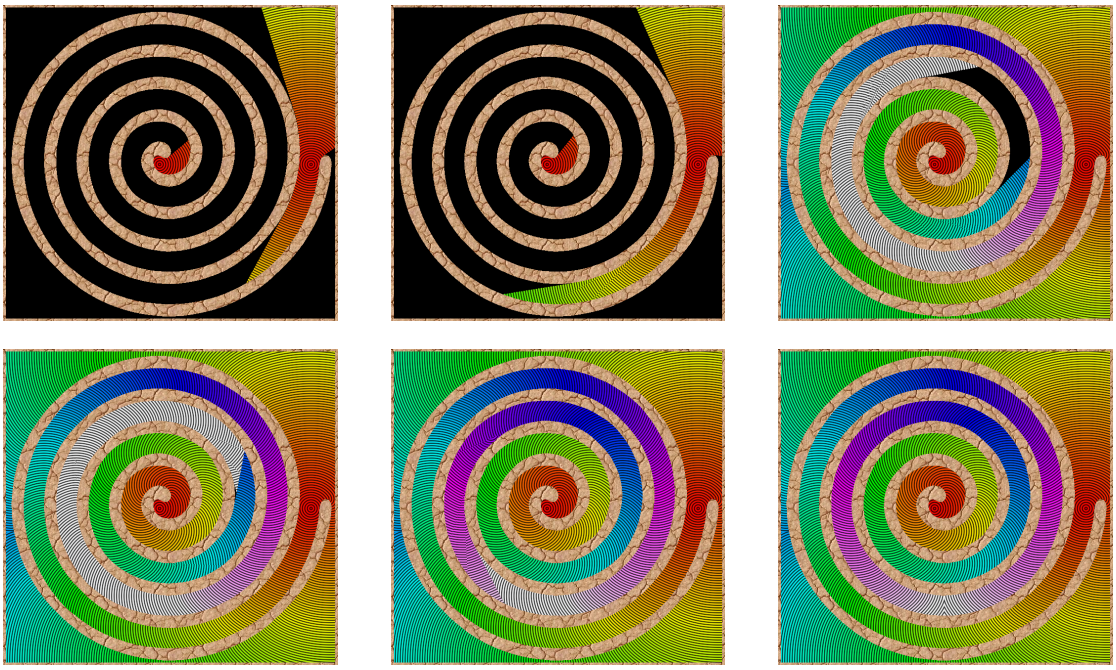


**Figure 1:** Steps $1, 2, 34, 37, 51$ and 161 (final) of MHydra1. Colors represent the minimum distance to a seed. Wavefronts issued from the two seeds (middle and right) propagate within the spiral until they meet and merge. Propagation continues until no distance to an existing hop pixel can be minimized and no new hop pixel can be found. The obtained distance field is perfectly smooth, without any artifact.

**Algorithm 16** MHydra1

```
procedure MAIN(grid G(m × n), seeds S ⊂ G)
  | L ← empty m × n grid //to store vector norms
  | V ← empty m × n grid of 2D vectors, I ← empty m × n grid of integers
  | fill all components of L and V with value ∞
  | for all pixel p ∈ G do
  |   | if p ∈ S then V(p) ← (0,0), I(p) ← 1, L(p) ← 0
  |   | else if G(p) > 0 then I(p) ← 0 //occupied pixel will not be modified
  |   | else I(p) ← ∞ //free pixel is allowed to be modified
  | c ← 0 //current hop counter
  | repeat
  |   | MPROPAGATE1(c, G, I, V, L), c ← c + 2
  | until Not MFINDHOPPIXELS(c, G, I, V)
procedure MPROPAGATE1(c, G, I, V, L)
  | directions ← {TLtoBR, TRtoBL, BRtoTL, BLtoTR}
  | for all d ∈ directions do //TLtoBR = top left to bottom right
  |   | xstep ← ±1, ystep ← ±1 corresponding to d //(1,1) for BLtoTR
  |   | for all pixel p ∈ G iterated according to d do
  |   |   | for all dp ∈ {(xstep, 0), (0, ystep), (xstep, ystep)} do
  |   |   |   | if I(p + dp) ≠ c + 1 and I(p + dp) ≠ 0 then
  |   |   |   |   | if I(p + dp) = c + 1 then
  |   |   |   |   |   | if L(p) + ||dp|| < L(p + dp) then
  |   |   |   |   |   |   | L(p + dp) ← L(p) + ||dp||, V(p + dp) ← −dp //p is a hop pixel
  |   |   |   |   | else//we have to check our neighbors for a shared hop pixel
  |   |   |   |   |   | b ← (L(p+V(p))+||V(p)−dp|| < L(p+dp)) //temporary boolean
  |   |   |   |   |   | if b and lineofsight(p+dp, p+V(p)) then
  |   |   |   |   |   |   | L(p + dp) ← L(p + V(p)) + ||V(p) − dp||
  |   |   |   |   |   |   | V(p + dp) ← V(p) − dp //point p + dp to p's hop pixel
function MFINDHOPPIXELS(c, G, I, V, L)
  | tL ← L, haschanged ← false
  | for all p ∈ G excluding a one pixel border do
  |   | if I(p) ≠ c and I(p) ≠ 0 and V(p) ≠ ∞ then
  |   |   | for all dp ∈ {(0, ±1), (±1, 0), (±1, ±1)} do
  |   |   |   | if V(p + dp) ≠ 0 and tL(p) + ||dp|| < tL(p + dp) then
  |   |   |   |   | tL(p + dp) ← tL(p) + ||dp||, I(p) = c + 1, I(p + dp) = c
  |   |   |   |   | haschanged ← true
  | return haschanged
```

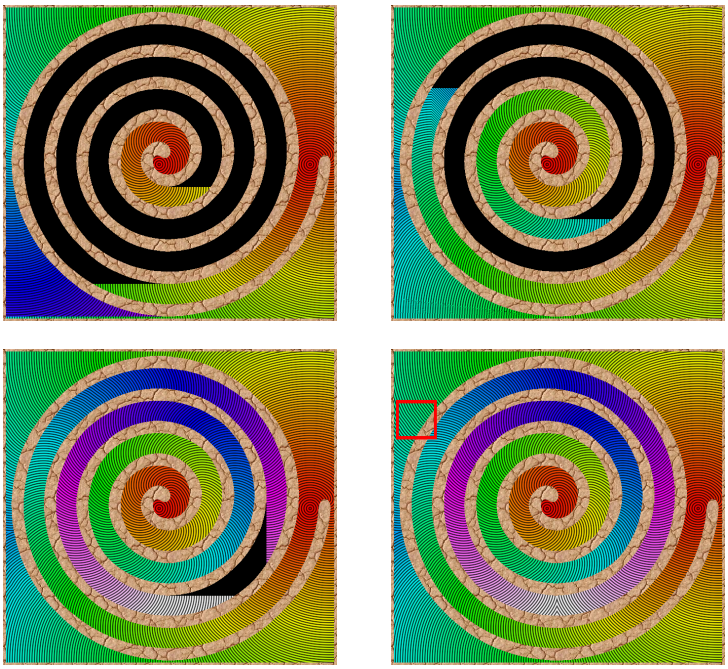


**Figure 2:** Steps $1, 2, 3$ and $7$ (final) of MHydra2. Colors represent the minimum distance to a seed. Wavefronts issued from the two seeds (middle and right) propagate within the spiral until they meet and merge. Propagation continues until no distance to an existing hop pixel can be minimized and no new hop pixel can be found. The obtained distance field is almost as smooth as that obtained with MHydra1, albeit with a few artifacts notably in the zone marked with a red square.

**Algorithm 17** MHydra2

```
procedure MAIN(grid G(m × n), seeds S ⊂ G)
    L ← empty m × n grid //to store vector norms
    V ← empty m × n grid of 2D vectors, I ← empty m × n grid of integers
    fill all components of L and V with value ∞
    for all pixel p ∈ G do
        if p ∈ S then V(p) ← (0,0), I(p) ← 1, L(p) ← 0
        else if G(p) > 0 then I(p) ← 0 //occupied pixel will not be modified
        else I(p) ← ∞ //free pixel is allowed to be modified
    c ← 2 //current hop counter
    repeat c ← c + 1
    until Not MPROPAGATE2(c, G, I, V)
function MPROPAGATE2(c, G, I, V, L)
    mustcontinue ← false
    directions ← {TLtoBR, TRtoBL, BRtoTL, BLtoTR}
    for all d ∈ directions do //TLtoBR = top left to bottom right
        xstep ← ±1, ystep ← ±1 corresponding to d //(1,1) for BLtoTR
        for all pixel p ∈ G iterated according to d do
            for all dp ∈ {(xstep, 0), (0, ystep), (xstep, ystep)} do
                if I(p + dp) ≥ c − 1 and I(p + dp) ≠ ∞ then
                    if V(p) ≠ V(p + dp) + dp then
                        b ← (L(p + dp + V(p + dp)) + ||V(p + dp) + dp|| < L(p + dp))
                        if b and lineofsight(p, p + dp + V(p + dp)) then
                            L(p) ← L(p + dp + V(p + dp)) + ||V(p + dp) + dp||
                            V(p) = V(p + dp) + dp, I(p) = c
                            mustcontinue ← true
            for all dp ∈ {(xstep, 0), (0, ystep), (xstep, ystep)} do
                if I(p + dp) ≥ c − 1 and I(p + dp) ≠ ∞ then
                    if V(p) ≠ dp then
                        if L(p + dp) + ||dp|| < L(p) then
                            L(p) ← L(p + dp) + ||dp||, V(p) = dp, I(p) = c
                            mustcontinue ← true
    return mustcontinue
```

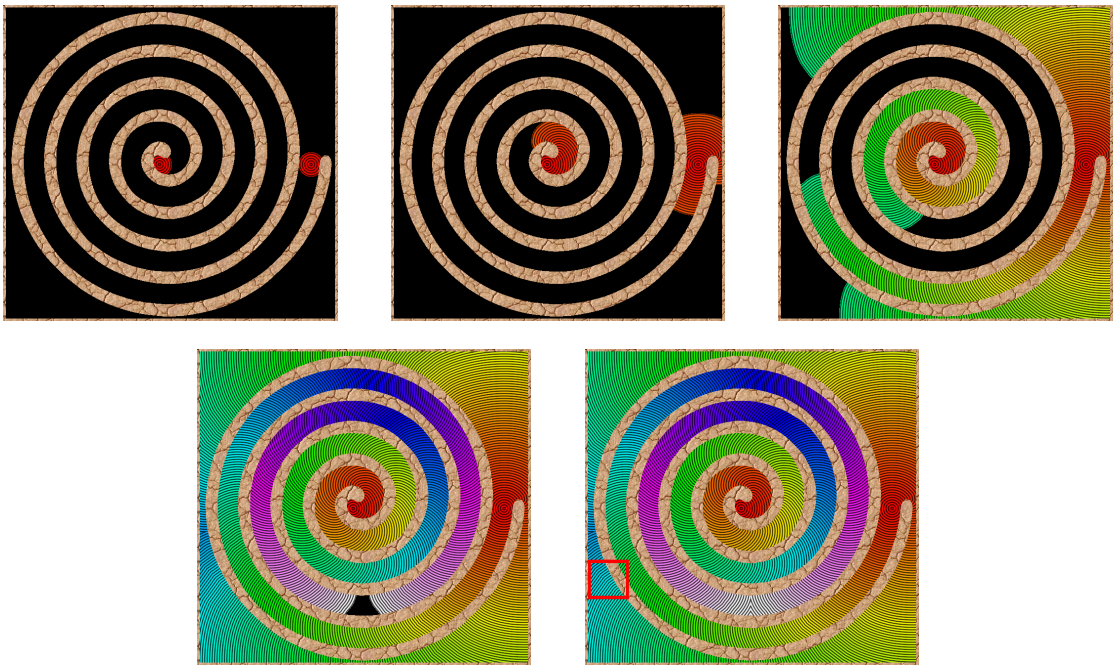


**Figure 3:** Steps of Multi Theta*. Colors represent the minimum distance to a seed. Wavefronts issued from the two seeds (middle and right) propagate within the spiral until they meet and merge. Propagation continues until no distance to an existing hop pixel can be minimized and no new hop pixel can be found. The obtained distance field is the least smooth of all algorithms presented in this section, with a lot of artifacts especially in the zone marked by a red square.

**Algorithm 18** Multi Theta*

```
procedure Main(G, S ⊂ V)
    open ← ∅, closed ← ∅, g(∀O ∈ S) ← 0
    ∀O ∈ S, parent(O) ← o
    ∀O ∈ S, open.insert(O, g(O))
    while open ≠ ∅ do
        s ← open.Pop()
        closed ← closed ∪{s}
        for all s' ∈ neighbr_vis(s) do
            if s' ∉ closed then
                if s' ∉ open then
                    g(s') ← ∞
                    parent(s') ← NULL
                UpdateVertex(s, s')
    return success
procedure UpdateVertex(s,s')
    g_old ← g(s')
    ComputeCost(s, s')
    if g(s') < g_old then
        if s' ∈ open then
            open.Remove(s')
        open.Insert(s', g(s'))
procedure ComputeCost(s,s')
    if lineofsight(parent(s),s') then
        /* Path 2*/
        if g(parent(s)) + c(parent(s),s') < g(s') then
            parent(s') ← parent(s)
            g(s') ← g(parent(s)) +c(parent(s), s')
    else
        /* Path 1*/
        if g(s) + c(s, s') < g(s') then
            parent(s') ← s
            g(s') ← g(s) + c(s, s')
```

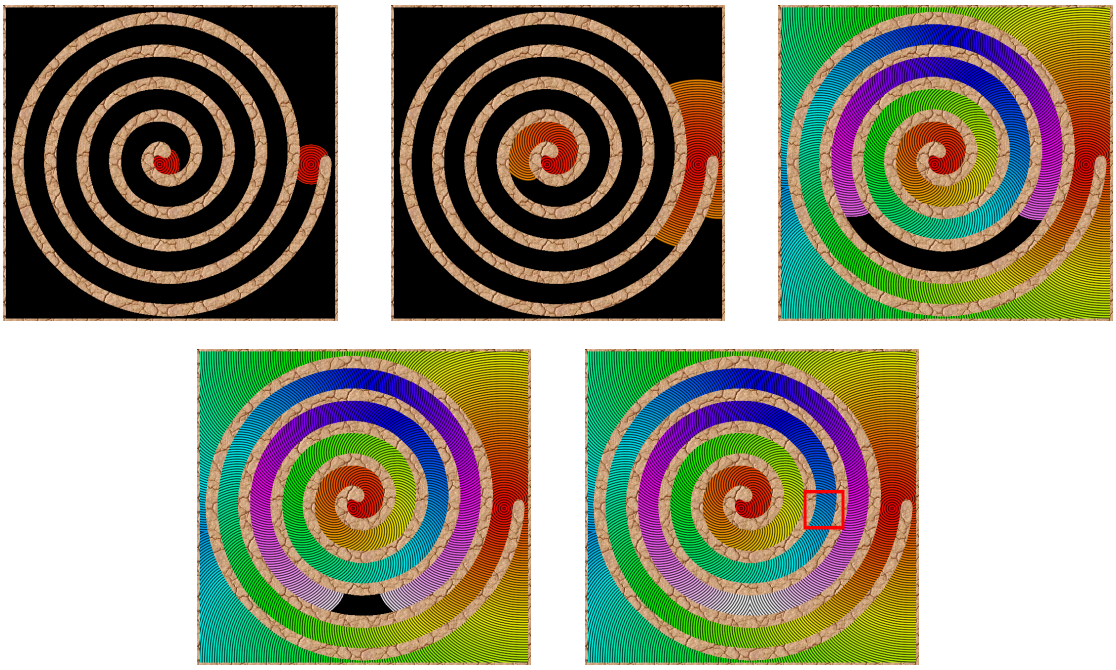


**Figure 4:** Steps of Lazy Multi Theta*. Colors represent the minimum distance to a seed. Wavefronts issued from the two seeds (middle and right) propagate within the spiral until they meet and merge. Propagation continues until no distance to an existing hop pixel can be minimized and no new hop pixel can be found. The obtained distance field is quite smooth but still shows artifacts especially in the zone marked with a red square.

**Algorithm 19** Multi Lazy Theta*

**procedure** MAIN($\mathcal{G}$, $S \subset \mathcal{V}$)
  open $\leftarrow \varnothing$, closed $\leftarrow \varnothing$, $g(\forall O \in S) \leftarrow 0$
  $\forall O \in S$, parent$(O) \leftarrow o$
  $\forall O \in S$, open.insert$(O, g(O))$
  **while** open $\neq \varnothing$ **do**
    $s \leftarrow$ open.Pop()
    SetVertex$(s)$
    closed $\leftarrow$ closed $\cup\{s\}$
    **for all** $s' \in neighbr_{vis}(s)$ **do**
      **if** $s' \notin$ closed **then**
        **if** $s' \notin$ open **then**
          $g(s') \leftarrow \infty$
          parent$(s') \leftarrow$ NULL
        UpdateVertex$(s, s')$
  **return** success
**procedure** UPDATEVERTEX(s,s')
  $g_{old} \leftarrow g(s')$
  ComputeCost$(s, s')$
  **if** $g(s') < g_{old}$ **then**
    **if** $s' \in$ open **then**
      open.Remove$(s')$
    open.Insert$(s', g(s'))$
**procedure** COMPUTECOST(s,s')
  /* Path 2*/
  **if** $g($parent$(s))$ + $c($parent$(s)$,$s')$ < $g(s')$ **then**
    parent$(s') \leftarrow$ parent$(s)$
    $g(s') \leftarrow g($parent$(s))$ +$c($parent$(s)$, $s')$
**procedure** SETVERTEX(s)
  **if not** lineofsight(parent$(s)$, $s$) **then**
    /* Path 1*/
    parent$(s) \leftarrow argmin_{s' \in neighbor_{vis}(s) \cap closed}(g(s') + c(s', s))$
    $g(s) \leftarrow min_{s' \in neighbor_{vis}(s) \cap closed}(g(s') + c(s', s))$

### 2.3 Comparison of MHydra and Multi Theta* algorithms

While (Lazy) Multi Theta* requires dynamical memory allocation, it may run faster than MHydra1/2 on a grid of pixels. Moreover, (Lazy) Multi Theta* can operate on arbitrary graphs and not just grids. A quick performance benchmark of the relative execution times of MHydra1, MHydra2, Multi Theta* and Lazy Multi Theta* shows that MHydra1, MHydra2 and Multi Theta* are respectively 75, 2.5 and 1.8 times slower than Lazy Multi Theta* on the "spiral" dataset. The "spiral" dataset is however highly unfavorable to MHydra algorithms since the number of segments on a path from a seed to a pixel is very high in a spiral (MHydra1 found 161 segments). Thus, we expect MHydra1 and MHydra2 to perform similarly to Lazy Multi Theta* on simpler datasets in terms of execution speed. There are slight difference of the order of magnitude of 1 pixel between results produced by all four algorithms due to slightly different definitions of traversable and blocked space used by each algorithm. The most accurate and glitch-free result is obtained by MHydra1, but the 75× execution time degradation is a serious drawback of this algorithm. Lazy Multi Theta* is faster than Multi Theta* and at least as accurate, so that we do not see a reason to use Multi Theta* altogether.

## 3 Data obsolescence in the occupancy grid for chapter 4

When the robot moves further than a certain distance $d$ from its previous position $P_{n-1}$, an aging process is applied to the occupancy grid. This aging process is necessary since when the grid scrolls, pixels on the far left will appear on the far right (for instance) due to wrap-around addressing. Let $n \times n$ be the dimensions of the grid and $m < n$ be the maximum sensor range. Let $P_n$ be the current position of the robot in the world, measured by odometry and eventually corrected by a local SLAM process. The grid is addressed so that $(0, 0)$ in the grid corresponds to $P_n$. Let $p_{xy}$ be a pixel at position $(x, y)$ in the grid, where $(x, y) \in [-n/2; n/2]^2$ The value of $p_{xy}$ is a log-odd describing whether space is occupied ($p_{xy} > 0$) or empty ($p_{xy} < 0$). This pixel was last updated $t_{xy}$ time steps ago (Figure 5), where one time step can correspond for instance to one sensor frame or to one millisecond (for most sensors with a steady update rate, both measurements should be equivalent). Algorithm 20 is applied:

In algorithm 20, temporal and spatial aging is applied. *aspeed* is the global aging speed, which can vary from 0 (no aging) to 1 (fastest aging). We used $d = m - n, aspeed = 1, r_{min} = d$ and $r_{max} = m = n - d$ in this thesis. $t_{min}$ and $t_{max}$ are chosen according to the sensor configuration of the robot through trial and error. With a too low $t_{min}$, useful information may be removed early from the grid while with too high $t_{max}$, the grid will be polluted by old data, potentially inaccurate and misplaced due to odometric drift. Figure 6 presents multiple occupancy grids collected on a (simulated) robot trajectory, with aging applied.

**Algorithm 20** Occupancy grid obsolescence

**Input:** $r_{min} \in [0; n], r_{max} \in ]r_{min}; n], t_{min} \in [0; +\infty], t_{max} \in ]t_{min}; +\infty], aspeed \in [0, 1]$
**for all** pixels $p_{xy}$ of the grid **do**
  $rsq \leftarrow x^2 + y^2$
  **if** $t_{xy} \geq t_{max}$ **or** $rsq \geq r_{max}^2$ **then**//is pixel obsolete or too far?
    $p_{xy} \leftarrow 0$ //reset the pixel because it is too obsolete
  **else if** $t_{xy} \geq t_{min}$ **or** $rsq \geq r_{min}^2$ **then**//should aging be applied?
    $afactor \leftarrow 1 - \frac{\max(rsq, r_{min}^2) - r_{min}^2}{r_{max}^2 - r_{min}^2}$
    **if** $t_{xy} \geq t_{min}$ **then**
      $afactor \leftarrow (1 - \frac{t_{min} - t_{xy}}{t_{min} - t_{max}}).afactor$
    $p_{xy} \leftarrow ((afactor - 1).aspeed + 1).p_{xy}$

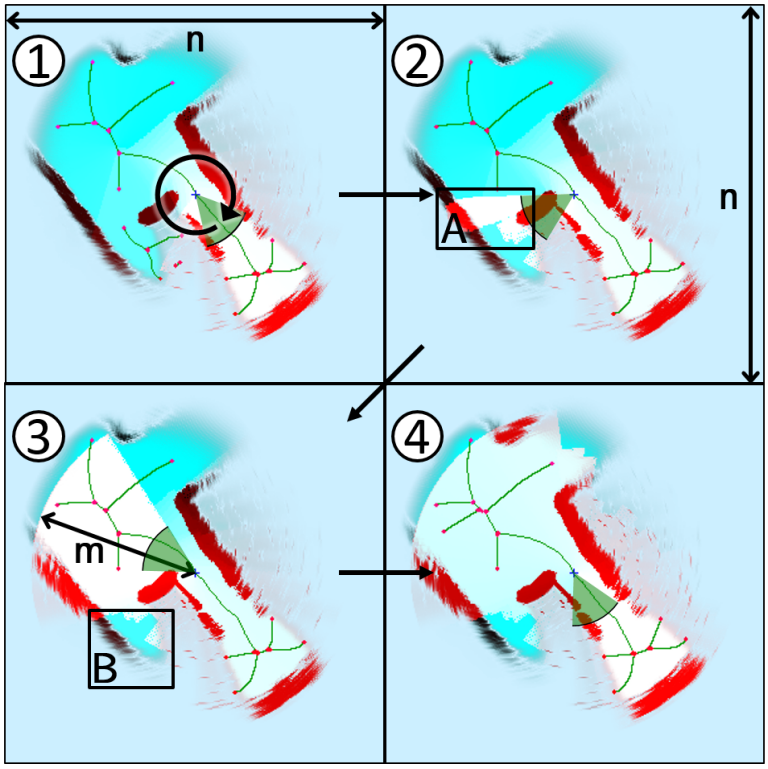



**Figure 5:** Evolution of $t_{xy}$ observed while the robot is performing a 360° clockwise rotation. Pixels recently updated in shades of red and pixels not recently updated in shades of cyan. (1): before rotation, (2) (3): during rotation, (4): after rotation. The effect of the rotation is to update most pixels located in range of the distance sensor whose field of view is represented with a green cone. For instance, pixels in zone A got updated during the rotation. Pixels in zone B were occluded from the point of view of the robot and did not get updated.

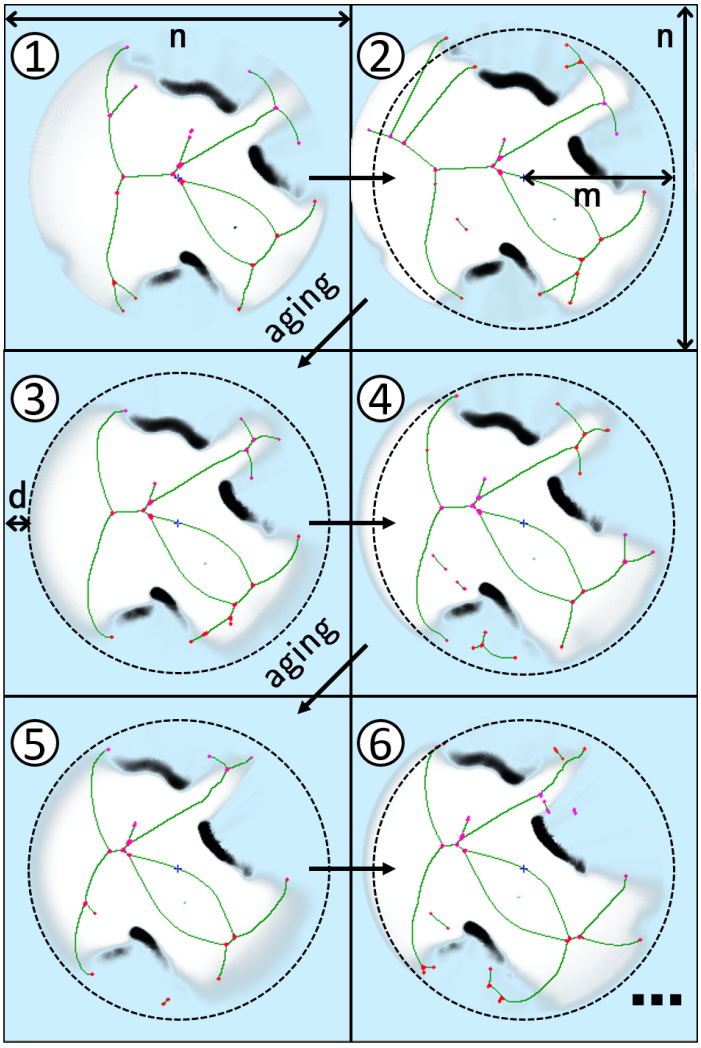



**Figure 6:** Occupancy grids acquired during traversal of an environment. From its starting position (1), the robot moved a distance $d$ (2), where an aging process occurs (3), which notably resets the pixels further than $m$ away from the robot (dashed circle). The aging process also resets pixels not updated for a long period. The process is repeated again later at a distance $d$ (4-5). (6) The aging process continues as long as the navigation algorithm is running.

# 4 Large-Scale Angular Drift Compensation for chapter 5, section 5.3

A robot can have two types of angular measurements: absolute measurements such as a compass or tracking stars in the sky, further abridged $\theta_{abs}$, and relative measurements such as odometric integration or rotation estimation from point tracking, further abridged $\theta_{rel}$. While relative angular measurements drift in time, they are usually quite accurate and smooth on a short time- and space scale. On the contrary, compass measurements are not reliable on a small scale due to magnetic field-emitting objects and stars may be occluded. However, absolute measurements do not suffer from drift over time.

The purpose of Large-scale Angular Drift Compensation (LSADC) is to correct the heading (only the yaw axis is considered in this thesis) obtained by relative measurements using that obtained by absolute measurements. The correction is performed over a long time period in order to account for absolute measurements not being reliable locally. There are multiple ways to implement this correction and algorithm 21 only describes one of them. Figure 6.17 page 169 shows the amount of heading drift that can be compensated by LSADC during a real-life indoor experiment.

We found experimentally that integration of a compass over a distance of 50 meters was a good compromise between robustness to local magnetic fields and delay due to integration. nsteps= 100 intermediary integration steps are used between each averaged sample. In algorithm 21, static variables retain their value between executions of the algorithm. We use a median instead of a mean since the median eliminates small-scale asymmetric perturbations, which are typically produced by magnetic field emitters in the environment.

**Algorithm 21** Large-scale Angular Drift Compensation

**Input:** $\theta_{rel} \in ]-\pi;\pi], \theta_{abs} \in ]-\pi;\pi], \vec{r}_{odom}$, distance, nsteps
**static variables** $old\delta_\theta$, initialization_counter = $\infty$, slot_counter, circular_buffer[nsteps]
**static variables** norm_accumulator, buffer_pos, angle_accumulator
vnorm = $||\vec{r}_{odom}||$
**if** initialization_counter = $\infty$ **then**//first time executing the function?
| $old\delta_\theta \leftarrow \theta_{abs} - \theta_{rel}$
| **if** $old\delta_\theta > \pi$ **then**
| | $old\delta_\theta \leftarrow old\delta_\theta - 2\pi$
| **else if** $old\delta_\theta \leq -\pi$ **then**
| | $old\delta_\theta \leftarrow old\delta_\theta + 2\pi$
| med $\leftarrow old\delta_\theta$
| norm_accumulator $\leftarrow 0$, buffer_pos $\leftarrow 0$, angle_accumulator $\leftarrow 0$
| initialization_counter $\leftarrow 0$, slot_counter $\leftarrow 0$
$\delta_\theta \leftarrow \theta_{abs} - \theta_{rel}$//difference between absolute and relative angle measurements
$\delta_\theta \leftarrow \delta_\theta + 2\pi.floor(\frac{old\delta_\theta - \delta_\theta}{2\pi})$//bring old and new $\delta_\theta$ closer ($+k.2\pi$, $k \in \mathbb{Z}$)
**while** $\delta_\theta - old\delta_\theta > \pi$ **do**//correct a $+2\pi$ offset if present
| $\delta_\theta \leftarrow \delta_\theta - 2\pi$
**while** $\delta_\theta - old\delta_\theta \leq -\pi$ **do**//correct a $-2\pi$ offset if present
| $\delta_\theta \leftarrow \delta_\theta + 2\pi$
$old\delta_\theta \leftarrow \delta_\theta$
angle_accumulator $\leftarrow$ angle_accumulator + vnorm.$\delta_\theta$//accumulate weighted $\delta_\theta$
norm_accumulator $\leftarrow$ norm_accumulator + vnorm//accumulate weights
**if** angle_accumulator $\geq$ distance/nsteps **then**
| //new averaged $\delta_\theta$ sample should be collected in circular_buffer
| **if** initialization_counter $<$ nsteps **then**
| | initialization_counter $\leftarrow$ initialization_counter +1
| circular_buffer[buffer_pos] $\leftarrow$ angle_accumulator / norm_accumulator
| **if** buffer_pos = nsteps$-1$ **then**//wrap-around array index
| | buffer_pos $\leftarrow 0$
| **else**
| | buffer_pos $\leftarrow$ buffer_pos+1
| med $\leftarrow$ median(circular_buffer[0..initialization_counter-1])//integrated $\delta_\theta$
| angle_accumulator $\leftarrow 0$
| norm_accumulator $\leftarrow 0$
$\alpha \leftarrow atan2(\vec{r}^{\,y}_{odom}, \vec{r}^{\,x}_{odom})$//find direction of odometric vector
$\vec{r}_{odom} \leftarrow vnorm.(\cos(\alpha+\text{med}), \sin(\alpha+\text{med}))$//rotate odometric vector

# Bibliography


Aine, Sandip, Siddharth Swaminathan, Venkatraman Narayanan, Victor Hwang, and Maxim Likhachev (2016). "Multi-Heuristic A*". In: *The International Journal of Robotics Research*. DOI: 10.1177/0278364915594029.

Albers, Susanne and Monika Rauch Henzinger (1997). "Exploring Unknown Environments". In: *SIAM Journal on Computing*, pp. 416–425. DOI: 10.1.1.36.42. URL: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.36.42.

Alyan, Sofyan and Rudolf Jander (1994). "Short-range homing in the house mouse, Mus musculus: stages in the learning of directions". In: *Animal Behaviour* 48.2, pp. 285–298. ISSN: 0003-3472. DOI: 10.1006/anbe.1994.1242. URL: http://www.sciencedirect.com/science/article/pii/S0003347284712425.

Arcelli, C. and G. Sanniti di Baja (1989). "A one-pass two-operation process to detect the skeletal pixels on the 4-distance transform". In: *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 11.4, pp. 411–414. ISSN: 0162-8828. DOI: 10.1109/34.19037.

Argamon-Engelson, Shlomo, Sarit Kraus, and Sigalit Sina (1998). "Utility-based on-line exploration for repeated navigation in an embedded graph". In: *Artificial Intelligence* 101.1–2, pp. 267–284. ISSN: 0004-3702. DOI: 10.1016/S0004-3702(98)00014-9. URL: http://www.sciencedirect.com/science/article/pii/S0004370298000149%20http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.49.4834.

Attali, Dominique (1995). "2D and 3D skeletons and Voronoi graphs". French. Thesis. Université Joseph-Fourier - Grenoble I. URL: https://tel.archives-ouvertes.fr/tel-00346066.

Aulinas, Josep, Yvan Petillot, Joaquim Salvi, and Xavier Lladó (2008). "The SLAM Problem: A Survey". In: *Proceedings of the 2008 Conference on Artificial Intelligence Research and Development: Proceedings of the 11th International Conference of the Catalan Association for Artificial Intelligence*. Amsterdam, The Netherlands, The Netherlands: IOS Press, pp. 363–371. ISBN: 978-1-58603-925-7. URL: http://dl.acm.org/citation.cfm?id=1566899.1566949.

Awerbuch, Baruch, Margrit Betke, Ronald L. Rivest, and Mona Singh (1999). "Piecemeal Graph Exploration by a Mobile Robot". In: *Information and Computation* 152.2, pp. 155–172. ISSN: 0890-5401. DOI: 10.1006/inco.1999.2795. URL: http://www.sciencedirect.com/science/article/pii/S0890540199927955.

Bailey, Tim (2002). "Mobile Robot Localisation and Mapping in Extensive Outdoor Environments". PhD thesis. Australian Centre for Field Robotics, University of Sydney.

DOI: 10.1.1.4.1707. URL: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.4.1707.

Bailey, Tim and H. Durrant-Whyte (2006). "Simultaneous localization and mapping (SLAM): part II". In: *Robotics Automation Magazine, IEEE* 13.3, pp. 108–117. ISSN: 1070-9932. DOI: 10.1109/MRA.2006.1678144.

Bar-Noy, Amotz and Baruch Schieber (1991). "The Canadian Traveller Problem". In: *Proceedings of the Second Annual ACM-SIAM Symposium on Discrete Algorithms*. SODA '91. San Francisco, California, USA: Society for Industrial and Applied Mathematics, pp. 261–270. ISBN: 0-89791-376-0. URL: http://dl.acm.org/citation.cfm?id=127787.127835.

Barraquand, J., B. Langlois, and J.-C. Latombe (1992). "Numerical potential field techniques for robot path planning". In: *Systems, Man and Cybernetics, IEEE Transactions on* 22.2, pp. 224–241. ISSN: 0018-9472. DOI: 10.1109/21.148426.

Bartol Jr, Thomas M, Cailey Bromer, Justin Kinney, Michael A Chirillo, Jennifer N Bourne, Kristen M Harris, and Terrence J Sejnowski (2015). "Nanoconnectomic upper bound on the variability of synaptic plasticity". In: *eLife* 4. Ed. by Sacha B Nelson, p. 18. ISSN: 2050-084X. DOI: 10.7554/eLife.10778.

Beeson, Patrick, Nicholas K. Jong, and Benjamin Kuipers (2005). "Towards autonomous topological place detection using the Extended Voronoi Graph". In: *Proceedings of the IEEE International Conference on Robotics and Automaton (ICRA-05)*.

Beeson, Patrick, Joseph Modayil, and Benjamin Kuipers (2010). "Factoring the Mapping Problem: Mobile Robot Map-building in the Hybrid Spatial Semantic Hierarchy". In: *The International Journal of Robotics Research* 29.4, pp. 428–459. DOI: 10.1177/0278364909100586. eprint: http://ijr.sagepub.com/content/29/4/428.full.pdf+html. URL: http://ijr.sagepub.com/content/29/4/428.abstract.

Beranger, V. and J.-Y. Herve (1996). "Recognition of intersections in corridor environments". In: *Pattern Recognition, 1996., Proceedings of the 13th International Conference on*. Vol. 4, 133–137 vol.4. DOI: 10.1109/ICPR.1996.547248.

Betke, Margrit (1991). "Algorithms for exploring an unknown graph". MA thesis. MIT, Laboratory for Computer Science. URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.18.6636&rep=rep1&type=pdf.

Borenstein, J. and Liqiang Feng (1996). "Measurement and correction of systematic odometry errors in mobile robots". In: *Robotics and Automation, IEEE Transactions on* 12.6, pp. 869–880. ISSN: 1042-296X. DOI: 10.1109/70.544770.

Borenstein, J. and Y. Koren (1989). "Real-time obstacle avoidance for fast mobile robots". In: *Systems, Man and Cybernetics, IEEE Transactions on* 19.5, pp. 1179–1187. ISSN: 0018-9472. DOI: 10.1109/21.44033.

Bose, Prosenjit, Pat Morin, Ivan Stojmenović, and Jorge Urrutia (2001). "Routing with Guaranteed Delivery in Ad Hoc Wireless Networks". In: *Wireless Networks* 7.6, pp. 609–616. ISSN: 1572-8196. DOI: 10.1023/A:1012319418150.

Bosse, Michael, Paul Newman, John Leonard, and Seth Teller (2004). "Simultaneous Localization and Map Building in large-scale cyclic environments using the Atlas framework". In: *International Journal of Robotics Research* 23.12, pp. 1113–1139. DOI: 10.1177/0278364904049393.

Breu, H., J. Gil, D. Kirkpatrick, and M. Werman (1995). "Linear time Euclidean distance transform algorithms". In: *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 17.5, pp. 529–533. ISSN: 0162-8828. DOI: 10.1109/34.391389.

Byrne, R. W. (1979). "Memory for urban geography". In: *Quarterly Journal of Experimental Psychology* 31, pp. 147–154. DOI: 10.1080/14640747908400714.

Carlevaris-Bianco, N., M. Kaess, and R.M. Eustice (2014). "Generic Node Removal for Factor-Graph SLAM". In: *Robotics, IEEE Transactions on* 30.6, pp. 1371–1385. ISSN: 1552-3098. DOI: 10.1109/TRO.2014.2347571.

Carrillo, H., P. Dames, V. Kumar, and J.A. Castellanos (2015). "Autonomous robotic exploration using occupancy grid maps and graph SLAM based on Shannon and Rényi Entropy". In: *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, pp. 487–494. DOI: 10.1109/ICRA.2015.7139224.

Chatila, R. and J. Laumond (1985). "Position referencing and consistent world modeling for mobile robots". In: *Robotics and Automation. Proceedings. 1985 IEEE International Conference on*. Vol. 2, pp. 138–145. DOI: 10.1109/ROBOT.1985.1087373.

Choset, H., I. Konukseven, and A. Rizzi (1997). "Sensor based planning: a control law for generating the generalized Voronoi graph". In: *Advanced Robotics, 1997. ICAR '97. Proceedings., 8th International Conference on*, pp. 333–338. DOI: 10.1109/ICAR.1997.620203.

Choset, H. and Keiji Nagatani (2001). "Topological Simultaneous Localization And Mapping (SLAM): toward exact localization without explicit localization". In: *Robotics and Automation, IEEE Transactions on* 17.2, pp. 125–137. ISSN: 1042-296X. DOI: 10.1109/70.928558.

Collett, T. S. and J. A. Rees (1997). "View-based navigation in Hymenoptera: multiple strategies of landmark guidance in the approach to a feeder". English. In: *Journal of Comparative Physiology A* 181.1, pp. 47–58. ISSN: 0340-7594. DOI: 10.1007/s003590050092.

Couprie, Michel, David Coeurjolly, and Rita Zrour (2007). "Discrete bisector function and Euclidean skeleton in 2D and 3D". In: *Image and Vision Computing* 25.10. Discrete Geometry for Computer Imagery 2005, pp. 1543–1556. ISSN: 0262-8856. DOI: 10.1016/j.imavis.2006.06.020. URL: http://www.sciencedirect.com/science/article/pii/S0262885606003064.

Crowley, J.L. (1985). "Navigation for an intelligent mobile robot". In: *Robotics and Automation, IEEE Journal of* 1.1, pp. 31–41. ISSN: 0882-4967. DOI: 10.1109/JRA.1985.1087002.

Cummins, M. and P. Newman (2008). "Accelerated appearance-only SLAM". In: *2008 IEEE International Conference on Robotics and Automation*, pp. 1828–1833. DOI: 10.1109/ROBOT.2008.4543473.

Cummins, Mark and Paul Newman (2010). "Appearance-only SLAM at large scale with FAB-MAP 2.0". In: *The International Journal of Robotics Research*. DOI: 10.1177/0278364910385483. eprint: http://ijr.sagepub.com/content/early/2010/11/11/0278364910385483.full.pdf+html. URL: http://ijr.sagepub.com/content/early/2010/11/11/0278364910385483.abstract.

Danielsson, Per-Erik (1980). "Euclidean Distance Mapping". In: *Computer Graphics and Image Processing* 14, pp. 227–248. URL: http://webstaff.itn.liu.se/~stegu/JFA/Danielsson.pdf.

Dechter, Rina and Judea Pearl (1985). "Generalized Best-first Search Strategies and the Optimality of A*". In: *J. ACM* 32.3, pp. 505–536. ISSN: 0004-5411. DOI: 10.1145/3828.3830. URL: http://doi.acm.org/10.1145/3828.3830.

Dessmark, Anders and Andrzej Pelc (2004). "Optimal graph exploration without good maps". In: *Theoretical Computer Science* 326.1–3, pp. 343–362. ISSN: 0304-3975. DOI: 10.1016/j.tcs.2004.07.031. URL: http://www.sciencedirect.com/science/article/pii/S0304397504004773.

Dissanayake, M. W M G, P. Newman, S. Clark, H.F. Durrant-Whyte, and M. Csorba (2001). "A solution to the simultaneous localization and map building (SLAM) problem". In: *Robotics and Automation, IEEE Transactions on* 17.3, pp. 229–241. ISSN: 1042-296X. DOI: 10.1109/70.938381.

Doeller, Christian F, Caswell Barry, and Neil Burgess (2010). "Evidence for grid cells in a human memory network". In: *Nature* 463, pp. 657–661. DOI: 10.1038/nature08704.

Drumheller, Michael (1987). "Mobile robot localization using sonar". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 9, pp. 325–332. URL: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.18.4482.

Duckett, T., S. Marsland, and J. Shapiro (2000). "Learning globally consistent maps by relaxation". In: *Robotics and Automation, 2000. Proceedings. ICRA '00. IEEE International Conference on*. Vol. 4, 3841–3846 vol.4. DOI: 10.1109/ROBOT.2000.845330.

Duckett, Tom, Stephen Marsland, and Jonathan Shapiro (2002). "Fast, On-Line Learning of Globally Consistent Maps". English. In: *Autonomous Robots* 12.3, pp. 287–300. ISSN: 0929-5593. DOI: 10.1023/A:1015269615729.

Dudek, G., M. Jenkin, E. Milios, and D. Wilkes (1991). "Robotic exploration as graph construction". In: *Robotics and Automation, IEEE Transactions on* 7.6, pp. 859–865. ISSN: 1042-296X. DOI: 10.1109/70.105395.

Dudek, Gregory, Paul Freedman, and Souad Hadjres (1993). "Using Local Information in a Non-Local Way for Mapping Graph-Like Worlds". In: *Proceedings of the 13th International Joint Conference on Artificial Intelligence (IJCAI-93)*, pp. 1639–1645. URL: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.47.9824.

Durrant-Whyte, H. and Tim Bailey (2006). "Simultaneous localization and mapping: part I". In: *Robotics Automation Magazine, IEEE* 13.2, pp. 99–110. ISSN: 1070-9932. DOI: 10.1109/MRA.2006.1638022.

Durrant-Whyte, Hugh, David Rye, and Eduardo Nebot (1996). "Localization of Autonomous Guided Vehicles". English. In: *Robotics Research*. Ed. by Georges Giralt and Gerhard Hirzinger. Springer London, pp. 613–625. ISBN: 978-1-4471-1254-9. DOI: 10.1007/978-1-4471-0765-1_69.

Eckner, James T., Jeffrey S. Kutcher, and James K. Richardson (2010). "Pilot Evaluation of a Novel Clinical Test of Reaction Time in National Collegiate Athletic Association Division I Football Players". In: *Journal of Athletic Training* 45.4, pp. 327–332. DOI: 10.4085/1062-6050-45.4.327.

Ekstrom, Arne D., Michael J. Kahana, Jeremy B. Caplan, Tony A. Fields, Eve A. Isham, Ehren L. Newman, and Itzhak Fried (2003). "Cellular networks underlying human spatial navigation". In: *Nature* 425, pp. 184–188. DOI: 10.1038/nature01964.

Elfes, A. (1987). "Sonar-based real-world mapping and navigation". In: *Robotics and Automation, IEEE Journal of* 3.3, pp. 249–265. ISSN: 0882-4967. DOI: 10.1109/JRA.1987.1087096.

– (1989). "Using occupancy grids for mobile robot perception and navigation". In: *Computer* 22.6, pp. 46–57. ISSN: 0018-9162. DOI: 10.1109/2.30720.

Epstein, Russell and Nancy Kanwisher (1998). "A cortical representation of the local visual environment". In: *Nature* 392, pp. 598–601. DOI: 10.1038/33402. URL: http://dx.doi.org/10.1038/33402.

Estrada, C., J. Neira, and J.D. Tardos (2005). "Hierarchical SLAM: Real-Time Accurate Mapping of Large Environments". In: *Robotics, IEEE Transactions on* 21.4, pp. 588–596. ISSN: 1552-3098. DOI: 10.1109/TRO.2005.844673.

Etienne, Ariane S. and Kathryn J. Jeffery (2004). "Path Integration in Mammals". In: *Hippocampus* 14, pp. 180–192. URL: http://www.ucl.ac.uk/jefferylab/publications/2004_Etienne_and_Jeffery_Hippocampus.pdf.

Etienne, A.S., R. Maurer, and V. Séquinot (1996). "Path integration in mammals and its interaction with visual landmarks". In: *Journal of Experimental Biology* 199, pp. 201–209. URL: http://www.ncbi.nlm.nih.gov/pubmed/8576691.

Fabbri, Ricardo, Luciano Da F. Costa, Julio C. Torelli, and Odemir M. Bruno (2008). "2D Euclidean Distance Transform Algorithms: A Comparative Survey". In: *ACM Comput. Surv.* 40.1, 2:1–2:44. ISSN: 0360-0300. DOI: 10.1145/1322432.1322434. URL: http://doi.acm.org/10.1145/1322432.1322434.

Felner, A., R. Stern, A. Ben-Yair, S. Kraus, and N. Netanyahu (2004). "PHA*: Finding the Shortest Path with A* in An Unknown Physical Environment". In: *Journal of Artificial Intelligence Research* 21, pp. 631–670. DOI: 10.1613/jair.1373. URL: http://jair.org/papers/paper1373.html.

Ferguson, Dave, Maxim Likhachev, and Anthony Stentz (2005). "A Guide to Heuristic Based Path Planning". In: *Proceedings of the Workshop on Planning under Uncertainty for Autonomous Systems at The International Conference on Automated Planning and Scheduling (ICAPS)*. DOI: 10.1.1.68.2544. URL: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.68.2544.

Filliat, David (2001). "Cartographie et estimation globale de la position pour un robot mobile autonome". PhD thesis. Université Paris 6. URL: http://tel.archives-ouvertes.fr/tel-00655469.

Fleischer, Rudolf (2005). "Exploring an unknown graph efficiently". In: *Proc. 13th Annu. European Sympos. Algorithms*. Springer-Verlag, pp. 11–22. URL: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.120.7651.

Förster, Klaus-Tycho and Roger Wattenhofer (2012). "Directed Graph Exploration". English. In: *Principles of Distributed Systems*. Ed. by Roberto Baldoni, Paola Flocchini, and Ravindran Binoy. Vol. 7702. Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 151–165. ISBN: 978-3-642-35475-5. DOI: 10.1007/978-3-642-35476-2_11. URL: http://dx.doi.org/10.1007/978-3-642-35476-2_11.

Garrido, S., L. Moreno, M. Abderrahim, and F. Martin (2006). "Path Planning for Mobile Robot Navigation using Voronoi Diagram and Fast Marching". In: *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on*, pp. 2376–2381. DOI: 10.1109/IROS.2006.282649.

Gerkey, Brian P., Richard T. Vaughan, and Andrew Howard (2003). "The Player/Stage Project: Tools for Multi-Robot and Distributed Sensor Systems". In: *Proceedings of the 11th International Conference on Advanced Robotics*. Coimbra, Portugal, pp. 317–323. URL: http://playerstage.sourceforge.net/.

Ghaffari, M., B. Hariri, and S. Shirmohammadi (2010). "On the necessity of using Delaunay Triangulation substrate in greedy routing based networks". In: *Communications Letters, IEEE* 14.3, pp. 266–268. ISSN: 1089-7798. DOI: 10.1109/LCOMM.2010.03.092045.

Gibson, James J. (1950). *The Perception of the Visual World*. Ed. by Leonard Carmichael. The Riverside PRess, Cambridge.

Golfarelli, M., D. Maio, and S. Rizzi (1998). "Elastic correction of dead-reckoning errors in map building". In: *Intelligent Robots and Systems, 1998. Proceedings., 1998 IEEE/RSJ International Conference on*. Vol. 2, 905–911 vol.2. DOI: 10.1109/IROS.1998.727315.

Grevera, GeorgeJ. (2007). "Distance Transform Algorithms And Their Implementation And Evaluation". English. In: *Deformable Models*. Topics in Biomedical Engineering. International Book Series. Springer New York, pp. 33–60. ISBN: 978-0-387-31201-9. DOI: 10.1007/978-0-387-68413-0_2. URL: http://dx.doi.org/10.1007/978-0-387-68413-0_2.

Grisetti, G., R. Kümmerle, C. Stachniss, and W. Burgard (2010). "A Tutorial on Graph-Based SLAM". In: *Intelligent Transportation Systems Magazine, IEEE* 2.4, pp. 31–43. ISSN: 1939-1390. DOI: 10.1109/MITS.2010.939925.

Gutmann, J.-S. and K. Konolige (1999). "Incremental mapping of large cyclic environments". In: *Computational Intelligence in Robotics and Automation, 1999. CIRA '99. Proceedings. 1999 IEEE International Symposium on*, pp. 318–325. DOI: 10.1109/CIRA.1999.810068.

Hart, P.E., N.J. Nilsson, and B. Raphael (1968). "A Formal Basis for the Heuristic Determination of Minimum Cost Paths". In: *Systems Science and Cybernetics, IEEE Transactions on* 4.2, pp. 100–107. ISSN: 0536-1567. DOI: 10.1109/TSSC.1968.300136.

Havangi, Ramazan, Mohammad Teshnehlab, and Mohammad Ali Nekoui (2010). "A Neuro-Fuzzy Multi Swarm FastSLAM Framework". In: *Journal of Computing* 2.3, pp. 83–94. URL: http://arxiv.org/abs/1003.4075.

Hesselink, Wim H. (2007). "A linear-time algorithm for Euclidean feature transform sets". In: *Information Processing Letters* 102.5, pp. 181–186. ISSN: 0020-0190. DOI: 10.1016/j.ipl.2006.12.005. URL: http://www.sciencedirect.com/science/article/pii/S0020019006003681.

Hirata, Tomio (1996). "A Unified Linear-time Algorithm for Computing Distance Maps". In: *Inf. Process. Lett.* 58.3, pp. 129–133. ISSN: 0020-0190. DOI: 10.1016/0020-0190(96)00049-X. URL: http://dx.doi.org/10.1016/0020-0190(96)00049-X.

Hoare, C. A. R. (1961). "Algorithm 65: Find". In: *Commun. ACM* 4.7, pp. 321–322. ISSN: 0001-0782. DOI: 10.1145/366622.366647.

Ieropoulos, Ioannis, John Greenman, Chris Melhuish, and Ian Horsfield (2010). "Ecobot III: a robot with guts". In: *Proceedings of the Alife XII Conference*. Odense, Denmark. URL: https://mitpress.mit.edu/sites/default/files/titles/alife/0262290758chap131.pdf.

Karger, David and Evdokia Nikolova (2007). *Exact Algorithms for the Canadian Traveller Problem on Paths and Trees*. Technical report. www.csail.mit.edu: MIT, Artificial Intelligence Laboratory. URL: http://dspace.mit.edu/bitstream/handle/1721.1/40093/MIT?sequence=1.

Karp, Brad and H. T. Kung (2000). "GPSR: Greedy Perimeter Stateless Routing for Wireless Networks". In: *Proceedings of the 6th Annual International Conference on Mobile Computing and Networking*. MobiCom '00. Boston, Massachusetts, USA: ACM, pp. 243–254. ISBN: 1-58113-197-6. DOI: 10.1145/345910.345953.

Khoshelham, Kourosh and Sander Oude Elberink (2012). "Accuracy and Resolution of Kinect Depth Data for Indoor Mapping Applications". In: *Sensors* 12.2, p. 1437. ISSN: 1424-8220. DOI: 10.3390/s120201437. URL: http://www.mdpi.com/1424-8220/12/2/1437.

Kimchi, Tali, Ariane S. Etienne, and Joseph Terkel (2004). "A subterranean mammal uses the magnetic compass for path integration". In: *Proceedings of the National Academy of Sciences of the United States of America (PNAS)* 101.4, pp. 1105–1109. DOI: 10.1073/pnas.0307560100.

Klette, Gisela (2003). "Computer Analysis of Images and Patterns: 10th International Conference, CAIP 2003, Groningen, The Netherlands, August 25-27, 2003. Proceedings". In: ed. by Nicolai Petkov and Michel A. Westenberg. Berlin, Heidelberg: Springer Berlin Heidelberg. Chap. Simple Points in 2D and 3D Binary Images, pp. 57–64. ISBN: 978-3-540-45179-2. DOI: 10.1007/978-3-540-45179-2_8. URL: http://dx.doi.org/10.1007/978-3-540-45179-2_8.

Koenig, S. and M. Likhachev (2005). "Fast replanning for navigation in unknown terrain". In: *Robotics, IEEE Transactions on* 21.3, pp. 354–363. ISSN: 1552-3098. DOI: 10.1109/TRO.2004.838026.

Koenig, Sven, Maxim Likhachev, and David Furcy (2004). "Lifelong Planning A*". In: *Artificial Intelligence* 155.1–2, pp. 93–146. ISSN: 0004-3702. DOI: 10.1016/j.artint.2003.12.001. URL: http://www.sciencedirect.com/science/article/pii/S000437020300225X.

Koenig, Sven, Apurva Mudgal, and Craig Tovey (2006). "A Near-tight Approximation Lower Bound and Algorithm for the Kidnapped Robot Problem". In: *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithm*. SODA '06. Miami, Florida: Society for Industrial and Applied Mathematics, pp. 133–142. ISBN: 0-89871-605-5. URL: http://dl.acm.org/citation.cfm?id=1109557.1109574.

Koenig, Sven and Yury Smirnov (1996). "Graph Learning with a Nearest Neighbor Approach". In: *Proceedings of the Ninth Annual Conference on Computational Learning Theory*. COLT '96. Desenzano del Garda, Italy: ACM, pp. 19–28. ISBN: 0-89791-811-8. DOI: 10.1145/238061.238065. URL: http://doi.acm.org/10.1145/238061.238065.

Koenig, Sven, Craig Tovey, and Yuri Smirnov (2003). "Performance bounds for planning in unknown terrain". In: *Artificial Intelligence* 147.1–2. Planning with Uncertainty and Incomplete Information, pp. 253–279. ISSN: 0004-3702. DOI: 10.1016/S0004-3702(03)00062-6. URL: http://www.sciencedirect.com/science/article/pii/S0004370203000626.

Konolige, K., E. Marder-Eppstein, and B. Marthi (2011). "Navigation in hybrid metric-topological maps". In: *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pp. 3041–3047. DOI: 10.1109/ICRA.2011.5980074.

Korrapati, Hemanth and Youcef Mezouar (2014). "Vision-based sparse topological mapping". In: *Robotics and Autonomous Systems* 62.9, pp. 1259–1270. ISSN: 0921-8890. DOI: 10.1016/j.robot.2014.03.015. URL: http://www.sciencedirect.com/science/article/pii/S0921889014000591.

Kuipers, B., J. Modayil, P. Beeson, M. MacMahon, and F. Savelli (2004). "Local metrical and global topological maps in the hybrid spatial semantic hierarchy". In: *Robotics and Automation, 2004. Proceedings. ICRA '04. 2004 IEEE International Conference on*. Vol. 5, 4845–4851 Vol.5. DOI: 10.1109/ROBOT.2004.1302485.

Kuipers, Benjamin (2000). "The Spatial Semantic Hierarchy". In: *Artificial Intelligence* 119.1–2, pp. 191–233. ISSN: 0004-3702. DOI: 10.1016/S0004-3702(00)00017-5. URL: http://www.sciencedirect.com/science/article/pii/S0004370200000175.

Kuipers, Benjamin and Yung-Tai Byun (1991). "A robot exploration and mapping strategy based on a semantic hierarchy of spatial representations". In: *Robotics and Autonomous Systems* 8.1-2. Special Issue Toward Learning Robots, pp. 47–63. ISSN: 0921-8890. DOI: 10.1016/0921-8890(91)90014-C. URL: http://www.sciencedirect.com/science/article/pii/092188909190014C.

Lam, S.S. and Chen Qian (2013). "Geographic Routing in d -Dimensional Spaces With Guaranteed Delivery and Low Stretch". In: *Networking, IEEE/ACM Transactions on* 21.2, pp. 663–677. ISSN: 1063-6692. DOI: 10.1109/TNET.2012.2214056.

Lee, Seongsoo, Sukhan Lee, and Seungmin Baek (2011). "Vision-Based Kidnap Recovery with SLAM for Home Cleaning Robots". In: *Journal of Intelligent & Robotic Systems* 67.1, pp. 7–24. ISSN: 1573-0409. DOI: 10.1007/s10846-011-9647-4.

Leonard, J.J. and H.F. Durrant-Whyte (1991). "Simultaneous map building and localization for an autonomous mobile robot". In: *Intelligent Robots and Systems '91. 'Intelligence for Mechanical Systems, Proceedings IROS '91. IEEE/RSJ International Workshop on*, 1442–1447 vol.3. DOI: 10.1109/IROS.1991.174711.

Li, Hong and Albert M. Vossepoel (1998). "Generation of the Euclidean skeleton from the vector distance map by a bisector decision rule". In: *Computer Vision and Pattern Recognition, 1998. Proceedings. 1998 IEEE Computer Society Conference on*, pp. 66–71. DOI: 10.1109/CVPR.1998.698589.

Liang, Zhao, Huang Shoudong, and G. Dissanayake (2013). "Linear SLAM: A linear solution to the feature-based and pose graph SLAM based on submap joining". In: *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*, pp. 24–30. DOI: 10.1109/IROS.2013.6696327.

Lim, H., J. Lim, and H. J. Kim (2014). "Real-time 6-DOF monocular visual SLAM in a large-scale environment". In: *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, pp. 1532–1539. DOI: 10.1109/ICRA.2014.6907055.

Loui, Ronald Prescott (1983). "Optimal Paths in Graphs with Stochastic or Multidimensional Weights". In: *Commun. ACM* 26.9, pp. 670–676. ISSN: 0001-0782. DOI: 10.1145/358172.358406. URL: http://doi.acm.org/10.1145/358172.358406.

Lowry, S., N. Sunderhauf, P. Newman, J.J. Leonard, D. Cox, P. Corke, and M.J. Milford (2016). "Visual Place Recognition: A Survey". In: *Robotics, IEEE Transactions on* 32.1, pp. 1–19. ISSN: 1552-3098. DOI: 10.1109/TRO.2015.2496823.

Lu, F. and E. Milios (1997). "Globally Consistent Range Scan Alignment for Environment Mapping". English. In: *Autonomous Robots* 4.4, pp. 333–349. ISSN: 0929-5593. DOI: 10.1023/A:1008854305733.

Lumelsky, Vladimir J. and Alexander A. Stepanov (1987). "Path-planning strategies for a point mobile automaton moving amidst unknown obstacles of arbitrary shape". In: *ALGORITHMICA* 2, pp. 403–430. URL: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.156.7780.

Malandain, Grégoire and Sara Fernández-Vidal (1998). "Euclidean skeletons". In: *Image and Vision Computing* 16.5, pp. 317–327. ISSN: 0262-8856. DOI: 10.1016/S0262-8856(97)00074-7. URL: http://www.sciencedirect.com/science/article/pii/S0262885697000747.

Man, D., K. Uda, H. Ueyama, Y. Ito, and K. Nakano (2010). "Implementations of Parallel Computation of Euclidean Distance Map in Multicore Processors and GPUs". In: *Networking and Computing (ICNC), 2010 First International Conference on*, pp. 120–127. DOI: 10.1109/IC-NC.2010.55.

Mayran de Chamisso, Fabrice, Laurent Soulier, and Michael Aupetit (2016). "Robust topological skeleton extraction from occupancy grids for mobile robot navigation". In: *Proceedings of the twentieth national congress on Shape Recognition and Artificial Intelligence (RFIA'16)*.

Mayran de Chamisso, Fabrice, Laurent Soulier, and Michaël Aupetit (2015). "Exploratory Digraph Navigation using A*". In: *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence*. IJCAI. AAAI Press / International Joint Conferences on Artificial Intelligence. URL: http://ijcai.org/proceedings/2015.

Mazuran, Mladen, Wolfram Burgard, and Gian Diego Tipaldi (2016). "Nonlinear factor recovery for long-term SLAM". In: *The International Journal of Robotics Research*. DOI: 10.1177/0278364915581629.

McCormack, J. (2010). "Enhancing Creativity with Niche Construction". In: *Proceedings of the Twelfth International Conference on the Synthesis and Simulation of Living Systems (Artificial LifeXII)*. Editors: Harold Fellermann, Mark Dörr, Martin M. Hanczyc, Lone Ladegaard Laursen, Sarah Maurer, Daniel Merkle, Pierre-Alain Monnard, Kasper Stoy and Steen Rasmussen. MIT Press, pp. 525–532. URL: http://jonmccormack.info/~jonmc/sa/artworks/niche-constructions/%20http://mitpress.mit.edu/books/artificial-life-xii.

McCormack, Jon and Oliver Bown (2009). "Applications of Evolutionary Computing: EvoWorkshops 2009: EvoCOMNET, EvoENVIRONMENT, EvoFIN, EvoGAMES, EvoHOT, EvoIASP, EvoINTERACTION, EvoMUSART, EvoNUM, EvoSTOC, EvoTRANSLOG, Tübingen, Germany, April 15-17, 2009. Proceedings". In: ed. by Mario Giacobini, Anthony Brabazon, Stefano Cagnoni, Gianni A. Caro, Anikó Ekárt, Anna Isabel Esparcia-Alcázar, Muddassar Farooq, Andreas Fink, and Penousal Machado. Berlin, Heidelberg: Springer Berlin Heidelberg. Chap. Life's What You Make: Niche Construction and Evolutionary Art, pp. 528–537. ISBN: 978-3-642-01129-0. DOI: 10.1007/978-3-642-01129-0_59. URL: http://jonmccormack.info/~jonmc/sa/artworks/niche-constructions/.

McNamara, Timothy P., James K. Hardy, and Stephen C. Hirtle (1989). "Subjective hierarchies in spatial memory." In: *Journal of Experimental Psychology: Learning, Memory, and Cognition* 15(2).0278-7393 (Linking), pp. 211–227. URL: http://www.ncbi.nlm.nih.gov/pubmed/2522511.

McNaughton, Bruce L., Francesco P. Battaglia, Ole Jensen, Edvard I Moser, and May-Britt Moser (2006). "Path integration and the neural basis of the 'cognitive map'". In: *Nat Rev Neurosci* 7.8, pp. 663–678. ISSN: 1471-003X. DOI: 10.1038/nrn1932.

Megow, Nicole, Kurt Mehlhorn, and Pascal Schweitzer (2011). "Online Graph Exploration: New Results on Old and New Algorithms". English. In: *Automata, Languages and Programming*. Ed. by Luca Aceto, Monika Henzinger, and Jiří Sgall. Vol. 6756. Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 478–489. ISBN: 978-3-642-22011-1. DOI: 10.1007/978-3-642-22012-8_38.

Milford, M. J. and G. F. Wyeth (2012). "SeqSLAM: Visual route-based navigation for sunny summer days and stormy winter nights". In: *2012 IEEE International Conference on Robotics and Automation*, pp. 1643–1649. DOI: 10.1109/ICRA.2012.6224623.

Modayil, J., P. Beeson, and B. Kuipers (2004). "Using the topological skeleton for scalable global metrical map-building". In: *Intelligent Robots and Systems, 2004. (IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on*. Vol. 2, 1530–1536 vol.2. DOI: 10.1109/IROS.2004.1389613.

Montello, Daniel R. (1993). "Scale and multiple psychologies of space". In: *Spatial Information Theory A Theoretical Basis for GIS: European Conference, COSIT'93 Marciana Marina, Elba Island, Italy September 19–22, 1993 Proceedings*. Ed. by Andrew U. Frank and Irene Campari. Berlin, Heidelberg: Springer Berlin Heidelberg. Chap. 21, pp. 312–321. ISBN: 978-3-540-47966-6. DOI: 10.1007/3-540-57207-4_21. URL: http://dx.doi.org/10.1007/3-540-57207-4_21.

Montemerlo, Michael, Sebastian Thrun, Daphne Koller, and Ben Wegbreit (2002). "FastSLAM: A Factored Solution to the Simultaneous Localization and Mapping Problem". In: *Eighteenth National Conference on Artificial Intelligence*. Edmonton, Alberta, Canada: American Association for Artificial Intelligence, pp. 593–598. ISBN: 0-262-51129-0. URL: http://dl.acm.org/citation.cfm?id=777092.777184.

Moravec, H.P. and A. Elfes (1985). "High resolution maps from wide angle sonar". In: *Robotics and Automation. Proceedings. 1985 IEEE International Conference on*. Vol. 2, pp. 116–121. DOI: 10.1109/ROBOT.1985.1087316.

Nash, Alex, Kenny Daniel, Sven Koenig, and Ariel Feiner (2007). "Theta*: Any-angle Path Planning on Grids". In: *Proceedings of the 22Nd National Conference on Artificial Intelligence - Volume 2*. AAAI'07. Vancouver, British Columbia, Canada: AAAI Press, pp. 1177–1183. ISBN: 978-1-57735-323-2. URL: http://dl.acm.org/citation.cfm?id=1619797.1619835.

Nash, Alex, Sven Koenig, and Craig Tovey (2010). "Lazy Theta*: Any-angle Path Planning and Path Length Analysis in 3D". In: *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence (AAAI-10)*. URL: http://aigamedev.com/open/tutorial/lazy-theta-star/.

Newman, Ehren L, Jeremy B Caplan, Matthew P Kirschen, Igor O Korolev, Robert Sekuler, and Michael J Kahana (2007). "Learning your way around town: How virtual taxicab drivers learn to use both layout and landmark information". In: *Cognition* 104.2, pp. 231–253. URL: http://www.ncbi.nlm.nih.gov/pubmed/16879816.

Nieto, Juan I., Jose E. Guivant, and Eduardo M. Nebot (2004). "The HYbrid Metric Maps (HYMMS): A novel map representation for denseSLAM". In: *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 391–396. URL: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.206.75.

Nitsche, M., P. de Cristoforis, M. Kulich, and K. Kosnar (2011). "Hybrid mapping for autonomous mobile robot exploration". In: *Intelligent Data Acquisition and Advanced Computing Systems (IDAACS), 2011 IEEE 6th International Conference on*. Vol. 1, pp. 299–304. DOI: 10.1109/IDAACS.2011.6072761.

O'Keefe, J. and A. Speakman (1987). "Single unit activity in the rat hippocampus during a spatial memory task". English. In: *Experimental Brain Research* 68.1, pp. 1–27. ISSN: 0014-4819. DOI: 10.1007/BF00255230. URL: http://dx.doi.org/10.1007/BF00255230.

O'Keefe, John (1976). "Place units in the hippocampus of the freely moving rat". In: *Experimental Neurology* 51.1, pp. 78–109. ISSN: 0014-4886. DOI: 10.1016/0014-4886(76)90055-8. URL: http://www.sciencedirect.com/science/article/pii/0014488676900558.

Panaite, Petrişor and Andrzej Pelc (1999). "Exploring Unknown Undirected Graphs". In: *Journal of Algorithms* 33.2, pp. 281–295. ISSN: 0196-6774. DOI: 10.1006/jagm.1999.1043. URL: http://www.sciencedirect.com/science/article/pii/S019667749991043X.

Pearl, Judea (1984). *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc. ISBN: 0-201-05594-5.

Pfingsthorn, M. and A. Birk (2014). "Representing and solving local and global ambiguities as multimodal and hyperedge constraints in a generalized graph SLAM framework". In: *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, pp. 4276–4283. DOI: 10.1109/ICRA.2014.6907481.

Pinies, P., Lina M. Paz, and J.D. Tardos (2009). "CI-Graph: An efficient approach for large scale SLAM". In: *Robotics and Automation, 2009. ICRA '09. IEEE International Conference on*, pp. 3913–3920. DOI: 10.1109/ROBOT.2009.5152581.

Polychronopoulos, George H. and John N. Tsitsiklis (1996). "Stochastic Shortest Path Problems with Recourse". In: *Networks* 27.2, pp. 133–143. ISSN: 1097-0037. DOI: DOI:10.1002/(SICI)1097-0037(199603)27:2<133::AID-NET5>3.0.CO;2-L. URL: http://dx.doi.org/10.1002/(SICI)1097-0037(199603)27:2%3C133::AID-NET5%3E3.0.CO;2-L.

Ponulak, Filip Jan and John J Hopfield (2013). "Rapid, parallel path planning by propagating wavefronts of spiking neural activity". In: *Frontiers in Computational Neuroscience* 7.98. ISSN: 1662-5188. DOI: 10.3389/fncom.2013.00098. URL: http://www.frontiersin.org/computational_neuroscience/10.3389/fncom.2013.00098/abstract.

Pudney, Chris (1998). "Distance-Ordered Homotopic Thinning: A Skeletonization Algorithm for 3D Digital Images". In: *Computer Vision and Image Understanding* 72.3, pp. 404–413. ISSN: 1077-3142. DOI: 10.1006/cviu.1998.0680. URL: http://www.sciencedirect.com/science/article/pii/S1077314298906804.

Puterman, Martin L. (1994). *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Ed. by Wiley. 1st. ISBN: 978-0-471-72782-8. New York, NY, USA: John Wiley & Sons, Inc., p. 684. ISBN: 0471619779. URL: http://eu.wiley.com/WileyCDA/WileyTitle/productCd-0471727822.html.

Quinlan, S. and O. Khatib (1993). "Elastic bands: connecting path planning and control". In: *[1993] Proceedings IEEE International Conference on Robotics and Automation*, 802–807 vol.2. DOI: 10.1109/ROBOT.1993.291936.

Rao, Nageswara S.V., Srikumar Kareto, Weimin Shi, and S. Sitharama Iyengar (1993). *Robot Navigation in Unknown Terrains: Introductory Survey of Non-Heuristic Algorithms*. Survey. Oak Ridge, Tennessee 37831: Oak Ridge National Laboratory. URL: http://web.ornl.gov/info/reports/1993/3445603759939.pdf.

Robins, Anthony (1995). "Catastrophic forgetting, rehearsal and pseudorehearsal". In: *Connection Science: Journal of Neural Computing, Artificial Intelligence and Cognitive Research* 7, pp. 123–146. URL: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.108.3078.

Rueckert, Elmar, David Kappel, Daniel Tanneberg, Dejan Pecevski, and Jan Peters (2016). "Recurrent Spiking Networks Solve Planning Tasks". In: *Scientific Reports* 6, pp. 21142–. DOI: 10.1038/srep21142.

Russell, Stuart and Peter Norvig (2009). *Artificial Intelligence: A Modern Approach*. 3rd ed. Pearson.

Savelli, F. and B. Kuipers (2004). "Loop-closing and planarity in topological map-building". In: *Intelligent Robots and Systems, 2004. (IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on*. Vol. 2, 1511–1517 vol.2. DOI: 10.1109/IROS.2004.1389610.

Schuster, M.J., C. Brand, H. Hirschmuller, M. Suppa, and M. Beetz (2015). "Multi-robot 6D graph SLAM connecting decoupled local reference filters". In: *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*, pp. 5093–5100. DOI: 10.1109/IROS.2015.7354094.

Shewchuk, Jonathan Richard (1996). "Applied Computational Geometry Towards Geometric Engineering: FCRC'96 Workshop, WACG'96 Philadelphia, PA, May 27–28,

1996 Selected Papers". In: ed. by Ming C. Lin and Dinesh Manocha. Berlin, Heidelberg: Springer Berlin Heidelberg. Chap. Triangle: Engineering a 2D quality mesh generator and Delaunay triangulator, pp. 203–222. ISBN: 978-3-540-70680-9. DOI: 10.1007/BFb0014497. URL: http://dx.doi.org/10.1007/BFb0014497.

Shewchuk, Jonathan Richard (2002). "Delaunay refinement algorithms for triangular mesh generation". In: *Computational Geometry* 22.1–3. 16th {ACM} Symposium on Computational Geometry, pp. 21–74. ISSN: 0925-7721. DOI: 10.1016/S0925-7721(01)00047-5. URL: http://www.sciencedirect.com/science/article/pii/S0925772101000475.

Sibson, R. (1973). "SLINK: an optimally efficient algorithm for the single-link cluster method". In: *The Computer Journal (British Computer Society)* 16.1, pp. 30–34. DOI: 10.1093/comjnl/16.1.30.

Simons, Daniel J. and Ranxiao Frances Wang (1998). "Perceiving real-world viewpoint changes". In: *Psychological Science* 9.4, pp. 315–320. DOI: 10.1111/1467-9280.00062. URL: http://pss.sagepub.com/content/9/4/315.short.

Smirnov, Yury, Sven Koenig, Manuela M. Veloso, and Reid G. Simmons (1996). "Efficient Goal-Directed Exploration". In: *Proceedings of the National Conference on AI*. AAAI Press, pp. 292–297. DOI: 10.1.1.1.9941. URL: http://www.cs.cmu.edu/~mmv/papers/aaai96-yury.pdf.

Smith, Randall C. and Peter Cheeseman (1986). "On the Representation and Estimation of Spatial Uncertainly". In: *Int. J. Rob. Res.* 5.4, pp. 56–68. ISSN: 0278-3649. DOI: 10.1177/027836498600500404. URL: http://dx.doi.org/10.1177/027836498600500404.

Smith, Randall, Matthew Self, and Peter Cheeseman (1990). "Estimating Uncertain Spatial Relationships in Robotics". English. In: *Autonomous Robot Vehicles*. Ed. by IngemarJ. Cox and GordonT. Wilfong. Springer New York, pp. 167–193. ISBN: 978-1-4613-8999-6. DOI: 10.1007/978-1-4613-8997-2_14.

Steck, Sibylle D. and Hanspeter A. Mallot (2000). "The Role of Global and Local Landmarks in Virtual Environment Navigation". In: *Presence: Teleoper. Virtual Environ.* 9.1, pp. 69–83. ISSN: 1054-7460. DOI: 10.1162/105474600566628. URL: http://dx.doi.org/10.1162/105474600566628.

Stentz, Anthony (1995). "The Focussed D* Algorithm for Real-Time Replanning". In: *Proceedings of the International Joint Conference on Artificial Intelligence*. URL: http://www.frc.ri.cmu.edu/~axs/doc/ijcai95.pdf.

Sunderhauf, N. and P. Protzel (2013). "Switchable constraints vs. max-mixture models vs. RRR - A comparison of three approaches to robust pose graph SLAM". In: *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pp. 5198–5203. DOI: 10.1109/ICRA.2013.6631320.

Tapus, Adriana and Roland Siegwart (2008). "Topological SLAM". English. In: *Probabilistic Reasoning and Decision Making in Sensory-Motor Systems*. Ed. by Pierre Bessière, Christian Laugier, and Roland Siegwart. Vol. 46. Springer Tracts in Advanced Robotics. Springer Berlin Heidelberg, pp. 99–127. ISBN: 978-3-540-79006-8. DOI: 10.1007/978-3-540-79007-5_5. URL: http://dx.doi.org/10.1007/978-3-540-79007-5_5.

Thrun, S. (2001). "Learning occupancy grids with forward models". In: *Intelligent Robots and Systems, 2001. Proceedings. 2001 IEEE/RSJ International Conference on*. Vol. 3, 1676–1681 vol.3. DOI: 10.1109/IROS.2001.977219.

Thrun, S., W. Burgard, and D. Fox (2000). "A real-time algorithm for mobile robot mapping with applications to multi-robot and 3D mapping". In: *Robotics and Automation, 2000. Proceedings. ICRA '00. IEEE International Conference on*. Vol. 1, 321–328 vol.1. DOI: 10.1109/ROBOT.2000.844077.

Thrun, Sebastian (1998). "Learning metric-topological maps for indoor mobile robot navigation". In: *Artificial Intelligence* 99.1, pp. 21–71. ISSN: 0004-3702. DOI: 10.1016/S0004-3702(97)00078-7. URL: http://www.sciencedirect.com/science/article/pii/S0004370297000787.

– (2000). "Probabilistic Algorithms in Robotics". In: *AI Magazine* 21.4, p. 17. DOI: 10.1609/aimag.v21i4.1534.

Thrun, Sebastian, Jens-steffen Gutmann, Dieter Fox, Wolfram Burgard, and Benjamin J. Kuipers (1998). "Integrating topological and metric maps for mobile robot navigation: A statistical approach". In: *In Proceedings of the AAAI Fifteenth National Conference on Artificial Intelligence*. URL: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.187.7438.

Thrun, Sebastian and Michael Montemerlo (2006). "The Graph SLAM Algorithm with Applications to Large-Scale Mapping of Urban Structures". In: *Int. J. Rob. Res.* 25.5-6, pp. 403–429. ISSN: 0278-3649. DOI: 10.1177/0278364906065387. URL: http://dx.doi.org/10.1177/0278364906065387.

Tolman, Edward C. (1948). "Cognitive maps in rats and men." In: *Psychological Review* 55(4), pp. 189–208. URL: http://psycnet.apa.org/index.cfm?fa=search.displayRecord&uid=1949-00103-001.

Tomatis, Nicola and Illah Nourbakhsh (2002). "Hybrid simultaneous localization and map building: closing the loop with multi-hypothesis tracking". In: *Proc. 2002 IEEE Intl. Conf. on Robotics & Automation*. DOI: 10.1.1.149.6571. URL: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.149.6571.

Touretzky, David S. and A. David Redish (1996). "Theory of Rodent Navigation Based on Interacting Representations of Space". In: *Hippocampus* 6, pp. 247–270. DOI: 10.1002/(SICI)1098-1063(1996)6:3<247::AID-HIPO4>3.0.CO;2-K.

Trefethen, Lloyd N. and Bau David (1997). *Numerical Linear Algebra*. ISBN: 9780898713619. SIAM.

Trevizan, Felipe W. and Manuela M. Veloso (2014). "Depth-based Short-Sighted Stochastic Shortest Path Problems". In: *Artificial Intelligence* 216, pp. 179–205. ISSN: 0004-3702. DOI: 10.1016/j.artint.2014.07.001. URL: http://www.sciencedirect.com/science/article/pii/S0004370214000848.

Wagner, R., U. Frese, and B. Bauml (2014). "Graph SLAM with signed distance function maps on a humanoid robot". In: *Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on*, pp. 2691–2698. DOI: 10.1109/IROS.2014.6942930.

Werner, F., F. Maire, J. Sitte, H. Choset, S. Tully, and G. Kantor (2009). "Topological SLAM using neighbourhood information of places". In: *Intelligent Robots and Systems,*

*2009. IROS 2009. IEEE/RSJ International Conference on*, pp. 4937–4942. DOI: 10.1109/IROS.2009.5354748.

Whishaw, I.Q. (1991). "Latent learning in a swimming pool place task by rats: evidence for the use of associative and not cognitive mapping processes". In: *The Quarterly Journal of Experimental Psychology* 43(1), pp. 83–103. DOI: 10.1080/14640749108401260.

Zhang, Qiwen, D. Whitney, F. Shkurti, and I. Rekleitis (2014). "Ear-based exploration on hybrid metric/topological maps". In: *Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on*, pp. 3081–3088. DOI: 10.1109/IROS.2014.6942988.

Thèse de Doctorat
de
L'Université Paris-Saclay
Préparée à
L'université Paris-Sud

École doctorale N° 580
Sciences et technologies de l'information et de la communication
Spécialité *Robotique*

Par
**M. Fabrice Mayran de Chamisso**

# Navigation exploratoire au long de la vie

## une approche intégrant planification, navigation, cartographie et localisation pour des robots mobiles disposant de ressources finies

## [Résumé]

**Ce document est un résumé en français du manuscrit de thèse intitulé *Lifelong Exploratory Navigation : integrating planning, navigation and SLAM for autonomous mobile robots with finite resources*. Il ne peut en aucun cas être substitué à la version complète et ne doit pas être référencé dans une publication.**

# Table des matières

## 0.1 Introduction - de nouveaux paradigmes pour la robotique mobile

### 0.1.1 Robotique et mouvement

Si l'on exclut les *bots* ou *robots web* qui sont des robots virtuels (agents logiciels), tout robot peut être défini comme une machine capable de calculer ou un ordinateur capable d'interagir physiquement avec son environnement afin d'accomplir un objectif. Trois problématiques sous-tendent cette définition :

1. En quoi le monde physique conditionne-t-il les algorithmes constituant l'intelligence artificielle du robot, et, réciproquement, quel est l'impact des algorithmes sur les performances physiques du robot ?
2. Étant donné que le principal moyen d'action que possède un robot pour agir sur son environnement est un *mouvement* de toute ou partie de sa structure, comment ce mouvement doit-il être géré physiquement et algorithmiquement ?
3. Que penser de cette notion d'"accomplir un objectif". En particulier, quid de la notion de vie ou d'existence du robot, c'est à dire de l'enchaînement de missions confiées par un opérateur externe et d'objectifs propres ?

Il est possible de classer les mouvements d'un robot en quatre catégories (figure 1) :

1. les mouvements à grande échelle,
2. les mouvements à petite échelle ou manœuvres,
3. les manipulations et
4. les expressions

Les mouvements à *grande échelle* correspondent à des déplacements importants en comparaison de la taille du robot. Ceci inclut par exemple le déplacement d'une pièce à l'autre pour un robot aspirateur. Les mouvements à *petite échelle ou manœuvres* correspondent à des déplacements de l'ordre de grandeur de la taille du robot ou de l'ordre de grandeur de la portée des capteurs utilisés (*ligne de vue*). Toujours pour un robot aspirateur, des mouvements dont l'amplitude ne dépasse pas un ou deux mètres peuvent être considérés comme "à petite échelle". Cette échelle est celle de manœuvres comme le demi-tour ou l'évitement latéral d'un obstacle mouvant. La distinction entre petite et grande échelle est qu'à petite échelle, les valeurs de capteurs de distance et de mouvement peuvent être exploitées directement et ne nécessitent pas de correction de dérive ou de recalage. L'échelle suivante est constituée des *manipulations*, à savoir des mouvements d'une partie du robot dont l'objectif principal n'est pas de déplacer le centre de gravité du robot mais d'interagir avec un objet. Cette échelle de mouvement est typiquement celle des bras articulés utilisés par exemple dans les chaines d'assemblage automobiles. Enfin, les *expressions* sont des mouvements à très petite échelle dont l'objectif est de communiquer avec un opérateur ou un autre robot. Une mission typique pour un robot pourrait être la suivante : le robot commence par se déplacer pour atteindre son emplacement de travail (*grande échelle*). Il se positionne correctement (*petite échelle ou manœuvres*) avant d'agir sur un objet d'intérêt (*manipulations*). Si un opérateur entre de le champ des capteurs, le robot prévient de sa présence par un mouvement préalablement convenu (*expression*) doublé par exemple d'une alarme

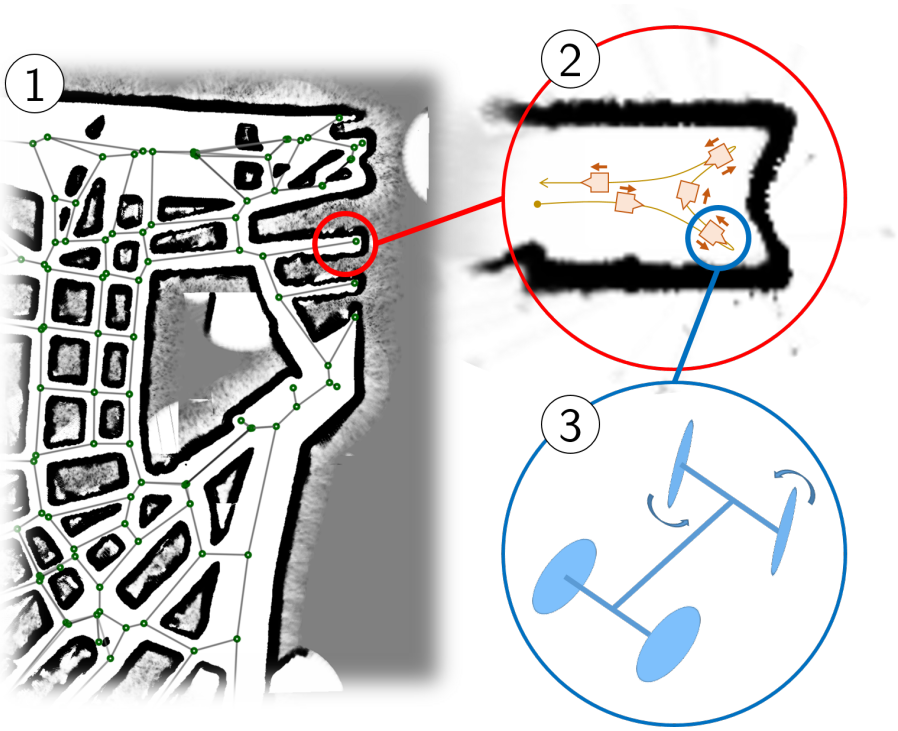



**Figure 1** – Une vision schématique des trois premières échelles de mouvement pour un robot mobile : (1) mouvements à grande échelle, (2) manœuvres et (3) manipulations.

sonore. Dans le cadre de cette thèse, nous ne considérons que les deux premières échelles de mouvement, l'hypothèse étant que les deux dernières échelles sont globalement indépendantes des deux premières ou peuvent être implémentées au-dessus de celles-ci dans une approche hiérarchique.

### 0.1.2 Planification, Navigation, Cartographie et Localisation

L'objectif de cette thèse est de concevoir un ensemble d'algorithmes permettant à un robot de maîtriser les deux premières échelles de mouvement, et ce quels que soient les objectifs à atteindre, l'environnement à traverser et la connaissance préalable de cet environnement - la *carte*. Le robot doit utiliser le moins de ressources (temps de calcul et mémoire) possible. Quatre composants sont nécessaires à la réalisation d'un mouvement : un algorithme de cartographie capable de créer une représentation « mentale » d'un environnement, un algorithme de localisation capable de situer le robot à l'intérieur de cette représentation, un algorithme de planification capable de calculer des trajectoires en utilisant la carte si nécessaire et un algorithme de navigation permettant d'exécuter ces trajectoires. Ces quatre algorithmes sont fortement interdépendants. En effet :

- la carte de l'environnement dépend des données collectées par les capteurs, qui dépendent de la trajectoire du robot,
- le robot se localise par rapport à la carte, qui elle-même est construite en utilisant la localisation du robot,
- la carte doit permettre à un algorithme de planification de calculer un chemin pour atteindre un objectif, sachant qu'il peut être nécessaire ou préférable d'explorer l'environnement plutôt que d'utiliser des chemins connus et
- le robot s'efforce de suivre le chemin calculé, sachant que celui-ci n'est pas nécessairement praticable.

Les problématiques de cartographie et de localisation ont été unifiées en un paradigme (Durrant-Whyte, Rye et Nebot, 1996) connu sous le nom de *SLAM* (pour *Simultaneous Localization and Mapping*). Cependant, le SLAM est toujours aujourd'hui considéré indépendamment des problématiques de planification et de navigation dans l'état de l'art. L'hypothèse fondamentale de cette thèse est qu'il est nécessaire d'intégrer (P)lanification, (N)avigation et (SLAM) au sein d'un unique paradigme que nous appelons PNSLAM. PNSLAM nécessite une adaptation des composants individuels :

- La planification devient *exploratoire*, c'est à dire que l'algorithme de planification considère à la fois des trajectoires calculées sur la carte et des trajectoires incluant des mouvements dans des zones non-cartographiées. Lors de la traversée d'environnements non-cartographiés, le SLAM met à jour la carte en ajoutant les nouvelles zones tout en informant l'algorithme de planification des modifications apportées.
- Le SLAM est autorisé à faire appel à la navigation (par l'intermédiaire de la planification) pour améliorer la qualité de la carte ou la localisation. En d'autres termes, le robot peut effectuer un mouvement dans le seul objectif d'améliorer sa carte ou sa précision de localisation. En outre, la carte maintenue par le SLAM est rendue compatible avec la planification.

— L'algorithme de navigation est adapté pour accepter les ordres de l'algorithme de planification et renvoyer des données de capteur au SLAM.

Nous décrivons dans les sections 0.2, 0.3 et 0.4 une implémentation possible des composants planification, navigation et SLAM modifiés. Le fonctionnement interne de chacun de ces composants est inspiré par le monde du vivant (le lecteur intéressé est invité à se reporter à la version complète de cette thèse). En particulier, une hiérarchie d'échelles de temps (réflexe, évitement, fonctionnement nominal, phase de "rêve") calquée sur le vivant est mise en place. Cette hiérarchie temporelle s'accompagne d'une hiérarchie spatiale séparant notamment manœuvres et mouvements à grande échelle. De plus, l'architecture logicielle utilisée est fortement asynchrone et permet une forte sureté de fonctionnement en séparant les processus critiques (évitement d'obstacles dynamiques entre autres) des processus n'impliquant pas de danger direct pour le robot et son environnement.

### 0.1.3 Vers une navigation exploratoire au long de la vie

Là où PNSLAM permet à un robot d'accomplir la plupart des missions constituées de mouvements à grande échelle et de manœuvres, le paradigme suppose de manière implicite que les ressources calculatoires et mémorielles du robot sont infinies. Si le robot doit accomplir plusieurs missions successives, sa mémoire et possiblement la charge de calcul associée (du fait de la complexité algorithmique) peuvent éventuellement saturer, particulièrement si l'environnement dans lequel le robot évolue est de grande taille et que le nombre de missions à accomplir est important. Le paradigme de navigation exploratoire au long de la vie (*LEN* pour *Lifelong Exploratory Navigation*) décrit la capacité pour un robot d'accomplir un nombre arbitraire de missions impliquant des mouvements dans un environnement arbitrairement grand et sans être réinitialisé entre chaque mission. Un robot suivant le paradigme LEN peut choisir de conserver des connaissances de l'environnement acquises précédemment ou de les oublier partiellement ou totalement afin de libérer des ressources et diminuer l'empreinte mémoire et le temps de calcul des algorithmes. Il apprend constamment, ce qui lui permet de s'adapter aux changements de l'environnement. L'architecture résultante est représentée sur la figure 2. Une version plus détaillée de cette architecture est donnée dans la version complète de cette thèse. Le contrôle des ressources mémorielles et calculatoires nécessaire à LEN est décrit dans les sections 0.6 et 0.7.

## 0.2 Planification exploratoire

Un algorithme de planification exploratoire est un algorithme de planification traditionnel (non exploratoire) qui est capable d'ordonner une phase d'exploration si celle-ci présente un bénéfice potentiel suffisant (figure 3). Afin de maximiser une heuristique donnée, le planificateur peut choisir de privilégier une stratégie d'exploration d'une partie encore inconnue de l'environnement ou au contraire d'utiliser des chemins déjà connus. Explorer l'environnement présente un risque (le robot peut se retrouver dans un cul-de-sac ou faire un détour) mais peut révéler des raccourcis, diminuant non seulement l'effort de navigation actuel du robot mais aussi les efforts futurs étant donné que la zone inconnue

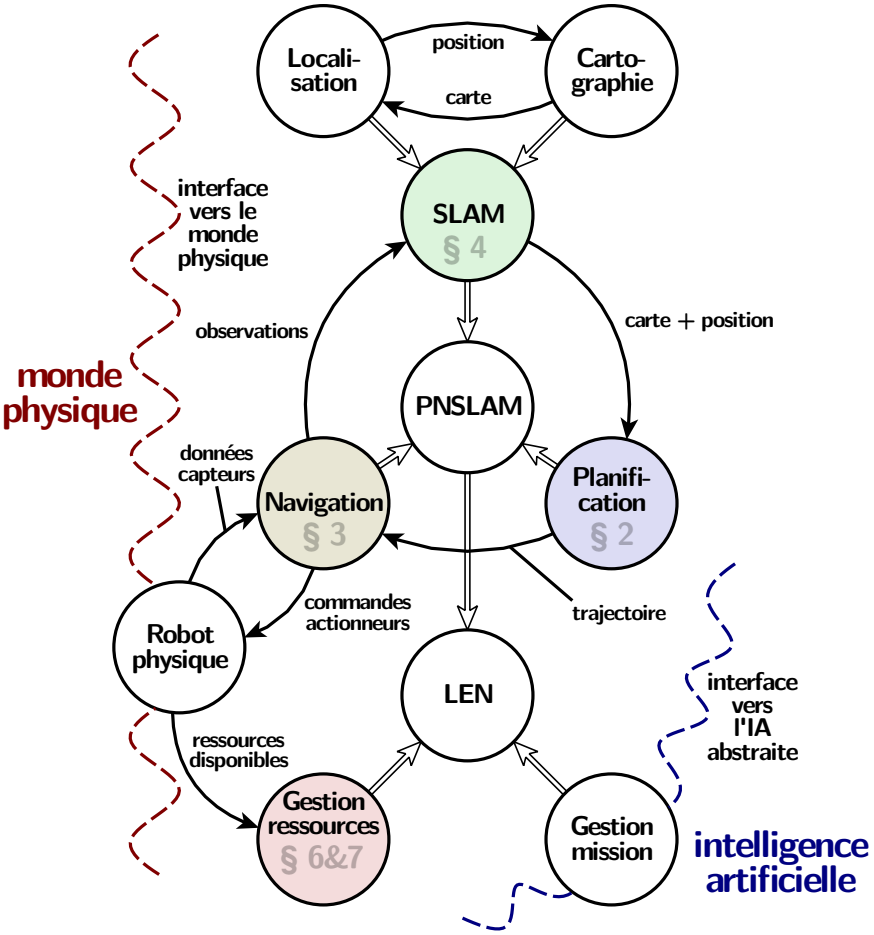



**Figure 2** – Une vue schématique du paradigme PNSLAM qui unifie (P)lanification, (N)avigation et (SLAM). Construite sur PNSLAM, la navigation exploratoire au long de la vie (LEN) ajoute la gestion de la mémoire et une unité de mission. LEN est aussi l'interface vers des processus d'intelligence artificielle de plus haut niveau. Sur ce schéma, les flèches en traits pleins indiquent une dépendance de donnée et les flèches doubles indiquent l'intégration des composants. Les ressources du robot incluent notamment la mémoire, la puissance de calcul et la batterie. Cette thèse propose une implémentation des composants *Planification*, *Navigation*, *SLAM* et *gestion de la mémoire* compatible avec le paradigme LEN.

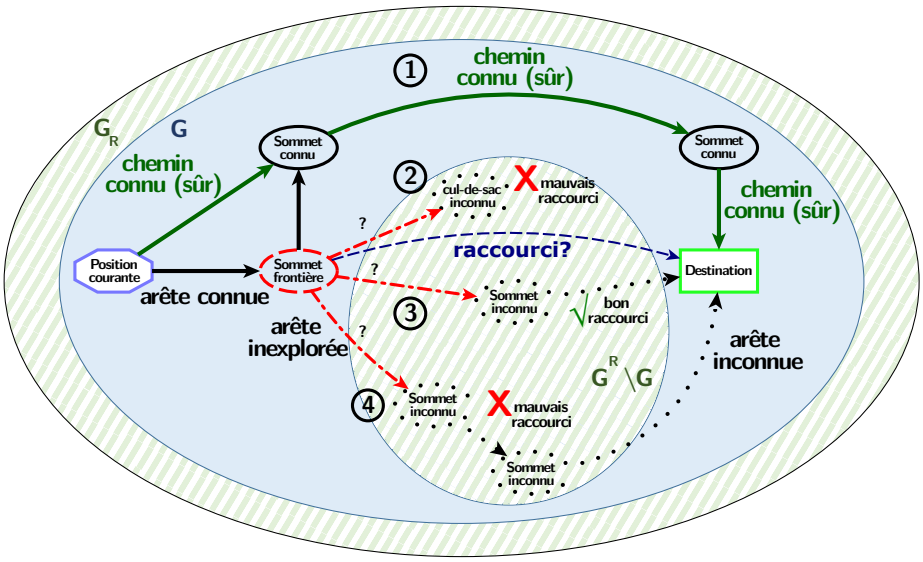



**Figure 3** – Un cas typique de navigation exploratoire : bien qu'il soit possible d'utiliser un chemin connu pour atteindre la destination (1), il peut être possible de diminuer la distance parcourue en utilisant un raccourci à travers une zone encore inexplorée (3). Cependant, de mauvais raccourcis (qui allongent la distance) peuvent aussi exister (2,4).

traversée est cartographiée à la volée par le SLAM. Nous proposons un algorithme de planification exploratoire sur graphe appelé EDNA* (Mayran de Chamisso, Soulier et Aupetit, 2015) qui est une modification simple d'A* (Hart, Nilsson et Raphael, 1968). Cette modification peut être appliquée à d'autres algorithmes de planification, notamment (Lazy-)Theta* (Nash et al., 2007 ; Nash, Koenig et Tovey, 2010) afin de le leur adjoindre une capacité d'exploration.

A* est un algorithme de calcul du plus court chemin entre deux sommets $O_0$ et $F$ d'un graphe dans lequel les sommets $X$ du graphe sont considérés par ordre d'heuristique $D + \mathcal{H}$ croissante, où $D$ est la distance de $O_0$ à $X$ en suivant les arêtes du graphe et $\mathcal{H}$ est une sous-estimation de la distance restant à parcourir de $X$ à $F$. EDNA* adjoint à cette heuristique une seconde heuristique $\mathcal{R}$, l'heuristique de *risque exploratoire*, calculée uniquement sur les sommets possédant une arête menant à une zone encore inexplorée. $\mathcal{R}$ (sur)estime la longueur d'un potentiel raccourci traversant la zone inexplorée. Le choix entre stratégie non-exploratoire et stratégie exploratoire dépend des valeurs relatives de $D + \mathcal{H}$ et $\mathcal{R}$. La figure 4 donne une explication géométrique des algorithmes A* et EDNA*.

Selon les environnements testés, l'utilisation d'EDNA* et ses facultés d'exploration permet de réduire la longueur des chemins empruntés par le robot de quelques pourcents à quelques dizaines de pourcents en moyenne. Il arrive fréquemment qu'EDNA* découvre un raccourci divisant par 10 voir plus la longueur du chemin parcouru. Il est possible de paramétrer EDNA* pour favoriser ou au contraire pénaliser l'exploration. En prenant $\mathcal{R}$ faible, EDNA* devient

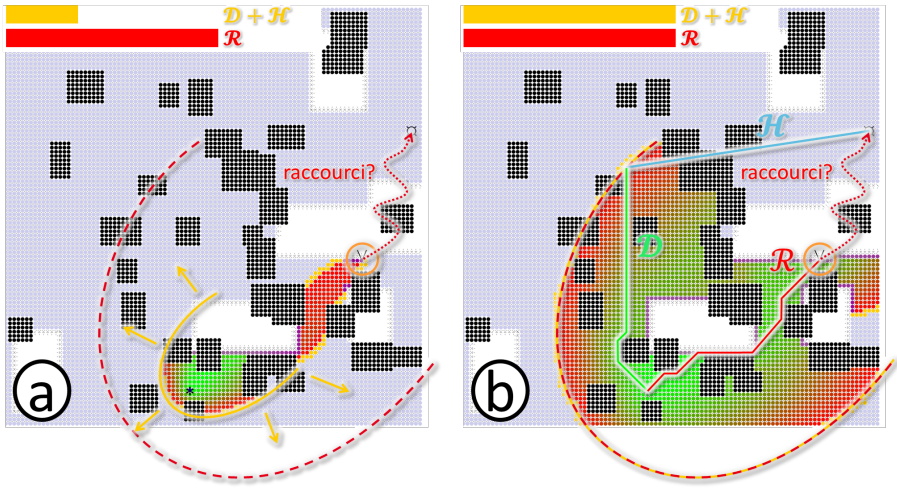



**Figure 4** – Fonctionneemnt d'A* et EDNA*. A* et EDNA* considèrent les sommets du graphe par ordre d'heuristique $D + \mathcal{H}$ croissante. Les sommets en jaune seront les prochains considérés par A* et EDNA*. Si, lors du parcours de graphe, un sommet possède une arête menant à une zone inexplorée (a), EDNA* calcule une seconde heuristique $\mathcal{R}$, l'heuristique de *risque exploratoire*, représentée par une ellipse rouge pointillée et qui (sur)estime la longueur d'un chemin contenant une phase d'exploration. L'algorithme continue de s'exécuter jusqu'à trouver le plus court chemin ou jusqu'à ce que $D + \mathcal{H}$ atteigne $\mathcal{R}$. Si les deux heuristiques se rejoignent (b), un raccourci a probablement été trouvé. En contrôlant à quel point l'heuristique $\mathcal{R}$ *surestime* la longueur d'un chemin exploratoire, il est possible de contrôler l'équilibre entre exploration et utilisation de la carte existante.

**Figure 5** – Une grille d'occupation. Espace libre en blanc, obstacles en noir, zones incertaines en nuances de cyan.

l'algorithme d'exploration "plus proche voisin inexploré". Au contraire, en prenant $\mathcal{R} \to \infty$, le robot n'utilisera que des chemins déjà connus, sauf si sa destination $F$ ne peut pas être atteinte sans explorer l'environnement, auquel cas EDNA* recourra à une phase d'exploration depuis l'endroit le plus prometteur de la carte. Il est possible de prouver qu'EDNA* permettra toujours au robot d'atteindre $F$ si cela est physiquement possible. De plus, dans le cas théorique où les heuristiques $D + \mathcal{H}$ et $\mathcal{R}$ sont idéales, l'algorithme EDNA* est optimal en terme de distance parcourue par le robot et en terme de charge de calcul et complexité algorithmique.

## 0.3 Navigation et topologie

Le composant de navigation est chargé d'exécuter les itinéraires donnés par l'algorithme de planification (exploratoire). L'algorithme EDNA* indique quel chemin doit être emprunté, à charge pour le composant de navigation de suivre ce chemin jusqu'à la prochaine instruction d'EDNA*.

Dans le cadre de cette thèse, nous avons choisi d'utiliser une grille d'occupation (figure 5) comme représentation intermédiaire entre les capteurs (odométrie, distance, ...) et les composants de SLAM et planification. La grille d'occupation décrit les obstacles se trouvant à proximité du robot. Elle est toujours centrée sur le robot et est utilisée pour planifier des *manœuvres* telles que l'évitement d'obstacles statiques ou dynamiques. La grille d'occupation est une représentation à petite échelle intégrant les valeurs de capteurs brutes. En particulier, elle ne nécessite pas l'utilisation d'une approche de type SLAM pour corriger l'inévitable dérive odométrique (voir section 0.4).

En plus d'utiliser la grille d'occupation pour l'évitement d'obstacles, nous avons choisi d'en extraire la topologie de l'environnement à travers le squelette topologique ou Graphe de Voronoï généralisé (GVG). Cette topologie pourra être utilisée par notre algorithme de SLAM présenté dans la section 0.4. Le processus d'extraction du squelette est basé sur une grille de vecteurs pointant de chaque pixel inoccupé de la grille vers le pixel occupé le plus proche ( *Vectorial Euclidean Distance Map* ou *VEDM*), calculée avec un algorithme proposé par Danielsson

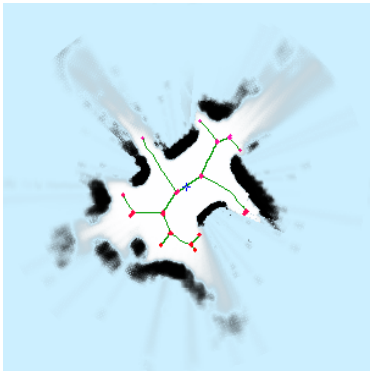


**Figure 6** – Une grille d'occupation avec squelette topologique.

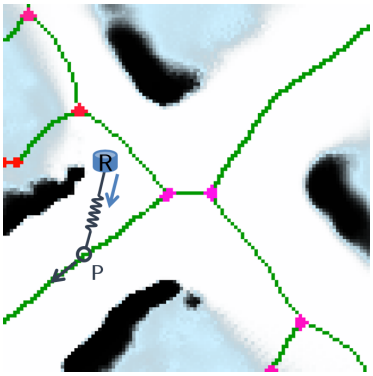



**Figure 7** – Le robot est attaché en $P$ à une arête du squelette topologique par une perche partiellement élastique (penser "téléski"). Cela lui garantit une certaine liberté de mouvement tout en garantissant le suivi d'une arête.

(1980). La détection des chemins dans l'environnement est paramétrée par la taille du robot, de sorte que le squelette calculé ne comporte que des chemins physiquement traversables par le robot. Contrairement aux méthodes de l'état de l'art, notre approche (figure 6) ne suppose aucune hypothèse sur les parties inconnues de la grille et permet au squelette de rester stable dans le temps alors même que la grille est mise à jour et que le robot se déplace.

Le passage de la représentation topologique (discrète) utilisée par EDNA* à des commandes d'actionneurs continues se fait par l'intermédiaire d'une approche représentée sur la figure 7 et analogue au fonctionnement d'un téléski. Une procédure d'évitement d'urgence utilisant la VEDM est également prévue si le robot s'approche trop près d'un obstacle. Le temps de réponse global des algorithmes de détection du squelette et de contrôle-commande (incluant l'évitement d'urgence) est strictement inférieur à 50ms et peut être ramené à moins de 20ms si besoin en utilisant une implémentation en nombre entiers au lieu de flottants.

## 0.4 Cartographie et localisation

Le problème du SLAM est probablement le problème canonique de la robotique mobile qui a reçu le plus d'attention (Durrant-Whyte et Bailey, 2006). Le problème est le suivant : le robot peut facilement établir une cartographie locale de son environnement, par exemple en intégrant les valeurs de capteurs dans une grille d'occupation. Cette approche est suffisante pour l'évitement d'obstacles statiques et dynamiques de petite taille. Elle n'est en revanche pas suffisante pour garantir la faculté de mouvement à *grande échelle* du robot, et cela du fait qu'il n'est pas possible de garantir la cohérence à grande échelle d'une grille d'occupation. En effet, le robot calcule toujours sa position actuelle en estimant le chemin parcouru depuis sa position précédente, que ce soit en estimant la vitesse de rotation des roues ou en suivant la position de points 3D dans le champ d'une caméra ou toute autre technique d'estimation. Du fait de l'intégration de ces mesures comportant chacune une petite erreur, l'estimation de position dérive au fil du temps. En conséquence, si le robot effectue une boucle dans l'environnement et revient à sa position de départ, la position finale estimée ne sera pas la position initiale et la carte comportera deux copies décalées de l'environnement autour de la position de départ (problème de *fermeture de boucle*). Un processus capable de détecter que le robot est de retour à un endroit déjà visité et de recalculer la carte en prenant en compte cet élément est nécessaire. Ce processus constitue l'élément central de notre approche de SLAM.

L'approche de SLAM que nous développons est une amélioration des SLAMs hybrides métriques/topologiques développés par l'équipe de Kuipers (Beeson, Modayil et Kuipers, 2010 ; Kuipers et al., 2004 ; Kuipers, 2000), Bailey (Bailey, 2002) et Bosse et al. (Bosse et al., 2004). Elle incorpore la notion clé de projection d'incertitude entre deux points de la carte (là où les SLAMs traditionnels (Durrant-Whyte et Bailey, 2006) sont cantonnés à projeter l'incertitude depuis l'origine). Par rapport à l'état de l'art, elle introduit également de nouveaux éléments dont :

- la possibilité de modéliser des relations unilatérales (chemins à sens unique),
- la distinction fondamentale entre incertitude liée à la détection d'un endroit et incertitude liée à l'estimation d'un mouvement,
- la prouvabilité de la cartographie et de la localisation et
- la possibilité pour le SLAM d'influer sur la trajectoire du robot afin de résoudre une ambiguïté dans l'environnement et plus généralement l'intégration Planification - Navigation - SLAM.

Nous avons prouvé que l'incertitude de position, qui est censée croître de manière monotone au fur-et-à-mesure que le robot se déplace du fait de l'estimation imparfaite du mouvement, peut en fait être ramenée à la seule incertitude de détection de l'endroit courant à partir du moment où cet endroit a déjà été traversé auparavant. En d'autre termes, l'incertitude de position sur une boucle dans la carte peut être ramenée à l'incertitude de détection du point de fermeture de la boucle (figure 8). Cette propriété rend la cartographie dans l'espace possible en permettant d'oublier la trajectoire passée du robot.

Notre SLAM utilise la notion d'endroit et de chemin sans se préoccuper de la manière dont ces endroits et chemins sont détectés, ce qui le rend fortement générique et indépendant des capteurs utilisés. L'utilisation des arêtes et sommets du squelette topologique extraits par le composant de navigation (section

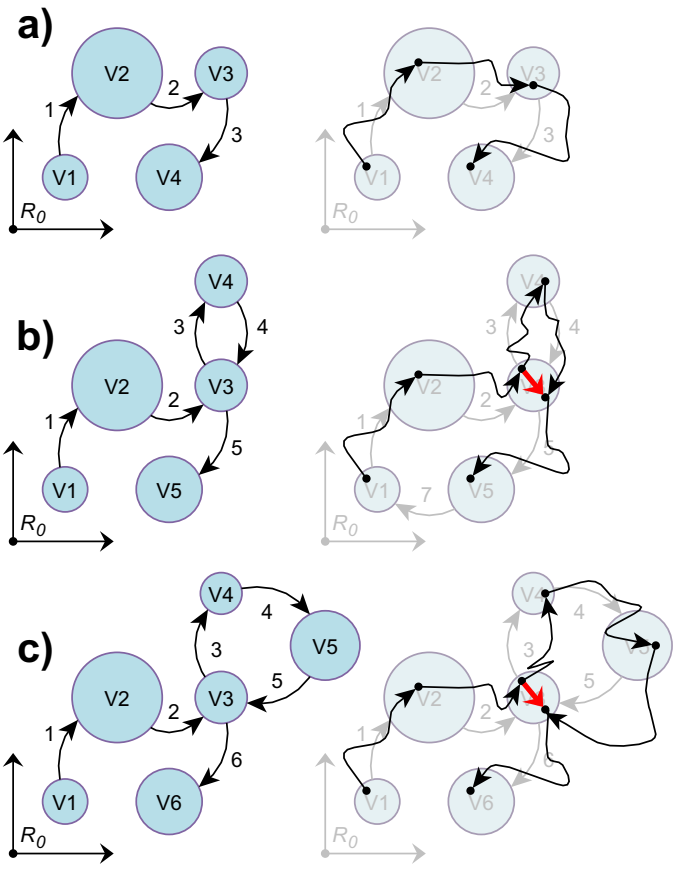



**Figure 8** – (gauche) vision topologique de la trajectoire du robot, avec les incertitudes de détection d'endroit modélisées par des sphères. (droite) trajectoire réelle du robot. Lors d'une fermeture de boucle (b,c), le robot revient près d'une position déjà visitée $V3$. Il peut donc se recaler dans l'espace avec une incertitude maximale représentée par une flèche rouge sur le schéma.

0.3) est un exemple possible. Afin de résoudre une ambiguïté dans la carte, le robot modélise chaque observation confirmant ou infirmant une hypothèse de position par une probabilité. Encore une fois, de nombreux capteurs différents peuvent être utilisés pour lever une ambiguïté.

## 0.5 Expériences de PNSLAM

Si toutes les incertitudes de capteurs sont bornées, que les endroits et chemins sont toujours détectés correctement et que l'environnement est statique, notre approche de SLAM garantit que la carte produite ainsi que la localisation du robot à l'intérieur de cette carte seront toujours topologiquement correctes. Cependant, lorsqu'endroits et chemins sont détectés avec une fiabilité inférieure à 100%, il se peut que la carte ne reflète pas fidèlement l'environnement traversé. Nous avons donc simulé et testé en conditions réelles l'ensemble planification, navigation et SLAM (PNSLAM). Le robot simulé ou réel se déplace tout en traçant la carte de l'environnement. La qualité de la carte est évaluée par rapport à une vérité terrain à l'aide de nouvelles mesures mettant en avant l'utilisation de cette carte pour la navigation. Ces mesures évaluent certes l'aspect métrique (distances, angles) mais aussi et surtout l'aspect topologique de la carte, ce qui consiste à vérifier si celle-ci retranscrit correctement l'ensemble des endroits et chemins présents dans l'environnement cartographié.

La figure 9 montre un exemple de carte générée par un robot (lui-même représenté sur la figure 10) en utilisant notre approche. Cette figure révèle également les nombreux défis qui ont du être surmontés pour parvenir à cette carte, qui reflète parfaitement la topologie de l'environnement et permet au robot de se déplacer partout tout en utilisant des trajectoires dont la longueur moyenne n'est supérieure à l'optimum théorique que de 0.05%.

L'approche PNSLAM permet de réaliser des missions autres que l'exploration ou la navigation entre deux points, comme par exemple rechercher un trésor autour d'une position approximative (figure 11).

## 0.6 Complexité et temps d'exécution

Soit $N$ le nombre d'endroits présents dans l'environnement. Alors, le composant de navigation fonctionne (dans le pire cas) en complexité $\mathcal{O}(1)$, la planification on $\mathcal{O}(N.log(N))$ et le SLAM utilise des processus en $\mathcal{O}(N)$ et $\mathcal{O}(N.log(N))$ pour la fermeture de boucles. En pratique au contraire, l'essentiel du temps de calcul provient du composant de navigation (le calcul du squelette topologique) car celui-ci s'exécute à une fréquence très importante.

Une analyse plus fine révèle que la complexité temporelle (temps de calcul) de l'algorithme de planification dépend exclusivement de la longueur de la trajectoire calculée et de la disposition des obstacles dans l'environnement. En particulier, elle ne dépend pas de la taille totale de l'environnement. Ainsi, la complexité effective (observée) d'EDNA* est $\mathcal{O}(1)$ en la taille de l'environnement. Quant au SLAM, l'utilisation et la propagation de bornes d'incertitude permettent de limiter la complexité de son opération la plus coûteuse, à savoir la fermeture de boucles, à $\mathcal{O}(1)$ également (figure 12). Ainsi, même dans un environnement arbitrairement grand, le temps de calcul de l'approche PNSLAM

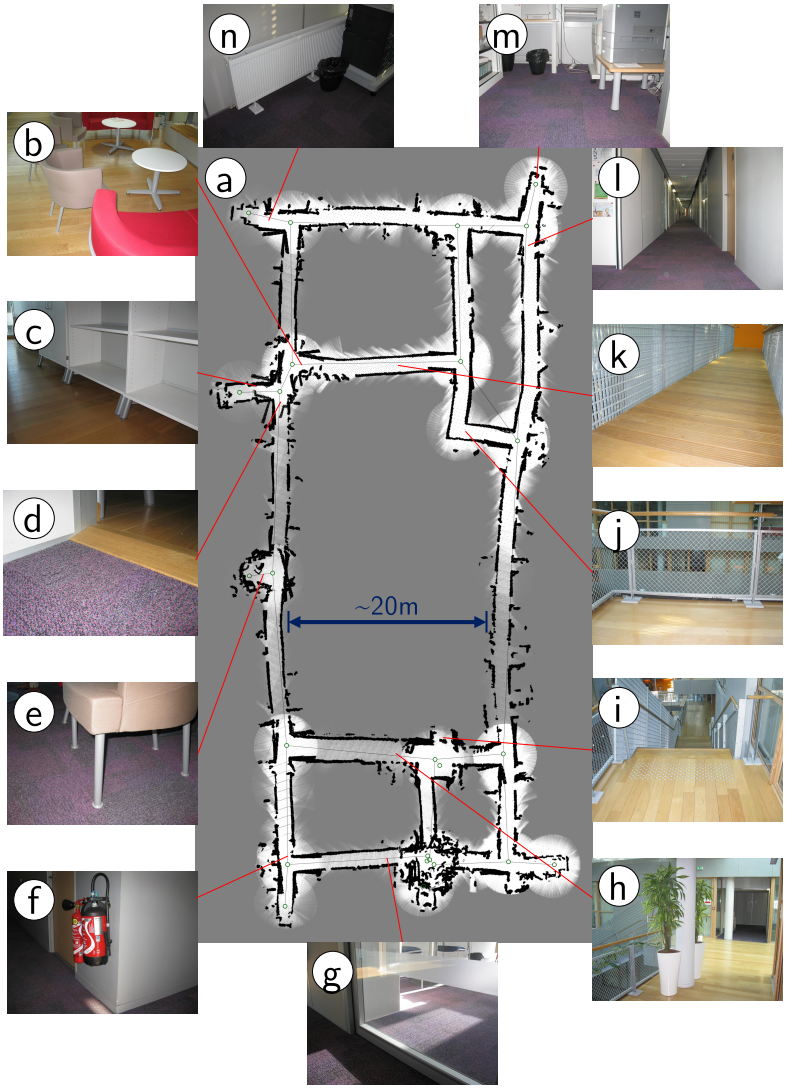



**Figure 9** – (a) Carte du bâtiment *Nano-Innov* utilisé pour les expériences, obtenue par notre approche à l'aide d'un robot pioneer 3, d'une Kinect 1 et d'un magnétomètre MEMS. Cet environnement présente de nombreuses difficultés comme : (b,c,e,f,m) - des obstacles de taille variable sur l'axe vertical, (d) - des marches et différents revêtements de sol, (k) - des portions de sol constituées de lattes inégales, (h,i,j,k) - des parois grillagées mal détectés par la Kinect 1, (g) - des portes vitrées difficilement visibles par la Kinect 1, (b,h,n) - des obstacles plus ou moins circulaires, (n,l,h) - une illumination fortement variable dans le visible et l'infrarouge, (d,i,j,n) - des masses métalliques créant localement de fortes perturbations du champ magnétique. Les escaliers (i) étaient bloqués par un carton durant les expériences.

**Figure 10** – Le robot pioneer 3 équipé avec une Kinect 1 et un magnétomètre MEMS. Le magnétomètre est monté sur une longue boîte en carton pour éviter les interférences magnétiques avec la structure métallique du robot.

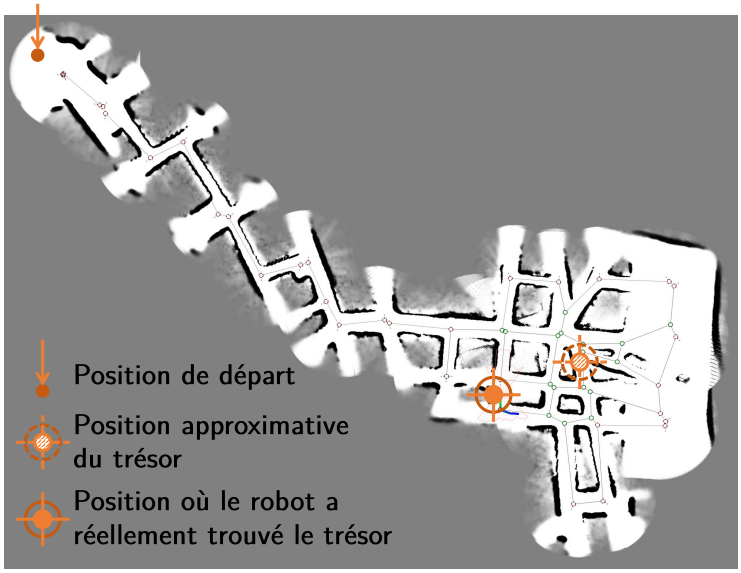



**Figure 11** – Le robot est chargé de découvrir un trésor se trouvant aux alentours d'une position donnée (indice). Il commence par se rendre à la position indice puis cherche radialement autour de celle-ci. Seule la partie nécessaire de l'environnement est cartographiée, ce qui est l'une des caractéristiques principales de l'approche PNSLAM. Le trésor est reconnu par un composant « boîte noire » non décrit dans le cadre de cette thèse.

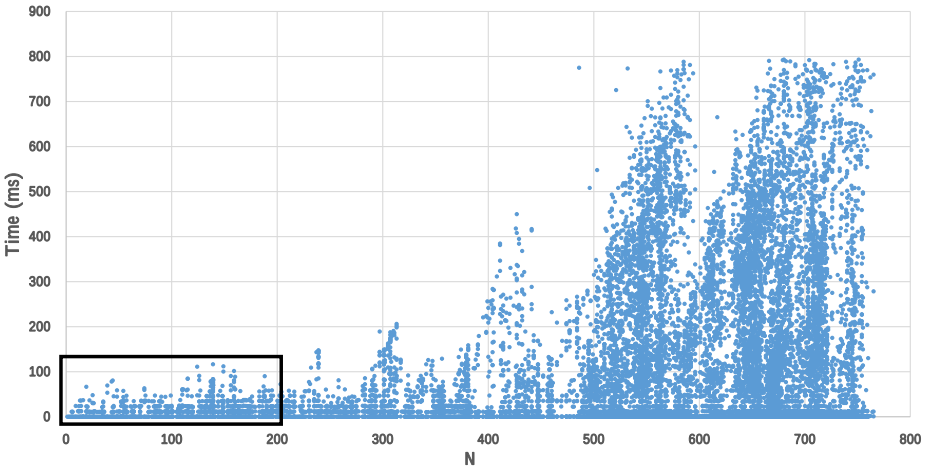



**Figure 12** – Courbe du temps d'exécution du SLAM pour chaque nouvel endroit visité, en fonction du nombre $N$ de sommets (endroits) déjà présents sur la carte. Un seul fil d'exécution à 2.7Ghz est utilisé pour les calculs. Au-delà de $N = 550$, le temps maximal d'exécution se stabilise autour de 800ms pour chaque nouvel endroit atteint (il devient *indépendant* de $N$). Ce plateau correspond à la densité maximale de sommets dans la carte. Un zoom dans le cadre noir révèle que la mesure est fortement granulaire, c'est à dire que le temps d'exécution est directement proportionnel au nombre (quantifié) de comparaisons de signatures de sommets (grilles d'occupation) réalisées lors d'une fermeture de boucle.

reste à peu près constant.

## 0.7 Consommation mémoire

Même si le temps d'exécution des algorithmes est indépendant de la taille totale de l'environnement, la consommation de mémoire nécessaire pour stocker la carte croît linéairement avec la taille de celle-ci. Si la carte couvre l'intégralité de l'environnement et si celui-ci est de très grande taille, la mémoire d'un robot peut ne pas se révéler suffisante, ce qui est inadmissible dans le cadre d'une approche de type LEN ou le robot n'est jamais réinitialisé. Nous proposons donc de compresser la carte pour réduire son empreinte mémoire en élaguant certaines zones judicieusement choisies.

Du point de vue de la navigation et grâce à la capacité d'EDNA* à exploiter des zones non représentées sur la carte (incluant donc les zones volontairement élaguées), le seul effet visible de l'élagage est de rallonger les trajets moyens. Ainsi, le moment où la compression est exécutée ainsi que le taux de compression ne sont pas contraints. La compression peut soit être effectuée régulièrement, soit lorsqu'un seuil de mémoire prédéfini est dépassé. Dans certains cas, d'autres algorithmes qu'EDNA* sont capables de navigation exploratoire, par exemple des variantes de l'algorithme glouton optimisées pour les graphes planaires. Cependant, la mise en œuvre de ceux-ci est plus complexe et moins robuste qu'EDNA*.

Nous avons envisagé plusieurs algorithmes d'élagage. L'un d'entre eux est

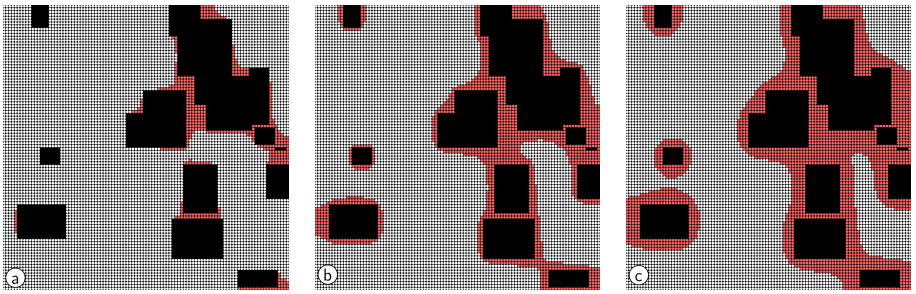



**Figure 13** – Les obstacles ne comportant pas de creux et d'irrégularités sont plus facilement gérés par les algorithmes de navigation. Une approche de compression de carte consiste donc à supprimer tous les sommets et arêtes de la carte (en blanc), en ne gardant que ceux (en rouge) qui entourent un obstacle, dans le but de rendre les obstacles plus réguliers (les sous-figures représentent des seuils de suppression différents). Cet algorithme offre des performances de navigation correctes sur des graphes sous forme de grilles.

représenté sur la figure 13. Le meilleur algorithme que nous ayons trouvé (non représenté sur la figure) prend en compte la fréquence d'utilisation de chaque chemin dans le graphe et supprime les chemins les moins utilisés. En utilisant cet algorithme, il est possible de dégrader fortement l'empreinte mémoire de la carte avec seulement un surcoût marginal en termes de navigation (14). Il est possible de faire varier le taux de compression de manière continue dans $[1; +\infty[$. Pour le jeu de données le plus difficile (l'ensemble des transports de la RATP, figure 15), à taux de compression infini de la carte, les temps de trajets en utilisant EDNA* sont allongés d'un facteur 7, sachant que la version d'EDNA* utilisée ne fait pas la différence entre mode de transports et ne sait pas que les trains vont en général plus vite que les bus.

## 0.8 Récapitulatif et conclusion

Nous avons démontré que l'intégration de la planification (éventuellement exploratoire), de la navigation, de la cartographie et de la localisation en un unique paradigme PNSLAM, essentielle pour permettre le déplacement totalement autonome d'un robot, pouvait être effectuée en apportant quelques modifications simples à des composants existants :

- l'algorithme de planification A* hérite de capacités exploratoires, pour un nouvel algorithme intitulé EDNA*,
- une grille d'occupation est utilisée pour extraire la topologie de l'environnement, cette topologie étant mise à profit pour permettre une navigation basée sur la topologie,
- un algorithme de SLAM hybride métrique/topologique est utilisé pour mener à bien la cartographie à grande échelle. Il utilise un modèle d'incertitude topologique et peut interagir avec l'algorithme de planification (EDNA*) pour demander au robot de modifier sa trajectoire afin de résoudre une ambiguïté dans l'environnement.

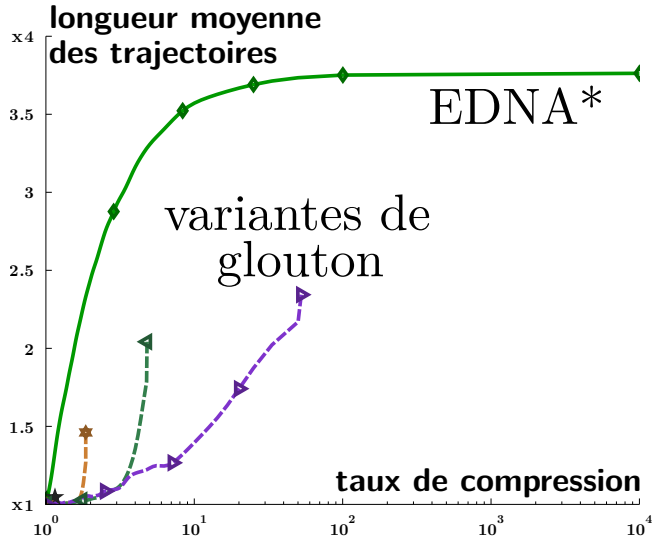



**Figure 14** – Lorsque le taux de compression de la carte augmente, la longueur moyenne des chemins empruntés augmente elle aussi. Il est donc nécessaire de trouver un compromis entre performances de navigation et empreinte mémoire. Les algorithmes inspirés de glouton peuvent obtenir de meilleures performances de navigation qu'EDNA* mais sont limités en matière de taux de compression tout en nécessitant une phase de précalcul coûteuse et difficile à mettre en œuvre avant l'élagage à proprement parler.

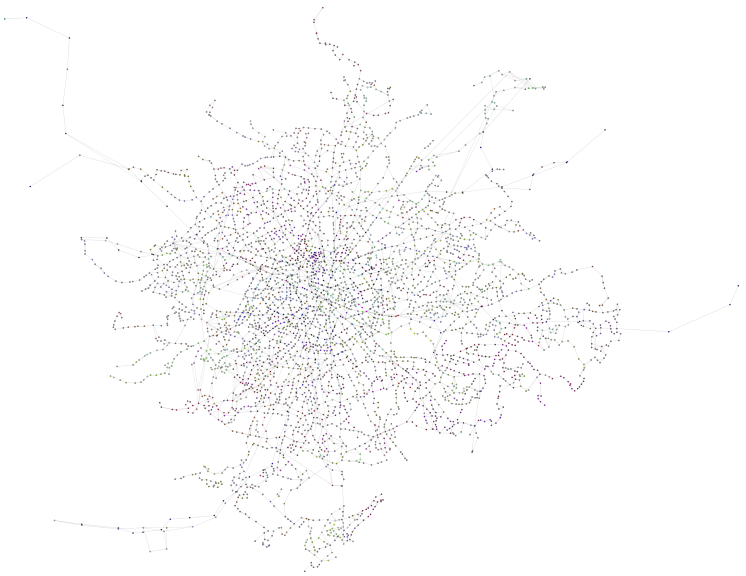


**Figure 15** – Graphe du réseau RATP montrant les arrêts, trajets et correspondances des transports publics autour de Paris. Données disponibles sur `http://data.ratp.fr/`.

Afin de rendre PNSLAM compatible avec une approche au long de la vie (c'est à dire où le robot n'est jamais réinitialisé), nous avons prouvé que la complexité algorithmique de tous les algorithmes utilisés pouvait être rendue indépendante de la taille de l'environnement dans lequel le robot circule. Nous avons également proposé une approche de compression de la carte en mémoire pour éviter la saturation mémoire. La maîtrise conjointe de la charge de calcul et de la mémoire permet d'envisager un fonctionnement dans des environnements potentiellement infinis, et donc le paradigme LEN.

Un certain nombre d'améliorations pourraient encore être apportées à l'approche, par exemple autour de la discrimination possible entre obstacles dynamiques et structure de l'environnement, autour de l'amélioration des heuristiques utilisées par les algorithmes de planification exploratoire ou encore autour de la coordination entre plusieurs robots (voire toute une flotte de robots).

# Table des figures

# Bibliographie


Bailey, Tim (2002). "Mobile Robot Localisation and Mapping in Extensive Outdoor Environments". Thèse de doct. Australian Centre for Field Robotics, University of Sydney. doi : 10.1.1.4.1707. url : http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.4.1707.

Beeson, Patrick, Joseph Modayil et Benjamin Kuipers (2010). "Factoring the Mapping Problem : Mobile Robot Map-building in the Hybrid Spatial Semantic Hierarchy". In : *The International Journal of Robotics Research* 29.4, p. 428–459. doi : 10.1177/0278364909100586. eprint : http://ijr.sagepub.com/content/29/4/428.full.pdf+html. url : http://ijr.sagepub.com/content/29/4/428.abstract.

Bosse, Michael, Paul Newman, John Leonard et Seth Teller (2004). "Simultaneous Localization and Map Building in large-scale cyclic environments using the Atlas framework". In : *International Journal of Robotics Research* 23.12, p. 1113–1139. doi : 10.1177/0278364904049393.

Danielsson, Per-Erik (1980). "Euclidean Distance Mapping". In : *Computer Graphics and Image Processing* 14, p. 227–248. url : http://webstaff.itn.liu.se/~stegu/JFA/Danielsson.pdf.

Durrant-Whyte, H. et Tim Bailey (2006). "Simultaneous localization and mapping : part I". In : *Robotics Automation Magazine, IEEE* 13.2, p. 99–110. issn : 1070-9932. doi : 10.1109/MRA.2006.1638022.

Durrant-Whyte, Hugh, David Rye et Eduardo Nebot (1996). "Localization of Autonomous Guided Vehicles". English. In : *Robotics Research*. Sous la dir. de Georges Giralt et Gerhard Hirzinger. Springer London, p. 613–625. isbn : 978-1-4471-1254-9. doi : 10.1007/978-1-4471-0765-1_69.

Hart, P.E., N.J. Nilsson et B. Raphael (1968). "A Formal Basis for the Heuristic Determination of Minimum Cost Paths". In : *Systems Science and Cybernetics, IEEE Transactions on* 4.2, p. 100–107. issn : 0536-1567. doi : 10.1109/TSSC.1968.300136.

Kuipers, B., J. Modayil, P. Beeson, M. MacMahon et F. Savelli (2004). "Local metrical and global topological maps in the hybrid spatial semantic hierarchy". In : *Robotics and Automation, 2004. Proceedings. ICRA '04. 2004 IEEE International Conference on*. T. 5, 4845–4851 Vol.5. doi : 10.1109/ROBOT.2004.1302485.

Kuipers, Benjamin (2000). "The Spatial Semantic Hierarchy". In : *Artificial Intelligence* 119.1–2, p. 191–233. issn : 0004-3702. doi : 10.1016/S0004-3702(00)00017-5. url : http://www.sciencedirect.com/science/article/pii/S0004370200000175.

Mayran de Chamisso, Fabrice, Laurent Soulier et Michaël Aupetit (2015). "Exploratory Digraph Navigation using A*". In : *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence*. IJCAI. AAAI Press / International Joint Conferences on Artificial Intelligence. URL : http://ijcai.org/proceedings/2015.

Nash, Alex, Kenny Daniel, Sven Koenig et Ariel Feiner (2007). "Theta* : Any-angle Path Planning on Grids". In : *Proceedings of the 22Nd National Conference on Artificial Intelligence - Volume 2*. AAAI'07. Vancouver, British Columbia, Canada : AAAI Press, p. 1177–1183. ISBN : 978-1-57735-323-2. URL : http://dl.acm.org/citation.cfm?id=1619797.1619835.

Nash, Alex, Sven Koenig et Craig Tovey (2010). "Lazy Theta* : Any-angle Path Planning and Path Length Analysis in 3D". In : *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence (AAAI-10)*. URL : http://aigamedev.com/open/tutorial/lazy-theta-star/.