Path planning for mobile robots in the real world

Handling multiple objectives, hierarchical structures and partial information

Doctoral Dissertation submitted to the

Faculty of Informatics of the Università della Svizzera italiana

in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

presented by

Jérôme Guzzi

under the supervision of

Prof. Luca Maria Gambardella and Dr. Alessandro Giusti

July 2018

Dissertation Committee

Dissertation accepted on 12 July 2018

I certify that except where due acknowledgement has been given, the work presented in this thesis is that of the author alone; the work has not been submitted previously, in whole or in part, to qualify for any other academic award; and the content of the thesis is the result of work which has been carried out since the official commencement date of the approved research program.

Jérôme Guzzi
Lugano, 12 July 2018

Abstract

Autonomous robots in real-world environments face a number of challenges even to accomplish apparently simple tasks like moving to a given location. We present four realistic scenarios in which robot navigation takes into account partial information, hierarchical structures, and multiple objectives. We start by discussing navigation in indoor environments shared with people, where routes are characterized by effort, risk, and social impact. Next, we improve navigation by computing optimal trajectories and implementing human-friendly local navigation behaviors. Finally, we move to outdoor environments, where robots rely on uncertain traversability estimations and need to account for the risk of getting stuck or having to change route.

Acknowledgements

I would like to thank my supervisors Luca Maria Gambardella and Alessandro Giusti for their invaluable support, and all people that traveled with me on my journey in robotics research: Gabriele Scascighini who kicked off the adventure, Andrea Rizzoli and Alexander Forster who, together with Luca Maria Gambardella, welcomed me at IDSIA, Gianni di Caro who supervised the first part of the thesis, and my colleagues Frederick Ducatelle, Eduardo Feo Flushing, Jawad Nagi, Jacopo Banfi, Fang-Lin He, Juan Pablo Rodríguez Gómez, Boris Gromov, and Omar Chavez-Garcia, for their precious help.

This work was partially supported by the Fondazione Informatica per la Promozione della Persona Disabile (FIPPD), the European Union Active and Assisted Living Programme (AAL), and the Swiss National Competence Center for Research in Robotics (NCCR-Robotics).

Contents

Introduction

Making a robot solve a *simple* task is usually a *complex* exercise for researchers, even without considering the challenge to design and build a working piece of hardware. Complexity arises when we instantiate the robot's task in the real world, even more so when the robot interacts with people.

In the thesis we focus on a ubiquitous task for autonomous mobile robots: reach a given target position using the robot's locomotion capabilities. In fact, there are instances of this problem, illustrated in any textbook, that are simple to model and relatively easy to solve, e.g., computing a feasible path given a grid map whose cells are either obstacles or free space. Yet, modeling *realistic* navigation tasks is complex. Simply reaching the target along the shortest possible path is often not enough; instead, we expect the robot to follow a trajectory that satisfies multiple other implicit goals.

As an illustration, let us consider the navigation task that a service robot needs to solve to go somewhere within a city using a map of the road (and sidewalk) network. From our experience as pedestrians, we quickly realize that the robot needs to consider many aspects. The robot should comply with traffic rules and move in a considerate, predictable manner when crossing people along the sidewalks. The robot should work well even with maps that are not always accurate; for example, maintenance works may block the access to some sidewalks on a given day. Finally, the robot performance depends on the environment; for example, the robot may need to avoid direct sunlight to be able to rely on infrared sensors and avoid obstacles; the robot may be equipped with wheels able to overcome small steps but not large ones, and may get stuck on streetcar rails when crossing them at an oblique angle. This scenario describes a complex navigation task.

In the thesis, we instantiate and solve navigation and path planning problems in different contexts and different level of abstraction, as illustrated in Figure 1.

In Chapter 1, the robot is traveling from one side to the other of a building, following a path that requires low effort, has a low risk of accidents, and minimizes discomfort caused to other users (in particular to people). The environment is

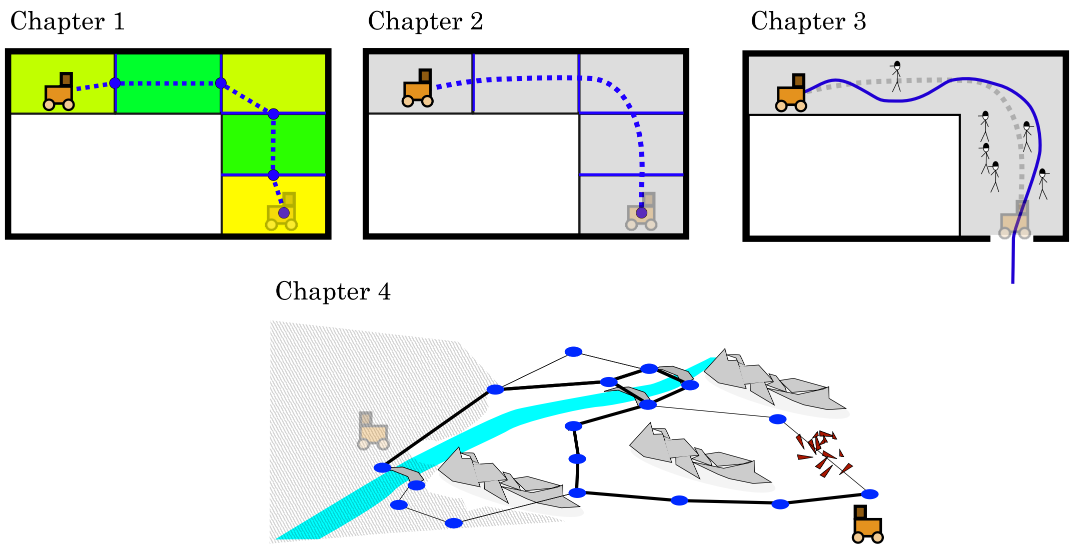


Figure 1. Outline of the thesis. Chapter 1: accounting for many factors, the robot plans the best path and identifies a sequence of cell boundaries to cross (in blue). Chapter 2: the robot chooses an optimal (short, smooth and legible to others) trajectory (blue) across these cell boundaries. Chapter 3: the robot smoothly navigates around people. Chapter 4: the robot now ventures outside in the unstructured world. The robot is not sure if it will be able to traverse a part of rough terrain and decides that is better to take the longer path instead of the shorter but riskier path. At the same time, the robot prefers crossing the river where there are *two* visible nearby bridges: in case one of them is found to be not traversable, backtracking to the other one will be cheap.

modeled as a set of cells (i.e., a non-regular decomposition of two-dimensional space into polygons) that contain a rich description of their relationship with the robot's navigation task. We compute a high-level topological route: a sequence of cells to traverse to reach the goal.

In Chapter 2, we go one step further and compute the continuous trajectory that the robot should follow along the high-level route. We consider the case when the robot is a smart wheelchair that has to be comfortable for the user (turn gently and avoid excessive accelerations). Moreover, the robot should account for the presence of other people, and, in particular, make its current goal easy to infer.

In Chapter 3, the robot is following a trajectory across a space crowded with people. Like them, the robot needs to continuously adjust its motion to avoid collisions. In addition, the robot's motion has to be as *human-friendly* as possible: in particular, it should strive to conform to the way humans use to navigate. Are humans proceeding along lanes? Then the robot should queue itself and follow the flow.

In Chapter 4, the robot has arrived at the other end of the building and ventures outside to navigate in a less structured environment. The robot needs to estimate whether it is able to traverse several stretches of difficult outdoor terrain. The estimate is computed by a statistical classifier using incomplete data, which has non-ideal accuracy; the robot, when planning a path, has to take into account the uncertainty in traversability estimates. When the robot arrives in place, it is able to refine the traversability estimates using data from its onboard sensors. The robot implements navigation strategies that are *resilient* to discoveries made along the path: in particular a resilient navigation strategy favors paths that offer cheap alternatives should a portion turn out to be not traversable.

When dealing with these issues, we observe three recurrent, connected themes: hierarchical structures, partial information and multiple objectives.

Hierarchical structures A realistic robotic task usually needs to be broken down to manageable subtasks that interact with each other. A prominent example is given by the first three chapters, where the task of reaching a target point is broken down into three subtasks, which iteratively refine the solution: (1) compute a high-level route; (2) compute a geometric trajectory along the route; (3) avoid dynamic obstacles along the trajectory. The subtasks are related in multiple ways. In direction $1 \rightarrow 2 \rightarrow 3$, the output of one planner serves as the input (goals or constraints) of the next planner. In the opposite direction, lower level planners serve to model costs. For example, a high-level planner estimates the effort needed to cross a room with a given occupancy based on the average performance of a local navigation controller in that setting.

Partial information In Chapter 1, we model heterogeneous spatial descriptors: attributes that may be subjective or objective, personal or general. We assume that, at planning time, information is complete and that the robot will be able to handle dynamic changes along its way, as e.g., in Chapter 3, where we handle unforeseen obstacles that were previously outside of the robot's view and we show that a reactive navigation policy is a good strategy to cope with them.

In Chapter 4, we deal with partial information in a different context, i.e., uncertain estimates concerning ground traversability. We discuss how the appropriate way to handle these uncertain estimates within a path-planning problem crucially depends on the consequences of attempting to cross a terrain that is not traversable. In particular, we distinguish the case in which the robot gets irreversibly stuck, and the case in which the robot can backtrack and continue traveling on a different route; we show that each case yields a very different formalization of the path planning problem.

Multiple objectives When framed in realistic contexts, a navigation task usually requires multiple objectives to be satisfied. In Chapter 1, we introduce the topic through an explicit model. The model derives, from a rich spatial description of navigability, a summary representation defining a few costs that are well suited to address many different scenarios, in particular when people share spaces with robots. In Chapter 2, we interpret some objectives (in particular *legibility*) as constraints and search for trajectories that are smooth and short. In Chapter 3, we proceed in a different way: one of the objectives (human-friendliness) is satisfied by design through the selection of the navigation behavior, instead of by explicit optimization. In Chapter 4, we finally link partial information to risk and formulate risk-aware multi-objective path planning problems.

When optimization over multiple objectives is suitable, we assume that we are computing plans for a strategic decision maker (or agent) that sits on top of the planning hierarchy. In Chapters 1 and 2, we compute exact solutions of multi-objective optimization path planning problems over (sparse) navigation graphs, whereas in Chapter 4 we compute approximations because the navigation graphs are too large.

Contributions

In this section I use the singular personal pronoun to clarify my contributions over the state-of-the-art and my own contributions in the collaborative research effort, and to summarize the scientific outcome in terms of publications, project milestones and software releases; the detailed list of scientific publications (Appendix A) and software releases (Appendix B) are provided as appendices.

Contributions over the state-of-the-art

Chapter 1: Path planning in indoor spaces I propose an extension of a recent standard for indoor maps [89] that adds a rich description of the relation between agents and spaces, with particular focus on robotic navigation. I propose an abstract model to describe planning costs in terms of geometry and environment state; I identify three macroscopic costs that describe many common scenarios both in robotics and in route planning for people.

Mine is the first effort to present a complete pipeline: from deriving a navigation graph, to defining and solving a multi-objective path planning problem in these spaces; this pipeline is composed by algorithms that are novel, namely: decomposing the space in few quasi-convex cells well suited for planning; computing the exact Pareto front of the multi-objective problem by iterative application of a state-of-the-art algorithm for the k-th shortest path on a graph [156]. Each of these algorithms solves a specific, narrow problem and therefore represents a minor contribution over the state-of-the-art [88].

Publications We introduced the pipeline for path planning in social spaces using three macroscopic costs (effort, safety risk, and social impact) in a workshop paper [56]. We elaborated on it to support users with restrained mobility in nursing homes in two project deliverables [53, 54].

Chapter 2: Optimal trajectory planning in indoor spaces I present a novel derivation of the cost associated with the discomfort of following a trajectory sitting on a robotic wheelchair as an intrinsic geometrical property. I use this cost and a parametrization of trajectories as composite Bézier curves, to define a non-linear optimization problem that searches for the trajectory that turns most gently. In this regard, I extend the state-of-the-art [28, 36, 147] in several directions: *a*) my approach can compute a long, complex trajectory that spans a whole building whereas previous work was limited to passing trough a single door; *b*) my approach adds constraints to increase predictability; *c*) I propose an effective heuristic to initialize the search; *d*) I propose a novel method to limit the dimensionality of the search space without compromising smoothness; *e*) I define and solve a novel multi-objective problem in the search for short trajectories that turn gently.

Publications We presented our optimal trajectory planner for wheelchairs in a conference paper [55].

Chapter 3: Human-friendly local navigation I present the first application to robotics of a state-of-the-art heuristic [103] used in social sciences to describe the navigation of pedestrians [105]. I compare it with two known algorithms for local robot navigation [14, 135], introducing new benchmarks for individual and group navigation performance; it is the first systematic comparison between reactive navigation algorithms with a large scale simulation campaign and real-robot tests. Moreover, I explore novel ideas on multi-robot implicit coordination, such as using artificial emotions to modulate the collective behavior.

Publications We described the extension of the pedestrian heuristic to robotics in a conference paper [60]. We compared it with other collision avoidance algorithms [14, 135] in conference paper [61]. Similar to what observed in crowds [66, 104], local navigation rules let macroscopic behaviors emerge in groups of robots, which we investigated in a conference paper [59]. We discussed how to improve navigation using artificial emotions in two conference papers [57, 58].

Chapter 4: Planning with traversability estimations In a joint effort with colleagues, I developed the first application of machine learning to estimate the traversability of a portion of an heightmap, which inspires the definition of path planning problems. In this framework, my specific contribution was to formalize risk-aware multi-objective path planning on maps with uncertain traversability estimations and develop a novel algorithm to approximate its solution; this is an incremental contribution over state-of-the-art multi-objective planners [35].

I also propose the first large scale investigation on a question that has not been tackled yet in the literature, namely: how does the quality of traversability estimations impact the performance of the navigation policies for the (very well-known) Canadian traveller problem [8, 113]?

Publications We present the complete pipeline — from estimation of terrain traversability to risk-aware multi-objective path planning — in a journal paper [26]. We discussed the impact of traversability estimation on the cost of navigation policies in a workshop paper [62].

Own contributions to collaborative research

A very large part of the work presented in the thesis originate from my direct contribution. Nonetheless, scientific research is not an individual effort and I could profit from many contributions from other researchers. I briefly go through each chapter to separate my contributions from the contributions of colleagues; the important contribution of the advisor (Prof. Luca Gambardella) and co-advisors (Prof. Gianni di Caro and Dr. Alessandro Giusti) in all my research activity and research output is always implied; in particular, the advisor and co-advisors actively participated in discussing and suggesting research ideas and assisting in the preparation of research papers.

Path planning in indoor spaces I developed the model, implemented the software and evaluated the planner performance. For the ALMA project, researchers from SUPSI-DSAN helped to define the attributes to describe social spaces; other partners have provided interfaces for my planner to the localization system, user interface and robotic wheelchair (POLIMI) and to the smart cameras (VCA); site tests were a collaborative effort of all project's partners. For the DFW project, I implemented the software on the light poles and on the drones with the help of Dr. Eduardo Feo Flushing.

Optimal trajectory planning in indoor spaces I developed the model, implemented all software and evaluated the planner performance. The robot controller makes use of the open source ROS framework and the planner uses on open source numerical solvers.

Human-friendly local navigation I adapted the pedestrian navigation model to robotics, compared it with alternatives, developed a custom simulator for large multi-agent scenarios, implemented the Human-like navigation behavior, and performed all the experiments; Dr. Alessandro Giusti contributed the model for simulating robot cameras. For the comparison, I use third-party, open source implementations of the other algorithms [69, 126]. The biomimetic model for tuning navigation based on artificial emotions is a joint idea of Prof. Gianni di Caro and me.

Planning with traversability estimations The design of the machine learning algorithms and the collection of the dataset in simulation was performed by Dr.

Omar Garcia-Chavez and Dr. Alessandro Giusti. I developed the risk-aware planner; I tested it on real maps and real robots with the help of Dr. Omar Garcia-Chavez and other partners from NCCR Robotics (ETHZ, EPFL and UNI-ZH). I designed the analysis of the Canadian traveler problem together with Dr. Alessandro Giusti. I implemented the policies, the random graph generators, performed the test and evaluated the results.

Scientific output

During the work that led to this thesis, I published, together with coauthors, 4 papers in major robotics journals (1 in Autonomous Robots, 2 in Robotics and Automation Letters, and 1 in Robotics Automation Magazine) and 13 papers in major robotics conferences: 4 at IROS, 2 at ICRA, 1 at GECCO, 1 at AAMAS, 2 at AAAI, 1 at AVICS, 1 at BIONETICS and 1 at ICSR. I was the responsible for providing mapping and planning services for a ALMA, a large EU AAL project, for DFW, a smaller Swiss CTI project, and for one work package in the search and rescue grand challenge of phase 2 of NCCR-Robotics that provided resilient path planning for ground robots on rough terrain. These efforts resulted in the open source release of 7 software libraries: a pipeline for indoor planning, its application in nursing homes and its extension to coordinate of a fleet of drones; a multi-agent simulator; a ROS package with various reactive navigation policies; a ROS package with a risk-aware path planner; and a library to perform large scale navigation experiments on random maps with uncertain traversability.

Chapter 1

Path planning in indoor spaces

1.1 Introduction

An autonomous robot is traveling towards its destination inside a building. To reach it, the robot will have to traverse several spaces with different characteristics (narrow corridors, slippery floors, crowded halls, dark rooms, steep stairs, and so on). Which one of the many alternative routes should it follow?

In this chapter, we focus on this question starting from the observation that indoor spaces have rich spatial descriptions. Spatial information gives us clues about difficulties and tradeoffs that the robot will face such as: it may fall or get stuck on some floor types; it should avoid private rooms; it may struggle to maintain good localization in dark spaces.

We approach the problem through a bottom-up analysis. We first discuss and model spatial representations that are well suited to encode rich spatial descriptions typical of environments shared with people. Then, we extract the most relevant information for a robot to navigate safely, efficiently and in a way that does not hinder people: a generic model that gives rise to a multi-objective path planning problem. Finally, we instantiate the generic problem on two real-world applications: computing appropriate routes for autonomous wheelchairs in nursing homes; coordinate drones along flyways, an outdoor scenario that features a similar spatial structure.

We present a model that lets agents account for rich information about themselves and their environment, and get a description of several optimal routes (each made of a sequence of cells to traverse) to reach a destination. In the next chapters, we will discuss how to refine the plans: in Chapter 2 we compute the best trajectory that passes through a sequence of cells; in Chapter 3 we study how the robot avoids dynamic obstacles along the route.

1.2 Related Work

1.2.1 Spatial representations

When humans describe a scenery, they normally label each part: "here is a grass field, there is a swamp, and in between a river". To encode this description in a map, we may list pairs of shapes and labels.

Typically shapes attached to different labels may overlap: the *grass* field and the *swamp* may be part of a larger *flatland* area. There are at least two alternatives to encode overlapping spatial information. We could use a single generating set of shapes and attach multiple labels to a shape, labeling areas as *flatland grass* or *flatland swamp*. Alternatively, we could separate conflicting labels into non-overlapping sets of shapes: a set of shapes would only carry labels about terrain grounds like *grass* and *swamp*, while another set would carry labels about terrain shape like *flatland*.

The second alternative, which we adopt in this chapter, is common in geographical information science, for which a family of (vectorial) multilayered [9, 158] formats (foremost the Geographic Markup Language, GML) contains tagged features linked to arbitrary polygonal shapes (also referred as *cells* in the following). This representation reflects the way humans reason about space, as hinted, for example, by Cognitive Maps [39] (a neuropsychological model of how humans and animals conceptualize spatial information and use it to navigate).

Indoor spaces are well modeled using multilayered structures for several reasons: *a*) spatial knowledge often originates from humans like buildings' designers[1], managers, and users; *b*) it's easy to subdivide indoor spaces into (tagged) cells like corridors, stairs, lunch rooms, or doors, because the spaces were designed by humans and share a common spatial semantics [95]; *c*) most information about geometry and designed usage of a space (e.g., *transiting* for a corridor) is known and static; *d*) many types of ambient sensors are linked to cells; for example, a thermometer measures the temperature of a room, a smart camera estimates the crowding in an area, or a light barrier detects passages through a door.

People naturally think in terms of cells: "*this room* is empty", "*this corridor* has low illumination", or "I have no access *to this area*". Psychometric attributes of places can be quantified through surveys among people [24]; people acquire spatial information by looking at maps (better for judging the relative location of places) or by navigating in the environments (better for orienting themselves

[1]Modern computer-aided design file formats, like IFC-XML, indeed contains tagged features, e.g., a door tag linked to its geometry and its material.

with respect to unseen objects and to estimate distances) [142]; in fact, visibility is very important for human's perception, representation and movement inside of a building [146]. The relation between spatial attributes and the use a person makes of an environment has been investigated in different contexts; for example, well designed, attractive open space favor public health by stimulating people to walk across them [48, 49].

These concepts also apply in the context of indoor robotics, in particular when robots have spatially situated interactions with people [92, 101, 108].

Geographic information system (GIS) maps are commonly used by robots to navigate outdoors [145]. In this chapter, we build upon indoorGML, a GIS data model for indoor spaces as a multilayered graph of cells, that has been recently approved as a standard by the Open Geographical Consortium [112]. It is expected that an increasing number of architectural plans will be provided in formats that contain rich geometric and semantic information such as IFC (Industry Foundation Classes) and that automatic tools will convert IFC data to IndoorGML maps to visualize maps and get navigation instructions inside buildings.

1.2.2 Rich maps and multi-objective path planning

For path planning, cells are preferably convex [88]; decomposing a map into a set of polygonal cells is a widely studied topic both in robotics [87] as well in computational geometry [27]. Besides resulting in a small number of cells, convex decomposition for robot path planning has additional objectives, e.g., adding diagonals that look as natural continuations of walls [134].

In this chapter, we focus on path planning problems for mobile robots [120], in particular on multi-objective path planning problems [2, 102], on maps with rich semantics, for which objectives arise as a compact description of the navigability of cells.

Multi-objective path planning is widely applied to compute the best routes, e.g., for ships [7], for (unmanned) aerial vehicles [86], for railway passengers looking for few train change, low prices, short travel time [153], for drivers to avoid congestions [75], or for network design [31]. Multi-objective path planning for robots, considers different costs, such, for example, minimal jerk and duration for robotic manipulators [123], or maximal clearance and short length for mobile robots [34].

We follow a common approach to solve a multi-objective problem: compute any solution that a rational strategy may select, i.e., the Pareto set [88, 138]. The strategic agent may be nonetheless overwhelmed when lots of possibilities are provided and it may help if we provide a more sparse set of solutions, different enough to represent significantly different strategic choices [42].

A Pareto set may be very large and contain all paths of a the graph, therefore viable solutions approximate the Pareto set, using for instance evolutionary methods that combine genetic algorithms with Dijkstra's algorithm [75, 161] to compute paths that are short, safe, and smooth [1, 74] for robots.

As an alternative, Particle Swarm Optimization allows fast computation of short and low risk paths for robots dangerous situations [159]. Variations of A*, such as MOA* [139] and NAMOA* [98], use heuristics to speed up the search [96, 97]. Multi-objective linear problem [10] can be solved iteratively with input from the strategic decision maker [106]. NBI (Normal-Boundary Intersection) [33] is a popular method to approximate Pareto sets starting from the convex hull of the solutions that minimize the single objectives before moving towards the boundary.

In robotics, picking one of the Pareto optimal solutions may be delegated to a human controller or to a higher-level planner, like a task scheduler [51]. For both cases, it's important that the path planner provides a compact, yet rich description of the solutions to facilitate the selection, similar to multi-objectives path planners that incorporate driver preferences [19], or methods based on fuzzy-optimization that take into account the preference of the decision maker [154].

1.2.3 Trajectory planning in social spaces

In this chapter, we discuss high-level route planning where the possible indirect interaction of robots and humans is encoded in attributes of cells. We will discuss more direct human-robot interaction in the context of navigation in Chapter 3.

Several existing planning approaches account for the fact that robots share space with humans: discomfort to humans can be minimized through the enforcement of specific constraints [30], additional cost terms [80, 132, 133] or rules implementing specific social conventions [77]; in the same framework, extensions were proposed for addressing more complex interaction scenarios like a joint working space [85], for ensuring that robots do not obstruct human's view of the environment [122], for navigating among moving humans [128, 140] and for taking into account the direction the human is facing [121].

A related line of research is concerned with learning-based prediction of human behavior [82, 114, 144], for avoiding unwanted interference with human activity [160] and for navigating shared spaces [47, 148], where, if crowded, the robot benefit from learning to exploit cooperation with people [81, 143].

1.3 Model

1.3.1 Problem formulation

The environment is mapped as a multilayered graph of cells $G = (N, E)$ and is populated by a set of agents X. Each cell in N represents a polygon and contains information about how suitable is for an agent to traverse it.

We want to answer path planning queries: given an agent's initial and target positions, return the best paths, where a path is defined as a list of cells to cross, in a given layer, to reach the target.

First, we describe the static map and how we process it to derive a topological navigation graph, whose edges will carry multiple traversability costs, which lead to the definition of a multi-objective path planning problem. Finally, we propose a simple algorithm to solve the optimization problem.

We plan for one agent at a time and only based on the current state of the environment; multi-agent coordination is partially accounted by spatial costs that depends on the movement of other agents.

1.3.2 IndoorGML

IndoorGML is a recent standard to describe indoor spaces and indoor navigation of heterogeneous agents [89]. An IndoorGML map is a graph $G = (N, E)$ of polygonal cells $c \in N$. The graph is partitioned in one or more non intersecting layers $G_i = (N_i, L_i)$, $i = 1, \ldots, n$. Cells belonging to the same layer do not overlap and edges between them represent (possibly traversable) shared boundaries. Edges between cells in different layers represent instead more general topological spatial relations, like *overlapping* or *contained in*, defined by DE-9IM (Dimensionally Extended nine-Intersection Model [29]).

Two IndoorGML schemas have been published. The core schema [109] defines the topology of the maps, while the navigation schema [110] add labels to cells useful for navigation, starting from the distinction between *traversable* and *non-traversable* cells; traversable cells are subdivided into general spaces (e.g., rooms), transition spaces (e.g., corridors and stairs) and connection spaces (e.g., doors), and carry fine grained attributes to specify their intended usage and function.

1.3.3 Microscopic attributes

The data structure defined by IndoorGML is very general but limited to geometry and minimal semantic for navigation, which is not sufficient for the richer description of indoor spaces we are interested in. In particular, we expect that cells contain any information that could be useful to estimate how well an agent would traverse them. This spatial information originates from various sources, like: *a*) ambient sensors deployed in the environment (ceiling cameras, tracking systems, thermometers, light sensors, ...); *b*) sensors mounted on robots (radios, cameras, lasers scanners, IMUs, wheel-encoders, ...), which for example can be used to measure wheel slippage; *c*) people traversing a space or passing nearby, who provide more subjective and fuzzy — yet very valuable — observations than sensors (e.g., "this corridor is a bit slippery"); *d*) floor maps, which not only contain the precise geometry of buildings but also label spaces according to their designed usage and characteristics ("this cell is a *corridor*", "this room is *private*", ...).

Therefore, in addition to geometry, we assume that the map contains a description of the environment in terms of spatial attributes $a \in A$ assigned to pairs of cells and agents. Some attributes, like *luminosity*, may not depend on the agent visiting that space. Other attributes, like *familiarity*, make only sense in relation to an agent. The values of all attributes give us a *microscopic* description of the environment with respect to an agent that is moving through it.

In general, the mappings $a \in A$ are partial functions $N \times X \twoheadrightarrow V_a$. They are defined only for subsets of cells, typically contained in one layer; for example, a map may store in one layer any information collected by temperature sensors spread in the environment. The same portion of physical space may be contained in multiple cells belonging different layers. Hence we need a way to retrieve attributes from overlapping cells.

Therefore we define an inheritance scheme in Algorithm 1 to let cells inherit attribute values defined in other layers through the topological relations encoded in IndoorGML inter-layer edges: a cell inherits missing attribute values from enclosing cells. More precisely, when the cell is enclosed in *multiple* cells from *different* layers, the cell inherits the attribute value from the *smallest enclosing* cell that has a defined value. Attributes' values are generally associated to pairs of cells and agents but we allow that, for some attributes and cells, no value may be associated to a particular agent; in this case, the value is inherited from the value associated to the cell alone (see Algorithm 1).

```
AttributeValue($a$, $c$, $x$)
if $c$ has value $v$ for $a$ with respect to $x$ then
    return $v$
end
forall cells $c' \in N$ that contain $c$, ordered by size do
    if $c'$ has value $v$ for $a$ with respect to $x$ then
        return $v$
    end
end
if $x$ is not undefined then
    return AttributeValue ($a$, $c$, undefined)
end
return undefined;
```

Algorithm 1: Computation of the value of attribute $a \in A$ of cell $c \in N$ with respect to agent $x \in X$, which may be left undefined.

1.3.4 Navigation layer

Map layers are characterized by the different attributes associated with their cells. In the following, we assume that *geometric layer* N_1 contains all geometric information about the indoor space, labelled according to the indoorGML navigation schema [110], which discriminates between navigable (e.g., a corridor) and not navigable cells (e.g., a wall). Other layers may contain information about sensors or semantic information specific to an agent, as illustrated in the example in Figure 1.1.

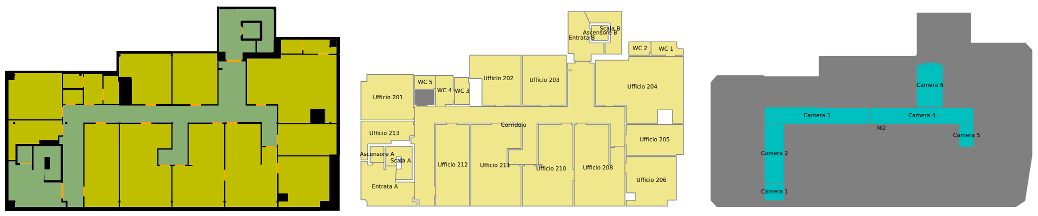


Figure 1.1. An example of an IndoorGML map. Left, *geometric layer*: cells are colored by their IndoorGML type — walls in black, transition spaces (corridors and stairs) in green, general spaces (rooms) in olive, and connection spaces (doors) in orange. Center, *semantic layer*: cells that carry labels (here rooms' names) are colored in yellow. Right, *ambient camera layer*: cells describe the range of view (blue) of an array of ambient monitoring cameras; grey cells are not visible by any camera.

From N_1, we derive, according to Algorithm 2, an (agent specific) *navigation layer* N_{n+1} composed only of *easily navigable* cells, i.e., cells that can be crossed by moving along a straight line between boundaries. We also require that cells in N_{n+1} have a well-defined value for each microscopic spatial attribute a according to the inheritance chain defined by Algorithm 1. These constraints are satisfied if the navigation layer consists of convex cells that cover the whole configuration space for the agent and that are covered by one cell from every other layer.

We start by computing the non-navigable area O as the union of walls and other obstacles in N_1, inflated by a radius r given by the agent's size plus a safety margin. Then, we add to the navigation layer any navigable cell in N_1, after removing its intersection with O. If the resulting cell is not contained in a single cell for each layer $N_2, \dots, N_n$, we split the cell accordingly. Finally, we decompose each cell into (quasi-) convex cells as described in the next section.

```
NavigationLayer($N_1,\dots,N_n$, $r$, $x$)
$N_{n+1} \leftarrow \emptyset$
$O \leftarrow \bigcup\{c \in N_1 \mid c$ is a wall$\}$
$O \leftarrow O$ inflated by $r$
forall $c \in N_1$ do
    if $c$ is navigable by $x$ then
        $c \leftarrow c \setminus O$
        $M \leftarrow$ Split($c$, $N_2, \dots, N_n$)
        forall $c$ in $M$ do
            $N_{n+1} \leftarrow N_{n+1} \cup$ Decompose($c$)
        end
    end
end
return $N_{n+1}$
```

Algorithm 2: Computation of the set N_{n+1} of cells of the navigation layer for agent x with radius r, from map layers $N_1, \dots, N_n$, where N_1 is the geometric layer. Function `Decompose` is covered by Algorithm 3. Function `Split(`c, N_2, ..., N_n`)` returns a partitioning of c into a set of cells $\{\bar{c}\}$, such that, for any cell $\bar{c}$, there exist cells in every layer $N_2, \dots, N_n$ that contain $\bar{c}$.

The navigation layer's graph $G_{n+1} = (N_{n+1}, E_{n+1})$ is built by adding, for each pair of adjacent navigable cells, an edge corresponding to the common boundary. The resulting navigation layer represents the configuration space for an agent as a *topological navigation graph* of cells, where routes are specified as a list of cells' boundaries to cross.

An example of automatic derivation of the navigation layer for a real building floor plan is illustrated in Figure 1.2.

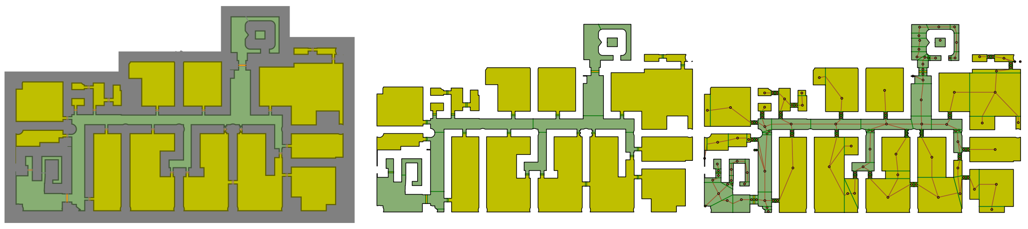


Figure 1.2. The navigation layer is automatically derived from the geometric layer as described in Section 1.3.4. *Left*: walls (gray) are inflated and subtracted from the navigable cells. *Center*: cells are subdivided according to all other layers in Figure 1.1. *Right*: cells are further partitioned into convex cells and the topological navigation graph G_{n+1} is computed; nodes are drawn as a brown dots at the center of the cell; edges are drawn as polylines (brown) from one cell center to the other, passing by the center of the common boundary.

Convex decomposition

We introduce a (quasi-) convex partitioning [27, 87] algorithm that simplifies path planning by generating small navigation graphs with portions where trajectories are very constrained (see Chapter 2).

Some convex partitioning schemes are more suitable than others to compute routes: *a*) small violations of convexity, like those caused by small irregularities along the boundary, won't make navigation across a cell significantly more difficult; *b*) the computational cost of path planning is greatly reduced by having few large cells; *c*) to split a (non convex) polygon, we prefer to cut along short diagonals because they add constraints that simplify the task of geometric planners (see Chapter 2).

Therefore, to keep the number of cells small, we relax the convexity constraint: we accept *quasi-convex* cells when the difference between a cell and its convex hull is small (according to criteria such as a small relative and absolute difference of area, *and* a small Hausdorff distance between boundaries). The additional safety margin, used for inflating O, guarantees that the robots' trajectories will remain collision free.

The cell decomposition Algorithm 3 iteratively subdivides non convex cells into quasi-convex cells by cutting along the *shortest* diagonal that removes a non regular vertex. An example of the decomposition of a cell is depicted in Figure 1.3. The algorithm uses brute force to select the next diagonal to cut, which leads to a quadratic complexity in the number of cell's edges. Nonetheless, it decomposes floor maps of large building, like the one depicted in Figure 1.10, in less than a minute.

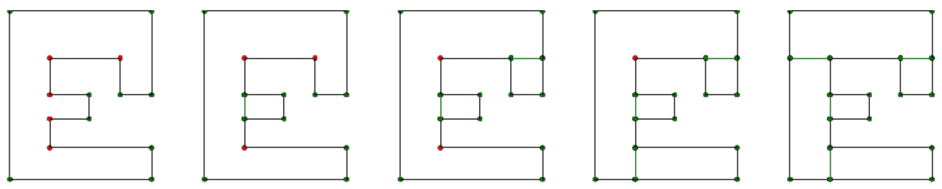


Figure 1.3. From left to right, progressive segmentation of a non convex cell into a set of convex cells by Algorithm 3. Red vertices are candidate vertices for which the shortest diagonals are computed and compared. Cells are cut along diagonals in ascending order of length.

1.3.5 Macroscopic attributes

Some spatial attributes (more precisely, their impact on traversability) may not be directly comparable. For example, which alternative solutions should we choose between taking the stairs (i.e., increasing our effort) or taking the elevator (i.e., increasing the management cost and possibly hindering other people movements)? We argue that forcing a comparison, e.g., by estimating how much we would be willing to pay to avoid climbing the stairs, would constrain the model too much and be difficult to assess objectively.

Instead of comparing all attributes together, we prefer to isolate some macroscopic attributes m whose value is derived from (possibly overlapping) subsets of microscopic attributes $A_m \subseteq A$. For any microscopic attribute in A_m, there should be a straightforward way to compare their effect on m. For examples given the choices of climbing a stair or taking the elevator, *with respect to the effort required*, we can score the two choices. This way we identify *effort* as a possible macro-attribute.

We formulate a generic description of traversability in terms of macroscopic costs that depend on the agent's characteristics, microscopic attributes, and topological/geometric features. In particular, *three macroscopic costs*, namely *effort*, *safety*, and *social cost*, model well many planning instances; this holds both for scenarios involving only robots, and in scenarios involving robots and humans.

```
Decompose($c$)
if $c$ is quasi-convex then
    return $\{c\}$
diagonal ← BestDiagonal($c$)
$\{c_1, c_2\}$ ← SplitCellAlongDiagonal($c$, diagonal)
return Decompose($c_1$) ∪ Decompose($c_2$)


BestDiagonal($c$)
diagonals ← ∅
vertices ← NonRegularVertices($c$)
for v ∈ vertices do
    d ← shortest line segment from v to any edge of $c$ that is separated
      by at least 2 edges from v
    add d to diagonals
end
diagonal ← the shortest among diagonals
return diagonal
```

Algorithm 3: Quasi convex decomposition of c by iteratively splitting along the shortest diagonal.

The actual interpretation of a cost depends on the particular instance, of which we provide examples in Section 1.5.

The model also describes a variety of path planning problems for *humans* in social spaces. For instance, we could model large shopping malls where consumers are attracted by different goods and shopping areas: then, we could compute routes that take into account the users preferences, the accessibility of the route and the current occupancy of the stores.

The cost of a path

Before introducing three concrete macroscopic costs, we discuss how a macroscopic cost $\mathscr{C}(\gamma, x)$, paid by an agent x for traveling along a trajectory γ, arises from the interaction between the geometry of the trajectory and the (dynamic) state, with respect to agent x, of the cells $(c_1, \ldots, c_n)$ that it traverses, which is given by the collection of micro-attribute values introduced in Section 1.3.3. The effect of micro-attributes on traversability may depend on the agent; for instance, having to travel a long uphill path may be perfectly fine for a powerful robot, but less so for a weaker robot running low on batteries.

We introduce three basic assumptions.

Additivity We assume that the cost is additive, i.e., when γ is subdivided into segments γ_i (one for each traversed cell c_i) the total cost is the sum of the cost of the segments:

$$\mathscr{C}(\gamma, x) = \sum_{i=1}^{n} \mathscr{C}(\gamma_i, x). \tag{1.1}$$

Compact segment description We further assume that, with respect to the cost, the geometry of each segment can be completely described by a small *discrete set of features* $\boldsymbol{\omega}(\gamma) \in \mathbb{R}^k$

$$\mathscr{C}(\gamma_i, x) = \mathscr{C}(\boldsymbol{\omega}(\gamma_i), x). \tag{1.2}$$

Decoupling between cell and segment Finally, we assume that the contribution of the (static) segment's geometry decouples from the contribution of the dynamic spatial information and the agent's characteristics as

$$\mathscr{C}(\gamma_i, x) = \sum_{j=1}^{k} \omega_j(\gamma_i) \rho_j^{\mathscr{C}}(c_i, x), \tag{1.3}$$

where $\rho_j^{\mathscr{C}} : N \times X \to \mathbb{R}^+$ associates a cost density, with respect to the j-th geometric feature, to an agent traversing a cell.

For example, the simplest description of a segment could be given by its length, which drivers use to compute travelling costs: sum up equipment and gas to estimate $\rho_{\text{length}}(\text{road})$ — the cost per Km which depends on road conditions — and then multiply by length $\omega_{\text{length}}(\gamma)$.

The model is more general than this example, as it allows for a richer description of the trajectory geometry. For instance, in a robotic scenario, it is relevant how much the trajectory *turns* in addition to its length or elevation changes: *turning* — the integral of the absolute angular changes along the segment — provides clues about the difficulties that an agent could suffer when following the segment, such as slipping, or losing orientation. Other features are illustrated on a concrete application in Section 1.5.1.

From micro- to macro-attributes In Equation (1.3), the contribution $\boldsymbol{\rho}(c,x)$ of the segment geometry $\boldsymbol{\omega}$ to the path cost depends on the interaction between the agent and the environment. For instance, let us consider the very simple case of a flat corridor and ω_{length}. We start modelling it by fixing a cost per unit length that is the same for all agents and parts of the corridor. Now imaging that, in the middle of the corridor, the floor is a bit slippery (as observed and reported to the system by users passing nearby), the illumination is a bit too low or that there are some steps; the cost (for instance the energy expenditure) would increase depending on the agent's abilities to overcome these problems; for instance a robot with larger, high-friction wheels, and good localization sensors, would be less penalized.

We assume that the macroscopic cost density is a summary of the microscopic description given by attributes $a \in A$ that is given by a weighted sum

$$\rho_j^{\mathscr{C}}(c,x) = \sum_{a \in A} R_{j,a}(x)\sigma_a(a(c,x)), \tag{1.4}$$

where functions $\sigma_a : V_a \to [0,1]$ convert attributes' values to costs: $\sigma_a(\cdot) = 0$ represents the optimal value of attribute a with respect to the navigation task while $\sigma_a(\cdot) = 1$ the worst value (highest cost). The factors $R_{j,a} \in \mathbb{R}^+$ weights the impact of problematic a on the navigation cost depending on the agent's characteristics.

For example, high floor slipperiness, low luminosity and very large steps (with respect to the agent capability) would all be attribute value that map to a maximal (cost) contribution of 1; yet a robot with a laser scanner would not be hindered when passing through a dark area (i.e, $R_{\cdot,\text{luminosity}}(x) = 0$). We give a more complete example of attributes and their contribution to a cost in Section 1.5.1.

Effort

Travelling along a path requires a certain effort $\mathscr{E}$ that increases when the path crosses spaces with unfavorable conditions (e.g., a soft carpet on which wheels don't roll well), when the path turns a lot, climbs or descends steps, ...

In the simplest scenario (like moving along a straight flat corridor) the (minimal) energy expenditure along a path is given by a term proportional to the distance. Namely, the basic cost $\mathscr{E}_0$ of movement per unit length is given by the mean force $F(x)$ exerted by agent x

$$\mathscr{E}_0(\gamma, x) = \text{length}(\gamma) F(x), \tag{1.5}$$

which identify $F(x)$ as a lower bound for the cost density $\rho^{\mathscr{E}}_{\text{length}}(\cdot, x)$ in Equation (1.3). Several factors increase the required force, such as, moving on a carpet for a wheeled robot.

Our model also takes into account other kind of efforts, and related sources, such as increased computational time, or increased psycho-physical discomfort caused to a user sitting on top of a robotic wheelchair, for which cost densities with respect to different geometric features may contribute. For example, too much turning would cause an increased effort for a user sitting on a wheelchair, which translates to $\rho^{\mathscr{E}}_{\text{turning}}(c, x) > 0$.

These cost densities, and the related Equations (1.3), are estimated by comparison with the basic effort $\mathscr{E}_0$. For example, to estimate the cost density of c with respect to *turning*, we equate it with the additional length l_c that the agent would prefer to travel in order to avoid turning by a unit, i.e,

$$\rho^{\mathscr{E}}_{\text{turning}}(c, x) = l_c F(x).$$

This example clarifies the basic requirement for macroscopic costs (such as effort): the effects of micro-attributes on a macroscopic cost should be *comparable*. In this case, we could objectively define effort as energy expenditure and we could measure the effect of different spatial attributes on it.

Safety risk

An agent that travels along a path could incur into safety risks $\mathscr{R}$ for many reasons; for instance a mobile robot could capsize, break a motor, or hit a sensor.

We introduce basic concepts from survival analysis [125] in order to define the safety risks associated to traveling along a path. The survival $S(s)$ defines the probability that the agent will avoid any accident up to distance s. The *hazard* $h(s) \geq 0$ defines the probability of incurring in an accident in the next unit length, after having safely traveled to s (i.e., the *accident rate*) as

$$h(s) = -\frac{S'(s)}{S(s)} = -\frac{\mathrm{d}}{\mathrm{d}s}\log(S(s)). \tag{1.6}$$

If the hazard does not depend on the distance (i.e., if we are not including fatigue into the analysis), then the probability of *not having* an accident along a path of length l is given by

$$S(l) = e^{-hl}, \tag{1.7}$$

where the term hl is called *risk*, coherently with the more general definition of risk as the product of hazard and exposure. In our context, the exposure corresponds to the travelled distance (where at each step an accident may occur), while hazard is expressed as risk per unit of length.

The definitions of risk and hazard matches with the general form of Equation (1.2), which we use to identify hazards with cost densities (with respect to length) linked to safety risks

$$\rho^{\mathcal{R}}_{\text{length}}(c, x) = h(c, x). \tag{1.8}$$

When there are multiple threats, their hazards can be summed if we assume that they are (statistically) independent. This justifies the choice of safety risk as a macroscopic cost that conforms with equation (1.4): we sum hazards due to threats represented by different attributes. All hazards are quantified as number of accidents per unit length and per agent. For example, if a floor is slippery and illumination is low, we can sum the related hazards of falling down due to one of the two problems.

With similar arguments, we could introduce hazards related to exposures identified by other geometric features ω_j. For example, for a robotic wheelchair, we could consider the risk of losing localization because of too much turning, i.e., $\rho^{\mathcal{R}}_{\text{turning}}(c, x) > 0$.

Social impact

Agents traveling along a path may share the space with other agents (people, robots or vehicles). If we follow economic reasoning, space is a limited resource and could be assigned a (monetary) cost; for example, a person or robot taking an elevator forces others to wait or to use the stairs; moreover, a space may be reserved to a group of agents, i.e., we may associate privacy and accessibility rules, used for managing a building, with similar costs.

In the following, we assume that accessibility to a space has a monetary cost $\mathscr{S}$ that can be purchased and depends on the time that an agent will spend in that space. For simplicity, we also assume that agents move at constant speeds, which lead to a cost that is proportional to the length of the path crossing cell c

$$\mathscr{S}(\gamma_c) = \text{length}(\gamma_c)\rho^{\mathscr{S}}_{\text{length}}(c, x), \tag{1.9}$$

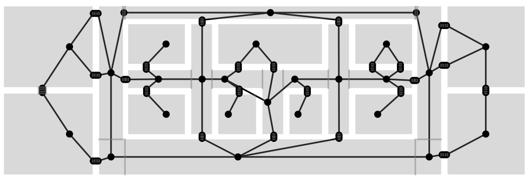
Comparing with the general form of Equation 1.3, we identify $\rho^{\mathscr{S}}_{\text{length}}(c)$ with the penalty (per unit length) for occupying cell c, which we measure in \$ per unit length.

Contrary to other costs, $\mathscr{S}$ seems to just be proportional to the length of the segment. Yet there are examples, such as when passing a toll bridge, for which a fixed cost is paid, independently on the actual path taken inside the cell.

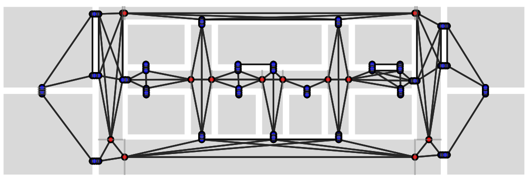
1.3.6 Navigation line graph

We have already discussed how when a segment crosses a single cell, the contribution to the cost from geometry ($\boldsymbol{\omega}$) and environment state ($\boldsymbol{\rho}$) are decoupled in Equation (1.3). We further simplify the planning problem by considering only paths composed of a *finite number of linear segments*, each segment approximating the optimal trajectory for crossing a cell from one boundary to the other.

This implies that the planning graph — whose nodes N_L corresponds to cell boundaries and edges to linear segments between them crossing a *single cell* — is the *line graph* $L = (N_L, E_L)$ of the topological navigation graph $G_{n+1} = (N_{n+1}, E_{n+1} \equiv N_L)$ (see Figure 1.4b).



(a) Navigation graph: cell boundaries are linked to edges in E_{n+1}.



(b) Navigation *line* graph: the same cell boundaries are now linked to nodes in N_L.

Figure 1.4. The navigation graphs.

In fact, a linear segment that crosses a (convex) cell from boundary center to boundary center is reasonably close to the optimal trajectory. Although the optimal trajectory generally depends on the characteristics of the agents (e.g., pedestrians prefer to follow more direct paths than users on robotics wheelchairs) and depends on the whole path, we would like to estimate the geometric features $\boldsymbol{\omega}$ of the optimal trajectory segment from the linear segment.

We can assume that the length of the linear segment is a good proxy for the length of the optimal segment. On the contrary, when estimating *turning*, we cannot simply assume that the optimal trajectory will pass though the centers of the boundaries as these zigzag trajectories have a much larger *turning*. Instead, as illustrated in Figure 1.5, we estimate *turning* by computing the minimal turning of a segment whose vertices are free to slide along the cell boundaries, while maintaining the assumption that the (optimal) path will enter and leave the cell perpendicularly to each boundary. Figure 1.6 shows that this approach leads to accurate estimations of the *length* and *turning* of (optimal) trajectories.

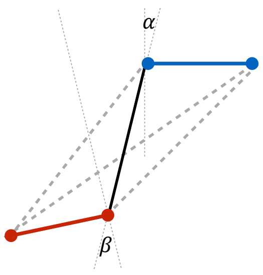


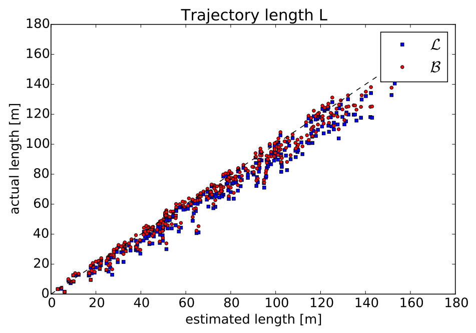
Figure 1.5. Computation of *turning* for an edge in E_L that connects the red with the blue boundary, assuming that the trajectory enters (red) or leaves (blue) perpendicularly through the boundary (i.e. needs to rotate by β at the red boundary and by α at the blue boundary): turning $= \min\{\alpha+\beta\}$, where the minimization (solid line) is over all linear segments connecting the two boundaries (among which there are the dashed lines).

The line graph topology and the geometric description of its edges are static and need therefore to be computed only once. Figure 1.7 displays the values of *length* and *turning* for all edges on a navigation line graph of a synthetic map.

1.3.7 Multi-objective optimization

In this section, given a navigation line graph whose edges carry well defined, non-comparable, positive, macroscopic costs, the path planning problem is formulated as a multi-objective optimization.

In the worst case, the Pareto set contains all simple paths between source and target (an exponential number with respect to the size of the graph). We



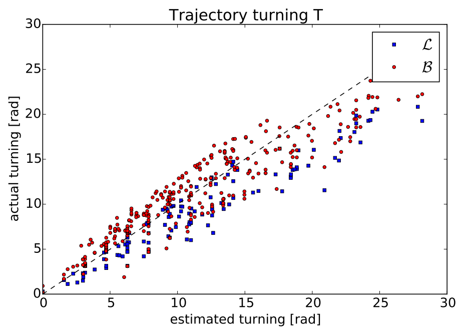
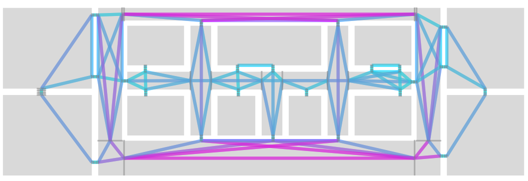
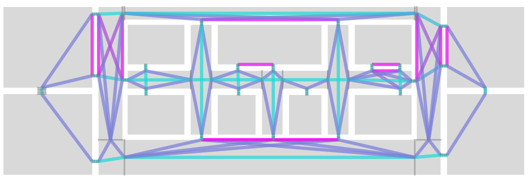


Figure 1.6. Estimation of the geometric features of the optimal trajectory from the navigation line graph L for 300 random trajectories in our laboratory (see Figure 1.10). Blue corresponds to optimal *pedestrian* trajectories, red to optimal *wheelchair* trajectories computed in Chapter 2. Perfect estimations lie on the dashed lines.



(a) Length



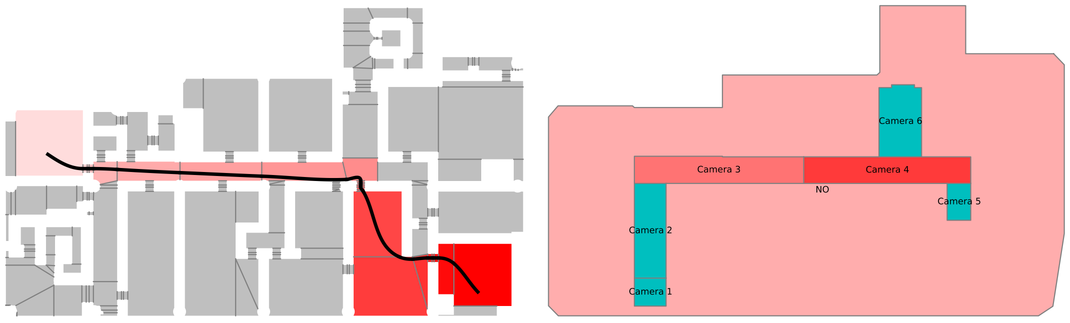
(b) Turning

Figure 1.7. Part of the geometric description of the edges of the line graph L; colder colors (blue) encode lower values.

argue that, for indoor spaces, the average complexity is significantly lower and we present a simple algorithm to compute all solutions in a short enough time. We explore computational costs on a real building map in Section 1.4.

Path planning instance We are given: an agent, its navigation line graph $L = (N_L, E_L)$, n positive cost functions $\mathscr{C}_{1\leq i\leq n}(e,x) \in \mathbb{R}^+$, source s and a target locations t. We complete L with any edge from s to neighboring cell boundaries, and from all neighbor cell boundaries to t.

The planning problem looks for the best set of paths $\pi \in \Pi$ on L between source and target according to costs $\mathscr{C}_{1\leq i\leq n}$. Figure 1.8 illustrates an example of a path given by a sequence of cells and cells boundaries to be crossed when travelling from s to t.



(a) A topological path is computed on the navigation layer from the lighter to the darker cells. The geometric information about the cells can be used to compute an smooth trajectory (in black), see Chapter 2.

(b) The same path, projected onto other map layers, provides a rich description. For example, the projection on the *smart camera layer* informs that the agent will be partially tracked by cameras 3 and 4.

Figure 1.8. A topological path π on a multi-layered map.

Optimal solutions In case of a single cost ($n = 1$), the best possible path is the one that minimizes the total cost in Equation (1.1). We can compute it in polynomial time with respect to the size of L using one of several algorithms for shortest paths on a graph, like Dijkstra's algorithm.

Instead, when multiple costs need to be minimized, there are *multiple* sensible ways to define and interpret a solution (see [99] for a complete survey).

Lexicographic order We can define a total ordering between the costs; then, we choose the path minimizing the first cost; if there are many such paths, we choose the one minimizing the second cost, and so on.

Costs as constraints We could interpret a subset of costs as constraints. That is, we could define thresholds C_j and impose $\mathscr{C}_j(\pi) \leq C_j$ for some j's and solve for the best path with respect to the remaining costs, using one of the other models presented in this list.

Combination of costs The most intuitive and simplest solution is to combine all costs in a single cost, for example using a weighted sum.

Pareto front We may define the best set of paths as the Pareto front of the multi-objective minimization problem, i.e. the set of all non dominated solutions. A dominated solution is a path for which there exists at least another path that is at better (or equal) to it with respect to *every* cost [20].

In general, to discriminate paths with respect to multiple costs, we need additional criteria that may not be specified at query time. In case of lexical ordering, cost as constraints and combination of costs, the strategic decision is taken before the path is computed by setting the order, the thresholds or the function to combine the costs respectively. On the contrary, the selection of one solution from the Pareto front is delegated (as an *informed strategic decision*) to the agent afterwards.

In the remaining of this chapter, we consider the multi-objective path planning problem associated with the computation of the Pareto front.

Computation of the Pareto front.

Yen's algorithm [156] computes iteratively the k-th shortest simple path on a graph and provides the key component to compute the set of optimal paths with respect to a set of costs, as illustrated by Algorithm 4.

The algorithm computes the Pareto front by iteratively applying Yen's algorithm (with respect to switching costs) and adding any non dominated solution until one solution π has been computed by Yen's algorithm with respect to *all* costs. In fact, any further application of Yen's algorithm would compute a path with higher costs than π. Figure 1.9 illustrates this search in cost space in the case of $n = 2$.

Complexity analysis The time complexity of Yen's algorithm depends on the shortest path algorithm that it uses. For example, for using Dijkstra's algorithm with Fibonacci heap, k applications of Yen's algorithm have a worst-case time complexity $O\left(k|N_L|(|E_L| + \log|N_L|)\right)$, which implies a worst-case time complexity $O\left(k\left(|\Pi| + |N_L|(|E_L| + |N_L|\log|N_L|)\right)\right)$ for Algorithm 4, where k are the number

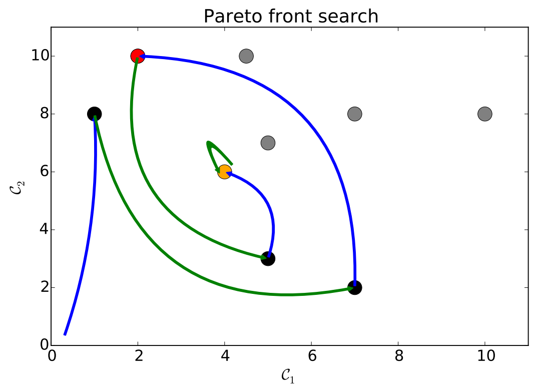


Figure 1.9. An illustrative example of the computation of the Pareto front Π by Algorithm 4. Each point represents the cost $(\mathscr{C}_1, \mathscr{C}_2)$ of a solution in the two dimensional cost space. The best solution with respect to cost $\mathscr{C}_1$ (left most point) is computed first; then the best solution with respect to cost $\mathscr{C}_2$ (bottom most point), and so on. We keep adding solutions (black dots) to the set Π, provided they are not dominated by one of the current members of Π (which is not the case for the red solution), following blue or green arrows, which correspond to the computation of the next shortest path using Yen's algorithm with respect to $\mathscr{C}_1$ and $\mathscr{C}_2$ respectively. We stop as soon as we find a solution (orange) that has been visited by arrows of all colors. The search is terminated before most gray solutions (which are all dominated) are even computed.

```
OptimalSolutions($G, s, t, \mathscr{C}_1, \ldots, \mathscr{C}_n$)
$\Pi \leftarrow \emptyset$
**forall** $i \in \{1, \ldots, n\}$ **do**
    /* Initialize n instances of Yen's algorithm */
    nextShortestPath$_i$ ← Initialize Yen's generator of shortest paths
      between $s$ and $t$ on $G$ with respect to cost $\mathscr{C}_i$
**end**
$i \leftarrow 1$
**do**
    $\pi \leftarrow$ nextShortestPath$_i$()
    **if** $\pi \in \Pi$ **then**
        add $\pi$ to $\Pi_i$
    **end**
    **else**
        **if** *not* dominated($\pi, \Pi_i, \mathscr{C}_1, \ldots, \mathscr{C}_n$) **then**
            add $\pi$ to $\Pi$
            add $\pi$ to $\Pi_i$
        **end**
    **end**
    $i \leftarrow i + 1 \mod n$
**while** $\pi \notin \Pi_i$ *for some* $i \in \{1, \ldots, n\}$
**return** $\Pi$


dominated($\pi, \Pi, \mathscr{C}_1, \ldots, \mathscr{C}_n$)
**forall** $\pi' \in \Pi$ **do**
    **if** $\mathscr{C}_i(\pi') < \mathscr{C}_i(\pi) \, \forall i = 1, \ldots, n$ **then**
        **return** true
    **end**
**end**
**return** false
```

Algorithm 4: Construction of the Pareto front of all paths between s and t on G by repeated application of Yen's algorithm with respect to costs $\mathscr{C}_1, \ldots, \mathscr{C}_n$.

of iterations that the algorithm needs to visit the whole Pareto front. In the worst case, the Pareto front may contain all simple paths from s to t, but in typical indoor maps and cost distributions, the size is usually very limited (see Section 1.4).

Approximations If an approximation of the Pareto front is sufficient, we can keep the query time limited by stopping Algorithm 4 after a maximal number of iterations. We may also prefer to compute solutions with *significantly* different costs to reduce the number of strategic choices. In this case, we relax the criteria on dominance to avoid adding *almost-dominated* solutions [42] in Algorithm 4.

1.4 Experiments

1.4.1 Experimental setup

We test the influence of the number of costs and their spatial distribution on the multi-objective path planning problem formalized in Section 1.3.7 and solved by Algorithm 4. We limit our analysis to a single map but we produce many different planning instances by randomly selecting source, target and cell costs.

Graph We use the (connected) navigation line graph $L = (N_L, E_L)$ depicted in Figure 1.10 that was automatically derived using Algorithm 2 from the floor map of our building, which has a size of about 120 m×70 m; $|N_L| = 1638$, $|E_L| = 2310$.

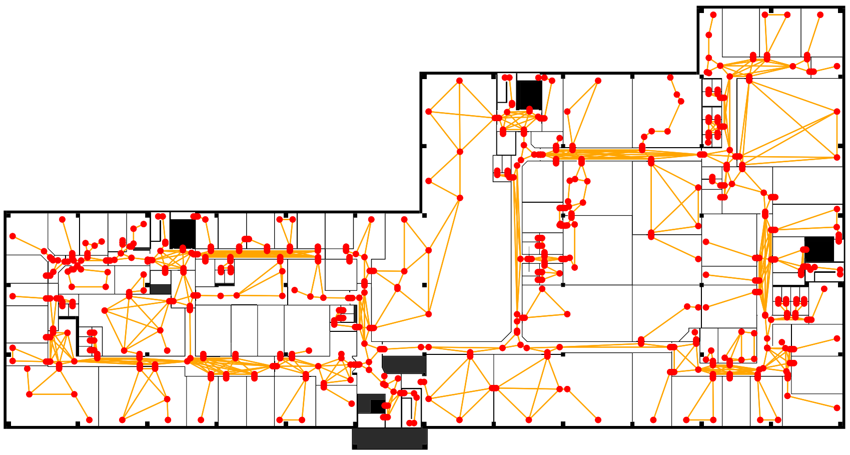


Figure 1.10. Map and navigation line graph used to test the impact of the number and the spatial distribution of costs on the planning problem.

Source and target Source s and target t are randomly selected from N_L with uniform probability.

Cell cost densities We limit the analysis to (n different) costs per unit of length $\rho^{\mathscr{C}}_{\text{length}}$. We define two distribution classes, with varying degrees of spatial correlation, that we use to sample cell cost densities: (1) a uniform distribution $\mathscr{M}_u = \mathscr{U}([0,1])$ without any spatial correlation between cell costs; and (2) a distribution $\mathscr{M}_g(k)$ that mimics painting the map with $k \geq 1$ blurred disks of radius $\frac{\rho}{\sqrt{k}}$, with $\rho = 50\,\text{m}$, centered in random location $\boldsymbol{x}_{1\leq j\leq k} \sim \mathscr{U}(A)$ on the map area $A \subset \mathbb{R}^2$. For such a set of disks, the cost of any cell is defined as

$$\rho^{\mathscr{C}}_{\text{length}}(c) = \sum_{j=1}^{k} e^{-\frac{|x_c - x_j|^2}{2k\rho^2}},$$

where $\boldsymbol{x}_c$ is the center of the cell. Large spatial correlations are produced by few, large disks (small k), while many small disks produce maps that are similar to those of $\mathscr{M}_u$ and have almost no spatial correlation. Note that the average area ($\pi\rho^2$) covered by the disks is independent of k. Figure 1.11 contains samples drawn from some of these distributions.

Random cost functions from $\mathscr{M}_u$ serve as an extreme scenario because they create large disparities in the costs of almost identical routes. In reality, many (microscopic) attributes have strong spatial correlations (e.g., luminosity does not usually change abruptly along a corridor) and are better modeled by $\mathscr{M}_g$.

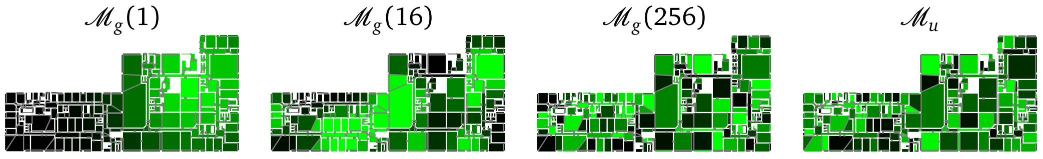


Figure 1.11. From left to right, samples drawn from cost map distributions that have a decreasing spatial correlation: a cell's brightness is proportional to its cost density.

Planning instance A planning instance is given by the tuple $(s, t, \mathscr{C}_1, \ldots \mathscr{C}_n)$. To compute the edge costs $\mathscr{C}_1, \ldots \mathscr{C}_n$, we sample n cost maps, as explained above, which we use to assign n cost densities (per unit length) to each cell; then we apply Equation (1.3) (limited to ω_{length}) to compute n costs of every edge in E_L.

We apply Algorithm 4 to compute the Pareto front Π of all paths between s and t with respect to cost functions $\mathscr{C}_1, \ldots \mathscr{C}_n$. For each instance, we measure the size of Π, the time it took to compute it, and the distance (the length of the shortest path on the graph) from s to t. We expect that distant s-t pairs will be connected by more alternative routes, therefore making the problem harder and the solution set larger.

For each experiment, we draw 10K planning instances and we average the results over all instances.

1.4.2 Results

Varying number of random costs Figure 1.12 illustrates the results when averaging 10K random planning instances. As expected the computation cost and the size of the Pareto set increase when source and target are distant. Not surprisingly, they also increase when more objectives are added. Yet, the average size of the Pareto set remains relatively small (less than 10) and the mean cost is small enough to serve real time queries.

Varying spatial correlation of a fixed number of costs Figure 1.13 reports results from an experiment with 10K planning instances using a fixed number of costs ($n = 3$); this time we vary the spatial distribution from which each cost is drawn. We find that the problem is harder for cost functions with average spatial correlation ($m = 4$, $m = 16$); in contrast, when there are many disks ($m = 256$) and with spatially-uncorrelated costs $\mathscr{M}_u$, the problem is significantly easier. At least on this map, cost functions with small spatial correlation result in simpler planning instances which yield smaller computational costs and smaller Pareto fronts.

1.5 Concrete instances

We now present two concrete applications of the abstract model presented in this chapter.

1.5.1 Optimal topological planning for smart wheelchairs

In environments like nursing homes and hospitals, (partially) automated smart wheelchairs are an important aid for people with restrained mobility. Computing routes for wheelchairs not only requires taking account of the wheelchairs' kinematics to ensure feasibility of routes, but also of the interaction between users and spaces. For example, some users may prefer a longer route that passes by the

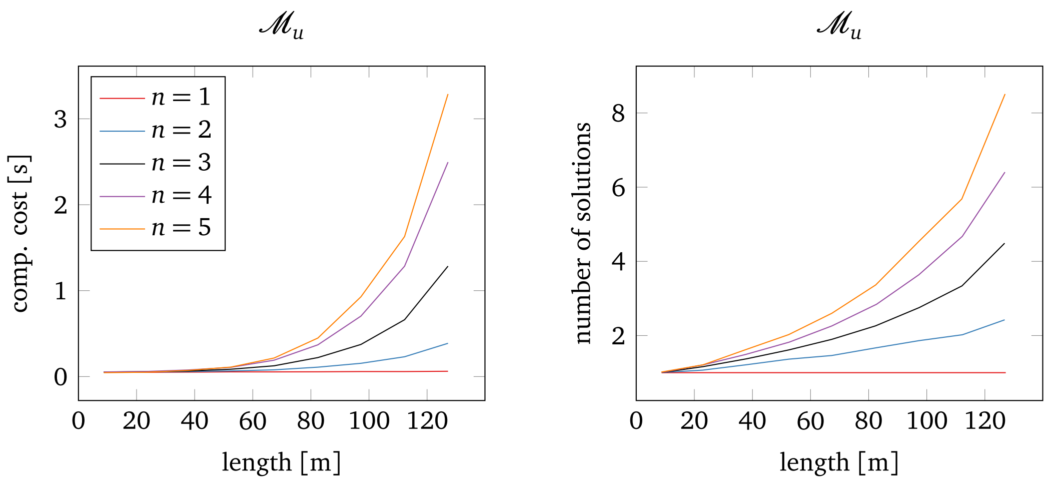


Figure 1.12. Results from 10K randomly drawn multi-objective planning instance with respect to $1 \leq n \leq 5$ costs, drawn using $\mathcal{M}_u$. *Left*: average computation cost. *Right*: average size of the Pareto front.

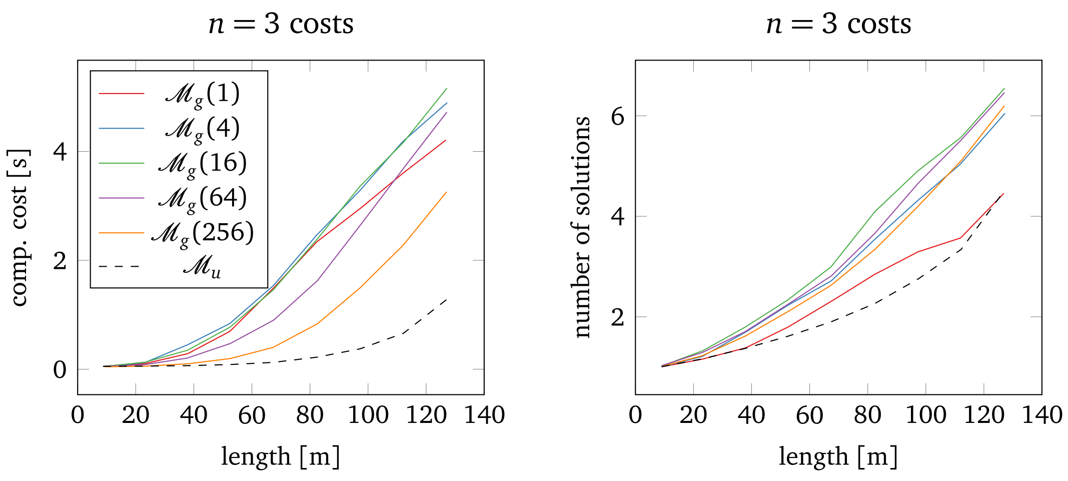


Figure 1.13. Results from 10K randomly drawn multi-objective planning instance with respect to 3 costs, drawn various distributions. *Left*: average computation cost. *Right*: average size of the Pareto front.

garden, others may feel uncomfortable crossing a narrow or dark corridor, and all routes should avoid passing through private rooms unless in an emergency.

The ALMA project

Ageing without Losing Mobility and Autonomy (ALMA) is a multidisciplinary research project [3], funded by the European Union Active and Assisted Living Programme (AAL), to support the autonomous mobility, navigation, and orientation of the (elderly) person with reduced mobility. ALMA specifically focuses on large facilities such as nursing homes or hospitals, where ambient sensors monitor the environment and track the location of people and objects. ALMA addresses scenarios like when an elder person, possibly with orientation difficulties, gets directions from a specialized navigator to reach a specific room; in some scenarios, the user may be equipped by a semi-autonomous motorized wheelchair that automatically takes him to the destination.

Several research groups have developed between 2013 and 2016 a set of inter-dependent modules: *a*) a distributed, radio-based localization module that tracks wheelchairs and users [Politecnico of Milano]; *b*) smart time-of-flight cameras that monitor users and wheelchairs, and detect crowding [VCA Technology, London]; *c*) specialized navigation interfaces for elderly users [Politecnico of Milano and SUPSI-DSAN]; *d*) an interfacing module that provides commercial motorized wheelchairs with (semi) autonomous navigation capabilities (i.e., a robotic wheelchair) [Politecnico of Milano and Degonda Rehab, Lausanne]; *e*) a central server, based on the model presented in this chapter, that stores spatial information and provides path planning services [IDSIA] (see Appendix B for the software implementation).

These modules have been deployed for testing in our labs and in a nursing home in Chiasso, Switzerland (see Figure 1.14); for every location we have derived planning graphs as described in Section 1.3.4 and offered online path planning services

Specific attributes and cost mapping

We list in Table 1.1 specific (micro) attributes to describe the navigation of people and robotic wheelchairs in the scenarios targeted by ALMA. Some attributes are linked to sensor measurements, others are meant to be assigned to spaces by secondary users like, e.g., building managers and nurses.

For ALMA, the geometry of graph E_L, is described by 4 features $\omega_{1\le j\le 4}$: *length*, *turning*, *diversity* (which checks if the edge enters a cell of different de-

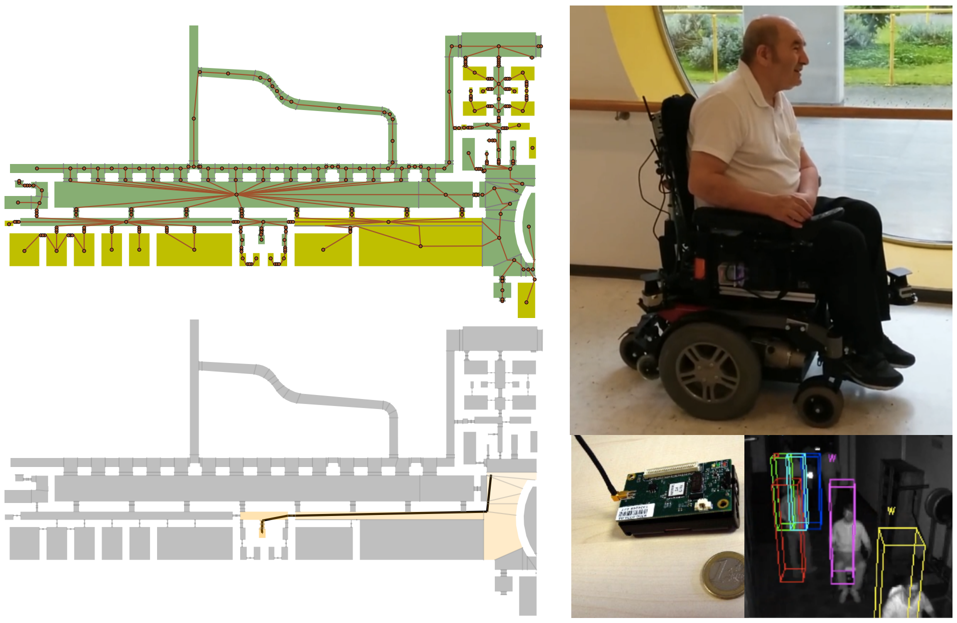
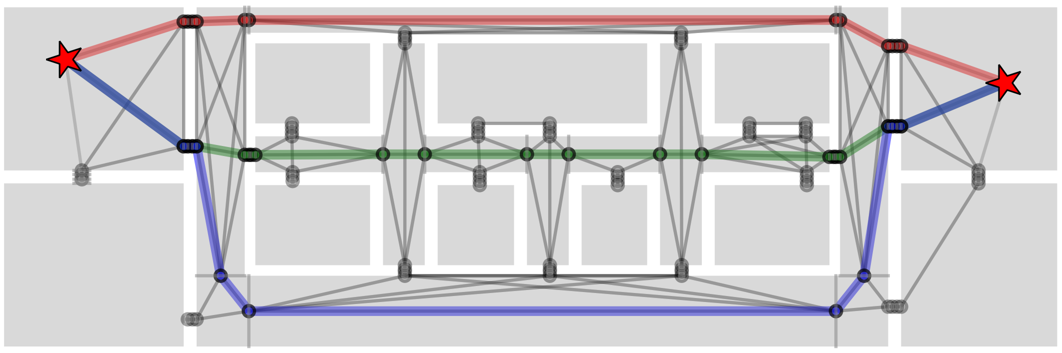
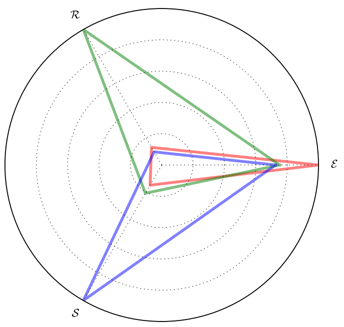
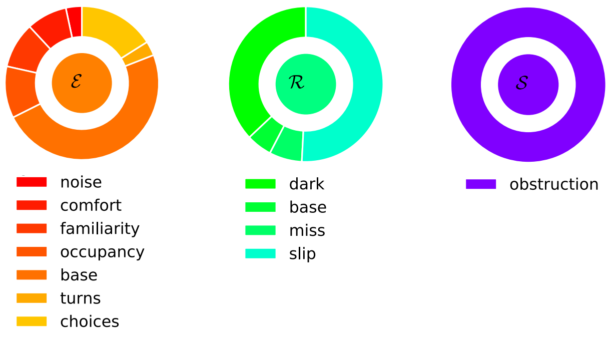


Figure 1.14. ALMA project. *Left*: Part of the navigation graph (top) that covers one floor plan of a nursing home in Chiasso, Switzerland, and one route (bottom) computed during the final tests on user demand. *Right*: Semi-autonomous smart wheelchair, radio-tag used for localizing users, and smart cameras detecting pedestrians and wheelchairs, developed by POLIMI and VCA Technology, are interacts with the mapping and path planning service.



(a) The Pareto set: we draw each optimal solution $\pi \in \Pi$ using a different color.





(b) Left: a radar chart of $\frac{\min_{\bar{\pi}\in\Pi}\mathscr{C}_i(\bar{\pi})}{\mathscr{C}_i(\pi)}$ for the three costs $\mathscr{C}_i \in \{\mathscr{E}, \mathscr{R}, \mathscr{S}\}$, drawn for each optimal solution $\pi \in \Pi$ using the same color as above, helps to describe routes to users: better solutions, with respect to a cost, have vertices toward the outer of the circle (all solutions being not dominated corresponds to triangles not being contained in other triangles). *Right*: the cost breakdown for a single solution (the one colored in red) identifies the main problems along the path.

Figure 1.15. Results from a multi-objective query on a synthetic map, with ALMA attributes and costs, for a path between the two starred locations on the navigation line graph.

attribute		meas.	$\mathscr{E}$	$\mathscr{R}$	$\mathscr{S}$
traversable	Is a cell traversable by the agent? Is a door locked?				
crowding	Smart cameras count people inside their field of view.	●	●		●
accessible	Is the agent allowed to access this cell?				●
obstruction	How much the agent would obstruct other agents?	●			●
noise	How much noise is these?		●		
comfort	Is traversing a cell comfortable?		●		
familiarity	Is the agent familiar with the ambient?		●		
luminous	How luminous is an ambient?			●	
slippery	How slippery is the floor?			●	
steps	Are there stairs or steps?		●	●	
help	Is help available in case of need?			●	

Table 1.1. Some spatial attributes of ALMA maps. Note that they cover navigation of pedestrians as well as of robotic wheelchairs. The value of some attributes depends on the agent, and some are measured or computed automatically (see "meas." column). The last three columns summarize the mapping σ between spatial attributes and macroscopic costs for a user on a robotic wheelchair; a dot means that the attribute influences the cost, i.e., $\sigma \not\equiv 0$.

signed use), *choices* (which counts the number of possible alternative edges, i.e. the order of the incident node); this description covers two cases: assisting the navigation of pedestrian users, and autonomous wheelchair navigation.

The planner provides a rich description of the routes in the Pareto set, such as the breakdown of the costs, to inform the agent about which parts of the environment may be problematic and how much each problem contributes to the macroscopic costs (see Figure 1.15 for an example).

1.5.2 Drones coordination on a flyway

Flyways are (virtual) corridors where airplanes are safe to travel. The Drone-FlyWay (DFW) project, funded by the Swiss Commission for Technology and Innovation (CTI), extends this idea to unmanned vehicles, with limited autonomy, flying in a city (see Figure 1.16:Top).

An industrial partner (Paradox Engineering, Switzerland) has equipped light poles with ultra-wide band radios to localize the drones along streets that act as flyways. The subdivision of the map in cells controlled by one or more smart light poles is well suited to the spatial model that we presented in this chapter. Moreover, we add cell attributes to represents regulations restricting flying above sensible areas, areas with strong wind or in congested air traffic. In fact, a model based on indoorGML is not restricted to indoor spaces; it describes any settings where spaces are naturally partitioned in cells and connectivity is an important information, even when their geometry is three dimensional.

As illustrated in Figure 1.16, we use the spatial representation and the high-level planner introduced in this chapter, to steer a fleet of drones along the flyways: we compute high-level routes and use a simple locking mechanism, which allow only one drone at time to enter one cell, to regulate traffic.

1.6 Conclusions and Perspectives

In this chapter, we presented a spatial modelling and planning approach for robots moving in indoor spaces. Spaces are partitioned into cells characterized by potentially many microscopic attributes; from these micro-attributes we extract three macroscopic attributes – individual effort, social discomfort, and safety hazard – that are well suited to describe a wide range of path planning problems. We show how to compute multi-objective plans, i.e., the best sequences of cells to traverse to reach a target. We test the computation requirements of planning instances on a real building map with randomly assigned costs, where the prob-

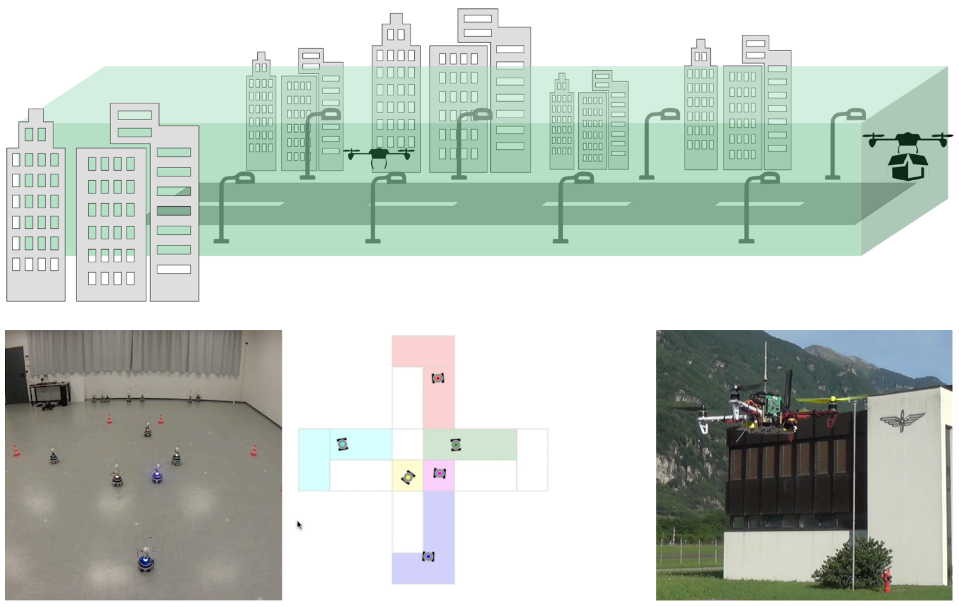


Figure 1.16. Top: A drone flyway (green) controlled by light poles equipped with UWB antennas to localize the drones. *Bottom-left*: coordination strategy for a group of robots (in this case MarXbots moving using the navigation algorithm of Chapter 3) which make robots reserve the next cell on their path; if the cell is not available, the robot request gets queued, as, for example, for the purple robot that is waiting for the green robot to cross the cell. *Bottom-right*: test with real drones have been performed in our lab and in an air base in Lodrino, Switzerland.

lem is small enough to be solved by the proposed algorithm, in particular when costs have small spatial correlations.

We conclude with two applications that validate the approach: planning paths in nursing home for robotic wheelchairs, and controlling drone traffic in a city. In the current implementation, some parameters need to be tuned manually (e.g., in the functions σ in Equation (1.4)), for which we don't have yet a well-tested procedure; moreover, the set of micro-attributes we used may need to be extended for other applications. One potential, larger topic for future research is automatically learning the mapping between micro- and macroscopic attributes from the behavior and feedback of users. In addition, we plan to compare the performance of the planning algorithm with approximations such as genetic programming, which allow to approximate Pareto fronts on much larger graphs.

Chapter 2

Optimal trajectory planning in indoor spaces

2.1 Introduction

In Chapter 1, we computed high-level paths using a rich description of the environment, where each solution consisted of a sequence of cell boundaries to cross. In this section, we study how to cross those cells boundaries. Which continuous geometric trajectory should the agent follow while traveling from a cell boundary to the next?

We expect that, in order to satisfy different types of agents, a planner should compute different geometrical trajectories. For instance, most pedestrians will be happy if we just provide them with the shortest possible path, since they can handle very tight turns without issues; yet, some pedestrians, such as elderly people using a walker, have difficulties when turning and may benefit from a different trajectory.

Planning a geometrical trajectory is definitely important for users sitting on wheelchairs, for whom a shorter length is not the most important feature of a suitable trajectory. Trajectories with small *curvature* are particularly beneficial [101] for wheelchairs for multiple reasons; the wheelchair can limit accelerations [52], increasing the comfort for a user sitting on it; furthermore, for a given linear speed, a trajectory with lower curvature yields a lower angular speed, which improves visual odometry accuracy due to better keypoint tracking and helps to reduce wheel slippage.

It is also important that trajectories look natural, predictable and legible, which are important features when autonomous wheelchairs share the space with people [90]. In particular, when crossing narrow spaces like a door, people try to infer future trajectories of people (and robots) surrounding them. Sometimes it is beneficial to reduce personal utility for an increased legibility. Imagine, for example, that we are standing close to a door while observing a person on a wheelchair traveling along the corridor. If the wheelchair trajectory clearly suggests that it is going to cross the door, we may recognize this and may be able to move away in time. Instead, if the wheelchair, in an attempt to optimize some performance metric, travels parallel to the corridor up to the door, before suddenly turning towards us, we may not be able to anticipate its intentions in time.

The spatial data model introduced in Chapter 1 contains all information required to plan trajectories that are safe, legible, and comfortable. In fact, we use the shape and spatial semantic label (“door”, “corridor”, . . .) of cells to define constraints on the trajectory to increase legibility and ensure safety; we then look for the smoothest (i.e., highest comfort) trajectory that satisfies the constraints. The constraints allow us also to significantly reduce the complexity of the optimization problem.

In case multiple routes are available, we formulate a new multi-objective problem that relies on geometrical information only: which are the best trajectories, between a start and goal pose, with minimal curvature *and* length?

In a hierarchy of controllers, geometrical trajectory planning lies between high-level planning and low-level control, which we further discuss in Chapter 3 where the robot, while following a geometrical trajectory, needs to avoid dynamic obstacles.

2.2 Related Work

Planning for smooth trajectories has been an important topic of research in robotics [111]. Robots controller have an easier task when following a smooth trajectory, which should ideally be G^3 continous [116]. Splines and composite Bézier curves are families of smooth curves that researchers often look at for optimal smooth trajectories, for which the bend energy [36] – i.e, the integrated squared curvature of a trajectory – has provided a natural objective to minimize. Curvature constraints can be added to ensure the feasibility of the trajectory by non-holonomic robots [68]. Recent research has investigated the use of Bézier composite curves for planning trajectories along corridors [28], and across doors [147]; in this chapter, we discuss how to plan an optimal trajectory that may traverse a whole building.

Interestingly, drawing Bézier curves on a screen may also provide a human friendly interface for the remote control of a robotic wheelchair [73].

2.3 Model

We introduce basic notation[1] to describe the two dimensional trajectory followed by a mobile robot, as illustrated in Figure 2.1. We model a differential-driven two wheeled robot as an oriented disk of radius ρ that travels along a trajectory $\gamma : [t_0, t_1] \to \mathbb{R}^2$, parametrized by time. Let $\{\boldsymbol{e}_1(t), \boldsymbol{e}_2(t)\}$ be an orthonormal frame attached to the robot such that $\boldsymbol{e}_1$ points forward (i.e., the Frenet-Serret frame of γ). The robot kinematics is described by linear speed $v(t) = |\dot{\gamma}(t)|$ and angular speed $\omega(t)$, which is related to the curvature κ (by the Frenet-Serret formula) as

$$\omega(t) = v(t)\kappa(t). \tag{2.1}$$

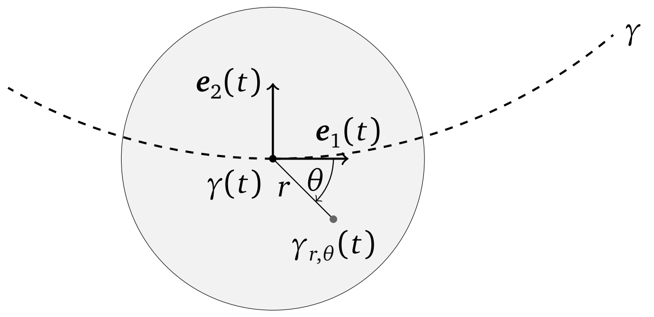


Figure 2.1. The robot trajectory γ and its Frenet-Serret frame $\{\boldsymbol{e}_1, \boldsymbol{e}_2\}$ at time t; $\gamma_{r,\theta}(t)$ is a (polar) parametrization of the robot's disk in the frame $\{\boldsymbol{e}_1, \boldsymbol{e}_2\}$.

2.3.1 The jerk along a trajectory as a cost

The jerk $\dddot{\gamma}(t)$ is a common quantity to quantify the smoothness of a trajectory; in fact, trajectories with low jerk magnitude are perceived as more comfortable by a user sitting in a vehicle [70, 130] and they put less stress on the mechanical parts like motors and gears [84].

We assume that average magnitude of jerk over the disk is a good proxy for the discomfort of a user sitting on top of a robotics wheelchair (in this case ρ would denote the radius of the user), and, more generally, is appropriate in our context to discriminate smooth trajectories.

[1]dots denote derivations with respect to time.

Discomfort should be accumulated over time, therefore we define the following cost, which was first introduced to measure coordination abilities of humans [45]:

$$\mathscr{J}(\gamma) = \int_{t_0}^{t_1} \left\langle \dddot{\gamma}_{r,\theta}(t)^2 \right\rangle_\rho \mathrm{d}t, \tag{2.2}$$

where

$$\gamma_{r,\theta} = \gamma(t) + rR_\theta \boldsymbol{e}_1(t)$$

denotes the position at time t of a point which has fixed polar coordinates (r, θ) in the robot frame, R_θ a rotation by angle θ in the robot frame, and $\langle \cdot \rangle_\rho$ the average over a disk or radius ρ.

We compute Eq. (2.2) by inserting Eq. (2.1):

$$\begin{aligned}\left\langle \dddot{\gamma}_{r,\theta}(t)^2 \right\rangle_\rho &= \left(\ddot{v}(t) - v(t)\omega(t)^2\right)^2 + (2\dot{v}(t)\omega(t) + v(t)\dot{\omega}(t))^2 \\ &+ \frac{\rho^2}{2}\left((3\dot{\omega}(t)\omega(t))^2 + \left(\ddot{\omega}(t) - \omega(t)^3\right)^2\right).\end{aligned}$$

In this chapter, we focus on the geometrical aspects only; therefore we assume that the robot moves at constant speed $v(t) \equiv v \Rightarrow \omega(t) = v\kappa(t)$ and we switch to a parametrization of γ by arc length s, for which

$$\left\langle \dddot{\gamma}_{r,\theta}(s)^2 \right\rangle_\rho = v^6 \left(\kappa(s)^4 + \left(\frac{\partial \kappa(s)}{\partial s}\right)^2 + \frac{\rho^2}{2}\left(\left(3\frac{\partial \kappa(s)}{\partial s}\kappa(s)\right)^2 + \left(\frac{\partial^2 \kappa(s)}{\partial s^2} - \kappa(s)^3\right)^2\right)\right).$$

When the trajectory turns relatively gently compared to the robot radius ρ, i.e., when $\rho\kappa \ll 1$, only the first two terms make a significant contribution, i.e.,

$$\mathscr{J}(\gamma) \approx v^5 \int_\gamma \left(\kappa(s)^4 + \left(\frac{\partial \kappa(s)}{\partial s}\right)^2\right) \mathrm{d}s, \tag{2.3}$$

which is the final form of the cost that we are going to optimize.

Alternative costs A similar, alternative cost that is discussed in the literature [28, 55, 147] is given by

$$\int_\gamma \left(\kappa(s)^2 + \Gamma\left(\frac{\partial \kappa(s)}{\partial s}\right)^2\right) \mathrm{d}s, \tag{2.4}$$

where Γ is a factor with dimension $[s]^2$. For $\Gamma = 0$, this cost is proportional to the trajectory bend energy [36], i.e., to the potential energy required to bend an elastic band to the shape given by γ.

An alternative approach would be to minimize the maximal discomfort, or more simply, to find the trajectory with the largest minimal curvature

$$\min_{s\in[0,\mathrm{length}(\gamma)]} |\kappa(s)|. \tag{2.5}$$

2.3.2 Problem formulation

Trajectories with small curvature are beneficial for multiple reasons: *a*) the robotic wheelchair keeps jerk small, increasing in this way the comfort for a user sitting on it; *b*) with smaller angular speeds for the same linear speed, the wheelchair slips less and reduces the risk of losing localization; *c*) gently turning paths also look more natural, predictable and legible, which are important features when autonomous wheelchairs share the space with people.

Bending cost optimization Given a list $(b_1, \ldots, b_n)$ of convex cells' boundaries to cross while traversing convex cells $(c_0, \ldots, c_n)$, and start s and end e poses located in the first and last cell, we look for the trajectory γ that passes through them in the correct order, is contained in the cells, and minimizes the *bending cost* derived through Eq. (2.3)

$$\mathscr{B}(\gamma) = \int_\gamma \left(\kappa(s)^4 + \left(\frac{\partial \kappa}{\partial s} \right)^2 \right) \mathrm{d}s, \tag{2.6}$$

where γ is parametrized by arc length s and κ is the curvature of γ.

We make use of the semantic labels attached to cell boundaries to constrain the trajectory to pass perpendicularly in the middle of doors.

Multi-objective geometrical optimization In general, there may be multiple topological paths connecting two cells in the navigation layer (see Section 1.3.6). In this case, looking for the minimal bending cost while neglecting other geometric features may be too limited. As a matter of fact, humans prefer *short* and less curved trajectories.

Therefore, a second optimization problem is defined for the same inputs that looks for trajectories that turn gently *and* are short. Namely, for a set of convex cells, a given source and target poses, we want compute the Pareto optimal trajectories (contained within navigable cells) according to length and bending cost $\mathscr{B}$.

2.3.3 Search space

The problem is defined as a non-linear optimization in the space of all curves. It becomes more manageable if we restrict the search to polynomial curves, which have a finite (small) number of parameters but approximate well enough any interesting trajectory.

In particular, composite Bézier curves [119] fit well into our problem [28, 147]. In fact, we could split the trajectory γ in many low-dimensional Bézier curves, one for each cell; each curve lies inside the (convex) cell if all control points lie inside the cell themselves; similarly, the curve will pass through two cell's boundaries if the first and last control points lie on the boundaries.

Therefore we search for the optimal γ in the space of *N-th order composite Bézier curves*.

Bézier curves

For each cell c_i, the curve segment $[0,1] \to c_i$ has the form[2]

$$t \mapsto \sum_{j=0}^{N} \boldsymbol{P}_j^{c_i} b_{j,N}(t), \tag{2.7}$$

where $b_{j,N}$ are Berstein polynomials of order N,

$$b_{j,N} : [0,1] \to [0,1] \tag{2.8}$$

$$b_{j,N}(t) = \binom{N}{j} t^j (1-t)^{N-j} \tag{2.9}$$

and $\boldsymbol{P}_{1\leq j\leq N}^{c_i} \in c_i$ are N control points that completely define the curve.

Bézier curves have limited degrees of freedom, allowing for efficient optimization, and are well known for approximating curves of any shape. They have several additional useful properties. For instance, Bézier curves are contained into the convex hull of their control points; thus, if the control points are contained in a convex cell, the corresponding Bézier curve will be contained in it too. Moreover, they are efficient to compute by a sequence of linear compositions using the de Casteljau's algorithm.

The control points adjacent to the segments' joins define the smoothness degree of a composite Bézier curve. More precisely, the n-th order derivative at a

[2]We switch to parameter t because, in this parametrization, curve segments are not parametrized by arc length.

join depends only on the adjacent n control points:

$$\begin{aligned}
\gamma^c(0) &= \boldsymbol{P}_0^c \\
\gamma^c(1) &= \boldsymbol{P}_N^c \\
\dot{\gamma}^c(0) &= N\left(\boldsymbol{P}_1^c - \boldsymbol{P}_0^c\right) \\
\dot{\gamma}^c(1) &= N\left(\boldsymbol{P}_N^c - \boldsymbol{P}_{N-1}^c\right) \\
\kappa_{\gamma^c}(0) &= \frac{N-1}{N}\frac{\left(\boldsymbol{P}_2^c - \boldsymbol{P}_1^c\right) \times \left(\boldsymbol{P}_1^c - \boldsymbol{P}_0^c\right)}{\left\|\boldsymbol{P}_1^c - \boldsymbol{P}_0^c\right\|^3} \\
\kappa_{\gamma^c}(1) &= \frac{N-1}{N}\frac{\left(\boldsymbol{P}_N^c - \boldsymbol{P}_{N-1}^c\right) \times \left(\boldsymbol{P}_{N-1}^c - \boldsymbol{P}_{N-2}^c\right)}{\left\|\boldsymbol{P}_N^c - \boldsymbol{P}_{N-1}^c\right\|^3}.
\end{aligned} \tag{2.10}$$

G^0 geometric continuity (i.e., C^0 continuity if the curve would be parametrized by arc length), requires that successive segments share the last, resp. first, control point. G^1 continuity needs that the three control points around the join are collinear.

Order

A necessary condition for a differential driven robot, like a typical wheelchair, to be able to follow a trajectory is that the trajectory should be G^2: the curvature (and therefore the angular speed, for continuous linear speeds) must be continuous. This puts a constraint on the 5 adjacent control points around a join, which generally requires $N \geq 6$ to be satisfied. Note that G^3 continuity may be necessary for a controller to follow a curve [116]; however, in most cases, the robot has to cope with localization errors and obstacles avoidance, therefore G^2, or even G^1 continuity, may be smooth enough for a trajectory that will be locally adjusted anyway.

The search for optimal curves is much faster when N is small, i.e., when their parameter space is low dimensional. Therefore we limit ourselves to $N \leq 6$. Often an optimal G^1 curve is sufficiently similar to an optimal G^2 to serve as an approximation. This offers the following possibilities to reduce the computation cost of the search of an optimal G^2 composite Bézier curve.

1. Search for optimal 4-th order (cubic) G^1 composite Bézier curve, without imposing curvature continuity at cell boundaries. Then, in a second phase, transform the curve into a G^2 curve that has continuous curvature at the boundaries.

2. Search for optimal 4-th order G^2 composite Bézier curve, while imposing curvature continuity at cell boundaries. If we are lucky (which is actually a common cases on indoor maps), we find a G^2 curve; else we revert to one of the other methods.

3. Search for the optimal 6-th order G^2 composite Bézier curve, imposing curvature continuity at cell boundaries. As discussed, these curves have enough degrees of freedom that feasible solution are always found.

For the rest of this chapter, we limit the discussion on the first option, which has been proven to be a viable solution in real-world scenarios; we discuss smoothing through degree elevation in Section 2.3.4.

Parametrization

We parametrize the $4n+4$ control points defining a composite cubic Bézier curve γ from s to e. Figure 2.2:Left illustrates the notation. In total, the curve has $4n+2$ degrees of freedom left for optimization after imposing G^1 continuity at the boundaries.

The curve starts and end in control points located in $\boldsymbol{P}_0^{c_0}$ and $\boldsymbol{P}_3^{c_n}$, and passes through control points $\boldsymbol{P}_0^{c_i} = \boldsymbol{P}_3^{c_{i-1}}$ located on each boundary. We parametrize their positions along b_i by a linear function l with parameters $s^i \in [0,1]$ ($s^i = 0, 1$ define points on the boundary of b_i)

$$\boldsymbol{P}_0^{c_i} = \boldsymbol{P}_3^{c_{i-1}} = l(s^i; b_i).$$

For the second, and second-last control points, which define the start and end derivatives, we use a polar parametrization p with origin in s, respectively e,

$$\boldsymbol{P}_1^{c_0} = p(u_1^0, \alpha_s; s),\ \boldsymbol{P}_2^{c_n} = p(u_2^n, \alpha_e; e),$$

where $u_1^0, u_2^n \in [0,1]$ are (normalized) radial distances divided by the cell length (the largest edge of the cell bounding box), and α_s, α_e are the headings of s and e respectively. Similarly, the points $\boldsymbol{P}_{1,2}^{c_i}$, for $1 \le i \le n$, which control the derivatives at the cell boundaries, are polar parametrized with respect to origins in $\boldsymbol{P}_0^{c_i}$ and axis oriented along b_i

$$\begin{aligned}\boldsymbol{P}_1^{c_i} &= p(u_1^i, \alpha^i; l(s^i; b_i), b_i),\\ \boldsymbol{P}_2^{c_i} &= p(u_2^{i+1}, \alpha^{i+1} + \pi, l(s^{i+1}; b_{i+1}); b_{i+1}),\end{aligned}$$

where α^i is the angle by which the trajectory crosses border b_i.

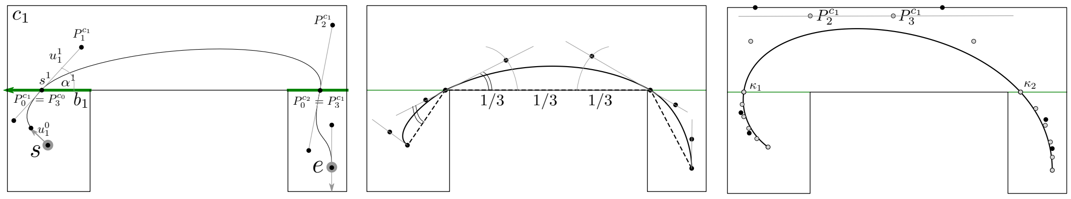


Figure 2.2. Towards the generation of the optimal Bézier composite curve. *Left:* start (s) and end (e) poses, and a topological path that traverses cells (c_0, c_1, c_2) and crosses boundaries (b_1, b_2), are given. The control points $\boldsymbol{P}^c_{0,1,2,3}$ define the curve. There are four degrees of freedom at boundaries (e.g., $\alpha^1, s^1, u^1_1, u^0_2$ at b_1) and one degree of freedom at start and end (e.g., u^0_1 at s). *Center:* the trajectory is initialized after computing the shortest path (dashed). *Right:* the G^2 trajectory is obtained by elevating the optimized 4-th order trajectory (black control points) to 6-th order (gray control points). The 6-th order control points $\boldsymbol{P}^{c_1}_2$ and $\boldsymbol{P}^{c_1}_3$ are moved slightly along the segment that connect them to make the curvature continuous at b_1 and b_2. respectively

2.3.4 Bending cost optimisation

We reformulate the optimization problem as a search in the space of the Bézier segments' control points.

Using the notation $\boldsymbol{u} = (u^i_1, u^i_2)_{0\le i\le n}$, $\boldsymbol{s} = (s^i)_{1\le i\le n}$ and $\boldsymbol{\alpha} = (\alpha^i)_{1\le i\le n}$, the $4n+2$ parameters defining the optimal $\gamma = \gamma(\boldsymbol{u}, \boldsymbol{s}, \boldsymbol{\alpha})$ with respect to $\mathscr{B}$ are obtained through the solution of the following non-linear constrained optimization problem:

$$\begin{aligned}
&\text{Minimize } \mathscr{B}(\gamma(\boldsymbol{u}, \boldsymbol{s}, \boldsymbol{\alpha}))\\
&\text{Subject to}\\
&\qquad u^0_1, u^i_1, u^i_2, u^n_2, s^i \in [0,1],\\
&\qquad \alpha^i \in [0,\pi],\\
&\qquad p(u^0_1, \alpha_s; s) \in c_0\\
&\qquad p(u^i_1, \alpha^i; l(s^i; b_i); b_i) \in c_i\\
&\qquad p(u^i_2, \alpha^i + \pi; l(s^i; b_i); b_i) \in c_{i-1}\\
&\qquad p(u^n_2, \alpha_e; e) \in c_n\\
&\qquad \forall i \in \{1, \ldots, n\}.
\end{aligned}$$

We can then use a numerical solver, such as "Constrained Optimization By Linear Approximations" (COBYLA) [118], to compute (locally) optimal trajectories. Since this solver works better with differentiable constraints, we reformu-

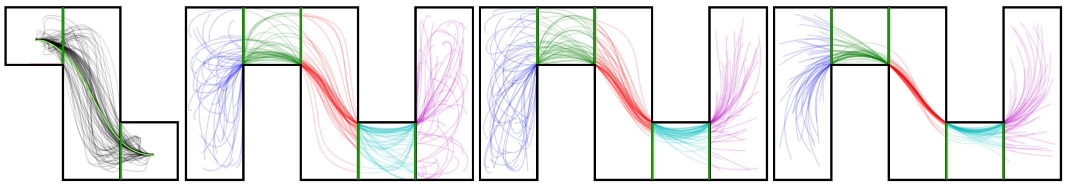


Figure 2.3. Optimized trajectories on a synthetic map. *Left*: Local optima found by the solver between the same source and target poses, initializing the search with random parameters (100 black lines) or using the heuristic (green line). *Right*: 50 optimized trajectory between two poses (left), a pose and a point (center), and two points (right); trajectories are colored by the cell they traverse.

late all constraints of form $\boldsymbol{P}_j^{c_i} \in c_i$ as:

$$\boldsymbol{P}_j^{c_i} \in c_i \Leftrightarrow d(\boldsymbol{P}_j^{c_i}, \partial c_i) \geq 0,$$

where $d(P, \partial c)$ is the distance between P and the boundary of cell c, negated if P lies outside of the cell.

Because of non linearity, the cost function may present several local minima to which the solver converges depending on the initial conditions; therefore a good initial estimation of the optimal parameters is needed. Figure 2.3:Left illustrates an example for which most randomly initialized searches do not converge to the global optimum, to the contrary of a search initialized with the heuristic described below. Figure 2.3:Right depicts several optimal trajectories between two random locations: two poses (i.e., α_s and α_e fixed), a point and a pose (i.e., only α_s fixed), and two points (i.e., α_s and α_e free to vary).

Initialization

To compute a good initial guess, we apply a heuristic that smoothens the shortest path γ_0 between start and end poses, as illustrated in Figure 2.2:Center. The shortest path itself is computed from the visibility graph of the navigation layer and is composed of a line segment for each cell.

The parameters $\boldsymbol{s}$ are chosen in order to place the points $\boldsymbol{P}_0^{c_i}$ at the intersection of γ_0 with each boundary; angles $\boldsymbol{\alpha}$ are initialized as a midway between the angles of the two incident segments of γ_0; $\boldsymbol{u}_{1,2}$ are set by placing the control points $\boldsymbol{P}_1^i$ and $\boldsymbol{P}_2^i$ at one third of the distance between $\boldsymbol{P}_0^i$ and $\boldsymbol{P}_0^{i+1}$ (or on the cell boundaries if they would end up outside of c_i).

We have verified that, in most cases, applying this heuristic leads to acceptable results even if the parameters are not further optimized.

G^2 trajectory

As discussed in Section 2.3.3, composite cubic Bézier curves are not generally G^2 because of discontinuities in the curvature at the joins. In general, we need to add at least two other control points for each cell to be able to adjust the curvature at one endpoint, without modifying the derivative and the curvature at the other end.

We apply the following procedure to transform the optimal 4-th order curve to a 6-th order G^2 curve; shape and cost $\mathscr{B}(\gamma)$ do not be change significantly. The degree of each segment is elevated to 6-th order [119], i.e., we find 6 control points that define the *same* Bézier curve. Then, for each boundary, we compute the curvatures $\kappa^i(0)$ and $\kappa^{i-1}(1)$ on the two sides (see Equation 2.10). They should be equal for the curve to be G^2. If they are different, we set the target curvature at boundary b_i as

$$\kappa^i = \begin{cases} \sqrt{\kappa^i(0)\kappa^{i-1}(1)}, & \text{if } \kappa^i(0)\kappa^{i-1}(1) \geq 0 \\ 0, & \text{else} \end{cases} \tag{2.11}$$

and move the control point $\boldsymbol{P}_2^{c_i}$ along the line passing by $\boldsymbol{P}_2^{c_i}$ and $\boldsymbol{P}_3^{c_i}$ as necessary to achieve curvature κ^i (see Figure 2.2:Right). Similarly, on the other side of the boundary, $\boldsymbol{P}_3^{c_{i-1}}$ is moved along the line passing by $\boldsymbol{P}_3^{c_{i-1}}$ and $\boldsymbol{P}_2^{c_{i-1}}$.

For some trajectories, which are uncommon on indoor building maps, this process may not be possible because the resulting control point would lie outside of the cell. In those cases, we have to rely on other smoothing techniques or switch to another option discussed in Section 2.3.3.

2.3.5 Optimal trajectories in indoor buildings

Doors and narrow passes Maps of indoor buildings generally have more structure than just being collection of cells. For instance, they contain corridors and rooms separated by doors.

We use these characteristics to add additional constraints to the control points of trajectories in indoor buildings. In particular, we want to ensure that the wheelchair when crossing doors passes perpendicularly in the middle, i.e., $\alpha^i = \pi/2$ and $s^i = 1/2$ for all i for which c_i or c_{i-1} are doors.

This brings two advantages: *a*) legibility and predictability of the wheelchair trajectory near doors is increased, where good negotiation of shared crossing with humans is critical; *b*) the optimization problem is split into smaller problems, since the path $(b_1, \dots, b_n)$ gets subdivided into m smaller chains

$$\big((b_1, \dots, b_{n_1}), (b_{n_1}, \dots, b_{n_2}), \dots, (b_{n_{m-1}}, \dots, b_{n_m})\big), \tag{2.12}$$

for which the trajectory is perpendicularly constrained into the middle point at $b_{n_i}, i = 1, \ldots, m$. The optimization of one chain is *independent* from the other chains.

We impose the same constraints on cell boundaries that are very narrow, losing a little bit of freedom but substantially reducing computational complexity. If all boundaries in a chain are constrained, like for a sequence of doors, optimization is trivial and results in a straight line.

Trajectories graph Contrary to start and end poses, which are only known at query time, all constrained boundaries can be identified at initialization. Moreover, the optimal trajectory between them can be pre-computed. In fact, they form the edges of a graph G_{opt} whose nodes consist of doors and narrow passages.

Based on these insights, we greatly reduce query time by proceeding as following. Instead of optimizing the full trajectory, we first subdivide the topological path into chains as described above. Then, we run the optimization only for the first and last chain (i.e., from s to the first door and from the last door to e), while we retrieve already optimized intermediate trajectory from G_{opt}. In this way, computational costs become almost independent of the number of boundaries crossed (see Section 2.4.2 for an experimental verification).

2.3.6 Geometric multi-objective optimization

The graph G_{opt} also serve us to compute Pareto optimal curves with minimal length and bending cost. First, we add to G_{opt} all optimal trajectories from s and to e that originate from a node of G_{opt} and do not cross any other node. Then, we apply Algorithm 4, without modifications, with respect to pre-computed costs $\mathscr{B}$ and length $\mathscr{L}$ for all edges in G_{opt}. Figure 2.4 illustrates an example.

The considerations discussed in Section 1.3.7 apply here as well. Namely, it is in general beneficial to reduce the number of solutions by pruning solutions with small cost differences.

2.4 Experiments

We report experimental results that evaluate the performance of the trajectory optimization and of the multi-objective planner. All experiments are performed on a single core modern architecture CPU using the implementation described in Appendix B.

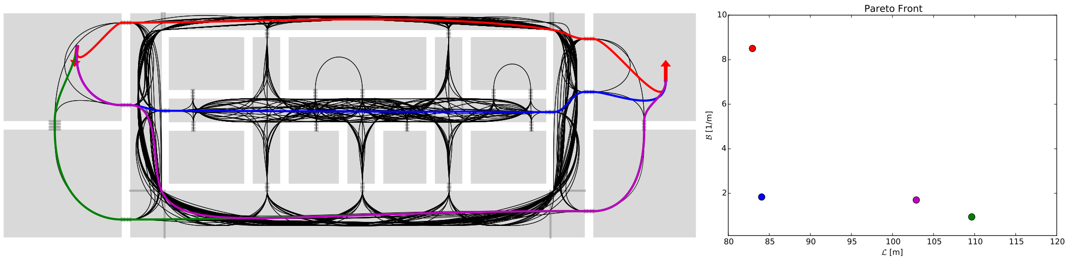


Figure 2.4. The graph G_{opt} used for multi-objective path planning. *Left*: Optimal solutions according to length $\mathscr{L}$ and bending cost $\mathscr{B}$ between two poses (red arrows) are drawn with different colors (green and purple paths overlap after a few cells); edges of G_{opt} are drawn in black. *Right*: Pareto front in cost space (with corresponding colors): the planner gives back the choice between longer (green), curvier (red) and two intermediate trade-offs trajectories.

2.4.1 Synthetic map

We measure the performance of the planner on a synthetic map, made by a succession of corners similar to a snake (see Figure 2.5:Top). Two examples of trajectory optimization converge slower (right) and faster to the optimal solution. Figure 2.5:Left reports how the computational cost grows super-linearly w.r.t. the length of the topological plan because of the linearly increasing number of degrees of freedom.

The numerical solver can be tuned for faster or more accurate results. Figure 2.5:Right illustrates the trade-off between computational cost and accuracy (bending cost nearer to the optimum) for paths of fixed length of 4 boundaries (i.e., with 18 degrees of freedom to optimize). For the experiments shown in the following, the planner is tuned for accuracy.

2.4.2 Real building map

We investigate the real world performance by testing the planner on the map of our building, which is 120 meters long. The navigation layer was obtained as described in Section 1.3.4 by inflating walls by 40 cm and then decomposing the navigable space into convex cells.

As detailed in Section 2.3.5, the planner initialization pre-computes all trajectories between constrained boundaries, which form the edges of the graph G_{opt}. For our building, this phase requires 3 minutes to compute about 2000 trajectories. Figure 2.6:Left displays the full map and an excerpt of the navigation layer together with trajectories in G_{opt}. Note how curves originate perpendicularly

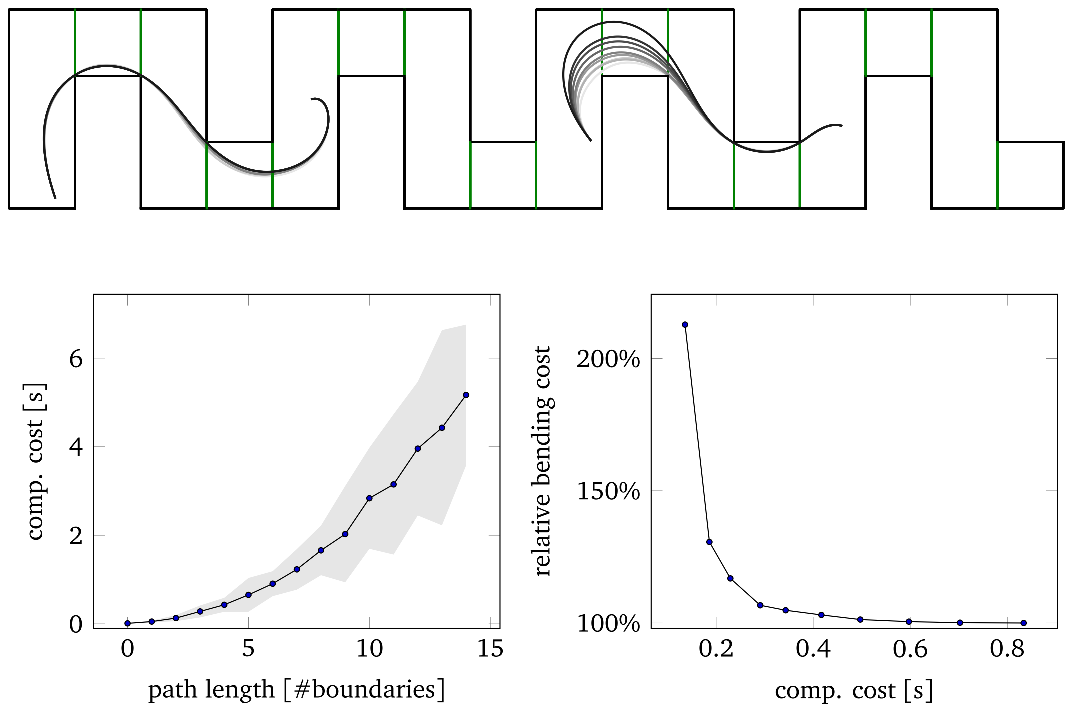


Figure 2.5. Top: synthetic map with two examples of trajectory optimization. Darker colors mean further optimization (higher computational cost). *Botttom-Left:* Computational cost for optimizing trajectories between random poses (50 per length) of increasing (topological) length: average (line) ± one standard deviation (grey area). *Botttom-Right:* average trade-off between computational cost and trajectory relative cost (w.r.t. the minimal trajectory cost) for 100 trajectories of (topological) length 4 between random poses when optimized by the numerical solver according to different accuracies.

from the middle of doors.

The planner performance is higher than on a synthetic map. On the one hand, the trajectories are more constrained (more doors and narrow passes). Therefore they can be split into chains of limited size. The total computational (see Figure 2.7:Left) cost becomes almost linear w.r.t. the length of the path. On the other hand, at query time, we take advantage of G_{opt} to compute only the first and last part of a trajectory. Thank to this caching, the computational cost becomes almost constant and remains well under 1 s for almost all queries. This confirms that the planner is well suited for online planning for robots moving in real buildings.

2.4.3 Multi-objective planning example

Figure 2.6:Right displays the solution for a single multi-objective ($\mathscr{C}$ vs $\mathscr{L}$) planning problem between two poses in our building. For this case, Algorithm 4 terminates after three paths have been evaluated and returns a Pareto front composed of two solutions, while only one additional dominated solution is computed. The planner gives us the choice between the longer (left) and curvier (right) trajectory.

Figure 2.7 reports the cost to compute the set of optimal trajectories, between 1000 randomly drawn poses, and its size. Problem instances with longer paths give rise to more solutions (up to 6 in average) and an increased cost, which nonetheless remains acceptable for online planning. We note that the longest topological paths do not correspond to the longest computational times: on this particular map, the few longer paths happen to be relatively simple to optimize (e.g., travelling along a long straight corridor).

2.4.4 Robot navigation

We tried to follow the optimal trajectories using a differentially driven, two wheeled, TurtleBot 2 robot. The robot localizes itself, using a depth sensor, on the same map used by the planner. We configure the robot navigation controller to keep its contribution to the final robot trajectory negligible. See Appendix B for details about robot's implementation, and Chapter 3 for a discussion about the controller used to follow an optimal trajectory.

Figure 2.8 shows some of the trajectories performed by the TurtleBot robot moving autonomously in our building. We qualitatively compare our planner with one of the planners implemented within the Robot Operating System (ROS)

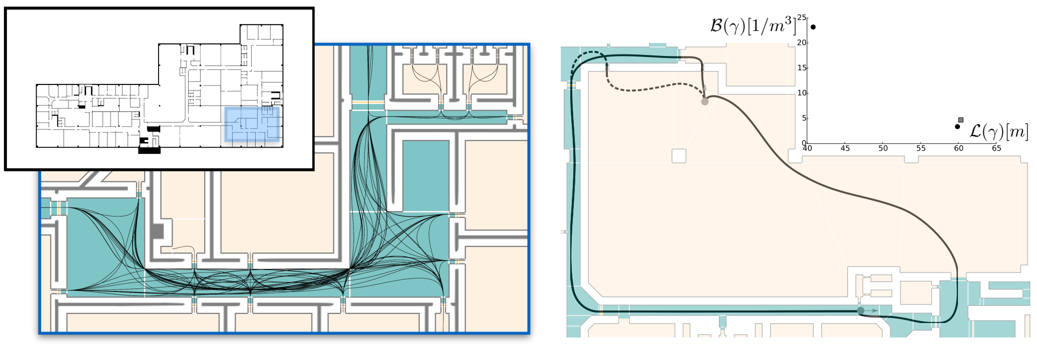


Figure 2.6. Left: part of our building IndoorGML map with the optimal trajectories forming the edges of G_{opt}. Cells are colored by type (corridors green, rooms yellow, doors orange). *Right:* multi-objective optimal paths between two poses (grey circles with arrow): Pareto optimal trajectories (solid lines) and the only visited dominated solution (dashed line). *Top Right:* costs of the three trajectories – Pareto front (black circle), dominated solution (grey squares).

navigation framework (*move_base*): *navfn*, a grid search planner, which optimizes a navigation function. The two planners have different goals. In fact, *navfn* is not trying to find a differentiable trajectory and is meant to be used together with a local planner that refines the trajectory. Still, it is interesting to note how qualitatively different the plans and the robot's traces are when using the two planners. Our planner produces smooth robot's traces that turn gently, making use of the free space, which are easier to predict for a human observer and more similar to those followed by a skilled human driver.

2.5 Conclusions and Perspectives

We discussed how to optimize a trajectory and perform multi-objective path planning to select short but comfortable trajectories for wheelchairs. We investigated the performance of the planner in synthetic maps. We showed that the planner is suitable for real time planning for robots in real world buildings.

In this work, we assumed that bending cost $\mathscr{B}$ is a good proxy for comfort and that, together with the constraints we impose, minimizing this cost also increases the acceptance of the robotic wheelchair by people sharing space with it. While this assumption is reasonable, an experimental validation would be an interesting (but difficult) future research effort.

One challenge for such validation is the separation of the contributions from the path planner and from the lower-level controller: a perfectly smooth target

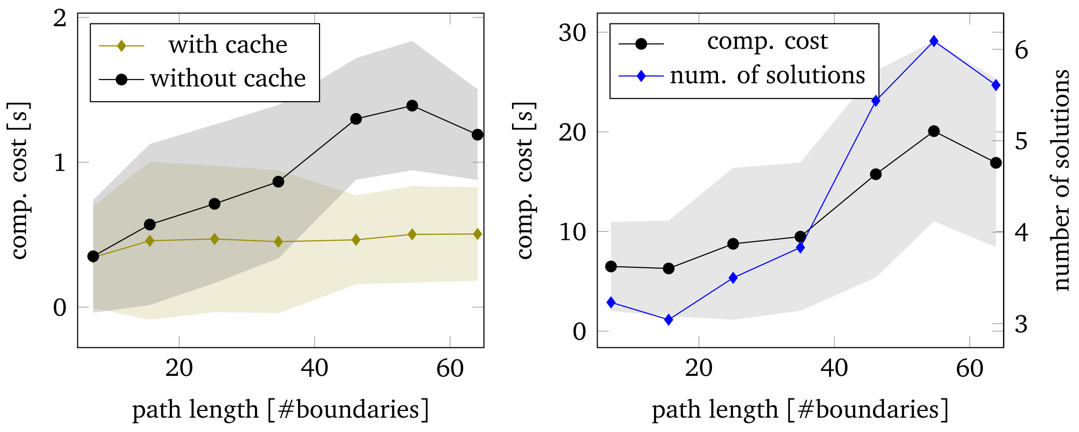


Figure 2.7. Computational cost of 1000 trajectories between random poses on the map depicted in Figure 2.6: average (solid line) and ±1 standard deviations (shaded area) vs. the number of boundaries crossed. *Left*: cost of trajectory optimization with/without using the pre-computed optimal trajectory graph G_{opt} as cache. *Right*: cost of the multi-objective path planning problem and, in blue, the average number of optimal solutions.

trajectory is not of much use, for the user, if the local navigation algorithm or the motor controller are very jerky, or when the wheelchair greatly deviates to avoid dynamic obstacles. This last point is the topic of the next chapter.

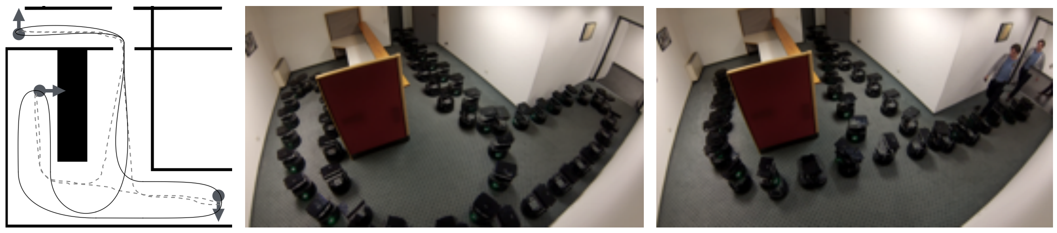


Figure 2.8. Left: qualitative comparison of the trajectories computed by our planner (solid black lines) and by the *navfn* planner (dashed grey lines). *Center and right:* the trajectories followed by the robot, which suffers slightly from inaccuracies in localization and in actuation (center: our planner, right: *navfn* planner).

Chapter 3

Human-friendly local navigation

3.1 Introduction

3.1.1 Local navigation

The planners discussed in Chapters 1 and 2 do not account for dynamic aspects. All obstacles have been modeled as static cells: they do not move and we know the position of all of them at planning time. Moreover, the planners are not concerned with time or velocity. The topological planner assumes that the robot will be able to move towards the next cell boundary and the geometric trajectory planner requires that the robot is able to follow a curve.

Nonetheless, dynamics is important. To remain safe, the robot should avoid colliding into unforeseen or moving obstacles; to be predictable, it should accelerate and decelerate smoothly. This should be implemented while diverging as little as possible from the optimal geometrical trajectory.

Local navigation and collision avoidance are fundamental challenges in mobile robotics. The most common solutions adopt reactive algorithms [22, 23] that iteratively, at each control step, select the desired motion towards the target taking into account the presence and the expected motion, over a short time horizon, of perceived obstacles.

From an engineering point of view, robot navigation needs to be effective in terms of followed trajectories, robust to sensing errors, scalable to differently crowded environments. At same time, when robots share space with people, robot navigation needs to be safe and *human-friendly* [90].

3.1.2 Human-friendly behavior

With human-friendly, we mean that robot movements must exhibit behaviors that are acceptable by humans [50, 93]. In terms of local navigation, this could be ensured by trajectories that a person, in a similar setting, would follow. Such trajectories have the property to be legible and predictable by humans [40], meaning that a person observing the robot motion can intuitively understand the spatial target the robot is heading to and predict future movements. Other than limiting emotional stress, this ensures that both humans and robots can navigate the space efficiently. In fact, if navigation behaviors generate unpredictable trajectories, humans have to frequently change their direction to move around robots, ultimately resulting in less efficiency for both groups.

How to reproduce human-like behaviors? In this chapter, we illustrate a robot navigation algorithm which mimics pedestrian behavior. In fact, we implement the same local navigation rule [103] that has been validated in large-scale comparative experiments and shown to closely model human trajectories in controlled conditions [105].

Our implementation provides a number of extensions to the basic model, which take into account the core differences between robots and humans. In particular, we: (1) enforce safety (lateral contacts and pushing are an integral part of human crowd motion but are unacceptable in robotics); (2) respect human personal spaces whenever possible; (3) prevent local crowding situations, which could naturally lead to reduced efficiency.

The microscopic description of human-like navigation models groups of humans and robots that share the same space (and the same navigation behavior); this helps us to estimate the effort required to a human or a robot to move through a crowded cell in the models introduced in Chapter 1. Yet, the behavior works equally well when only robots are in the environment. In this respect, it represents a more flexible alternative to existing methods.

The navigation behavior only relies on local sensing. Which is the impact of perception on the navigation performance? We expect that a robot with a smaller field of view is less safe because it doesn't perceive some obstacles. The impact on efficiency is also interesting: the short-sighted robot may go straighter at first only to have to steer strongly later.

3.1.3 Emerging collective behaviors

The navigation algorithm is a microscopic description of the navigation behavior, one small step at a time. Yet it let macroscopic patterns emerge, in the form of collective behaviors, which are similar to those observed in human crowds. For example, crowds in corridors spontaneously split in lanes going towards different directions; at crossing points crowds form a swirl pattern that minimizes stop and go (see Figure 3.1).

Collective behaviors benefit the single agent in multiple ways. They increase efficiency and safety by reducing the number of conflicts (i.e., collisions to be avoided). They also increase human acceptance of the robot behavior, both for single robots, as well for groups of robots.

Collective behaviors are also a form of implicit (proxemic) communication. In the navigation algorithm, collision avoidance is shared by agents on a collision course. The agents communicate their intention just by moving towards where they want to go. Does perception (i.e., partial information) play a significant role? Indeed, we show that the collective behaviors require that enough information is (implicitly) shared among agents. For instance, agents that only look in front of their eyes, like horses with blinkers, won't organize in lanes.

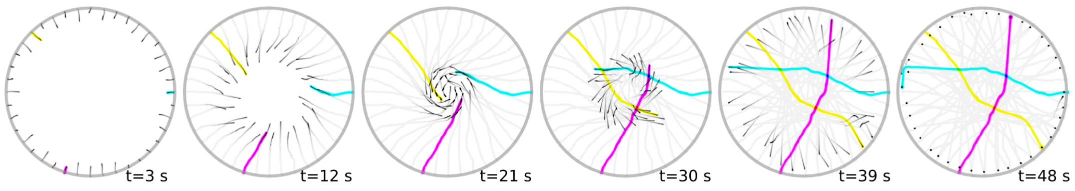


Figure 3.1. Visualization of trajectories for 40 simulated robots following the human-inspired navigation behavior. The robots show a collective behavior at a crossing: they swirl around each others without having to stop.

3.1.4 Outline

In this longer chapter, we present a bio-inspired, local navigation algorithm that generates human-like (and thus human-friendly) trajectories. In Section 3.3, we introduce the algorithm, which is conceptually simple, computationally light, independent of the specific sensing technique, and inherently able to handle heterogeneous agents. In Section 3.4, we compare the human-inspired behavior with two other state-of-the-art algorithms for local navigation of group of robots. In Section 3.5, we present a control strategy to let robots follow the (optimally smooth) path computed in Chapter 2.

We discuss and demonstrate the implementation of the navigation behavior on both *real* and *simulated robots* (Section 3.6); the simulation results are validated with real experiments; we also provide quantitative results obtained from large-scale simulations (Section 3.7) for all three navigation behaviors. In particular, we test the scalability of the approach to large robot swarms, the robustness to sensing inaccuracies, and the effect of various parameters on safety and efficiency. We additionally demonstrate that robot swarms implementing the proposed algorithm exhibit macroscopic behavioral patterns (e.g., the emergence of lanes of opposite flows in corridors) matching those observed in human crowds [105]. Finally, we show results on how simulated robots perform navigation in indoor building: how much navigation algorithms deviate from the prescribed trajectory and how much jerk increases as a results.

We only present experimental results that measures objective, quantitative metrics for safety and efficiency; ongoing experimental trials focused on acceptance as an inherently subjective quantity, using the tools described in Appendix B, are briefly discussed in Section 3.8.

3.2 Related Work

Dynamic obstacle avoidance is a challenge that has to be faced by any mobile agent. Therefore, this is a topic that has been extensively studied both in robotics and social sciences. In robotics, the aim is to effectively control the motion of one or more robots, while in social sciences the basic goal is to understand the behavioral models adopted by pedestrians in different situations. Moreover, since humans and robots may share common spaces, obstacle avoidance has been studied in reference to the simultaneous presence of both humans and robots, which raises up a number of issues regarding not only safety but also reciprocal predictability of trajectories and acceptance (of robots).

In this section, we present related works in the context of local navigation and human acceptance of robots' behaviors. We refer to Section 1.2.3 for the case when agents have access to global information to compute human-friendly paths.

3.2.1 Local navigation in robotics

In robotics, the most common approach is based on the concept of *velocity obstacle* [131], also known as *collision cone* or *forbidden velocity map*, which is the set of velocities that will lead a robot to collision: choosing a velocity outside such set ensures that no collision will occur. Different variants have been presented to: *a*) improve the prediction of the other agents' trajectories [13, 150, 155]; *b*) add recursion and include a probabilistic framework to account for sensing errors [78]; and *c*) ensure smooth trajectories by sharing the responsibility to avoid a collision with other robots (*reciprocal velocity obstacle, RVO* [12]), provided that the robots pass each other from the same relative side, to prevents oscillatory behaviors.

Two common approaches proposed to enforce that robots pass each other from the same side are: *a*) restricting the safe velocity space to half-planes constructed from the tangent spaces of velocity obstacles, truncated by a finite time horizon (*optimal reciprocal collision avoidance*, ORCA [14]) or *b*) artificially enlarging one side of the reciprocal velocity obstacle (*hybrid reciprocal velocity obstacle*, HRVO [135]). Both techniques can be applied to non-holonomic differential driven robots [5, 136] with localization and sensing uncertainties (an implementation for the Robot Operating System (ROS) is also available [67]).

Original velocity obstacle approaches assume that all agents and obstacles follow piece-wise linear motions and that agents adapt their velocity in a distributed way without explicit coordination. Extensions take into account robots' specific mobility constraints as well as the nonlinear or unpredictable motion of obstacles [11, 151], and robots following smooth feedback controlled trajectories [124]. A recent work investigates a centralized solution to select the optimal velocities and found that robots can increase their performance by operating in the joint velocity space [4]. Another recent study shows the utility of RVO as mesoscopic navigation strategy, where a robot treats similar neighbors as a larger group resembling a single obstacle, and avoid them as a whole [64].

All the mentioned works build on a mechanistic and artificial approach to navigation, which is primarily designed to ensure safety (collision-free motion), and is engineered to produce also smooth trajectories. Instead, our work stems from a heuristic modeling human behavior [103]. Since for humans a "contact" with another human while walking can be tolerated, the heuristic naturally produces paths of good efficiency, smoothness, and legibility, to which we add some modifications to ensure safety. We are first to apply this heuristic to robotics. Implementation-wise, the characteristics of the heuristic allows to decouple the computation of the desired heading and of the desired speed. This leads to a simpler implementation than velocity-obstacle approaches, which requires a search over the two-dimensional velocity space.

3.2.2 Local navigation in social sciences

Mutual avoidance and sharing of space among humans has been extensively studied in social sciences, mostly to predict the behavior of crowds. The original models are based on the study of *proxemics* [63], which formalizes the concept of *per-*

sonal and *social* space; pedestrian behavior based on *social forces* [65] enforces people to keep a minimum distance from neighbors whenever possible. Such a model was successfully used for crowd simulation and also inspired several human tracking and avoidance models in robotics [94, 141]. Simple rules applied in pedestrian navigation (passing on the left and shared collision avoidance responsibility) were incorporated in a sampling-based planner [79] for collision avoidance among robots. Moussaid et al. [103, 105] recently proposed a fundamentally different model, which we extend in this chapter for implementation in robots.

By adopting such a model for pedestrians, we aim to ensure that the robot will exhibit a human-like behavior. In turn, this can ensure that humans sharing space with the robot will be able to easily predict its intentions thus improving both efficiency and social acceptance.

3.2.3 Robot behavior acceptance

Research on the acceptance of a robot behavior by a human has identified two separate relevant characteristics of the robot behavior [40], which refers to the ability of humans to: *a*) infer the robot's intent by observing its behavior (*legibility*); and *b*) predict the robot's behavior while knowing its intent in advance (*predictability*).

In the context of robots navigating and sharing space with humans, the differentiation between a legible (i.e., when it is possible to infer which goal a trajectory is heading to) and a predictable (i.e., when it's possible to anticipate the trajectory towards a goal) behavior, may depend on the chosen experimental setup. First investigations report no significant distinctions after studying robots heading to possible different goals while being observed by humans [91]. For the same experiment setup, the interplay between local and global navigation behaviors has been shown to be of large importance for the legibility and acceptance of the resulting trajectories: in particular, human-aware global path planners only resulted in acceptable trajectories when coupled with human-like obstacle avoidance behaviors [90], which further highlights the importance of human-friendly local navigation behaviors. In addition to safety, efficiency (i.e., the amount of time or other resources required to accomplish a task) also improve the acceptance [50] of robots as social partners and was recently investigated for navigation [93].

3.3 Model

3.3.1 Problem formulation

At any given time, given the robot shape, an optimal speed, a target position, and a list of moving obstacles, compute the optimal velocity to avoid any collision and give rise to a efficient, legible and predictable motion towards the target.

3.3.2 Pedestrian heuristics

The pedestrian heuristics [103, 105], upon which we develop the robot navigation algorithm, can be summarized as following: move towards the direction that come closest to the target before colliding with any obstacles and keep a speed that allows breaking in time if needed. We introduce the following notation to formulate it more precisely (see Figure 3.2).

A time t, an agent at position $\boldsymbol{x}(t)$ and with heading $\alpha(t)$ in some fixed frame F, is directed, with velocity $\boldsymbol{v}(t) = v(t)\boldsymbol{e}_1(t)$, towards a (static) target point $\boldsymbol{O}$, where $\boldsymbol{e}_1(t) = \boldsymbol{e}(\alpha(t))$ is the unit vector in direction $\alpha(t)$. The agent is characterized by: (i) an optimal (open space) moving speed v_{opt}; (ii) a horizontal field of view $\text{fov}(t) = \text{fov}(\alpha(t)) = [\alpha(t) - \phi,\ \alpha(t) + \phi]$, for some $\phi \in [0, \pi]$ that depends on the perception capabilities of the agent; (iii) a ground occupancy that we approximate by a disk of radius r.

To direct its movements, the agent, based on visual information, makes use of a cognitive[1] function $f : \text{fov}(t) \to \mathbb{R}^+$ that maps each heading α within the field of view to the distance that the agent could travel at speed v_{opt} towards α before colliding with any visible obstacle. The distance is bounded by a maximum horizon H. When computing $f(\alpha)$, all obstacles are assumed to keep their current heading and speed, thus moving according to a uniform linear motion. Let $d_O : \text{fov}(t) \to \mathbb{R}^+$ be the minimal distance from $\boldsymbol{O}$ when moving in direction α before reaching horizon H or colliding with an obstacle, i.e., the distance from $\boldsymbol{O}$ to the segment that connects $\boldsymbol{x}(t)$ and $f(\alpha)\boldsymbol{e}_1(t)$.

Given the above notation, the navigation of a (pedestrian) agent can be explained by the following simple heuristic rules. First, the agent determines its desired heading $\alpha_{\text{des}}(t)$ as the direction allowing the most direct path to $\boldsymbol{O}$, taking into account the presence of obstacles

$$\alpha_{\text{des}}(t) = \underset{\alpha \in \text{fov}(t)}{\arg\min}\, d_O(\alpha). \tag{3.1}$$

[1]Humans feature a dedicated neural mechanisms to detect object motion [129] and predict the time-to-collision with obstacles.

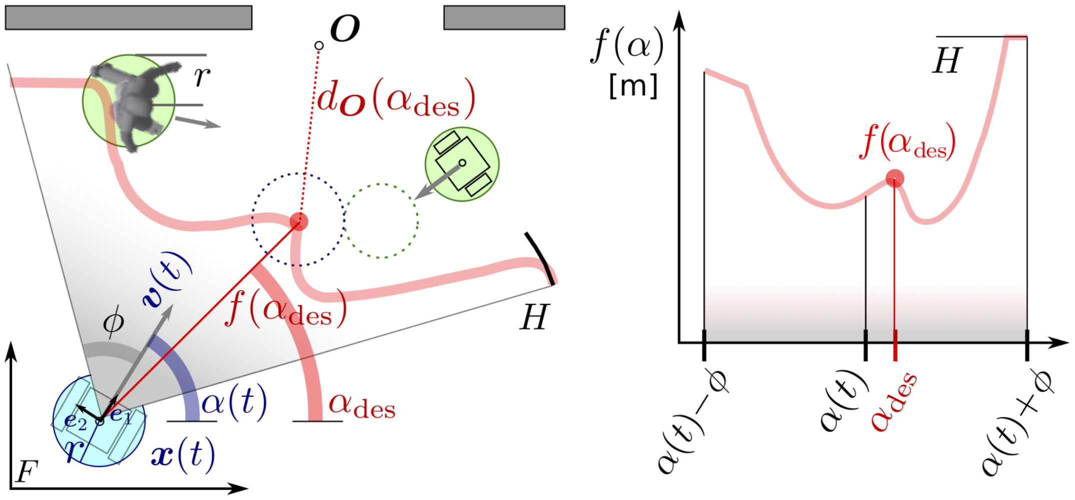


Figure 3.2. Human-inspired navigation behavior. The red curve $f(\alpha)$ is the estimated free distance that the blue robot can travel in direction α up to the first collision, that in direction α_{des} will happen at the red point where the light blue (robot) and green (robot neighbor) dotted circles will touch.

If the agent moves towards direction α_{des} at a constant speed v_{opt}, it will reach a point closer to the target than any point it would reach when moving in any other direction.

Then, the agent determines its desired speed to allow stopping in a fixed time $\eta > 0$ within the free distance $D(\alpha_{des}) \in [0, H]$, *currently* seen in direction α_{des}

$$v_{des}(t) = \min\left(v_{opt}, \frac{D(\alpha_{des})}{\eta}\right) \tag{3.2}$$

$$\boldsymbol{v}_{des}(t) = v_{des}\boldsymbol{e}(\alpha_{des}) \tag{3.3}$$

The actual velocity vector $\boldsymbol{v}(t)$ is continuously adjusted to obey

$$\frac{\partial \boldsymbol{v}(t)}{\partial t} = \frac{\boldsymbol{v}_{des}(t) - \boldsymbol{v}(t)}{\tau}, \tag{3.4}$$

where the fixed parameter τ represents the *time constant* characterizing the exponential speed profile, which modulates the smoothness of motion. Controlled laboratory experiments measured $\eta \approx \tau \approx 0.5\,\mathrm{s}$ for pedestrians in normal walking conditions [104].

Since computing $f(\alpha)$ involves a rough prediction of agent's and obstacles' future trajectories, the resulting behavior is *proactive* in that it attempts to avoid potential collisions well before they are expected to occur.

Pedestrian movements to reach a *pose* (i.e., when there is a target heading too, like when passing through a door) is different [114] than the heuristic we have illustrated.

3.3.3 Application to robot navigation

We now describe the application of the navigation heuristics for a two wheeled differential-driven robot with wheel axis A and current wheel speeds $w^{\text{left}}(t)$, $w^{\text{right}}(t)$ that are bounded by a maximal wheel speed w_{max}.

Robot kinematics

There are various possibilities to adapt the algorithm to the robot's simple kinematics [88]. We follow a common approach: from the desired heading computed in Equation (3.1), we compute a desired angular speed ω_{des} that makes the robot turn towards α_{des} in a fixed time τ_{rot}

$$\omega_{\text{des}}(t) = \left[\frac{\alpha_{\text{des}}(t) - \alpha(t)}{\tau_{\text{rot}}} \right]_{-\omega_{\text{max}}}^{+\omega_{\text{max}}}, \tag{3.5}$$

where, in order to avoid large slippages and sensing errors, we clamp the angular speed to an interval $[-\omega_{\text{max}}, \omega_{\text{max}}]$.

Then, left and right desired wheel speeds are computed from desired linear (Equation (3.2)) and angular speeds

$$w_{\text{des}}^{\text{left}}(t) = v_{\text{des}}(t) - \frac{A}{2}\omega_{\text{des}}(t), \tag{3.6}$$

$$w_{\text{des}}^{\text{right}}(t) = v_{\text{des}}(t) + \frac{A}{2}\omega_{\text{des}}(t). \tag{3.7}$$

Finally, we apply a modulation similar to Equation (3.4) to the speed of the wheels

$$\frac{\partial w^{\text{left,right}}}{\partial t}(t) = \frac{w^{\text{left,right}}(t) - w_{\text{des}(t)}^{\text{left,right}}}{\tau}, \tag{3.8}$$

and clamp their value to $[-w_{\text{max}}, w_{\text{max}}]$.

Increase safety

The model described above results in smooth paths, which have been shown to closely match the characteristics of pedestrian motion in large-scale controlled experiments, both for single trajectories and macroscopic crowd motion patterns [105]. Robots following the same rules would therefore exhibit a behavior which is predictable, legible, and acceptable by humans sharing the same environment with robots.

Nevertheless, the immediate application of the model to robotics is hindered by several shortcomings, with the main one related to the fact that trajectories are *not safe*: in fact, *collisions* among humans happen routinely (gentle pushing, shoulders rubbing) and, especially in crowded situations, contribute to define the motion of tightly-packed groups through reciprocal pushing forces. Even with sparse agents, collisions may happen when agents with a limited field of view are unable to perceive each other when traveling side by side, or in presence of sudden direction changes, which are only partially accounted for by the heuristic model.

Clearly, in our context, collisions (both among robots and between robots and humans) should be avoided as much as possible. Therefore, we extend the heuristic with the concept of *safety margin*, which is common to many obstacle avoidance approaches. In particular, when computing $f(\alpha)$, we account for an increased radius $r' = r + m_s$ for each agent, with m_s being a fixed safety margin parameter. Agents that enter into the safety margin of an obstacle are required to nullify the components of $\boldsymbol{v}_{\text{des}}$ which point towards it.

Under the unrealistic assumption of perfectly-accurate and omnidirectional sensing, choosing a sufficiently large value for m_s ensures that no collisions can occur. The trade-off is that a larger safety margin generally leads to worse performance because it reduces the available free space. In fact, we can estimate an *upper bound* M_s on the minimal safety margin required for collision-free behavior for agents that adjust their desired velocity once every *finite* time step Δt. For an agent with no constraints on acceleration, moving together with agents with the same upper bound on speed v_{opt} (Figure 3.3:Left), collision-free behavior is ensured if $m_s > M_s = 2v_{\text{opt}}(\Delta t + \tau)$, where the second term takes into account the additional amount of space to come to a complete stop. Non-holonomic agents demand extra care as their selection of desired velocity does not take the motion constraints into account and therefore need additional space to turn towards the desired heading. In the worst case scenario (Figure 3.3:Right), when two facing robots moving at full speed towards each other do 180° turn, they need an additional space proportional to $v_{\text{opt}}\tau_{\text{rot}}$, which has to be added to M_s. In practice, much shorter safety margins can be safely adopted (see Section 3.7.1).

Nonetheless, with realistic sensing inaccuracies and limited field of view, a completely safe behavior cannot be guaranteed, and too large safety margins would also lead to inefficient and unnatural trajectories. Therefore, given the

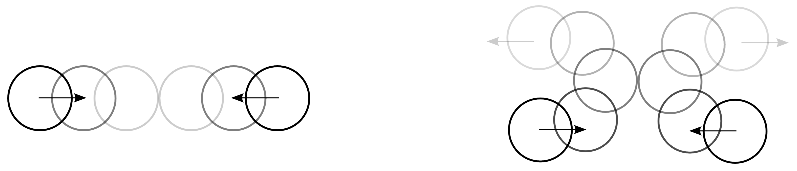


Figure 3.3. The worst case scenario considered to compute the upper bound M_s on the safety margin: two agents travel at maximal speed towards each other. *Left*: each needs at most $v_{\text{opt}}(\Delta t + \tau)$ space to stop before colliding. *Right*: non-holonomic agents need additional space proportional to $v_{\text{opt}}\tau_{\text{rot}}$ when they turn.

characteristics of the sensing subsystem, the safety margin m_s controls the trade-off between efficiency and safety of the trajectories, which is investigated in Section 3.7.2. Unlike Moussaïd et al. [103], if an obstacle is inside the safety margin, we set $f(\alpha) = D(\alpha) = 0$ for all angles α that would bring the robot closer to the obstacle.

Respect personal space

We observe that robots that follow the pedestrian navigation rule, when crossing in opposite directions, tend to pass each other (and humans) as close as allowed by the safety margin, regardless of how much space is available. While it may be an appropriate model for humans, this behavior is not always suited to robots, for two reasons: *a*) a robot passing a human should, *if possible*, keep a distance large enough to avoid invading its personal space and causing discomfort; *b*) groups of robots passing close to each other can induce a temporary situation of *local crowding*, which occasionally results in *deadlocks*.

Increasing m_s is not an appropriate solution for either problem, because agents should be allowed to come close to each other *when needed* (e.g., in order to negotiate tight spaces). To address the problem, we add, to the static safety margin, a *social margin* $m_t \geq m_s$ that varies with the distance between the agent and its closest neighbor. When possible, the social margin should enforce a personal space, while, when the robot negotiates tight spaces, it should dissolve to avoid deadlocks.

Therefore we redefine $r' = r + m(d)$, where $m : [0, H] \to [m_s, m_t]$ is a piecewise linear function of the distance d between the agent and its closest neighbor (see Figure 3.4). As a result, *when enough space is available*, d is large and the robots tend to keep a distance larger than strictly necessary. On the one hand, this increases social acceptance by humans; on the other hand, this reduces the likelihood of forming local high-density clusters of robots (further enhancing safety

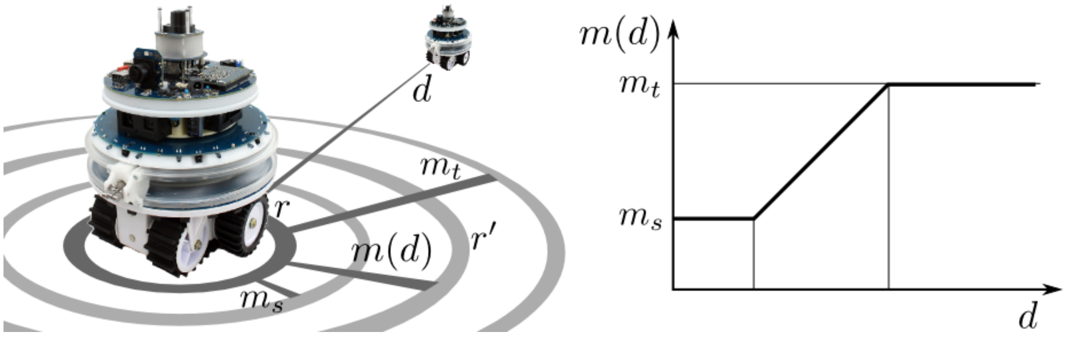


Figure 3.4. Illustration of safety margin m_s and social margin $m(d)$. As a function of the distance d of the closest neighbor, a margin $m_s \leq m(d) \leq m_t$ is added to each obstacle's physical radius r. The figure also illustrates the foot-bot robot, on which we implemented the navigation system.

as a side-effect), which may lead to deadlocks. Still, deadlocks cannot be completely ruled out and may occasionally occur, especially with large sensing errors and/or large numbers of robots packed in tight spaces.

The function $m(d)$ acts as a *soft* constraint, plays a similar role as potential fields in the social force formulation of navigation [65], and can be modulated to enforce interesting group behaviors [59].

Biomimetic tuning

All parameters of the navigation behavior have a simple social/bioinspired interpretation. For example, η can be interpreted as "caution": the higher η, the more time the robot keep away from the nearest obstacles. Can the dynamical, bio-inspired modulation of the social margin, be extended to other parameters to improve the robots' (collective) behavior?

In two related works [57, 58] we propose a model of *artificial* emotions for adaptation and implicit coordination in multi-robot systems that indeed improves the collective performance. Artificial emotions act as modulators of the individual robots' navigation behavior, and as means of communication for social coordination to: *a*) prevent deadlocks in crowded conditions; *b*) enabling efficient navigation of agents with time-critical tasks; *c*) assisting robots navigating despite of faulty sensors.

As an example of situation *a*), robots which progress slowly towards their target, i.e., for which v_{des} is small due to little free distance $D(\alpha_{\text{des}})$ in Equation (3.2) caused by local crowding, progressively become *frustrated*; this increases *fear* among neighbors to end up in a deadlock. By modulating free parameters ($v_{\text{opt}}, \phi, \eta, \tau, H$) depending on the emotional states, we show in these works that the agents avoid potential blockage. For instance, fearful agents slow down (v_{opt} decreases), are more attentive (ϕ increases), and more cautious (η increases).

3.4 Comparison with alternative navigation behaviors

We refer to the algorithm described in Section 3.3 as *Human-like* (HL); in this section, we present alternative algorithms and compare them with HL.

3.4.1 Behaviors based on Reciprocal Velocity Obstacle

Behaviors based on Reciprocal Velocity Obstacle are state-of-the-art for local robot navigation. They originated from the *Velocity Obstacle* concept [131]. The main idea is to first determine the set of velocities (denoted as VO_o) that will lead to collisions with an obstacle o, assuming that o will maintain its current velocity, and then select the best velocity outside of it, typically the one that is nearest to a prescribed preferred velocity. VO_o is constructed in the velocity space by translating the collision cone (i.e., the set of velocities that eventually lead to a collision if o remains at its current position) by the obstacle's velocity $\boldsymbol{v}_o$.

Nevertheless, when the obstacle is an agent also adjusting its velocity following the same rule, oscillations and unsafe trajectories may occur when using this behavior, because there is no guarantee that the desired velocity remains safe after the concurrent change of velocity by the neighboring agent. The issue is addressed by the *Reciprocal Velocity Obstacle* behavior [12], which modifies the construction of the safe velocity set, moving the collision cone by $\frac{1}{2}\boldsymbol{v}_o + \frac{1}{2}\boldsymbol{v}$ instead of by $\boldsymbol{v}_o$, (yielding the *reciprocal* velocity obstacle RVO_o) so to let each one of the agents take *half* of the responsibility to avoid the collision.

Still, the behavior requires that the agents choose to adjust the velocity towards the same (relative) side (i.e., either both are steering left, or both are steering right), in order to avoid oscillating behaviors. Researchers have proposed two approaches to enforce this *implicit* coordination for steering, depicted in Figure 3.5.

Hybrid Reciprocal Velocity Obstacle (HRVO [135]) The forbidden velocity set induced by obstacle o is given by the *hybrid reciprocal velocity obstacle* HRVO_o,

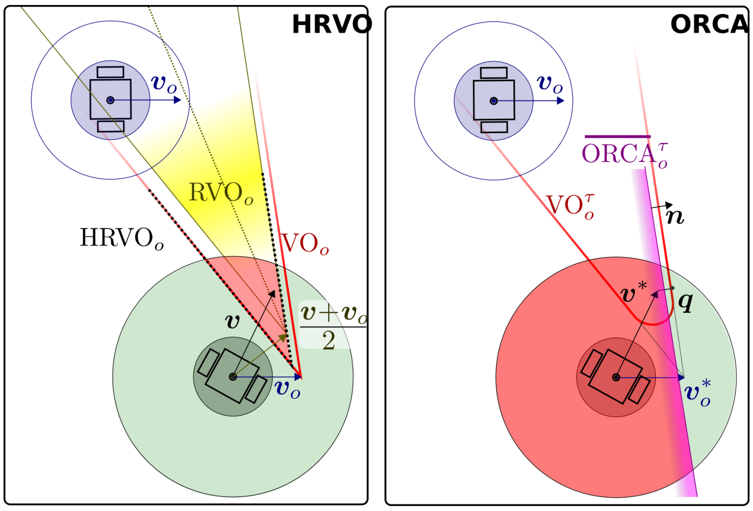


Figure 3.5. Illustration of the main entities and notations for HRVO and ORCA. The desired velocity is chosen inside the green region containing the feasible velocities that are outside of the HRVO/ORCA velocity obstacle (red region) induced by the top-right neighbor robot.

which is constructed by asymmetrically enlarging the reciprocal velocity obstacle RVO$_o$. In particular, if $\boldsymbol{v}$ is in the left half-plane respect to the bisector of RVO$_o$, then the right half of RVO$_o$ is substituted with the right half of VO$_o$. Intuitively, the agent takes half of the responsibility to avoid a collision when choosing to pass to the left, whereas full responsibility is taken if choosing to pass to the right. If $\boldsymbol{v}$ lies instead in the right half-plane, the opposite occurs.

Optimal Reciprocal Collision Avoidance (ORCA [14]) In the case of ORCA, the construction of the forbidden velocity set is more involved. Let $\boldsymbol{v}^*$ represent an optimal velocity the agent would like to maintain, that we fix to its current velocity $\boldsymbol{v}$ (see [14] for a discussion about the effect of different choices). In ORCA a finite time horizon τ is considered: beyond τ future collisions are ignored. Consequently, the velocity obstacle VO$_o$ (with apex at $\boldsymbol{v}_o^*$) is truncated to VO$_o^\tau$. In practice, this removes the apex of VO$_o$, which corresponds to the velocities that would lead to a collision after a large amount of time. Let $\boldsymbol{q}$ be the point on the boundary of VO$_o^\tau$ that is nearest to $\boldsymbol{v}^*$, and $\boldsymbol{u}$ be the vector connecting $\boldsymbol{q}$ to $\boldsymbol{v}^*$, $\boldsymbol{n}$ be the outwards normal of VO$_o^\tau$ at $\boldsymbol{q}$. The half-plane ORCA$_o$ is defined as the half-plane perpendicular to $\boldsymbol{n}$ at point $\boldsymbol{v}^* + \frac{1}{2}\boldsymbol{u}$ and define the set of forbidden velocities induced by obstacle o.

Both approaches lead to safe paths without oscillatory behaviors, even when more than two agents are involved. Moreover, for ORCA it is possible to formally prove smoothness and safety of the resulting paths (assuming that the agent's velocity update are synchronized and perfect knowledge about the neighbors is available).

There are several approaches to extend these algorithms behind holonomic agents that are capable to immediately adjust its speed in any direction. One is to follow Equations (3.5) and (3.8) to transform the desired Cartesian velocity to wheel speeds.

Non Holonomic Optimal Collision Reciprocal Avoidance (ORCA-NH [136]) Following [136], another possibility is to consider the agents as being contained within an effective circle with a forward-shifted center $\pi = \boldsymbol{x} + \rho_a \boldsymbol{e}(\alpha)$ and radius $r + \rho$ with $\rho > 0$. There is an invertible map between wheel speeds and velocities of the effective center, that allows the agent to follow an arbitrary path of its effective center like if it were holonomic. We denote the ORCA behavior applied on the effective circle with $\rho = A/2$ as ORCA-NH.

3.4.2 Comparison with the Human-like behavior

All presented behaviors have a common trait: they anticipate future collisions by using the current sensing information for position, velocity, and shape of the agent and of surrounding obstacles. All use a linear prediction of the obstacles' trajectories to compute a time-to-collision estimate, then select the velocity that minimizes their deviation from the straight line towards the target. The presence of obstacles enforces hard constraints on the agent that is not permitted to touch them. In contrast, methods based on potential fields (also known in sociology as *social forces*) do not explicitly perform a prediction of future trajectories; in this case obstacles generate soft constraints in the form of increased costs.

HRVO and ORCA explicitly share the collision avoidance responsibility among agents, which leads to improved performance. HL indirectly obtains the same effect by modulating velocities smoothly using the τ parameter: an agent, while smoothly turning to avoid others, has sufficient time to acknowledge the obstacles' actions.

All behaviors only use currently sensed information and bear no history or state information, i.e. they are purely *reactive* and *stateless*[2]; at the same time, all behaviors *proactively* avoid collisions and anticipate the motion of others.

[2]This does not necessarily apply to the sensing subsystems. For instance, history of obstacles' positions could be maintained in order to determine their speed.

The most prominent peculiarity of HL is that it performs a one-dimensional search over the direction of the desired velocity, choosing the one that minimizes the *spatial* distance to the target. RVO and all derivatives, instead, perform a search over (a subset of) the two-dimensional velocity space: then the desired speed is chosen in order to minimize the *velocity-space* distance to the optimal velocity, i.e., the velocity directed towards the target with maximal speed. Note that because HL acts to minimize spatial distance, in no circumstances it will dictate to move farther away from the target. Instead, ORCA and HRVO may exhibit such behavior when the forward half of the velocity space is forbidden (i.e., when moving backwards is the only solution to avoid a future collision).

Another peculiarity of the HL behavior is that it does not explicitly exclude directions that could lead to future collisions: such directions are just penalized in the search of the desired direction. Instead, velocity obstacle based behaviors forbid all velocities leading to a collision, unless forced by the absence of alternatives. In other words, they start by searching for the set of safe velocities, then select the one that maximizes performance, whereas HL optimistically selects a direction maximizing performance (accounting for obstacles), and, only as a second step, it adjusts speed to ensure safety.

3.5 Navigation along a geometrical trajectory

In the previous sections we presented controllers that steer an agent to reach a fixed target point while avoiding obstacles. In this section, we describe an extension that allow the robot to follow a geometrical path γ discussed in Chapter 2. The controller dynamically updates the target $\boldsymbol{O}_\gamma(t) = \boldsymbol{O}_\gamma(\boldsymbol{x}(t))$, used by the navigation algorithms, depending on the agent current position with respect to γ, which is a simple strategy known as *carrot planner* [88].

Let $\gamma : [0, l] \to \mathbb{R}^2$ be a curve parametrized by arc length. To get a reference point $\gamma(s)$ on the curve, we project the robot position $\boldsymbol{x}(t)$ on γ and advance by (at most) $\delta \geq 0$:

$$s(t) = \left[\operatorname*{arg\,min}_{s' \in [0,l]} |\boldsymbol{\gamma}(s') - \boldsymbol{x}(t)| + \delta \right]_0^l , \tag{3.9}$$

which we use to define the target point for the navigation algorithms at time t, as illustrated in Figure 3.6

$$\boldsymbol{O}_\gamma(t) = \gamma(s(t)) + h\boldsymbol{e}_1^\gamma(s(t)), \tag{3.10}$$

where $0 < h \leq H$ influences the planning horizon for the navigation behavior.

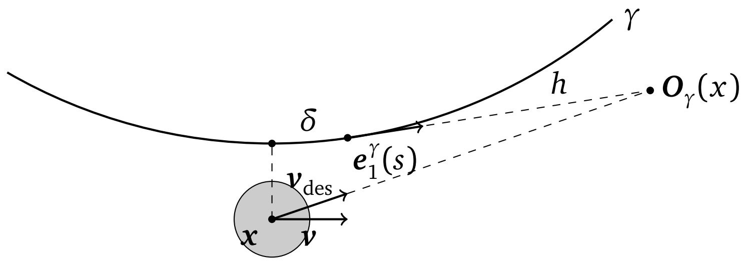

γ
h
$\boldsymbol{O}_\gamma(x)$
δ
$\boldsymbol{e}_1^\gamma(s)$
$\boldsymbol{v}_{\text{des}}$
$\boldsymbol{x}$
$\boldsymbol{v}$

Figure 3.6. The dynamic target point $\boldsymbol{O}_\gamma$ used to follow curve γ. When there are no obstacle, the desired velocity $\boldsymbol{v}_{\text{des}}$ computed by Equation (3.3) points to $\boldsymbol{O}_\gamma$ and steers the robot along the path.

The distance between the actual robot trajectory $\boldsymbol{x}(t)$ and the target trajectory γ depends on the navigation parameters τ_{rot} and v_{opt}, and on the carrot planner's head gap δ. When the robot is near and almost aligned with the trajectory, for small curvature $\delta k \ll 1$ and large horizon $\delta \ll h$, the robot rotation should be equal to the trajectory rotation, or more precisely

$$\omega\tau_{\text{rot}} = \Delta\alpha \approx k\delta = \frac{\omega}{v}\delta. \tag{3.11}$$

Therefore, in order for the controller to accurately follow the trajectory, we require that

$$\delta \approx \tau_{\text{rot}} v. \tag{3.12}$$

We verify in Section 3.7.7 that this in fact the best choice.

3.6 Experimental setup

3.6.1 Scenarios

We investigate the behavior of robots that navigate, using the algorithms described in Sections 3.3 and 3.4, in four scenarios.

Cross robots are initially randomly placed, and divided in two equally-sized groups; robots of each group need to travel back and forth between two targets located at the opposite vertices of a square with an edge of 3.4 meters. This creates a crossroad in the center where robots frequently need to adjust their trajectories in order to avoid collisions (Figure 3.7).

Circle robots are initially placed at regular intervals along a circumference with a given radius, and are tasked to reach a target at the diametrical point (Figure 3.8); because robots start moving at the same moment, crowding at the center of the circle occurs. This is a commonly used benchmark in related works.

Corridor two groups of robots, initially randomly placed, travel towards opposite directions along a straight corridor with finite width (Figure 3.15). The ends of the corridor "wrap around" and connect to each other, as if the corridor was wrapped around a cylinder; this simulates a corridor of infinite length. This setup is commonly considered in crowd analysis literature [104].

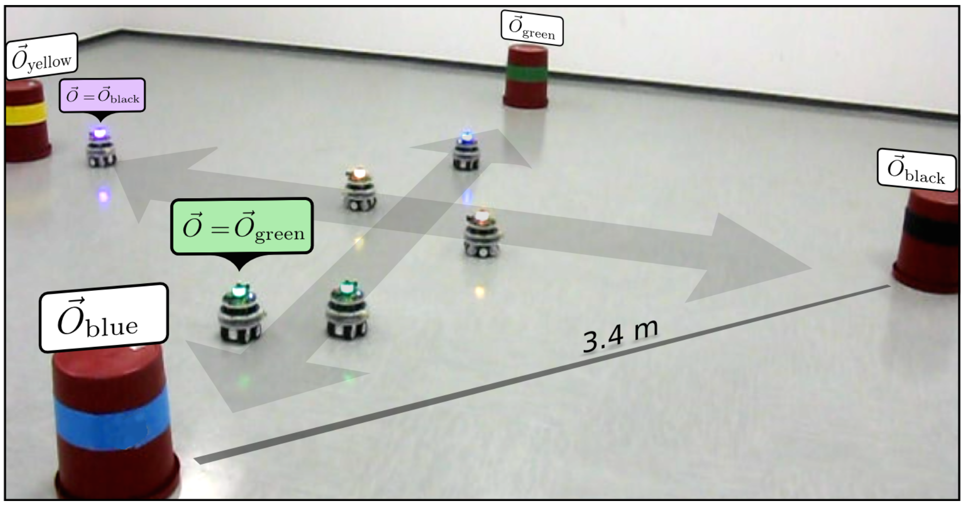

$\vec{O}_{\mathrm{yellow}}$
$\vec{O}_{\mathrm{green}}$
$\vec{O} = \vec{O}_{\mathrm{black}}$
$\vec{O}_{\mathrm{black}}$
$\vec{O} = \vec{O}_{\mathrm{green}}$
$\vec{O}_{\mathrm{blue}}$
3.4 m

Figure 3.7. Six foot-bot robots at play in the *Cross* scenario.

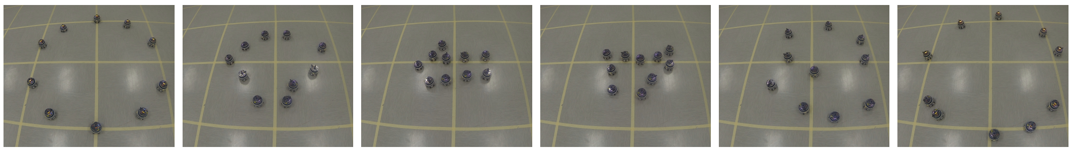


Figure 3.8. 10 foot-bot robots perform one run of the *Circle* scenario with radius 2.4 m. From left to right, images are taken after 0, 4, 8, 12, 16, and 22 seconds.

Indoor The robots move in a synthetic indoor map (Figure 3.17), where optimal trajectories are compute following the methods introduced in Chapters 1 and 2. Trajectory are constrained to pass in the middle of the shorter corridors where they are smoothly joined to form closed loops. Robots are randomly attributed to one of the different loops and randomly placed along their target trajectory, which they start to follow without interruption.

3.6.2 Robots and sensing

We have implemented on real *foot-bots* robots the navigation behaviors described in Sections 3.3, 3.4, together with a simple higher-level planner required to accomplish the tasks described in Section 3.6.1. The *foot-bot* robot (Figure 3.4) is a small mobile platform, directly derived from the *marXbot* [18], specifically designed for swarm robotics [38]. The robot is 30 cm wide and 20 cm tall, and is based on an on-board ARM-11 processor programmed in a Linux-based operating environment. Differential-driven motorized tracks allow mobility at speeds up to 30 cm/s.

Foot-bots use two distinct sensing modalities: *a*) an IR-based *range-and-bearing* sensor and communication system, which allows a robot to detect its *line-of-sight* robot neighbors within a 4 m range and estimate their relative distance and bearing; each robot also advertises its current speed and relative bearing to neighbors through the same system; *b*) a forward-facing camera with a $2\phi = 90°$ field of view and a resolution of 128×92 px, which is used for localizing neighbors (humans and other foot-bots), colored target markers, and walls at 25 frames-per-second.

Because our main focus is on navigation behaviors and not on sensing, we use straightforward techniques for processing camera images: entities of interest, (e.g., landmarks used to identify a destination point, or humans) are marked with differently colored bands at a known height from the floor. Robots convert each frame to the HSV color space, and segment pixels corresponding to each object. After performing connected component analysis, this results in a set of binary blobs. From the image coordinates of each blob's centroid, the robot computes distance and bearing of the corresponding entity by means of a homography transform, which can be estimated in advance given that the camera parameters and height of each entity are known. The velocity of neighbors is estimated as a finite difference, after smoothing position readings with a moving average filter defined over a period of 0.5 s. Note that the position of path markers (i.e., destination points) is sensed online through vision, and not given by an external observer.

3.6.3 Simulation

In addition to the real robots implementation, we developed a custom simulator (see Appendix B) for performing large-scale experiments with different kinds of agents that comprise: foot-bots, holonomic robots, and humans.

In real robot implementations, perception of the environment (i.e., positions of navigation targets and of other robots) is commonly affected by major sensing inaccuracies, which in turn affect navigation performance and safety. To address this issue, in our simulations we consider two different sensing models. A *perfect sensing* model, in which all robots within an assigned range are perfectly detected, and a *realistic*, camera-based sensing model, in which neighbors are only perceived when not occluded and within a given angular field of view (centered on the direction the robot is currently facing).

Simulated vision sensor readings approximate the statistical properties of localization errors from monocular, catadioptric, or stereo cameras. That is, precise and uniform bearing resolution but large uncertainty in depth estimation, which increases for objects farther away. More specifically, given an obstacle whose ground truth relative position is expressed in robot-centered polar coordinates as (ρ, θ), the observed position (ρ', θ') is given by $\theta' = \theta + \phi e$; $\rho' = \rho + k\rho\phi e$, where: $e \sim \mathcal{N}(0, \sigma)$ models the localization error in the normalized image space, ϕ denotes the camera field of view, and k is a constant depending on the characteristics of the depth estimation approach. In the following, we set $\sigma = 1/128$ (i.e., 1 pixel on a 128×96 sensor) and $k = 10$, which well fits the errors observed in real robots. We can evaluate the impact of sensing errors by tuning the σ parameter. As in the real robot implementation, velocity vectors are estimated as a finite difference. Simulated range and bearing sensors, which model well other sensing modalities like laser, ultrasound, time-of-flight or structured illumination are instead characterized by constant angular resolution for bearing and distance-independent uncertainty for range (within maximum limits) and a constant probability (set as 80%) for the message to be received.

We verify in Section 3.7.3 that indeed our simulation models the real navigating foot-bot accurately enough.

3.6.4 Implementation of navigation behaviors

Robot controllers operate on a 0.1 s time step and are not synchronized with each other. At each time-step, the robot updates its belief about the neighboring robots and fixed obstacles, and applies one of the behaviors described in Sections 3.3 and 3.4 to compute the desired heading and velocity. Robots are controlled with ORCA, HRVO and HL as described in Section 3.3.3, by translating the holonomic desired velocity into wheel speeds. In addition, we also consider ORCA-NH, which explicitly takes into account the non-holonomicity of the robots when computing the desired wheel velocities, as discussed in Section 3.4.1.

HL We provide HL robots with our own implementation (see Appendix B) of the behaviors described in Section 3.3. We fix $\tau_{rot} = 0.5\,\mathrm{s}$ and we limit the foot-bot angular speed to $\omega_{\max} = 90\,^{\circ}\,\mathrm{s}^{-1}$ to prevent excessive slipping and camera image blurring. Wheel speed is clipped to $w_{\max} = 30\,\mathrm{cm/s}$. Human motion characteristics are given by $\tau = \eta = 0.5\,\mathrm{s}$ [104]. We maintain $\eta = 0.5\,\mathrm{s}$ and decrease τ to $0.125\,\mathrm{s}$ to obtain a more reactive but still smooth behavior, which moves with increased caution.

HRVO We use an open source implementation [69] of the model described in Section 3.4.1. The desired velocity is found in the velocity space through a linear optimization technique. The implementation does not have free parameters.

ORCA We use an open source implementation [126] of the model described in Section 3.4.1. The desired velocity is found in the velocity space through a linear optimization technique. The time horizon is a free parameter: a large time horizon allows the robot to anticipate crowding and avoid congestion, but at the same time penalizes it with a reduction of speed and a longer, more conservative path. In the following, we select the time horizon with the best performance for each scenario.

ORCA-NH We use the same controller as ORCA, but apply it to the effective center and effective radius (see Section 3.4.1) of the non-holonomic robots.

3.7 Experiments

We investigate how the proposed human-like behavior compares with alternative local navigation behaviors detailed in Section 3.4. In particular, we aim to:

a) investigate how the safety margin affects the navigation safety for robots with error-free omnidirectional sensing and the trade-off between efficiency and safety for robots with realistic sensing (Section 3.7.1);

b) study the impact of imperfect sensing (Section 3.7.2);

c) validate simulated results by comparison with real-robot experiments performed in the same conditions (Section 3.7.3);

d) investigate how the navigation performance for the different behaviors scales with the number of robots (Section 3.7.4), and explore the impact of groups of heterogeneous agents implementing different navigation algorithms (Section 3.7.5);

e) study the emergence of macroscopic group behaviors (Section 3.7.6);

f) study how well the robots follow a prescribed smooth trajectory (Section 3.7.7).

For each experiment, we compute how a given parameter affects a number of performance metrics, detailed below; for each value of the parameter, we perform R simulation runs (replicas), each lasting T seconds after a random initialization. For the *Cross* scenario, $R = 50, T = 900\,\mathrm{s}$; for the *Circle* scenario, $R = 100, T = 100\,\mathrm{s}$; for the *Corridor* scenario, $R = 100, T = 180\,\mathrm{s}$; for the *Indoor* scenario, $R = 100, T = 300\,\mathrm{s}$.

Performance Metrics

We compute the following performance metrics, which quantify different aspects of the robots' trajectories.

Relative throughput indicates the *efficiency* in navigating towards the targets. This measure is defined for the *Cross* scenario as the total amount of targets that the robots were able to reach, divided by the number of targets that the robots could reach in the same time while traveling in straight lines (i.e., ignoring any collision). In the *Circle* scenario, throughput is defined as the minimal time it would take for one robot to reach the opposite side (when traveling in a straight line) divided by the actual time it took. In the *Corridor* scenario, the relative throughput is given by the average speed directed towards the target divided by the maximal admitted speed of the robot. The resulting quantity is a-dimensional, bounded between 0 (worst) and 1 (optimal), and is averaged over all the robots in the simulation.

Relative path length: the total length that the agents have traveled, divided by the length the agents would have covered while traveling in straight lines (i.e., ignoring any collision). This is negatively related to the energetic efficiency of the trajectories.

Path irregularity: the amount of *unnecessary turning* per unit path length performed by a robot; *unnecessary turning* corresponds to the total amount of robot rotation minus the minimum amount of rotation which would be needed to reach the same targets with the most direct path. Path irregularity is measured in rad/m, and is averaged over all the robots in the simulation. We propose this as an objective measure of the *legibility* (see Section 3.2) of the robot's behavior. In fact, it's difficult to infer the intention (the target) of a robot that is changing its direction often. We are currently researching the correlation of path irregularity with the legibility of the robot's behavior and the subjective judgment of its friendliness by humans.

Total number of collisions: the number of collisions per robot per meter of covered distance. Collisions are defined as discrete events, so pairs of agents repeatedly brushing against each other give rise to multiple collision events.

Safety margin violations: for a given robot r, the fraction of time during which at least one agent or obstacle penetrates the safety margin m_s by more than a given amount of space (*violation length*); the value is then averaged over all robots. The value is computed for every *violation length* between 0 and m_s. Compared to the number of collisions, this provides a more descriptive but less concrete measure of safety: for example, it allows to discriminate a case in which the safety margin is frequently violated, but only by a small amount, from a case in which the safety margin is rarely violated, but with robots almost coming into contact.

Line order: a metric computed only in the *Corridor* scenario, where it quantifies, for a given moment, the segmentation of robots of two different groups (corresponding to different optimal speeds or target directions) in longitudinal lines [103]. More specifically, we divide the corridor into narrow longitudinal bands with a width of 30 cm (i.e., roughly twice the width of a foot-bot) and count the number of robots of each group $n_1(B), n_2(B)$ inside a band B: the Yamori band [152] is defined as $Y(B) = \frac{|n_1(B)-n_2(B)|}{n_1(B)+n_2(B)}$. The line order O_L is defined as the average Yamori index over all bands. O_L is bounded between 0 and 1 (representing a perfect organization of the swarm classes in longitudinal lines).

Hausdorff distance:

$$d_H(\gamma, \gamma_t) = \max\left(\max_{\boldsymbol{x}\in\gamma} \min_{\boldsymbol{y}\in\gamma_t} |\boldsymbol{x}-\boldsymbol{y}|, \max_{\boldsymbol{y}\in\gamma_t} \min_{\boldsymbol{x}\in\gamma} |\boldsymbol{x}-\boldsymbol{y}| \right),$$

between the robot's trajectory γ and a target trajectory γ_t, quantifies deviations from the path.

Discomfort: the integrated squared magnitude of the jerk $\mathscr{J}$ quantifies the smoothness of the trajectory (see Section 2.3.1). Agents that would artificially reduce their speed while following a geometrical trajectory, would substantially decrease $\mathscr{J}$. We get a better metric $\bar{\mathscr{J}}$ when we divide $\mathscr{J}$ by the fifth power of the efficiency because it become independent of the mean speed (see Equation (2.3)).

3.7.1 Safety

Ideal sensing If the robots would have perfect omnidirectional sensing and implement the HL behavior, a lower bound M_s for the safety margin parameter m_s would give a theoretical guarantee of collision-free behavior as discussed in Section 3.3.3 ($M_s \approx 15\,\mathrm{cm}$ for foot-bots). In Figure 3.9:Left, we compare the *safety margin violations* metric for different behaviors, with the safety margin set to $m_s = 20\,\mathrm{cm}$, for simulated robots with ideal sensing.

The ORCA-NH variant is safer than plain ORCA; yet HL is the safest behavior and a safety margin $m_s = 10\,\mathrm{cm}$ minimizes the probability of collisions for HL.

Realistic sensing When sensing is not ideal and not omnidirectional, safety cannot be theoretically guaranteed by any of the considered behaviors; for example, collisions may occur between pairs of robots traveling in parallel to each other — like those that happen shoulder to shoulder in human crowds — because of missing lateral view. In Figure 3.9:Right, we report the trade-off between safety and efficiency, determined by safety margin m_s, in case of sensing parameters ($2\phi = 90°, \sigma = 0.008$) that models well real foot-bots. Choosing a larger value for m_s improves safety, but hinders efficiency. The HL behavior performs well (less than 10 collisions per km and large throughput) for any safety margin. In contrast, safety of ORCA strongly depends on the choice of m_s. The performance of ORCA-NH is significantly worse than other behaviors. In the following experiments m_s is set to $6\,\mathrm{cm}$ and we do not further report results for ORCA-NH.

3.7.2 Sensing

We study the impact of the quality of a forward-looking camera used by simulated foot-bots to navigate in the *Cross* following the HL behavior. As expected,

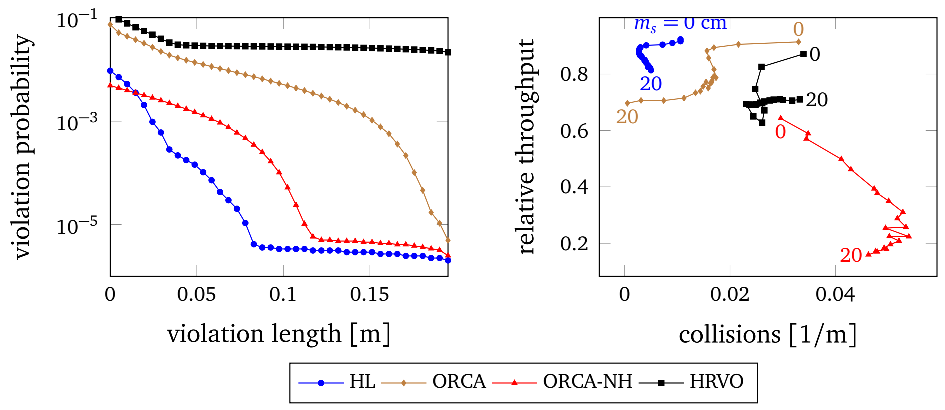


Figure 3.9. Safety in the *Cross* scenario with 20 simulated foot-bots. *Left:* safety margin violation probability (i.e., fraction of time during which a violation occurs by a given margin, per robot, per time step) with ideal omnidirectional sensing and $m_s = 20\,\text{cm}$; *Right*: tradeoff between safety and efficiency with realistic sensing parameters ($\sigma = 0.008, 2\phi = 90°$) averaged over 50 runs.

unreliable sensing (larger error σ) leads to more collisions (Figure 3.10:Left); at the same time, trajectories are less efficient because robots are misled to often change desired heading. A narrow field of view 2ϕ also leads to collisions (Figure 3.10:Right) and generates less efficient trajectories because robots are unable to navigate around crowded regions.

3.7.3 Validation with real robots

Within scenarios *Cross* and *Circle*, we validate the simulation results by comparison with the performance measured in real experiments with foot-bot robots. Results are reported in Figure 3.11. We can observe that the results obtained with real robots in the same conditions closely match simulations.

In the real robot implementation, despite the severe hardware limitations, the navigation controller requires invariably less than 20 ms of computation time per time-step. In simulation, we also tested robustness to time-steps longer than 0.1 s and found that in all considered scenarios, performance begins to degrade only when the time-step exceeds 0.4 s. When larger swarms are considered, the path irregularity increases because robots need to follow more curvy (and longer) trajectories in order to avoid collisions. Even in very crowded scenarios, paths remain smooth and predictable.

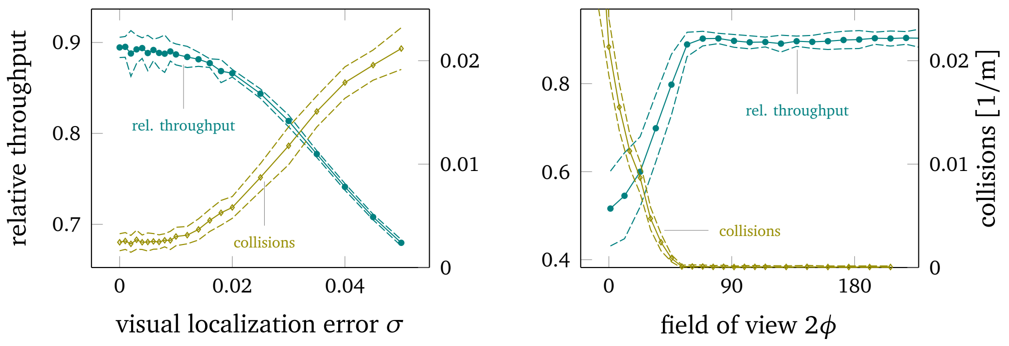


Figure 3.10. Average impact of sensing quality on 20 simulated foot-bots that follow the HL behavior in the *Cross* scenario (dashed lines are at ± one standard deviation over 50 runs). *Left*: fixed field of view $2\phi = 90°$ and variable error σ ($\sigma = 0.008$ corresponds to 1 pixel in a 128×96 image and matches the error observed on real robots). *Right*: fixed $\sigma = 0$ (no visual error) and variable field of view 2ϕ.

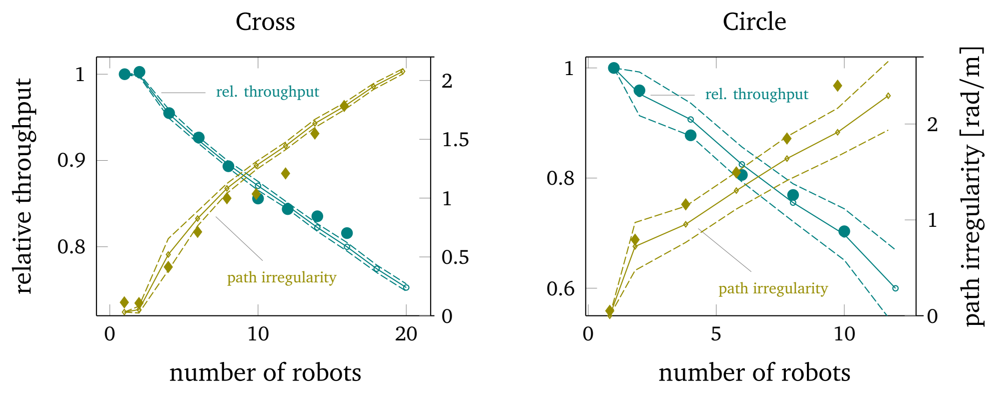


Figure 3.11. Experimental results with real and simulated robots in the *Cross* (*left*) and *Circle* with radius 2.4 m (*right*) scenarios. Large filled markers correspond to results measured on real robots; small markers correspond to simulated results. Dashed lines delimit ± standard deviation over 50 (Cross) and 300 (Circle) simulation replicas. In this experiment, both real and simulated robots use 360° range-and-bearing sensing, $m_s = 10$ cm.

3.7.4 Scalability

Figure 3.12 shows how different behaviors cope with increasingly crowded scenarios. In order to focus on the behaviors, we report the results for simulated robots with ideal sensing. As expected, the performance of all behaviors decreases when more robots are used, because longer and more complicated trajectories are required to navigate around others and avoid collisions. HL outperforms other behaviors in these scenarios, especially when a relatively large number of robots is considered.

In Figure 3.12:Right we compare the computational cost of different behaviors. We observe that the ORCA algorithm is faster than HL, which in turn is faster than HRVO. Note that for all algorithms the computation cost is low enough to run in real time on the embedded CPU of real foot-bot robots.

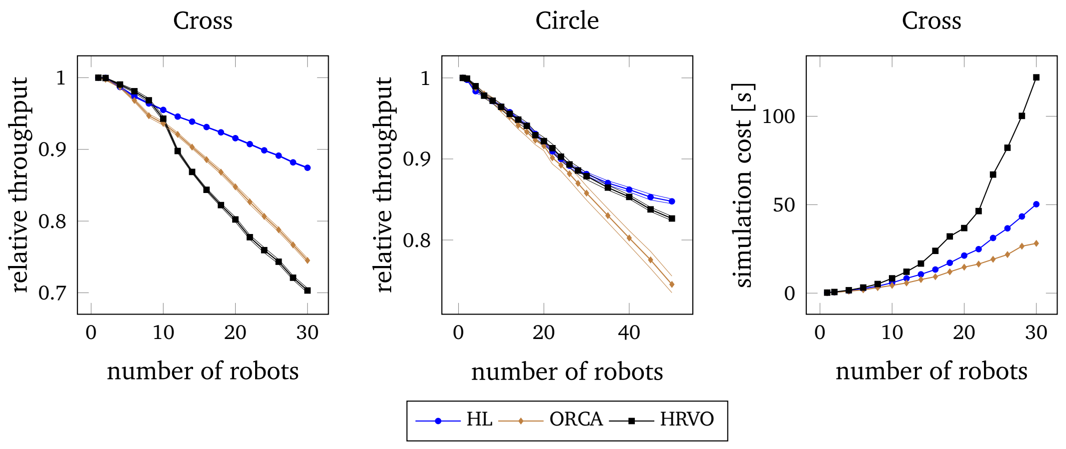


Figure 3.12. Left: Impact of the number of robots on trajectory efficiency in the *Cross* (*left*) and *Circle* with radius 5m (*center*) scenarios, using simulated foot-bots with ideal sensing. *Right*: the time required to simulate 900 s in the *Cross* scenario as a function of the number of robots; simulation time includes the time needed to simulate physics and sensing, but is dominated in the simulation by the time required to execute the navigation algorithm for all robots. Simulation are run on a 2GHz dual core laptop. Narrow lines represent 99% confidence intervals.

3.7.5 Heterogeneous swarms

For a given robot, the performance of an obstacle avoidance behavior is affected by the obstacle avoidance algorithms implemented by other agents. For example, HRVO assumes that all robots implement the same rules and the collision avoidance responsibility is shared between the robots. What happens when some agents implement a different behavior? In the *Cross* scenario, we investigate the performance of a 20-robot swarm whose members are divided into two groups, each following a different navigation behavior.

In Figure 3.13:Left one of the groups implements HL and the other group implements ORCA: we observe that the overall performance is maximized when most robots belong to the HL group (low values on the x-axis); as we move along the x-axis, we increase the number of ORCA robots (and correspondingly decrease the number of HL robots): overall efficiency decreases and average path length increases. A similar pattern in the overall performance is observed in Figure 3.13:Right, where a group implements HL, and the other group implements HRVO.

The performance metrics are also reported separately for each group. We observe that the behavior implemented by neighbors has a limited effect on the performance of HL robots: in particular, the relative throughput of HL robots marginally increases when neighbors switch from HL to ORCA, whereas it decreases when neighbors switch from HL to HRVO; we observe that the behavior of HRVO robots is generally less proactive than the behavior of ORCA robots (which does not translate to improved safety, as shown in Section 3.7.1): HL robots need a larger effort to navigate around HRVO robots than to avoid the ORCA robots.

Surprisingly, we also observe that ORCA (or HRVO) robots perform significantly better when their neighbors implement HL, rather than in a homogeneous group; HL robots directed to the same target tend to form short-lived, compact lines, which are not observed in other behaviors; these dynamic structures allow for easier and more efficient navigation also for other robots, and may explain for the performance difference.

3.7.6 Emerging collective behaviors

Recent research [66] in the field of anthropology has shown that groups of pedestrians exhibit in certain scenarios specific *emergent collective behaviors*; for example, when a corridor is traversed by a lot of people traveling in opposite directions, people tend to self-organize in longitudinal flow lines [104].

We investigate whether robot navigation algorithms also lead to emergence of collective behaviors in two specific scenarios, where such behaviors are favorable, namely *Circle* and *Corridor*.

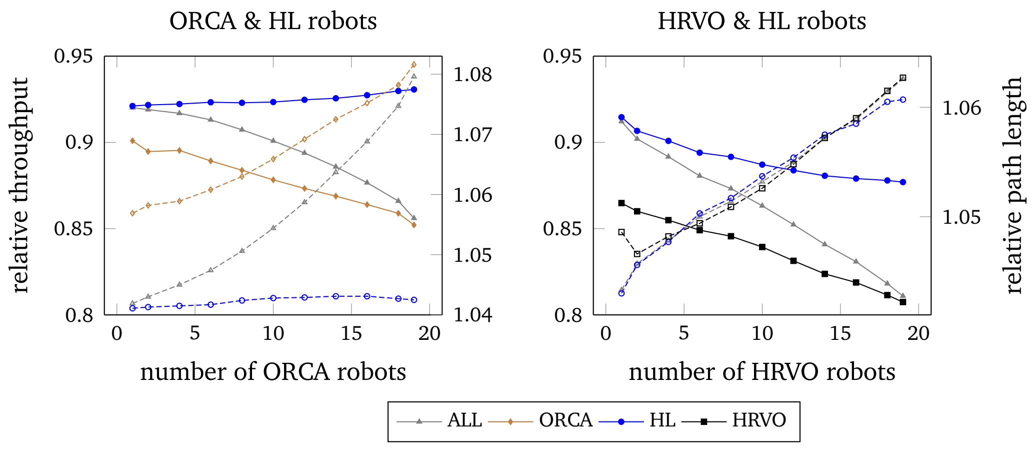


Figure 3.13. A swarm of 20 simulated robots with ideal sensing divided into two groups that implement different navigation behaviors in the *Cross* scenario: ORCA & HL (*left*), HRVO & HL (*right*). We report the average relative throughput (solid lines) and the average relative path length (dashed lines). The gray lines (labeled ALL) report the metrics computed for all members of the swarm; lines labeled HL, HRVO and ORCA represent the metrics computed only on robots implementing the corresponding algorithm.

Circle

Figure 3.1 shows the trajectories followed by 40 simulated HL robots in the *Circle* scenario (radius 5 m) with $\phi = 50°, \sigma = 0.008$.

One can notice that most robots tend to deviate from the straight path to the target, by passing by the same side with respect to the center of the circle — akin to cars in a roundabout. This is a very efficient emergent behavior, which minimizes the need to steer and avoid others; the behavior occurs without any explicit communication among robots, nor any social convention prioritizing steering to one side with respect to the other[3]; in fact, different simulation runs result in different directions of the swirl and both outcomes are equiprobable.

The same behavior is observed, albeit to a lesser extent, with HRVO robots; conversely, ORCA robots tend to follow straighter trajectories which do not exhibit any coordinated behavior. The impact on trajectory efficiency is shown in Figure 3.12:Center, where we observe that the HL algorithm tends to be the best performing option as soon as more than 30 robots are considered, followed by

[3]Conversely, a recent feedback-controlled variant of RVO [124] integrates an explicit preference over which direction to steer around an obstacle, which also causes the emergence of the same collective behavior.

HRVO and ORCA.

In this context, the amount of available sensing information plays an important role on efficiency; we study this relation in Figure 3.14, which compares the trajectory efficiency for different values of sensing range and field of view, for HL (*left*) and ORCA (*right*) robots. We observe that the performance of HL robots abruptly improves when the semi-field of view ϕ exceeds 70° (corresponding to a field of view $2\phi = 140°$), regardless of the range of view; in this case, robots can perceive the neighbors close at their side, which promotes the emergence of the efficient swirling collective behavior. This highlights that the collective behavior stems from an implicit communication occurring among agents by means of their occupation of space (a topic studied by *proxemics* [63]): when agents can not sense each other, such implicit communication does not occur, and collective behaviors do not emerge, yielding worse performance. In comparison, ORCA robots do not manifest such a transition in behavior but steadily increase their efficiency when more information is available.

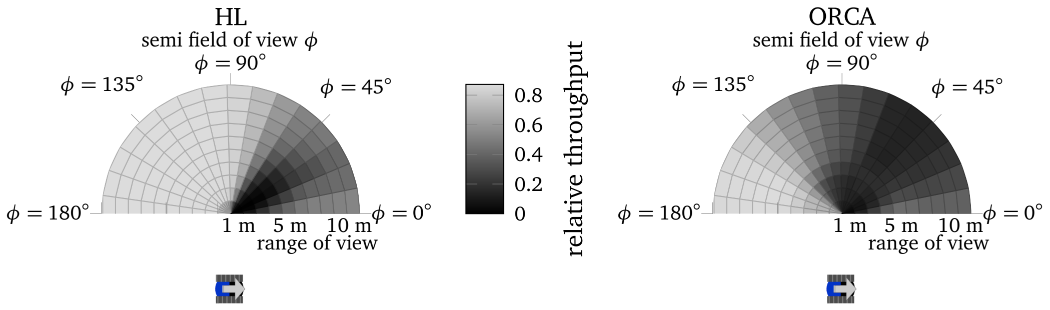


Figure 3.14. Influence of the amount of sensing information on the efficiency of trajectories in the *Circle* scenario with radius 5 m: 50 simulated HL robots (*left*), 50 simulated ORCA robots (*right*). The shade of gray at a given radius and angle represents the relative throughput observed when robots have the corresponding range of view and semi field of view ϕ.

Corridor

In the *Corridor* scenario, HL robots traveling in opposite directions exhibit collective behaviors matching those observed in studies of human crowds [104]: they tend to form ordered *flow lines* (despite being randomly initialized and implementing no explicit rules promoting such behavior), which minimizes the need to avoid others.

Figure 3.16:Left explores the transition from the random configuration at initialization time towards the configuration reached after 40 seconds. We observe that the line order metric (defined in Section 3.7) approaches 1, meaning that robots organize themselves in longitudinal lines of opposite flow. ORCA robots also evolve towards an ordered configuration, albeit more slowly than HL robots. Figure 3.16:Center shows that there is a critical corridor width below which robots are too packed to reach an ordered configuration.

We investigate whether coordination also arises when heterogeneous agents that share the same space. In Figure 3.15 we analyze how 30 humans and 30 foot-bot robots behave within a corridor, where each group is divided in equal-sized subgroups traveling in opposite directions. We observe that after 60 seconds, lines emerge formed by homogeneous agents (i.e., all humans or all robots) traveling in the same direction: this minimizes the need to avoid agents traveling in opposite directions, and also does not require humans to steer in order to overtake foot-bots traveling in the same direction (which are significantly slower). In this experiment, humans are simulated by adopting a realistic sensing model with 150° field of view, have a circular shape with radius 25 cm, and travel at a normal walking speed of 1.3 m/s; they are controlled by means of the heuristic introduced by Moussaïd et al. [103], which is also implemented by the HL algorithm and has been shown [105] to accurately model the behavior of pedestrians for $\tau = \eta = 0.5\,\mathrm{s}$ and $m_s = 0\,\mathrm{cm}$.

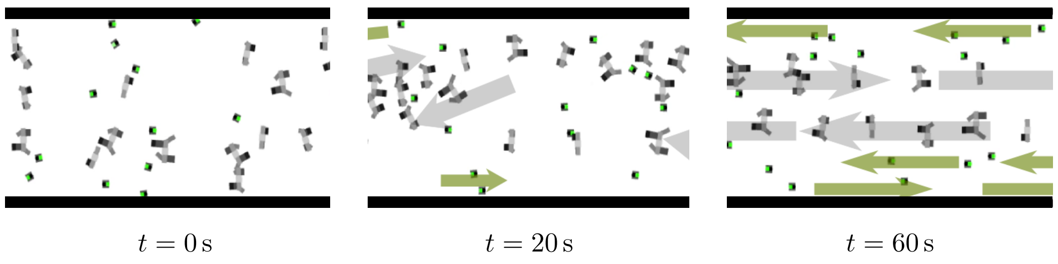


Figure 3.15. Simulation results on how humans and robots (which travel at significantly different speeds of 1.3 m/s and 0.3 m/s) tend to auto-organize themselves in vertical flow lines characterized by homogeneous agents traveling in the same direction

.

Another example is provided by two groups of robots with very different speeds, moving all towards the same direction. Figure 3.16:Right shows how the field of view of the slow robots affects the ability to reach an ordered configuration. When provided with a narrow field of view, slow robots cannot perceive and react to fast robots approaching behind them. In contrast, fast robots frequently need to steer around slow ones. As a consequence, fast robots tend to form ordered but curvy line-like structures, whereas slower robots remain scattered because they rarely need to navigate around obstacles. On the contrary, when slow robots are able to perceive neighbors in a large field of view (e.g., by using omnidirectional cameras or additional back-pointing sensors), both robot types reach a well-ordered configuration. In fact, slow robots are now able to anticipate that they are being overtaken and steer accordingly. This enables the formation of flow lines for both groups, therefore resulting in a very efficient configuration.

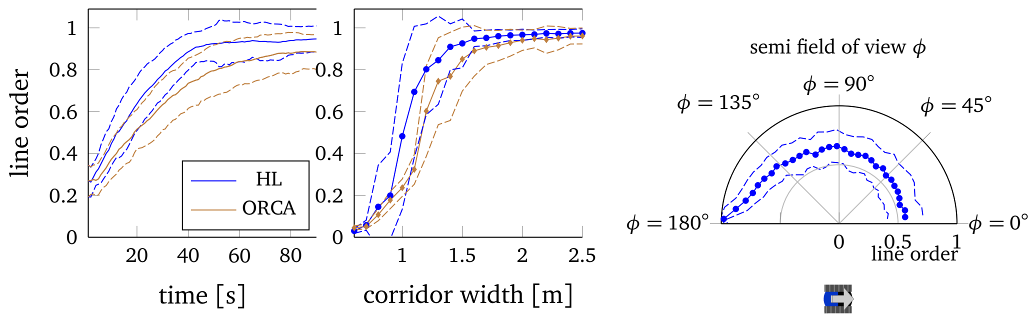


Figure 3.16. *Left* and *center*: 60 simulated foot-bots moving in a corridor in opposite directions: the progressive ordering in lines (*left*, within a corridor of width 2 m) and the dependence of final order (after 180 s) on the corridor width (*center*). *Right*: 30 slow ($v_{\text{opt}} = 10\,\text{cm/s}$) and 30 fast ($v_{\text{opt}} = 30\,\text{cm/s}$) foot-bots traveling in the same direction in a corridor of width 2 m, for different values of the field of view 2ϕ of the slow robots. Dashed lines represent ± 1 standard deviation over 50 runs.

3.7.7 Trajectory following

Single robot Figure 3.17 illustrates the behavior of a simulated robot that is alone in the *Indoor* scenario and follows closed loop trajectories. The results are consistent with the analysis (see Equation (3.12)) that sets $\tau_{\text{rot}} \approx \delta / v_{\text{opt}}$: in this case, the trajectory deviates minimally from the target trajectory. Larger values of τ_{rot} (green traces Figure 3.17:Left), slightly decrease the discomfort cost by ignoring the constraints imposed by the target trajectory and overshooting during turns.

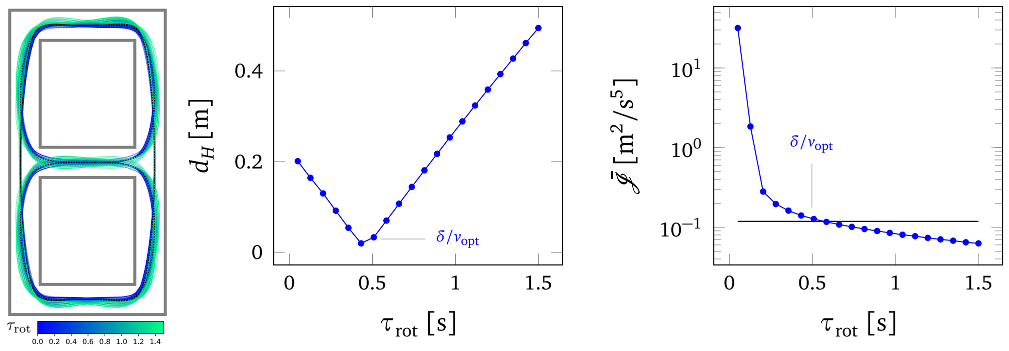


Figure 3.17. Controllers following the target path (black) at 0.5 m/s with no dynamic obstacles (only walls) for varying values of τ_{rot}. *Left*: Resulting robot trajectories. *Center*: Hausdorff-distance d_H between target trajectory and actual trajectory. *Right*: actual cost (blue) compared to the target trajectory's cost (black).

Group of robots We collect in Figure 3.18:Top traces of groups (of different size) of simulated robots in the *Indoor* scenario. We note that HL robots (top row), deviate less from the target trajectory and their trajectories are more predictable, as confirmed by the average Hausdorff distance in Figure3.18:Bottom-Left.

The discomfort cost reported in Figure 3.18:Bottom-Right increases significantly with respect to the target trajectory and the single robot scenario; the increase is due to manoeuvres to avoid collisions that cause sudden velocity changes. This is not surprising: the navigation algorithms we are considering were not designed to optimize smoothness, and should be adapted if an high degree of smoothness is required, such as for robotic wheelchair navigation. Nonetheless we observe that HL has a cost that is several orders of magnitude smaller than ORCA, in part due to the better group coordination we have already noted in other experiments, and in part because the HL behavior modulates the velocity over time τ, which therefore is less affected by sudden changes of desired velocity.

3.8 Discussion

The experimental validation presented in the previous section yields a number of interesting and counter-intuitive results, which we discuss in the following.

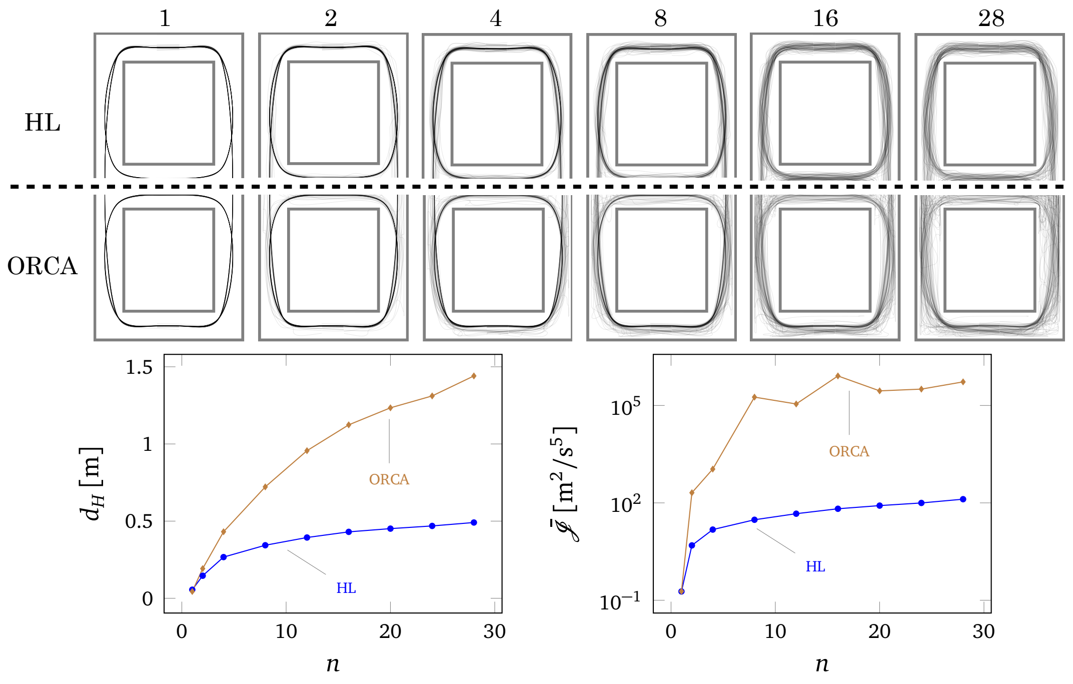


Figure 3.18. Groups of robots following the target trajectory (top left) using either HL or ORCA; results are from 100 randomly initialized replicas. *Top:* Robots' trajectory for different sized groups; to simplify comparison, we split the figure in two parts: HL robots above and ORCA robots below; note that the scenario is symmetric, therefore the two halves look very similar for the same algorithm. *Bottom:* Average Hausdorff distance from the target path (left) and average discomfort cost (right).

Performance compared to alternative algorithms Sections 3.7.2-3.7.5 compared the proposed human-like behavior with two state-of-the-art variants of the Reciprocal Velocity Obstacle (RVO) behavior, namely Hybrid Reciprocal Velocity Obstacle (HRVO) and Optimal Reciprocal Collision Avoidance (ORCA). We focused on the safety and efficiency of the trajectories of simulated robots, and showed that in most considered scenarios the human-like behavior achieves a better trade-off between safety and efficiency, meaning that it's safer for a given efficiency and it's more efficient (and requires a smaller safety margin) for a given safety requirement.

We identify two main factors which may contribute to this improvement.

- The most important factor affecting performance in *Circle* and *Corridor* is the emergence of collective behaviors, which lead to extremely efficient configurations; as shown in Section 3.7.6, these occur earlier and most often in HL than in RVO algorithms, which explains the difference in performance. Notably, HL robots in the *Cross* scenario also tend to organize in compact, short-lived queues formed by small groups of robots heading to the same target; these structures are not observed in other algorithms. Navigating around a compact group of obstacles traveling in a line grants more free space than when such obstacles are moving in an unorganized way: therefore, we hypothesize that the formation of these structures in HL algorithms may contribute to the observed performance differences in the *Cross* scenario. In support of this hypothesis, we recall the results in Section 3.7.5, which show that also ORCA (or HRVO) robots perform better when their neighbors implement HL rather than ORCA (or HRVO). We elaborate further on collective behaviors below.

- An additional factor contributing to performance disparity may directly stem from the algorithms themselves: the most obvious difference (a more detailed analysis is in Section 3.4.2) is that the HL behavior first selects the desired heading — *assuming to adopt the maximum speed* v_{opt} — only later determines a safe speed for such heading; in contrast, RVO approaches operate in the space of velocities and jointly optimize both parameters. In this respect, the HL algorithm determines the desired heading under an optimistic assumption (i.e., the ability to move at optimal speed) than RVO algorithms, which may explain its better efficiency in crowded scenarios. In contrast, ORCA restricts the space of safe velocities far more than HL (Figures 3.2,3.5). We observe that, when heading towards a moving obstacle in a crowded scenario, ORCA robots often reduce speed until enough free space is available, whereas HL tends to navigate around the obstacle while moving fast.

We highlight that all considered algorithms provide local reactive navigation rather than high-level path planning: therefore, in very crowded or constrained environments, the formation of deadlocks can not be excluded without an coordination mechanism like the one discussed in Section 3.3.3.

Requisites for emergent macroscopic behaviors Crowds of human pedestrians, like other biological multi-agent systems, exhibit emergent macroscopic behaviors such as lanes of flow, oscillations, and roundabout-like motion at intersections [66]. In general, a macroscopic behavior emerges when choices made locally by individuals promote the adoption of coordinated choices by neighbors, in a process which progressively propagates to the whole group. One requirement for this to happen is the sharing of information among agents. In the context of navigation, agents communicate *implicitly* by means of their position and motion, and acquire all the information they rely upon (position, velocity and shape of neighbors) through sensing.

Experiments in Section 3.7.6 verify that the ability to sense (and thus, implicitly communicate with) neighboring robots enables the emergence of collective behaviors. More specifically, the emergence of the *swirling* collective behavior in the *Circle* scenario requires coordination (and thus the ability to sense each other) among neighbors traveling side-by-side, and is hindered when the field of view does not allow to perceive such neighbors. In contrast, the formation of lines of flow among agents traveling in the same direction at different speeds in the *Corridor* scenario requires that slow agents coordinate with faster ones approaching from the back, and is only observed when the field of view approaches 360°.

The *Corridor* experiments also show that the emergence of organized structures requires time and some amount of free space. We observe that due to lack of free space, no lanes are formed in a corridor narrower than a critical width. Other critical values for density, v_{opt}, τ, η, or H yield to a with a similar effect [59].

Acceptance of navigation behaviors by humans Acceptance of a robot navigation behavior by humans sharing the same space is influenced by several factors, whose precise definition is still an open scientific question. Several of these factors are dependent on the specific application scenario (e.g., acceptable behaviors in a public sidewalk are likely to be a subset of those acceptable in an industrial setting shared with skilled workers), or related to the robots' physical characteristics (such as size and appearance [72]).

If we focus our attention on the more general properties of the robot *trajectories*, it is reasonable to assume that the key factors influencing acceptance are safety, efficiency, legibility and predictability. A safe robot which does not collide is more acceptable than an unsafe one; an efficient robot, which moves in a rational, goal-oriented fashion is more acceptable (other than more desirable) than an inefficient one. Safety and efficiency are objective, measurable quantities, on which we focused our experimental analysis. In contrast, legibility and predictability [40] are subjective properties, deeply linked to human psychology, whose quantitative measurement in robot navigation scenarios is an open scientific challenge. How legible and predictable are the trajectories generated by robot navigation algorithms?

We argue that the HL algorithm generates legible and predictable trajectories by design. In fact, the HL algorithm implements an heuristic which has been found to well model pedestrian behavior [103]: therefore, we expect that the resulting trajectories are similar to the trajectories that a pedestrian would follow in the same scenario. Such trajectories are legible and predictable for humans, because humans routinely solve local navigation tasks among other pedestrians with minimal cognitive effort. In other words, we claim that a robot implementing the HL algorithm is acceptable because: *a)* it behaves as a pedestrian would behave; and *b)* it is also safe and efficient, as our experimental analysis shows.

Still, the issue to objectively measure acceptance of different navigation algorithms remains open: are RVO algorithms less acceptable than HL? Is it possible to devise an algorithm which is even more acceptable than the behavior an human would have, without compromising efficiency or safety?

We plan to quantify the robots' legibility and predictability by analyzing the trajectories of *people*: in particular, people sharing the environment with legible and predictable robots are expected to follow smooth, rational and efficient trajectories with fewer changes of speed and direction. In order to have a natural (navigation) interaction with people, in addition to a friendly behavior, the robots need to have a similar size and speed; therefore, slow and small robots, like the one we tested in this chapter, are not well suited to study interaction with people.

As an intermediate step to overcome this problem, we are working on an immersive virtual reality scenario (see Appendix B) where a person, using a joypad, moves around among a crowd of robots that display different navigation behaviors. We will measure the (virtual) jerk that users experience while manually driving a wheelchair, compare it with the jerk experienced by the robots, and study the impact of the robots' behavior on the user navigation. It will be an interesting scenario where to test human acceptance of robots and verify if the human-like behavior is indeed more friendlier (at least in a virtual environment).

We also plan to test which effect macroscopic behaviors have on acceptance. We hypothesize that algorithms exhibiting macroscopic behaviors also observed in humans increase the acceptance of robots, because people are used to them and expect robots to act accordingly.

3.9 Conclusions and Perspectives

In this longer chapter, we introduced a local navigation algorithm for ground robots, based on a simple obstacle avoidance heuristic which well models pedestrian behavior [105]. We adapted the heuristic to a robotic context and extended it in order to ensure effective, safe, and smooth behavior in challenging settings. We compared with two state-of-the art reactive local navigation algorithms that share the same goals. We introduced a simple control strategy to follow a target (optimal) trajectory, providing an link with the geometrical planner discussed in Chapter 2.

The algorithm is demonstrated on real robots and on large-scale simulations considering multiple scenarios with different characteristics, in which we measured better performance of the human-like behavior compared to the alternatives: human-like trajectories were safer, smoother, and more efficient. We explored the impact of the parameters on efficiency, smoothness, and safety. We showed that the algorithm can handle heterogeneous agents, such as robots at different speeds, dynamics and sensing ability. The simulated experiments also demonstrate good performance when sharing spaces with humans.

Trajectories produced by the human-like behavior look legible and predictable but we did not yet validate these features through a user study. As a first step towards this goal, we are testing the interaction between a person, manually controlling a virtual wheelchair in an immersive simulated scenario, shared with a group of autonomous robotic wheelchairs.

Jerk, and the related discomfort metric we discussed in Chapter 2, necessarily increases when the robots need to avoid collisions; this is an important consideration in case the human-like navigation behavior is implemented on robotic wheelchairs (or other similar vehicles). In particular, we believe that collision avoidance algorithms that simultaneously yield legible and comfortable trajectories are an interesting, open research question.

Chapter 4

Planning with traversability estimations

4.1 Introduction

In Chapter 1, we discussed how traversability can be modeled by rules that associate spatial attributes and segments' geometry to costs. This approach works well for indoor environments, and, in general, for any structured environment that carries a natural partitioning into cells. Instead, in this section, we focus on unstructured (outdoor) environments and on traversability defined in terms of *traversal probabilities*.

In unstructured environments, or when the agent locomotion is complex, it is difficult to define rules that cover the many reasons a robot may fail to traverse a terrain. For example, consider a four-wheeled mobile robot that is trying to traverse a patch of outdoor terrain: can the robot pass over a tree branch, climb a sand dune, or make its way through a narrow aperture? Ideally, traversability estimates for real environments should also account for uncertainty.

In this chapter, we compute traversability estimates using a classifier that has been trained from the experiences collected by a simulated mobile robot while moving on rough terrain [25]. At runtime, this *traversability estimator* provides a map between terrain patches and the probability $q \in [0, 1]$ that the robot will traverse them safely. In this thesis, we don't focus on perception problems; nonetheless we provide a brief summary of how a traversability estimator is trained in Section 4.3.

4.1.1 Planning according to traversability estimations

Given an estimator that returns (soft) probabilistic information about traversability, how to compute the best sequence of patches to traverse towards a destination? The answer depends on the actual interpretation of the traversability estimator's output: the meaning of the labels used in training and of "traversal probability".

We consider the case of path planning on graphs whose edges have an associated traversal probability. Once the robot is at node n, incident to edge e that has traversal probability $q(e)$, it may turn out that e is in fact not traversable. Four alternative interpretations for this event are meaningful in robotics and are linked to different classes of path planning problems.

1. The robot tries to traverse e but fails, therefore remains at n (and possibly pays a cost). However, attempting to cross e again may work, with the same probability $q(e)$.

2. The robot tries to traverse e, doesn't get stuck, but ends up in a different node than expected.

3. The robot gets stuck while attempting to traverse e, i.e., it can not proceed nor backtrack.

4. The robot observes, before attempting to traverse it, that e is in fact non-traversable: e is removed from the graph and the robot has to plan an alternative path to its target.

In the first scenario, we assume that, with a slightly different speed or initial conditions, the robot may be able to pass, and that the robot may retry to pass the edge until successful. The lower the traversal probability q, the more times the robot will have to try before succeeding. The cost, measured as time or energy, to traverse a patch is then a stochastic variable whose distribution depends on q. This yields a path planning problem with stochastic costs, which is widely researched [137] and has many real-world applications. If we look for the path with lowest expected cost, the problem translates to a shortest path problem [88].

In the second scenario, the robot does not experience a complete failure and may be able to reach the goal along the same route. We would actually need to fix more details to define a sensible planning problem. For instance, if the estimator could predict the pose where the robot will end up, we would formulate the problem as a Markov Decision Problem [6]. In this interpretation, the stochastic outcome of actions is the sole source of uncertainty, while spatial knowledge is complete, which is the opposite interpretation as for the Canadian traveller problem that we discuss below.

The last two scenarios are the topic of this chapter.

4.1.2 Risk-aware path planning

The third scenario, or, more generally, when an estimator just returns if a robot can traverse a patch, would force us to act pessimistically. Should the robot take more risk for a reduced effort? The answer depends on contextual information we are missing: what is the task that the robot is doing? how bad is to get stuck and abort the mission? can other robots take on and complete the mission? and so on. Similar to Chapter 1, we are not able to directly compare different costs. The only sensible planning problem would be to find the best paths according to length (or other metrics connected to effort) and risk of getting stuck [159]. These two objectives can be handled with multi-objective optimization techniques [35], which have been applied to path planning for vehicles on uneven ground in previous works [120, 159] (Section 1.2.2 discusses additional related works on multi-objective path planning). We focus on this interpretation, the related multi-objective path planning problem, and its application to real world instances in Section 4.3.

4.1.3 Resilent path planning

Finally, in the last scenario the robot has no chance to pass even when reattempting. In this case, we assume that the estimator uncertainty originates from lack of information which the robot is able to discover once it arrives on site. The resulting planning problems are instances of the Canadian traveller problem (CTP) [8, 113], or stochastic path planning *with recourse* [117].

In winter, Canadians face the following problem. They want to reach a destination but some roads may be blocked by a deep cover of snow. At the time they leave home, they don't know which routes are free. They may have beliefs about the state of some roads based on their location, their importance, or their state during past winters. They know that they may have to backtrack if they discover that a road is indeed blocked. Which road should they pick based on their current belief? That is, which navigation policy should the Canadian traveller follow?

Not just Canadians need to reason about uncertain traversability information. Imagine that a fire alarm triggers. We look around and see smoke around us. We have a couple of possible routes to exit the building. We are unsure about which one to take: maybe some doors are blocked or some passage is no more viable.

Changes of directions, each time we get blocked, would cost us precious time. Do we go for the shortest route? For the route with lowest blockage risk? How to balance backtracking risk and route length?

Robots are confronted with the same problem if they are able, using local sensing, to refine traversability estimations on the spot. When they arrive near an area marked on their map with uncertain traversability, they inspect it with sensors and reveal information such as: the bridge is indeed fallen; the grass is too wet; the slope is too steep. The robot, after an observation, becomes sure that an area is traversable or not.

Formally, the Canadian traveller problem (CTP) searches for the optimal policy to navigate a graph with some *hidden edges* of unknown traversability: an information that will be revealed only upon arrival on an incident node. In this chapter, we do not look for better algorithms to approximate or compute the optimal policy. Instead, we are interested in the actual cost than an agent will pay when following the policy on a given map realization.

In Section 4.4 we introduce a model to study the impact of traversability estimations on CTP policies to tackle the following questions. When using a very accurate estimator, is it worth to account for small uncertainties? Can we rely on optimistic heuristics that treat as traversable all edges tagged with sufficiently high traversal scores? On the other end, when the estimator accuracy is very low, is it worth trusting the estimations at all? Are we not better served by a pessimistic policy that would simply assume that all hidden edges are not traversable and therefore choose the path with highest traversal probability?

To answer these questions, and more generally model the relation between traversability estimation and cost of a policy, we simulate a noisy estimator, i.e. an imperfect, realistic classifier. Then, we measure the cost of policies, on real and randomly generated maps that store traversal probabilities sampled from (hidden) probability distributions.

4.2 Related work on the Canadian traveller problem

Computing a policy for the Canadian traveller problem that minimizes the expected path cost is a #P-hard problem [46]. If we frame the problem as a POMDP and use Value Iteration to compute the optimal policy, computation cost growths $o(3^n)$ with the number n of hidden edges. This makes practically impossible to solve CTP for more than a few tens of hidden edges, except for particular types of graphs, like DAGs, for which exact policy have been explicitly formulated [107].

Researchers have developed heuristics, based on Monte Carlo Sampling, to approximate the optimal policy [41, 127]. Another method uses heuristics to speed up search of solutions in AND-OR trees [44].

Extensions of CTP account for remote sensing [16] and multi-agent systems [17]. Adjusting traversability beliefs along the path, according to Gaussian Processes, allows to model realistic problem, where uncertainty on different edges is not independent [37].

In this chapter, contrary to the mentioned related works, we do not investigate how to efficiently compute or approximate the optimal policy for CTP; instead, we study the impact of uncertain estimations on the *quality* of policies in Section 4.4.

4.3 Risk-aware path planning

In this Section, we assume that a binary classifier estimates the probability that a patch of terrain is traversable by a particular robot.

Training a traversability estimator In previous work [25], we presented the following method to train a traversability classifier (illustrated in Figure 4.1): (1) we generate a wide spectrum of synthetic terrains; (2) using an accurate simulator, we test if the robot, spawn at random locations, can move for at least Δ units of length, and use the result to label a patch of terrain as traversable or non-traversable; (3) we train a Convoluted Neural Network on a large dataset of labelled terrain patches, which, given a new patch, will output a traversability score $q \in [0, 1]$, which we interpret as a probability. We tested [26] the classifier estimation's quality on synthetic maps (AUC $= 0.926$) and on real terrain maps (AUC $\in [0.819, 0.961]$). On rough terrain, a differential-driven wheeled robot may not be able to rotate in place everywhere. Therefore, we extended the classifier so that it also estimates the probability that the robot can rotate in place by at least 45°, clock or counter-clock wise (with AUC $= 0.926$ on synthetic maps and AUC $= 0.834$ on real terrain maps).

Next, we introduce a related path planning problem and apply this classifier to compute optimal paths for a mobile outdoor robot on rough terrain.

4.3.1 Problem formulation

Given a map of a terrain, a robotic agent, start and a goal poses, and a binary probabilistic traversability estimator that assign a probability q to the event that the robot will be able to traverse or turn on a patch of terrain, find the best set

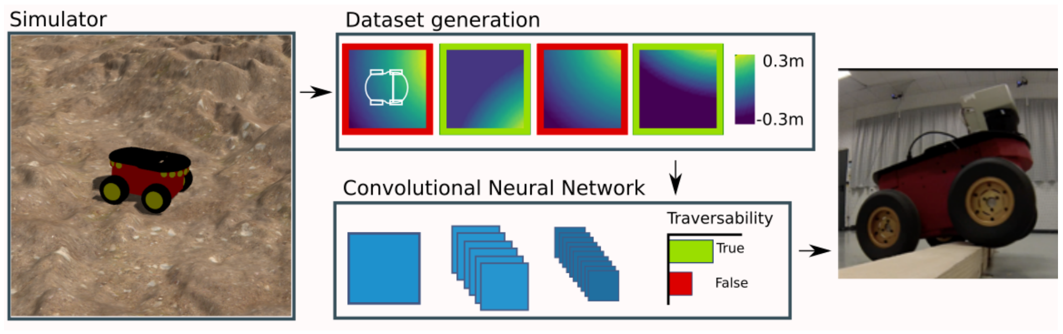


Figure 4.1. The robot model runs in simulation on procedurally generated terrains (left) to generate datasets linking heightmap patches with their traversability (top); on these datasets we train classifiers to estimate the probability that a given heightmap patch is traversable or not (bottom). The learned classifier correctly predicts terrain traversability for a real robot (right).

of paths $\{\pi\}$ from source to target that take into account that the robot may get stuck.

Planning graph

The robot, a Pioneer 3-AT (see Figure 4.1:Right), is a skid-steer outdoor robot with four wheels (i.e., wheels on the same side turn at the same speed).

We compute paths on a graph $G = (N, E)$ of nodes regularly distributed (every $\Delta = 18\,\text{cm}$ and $45°$) on a horizontal grid of poses. Edges $e \in E$ comprise: (1) rotations in place by $\pm 45°$, and (2) segments connecting neighboring poses of the same orientation. For every edge, we compute length and traversability score $p(e) = q(\text{patch}(e))$, which is the result of the traversability/turnability estimator q applied to a patch(e) centered at the edge's origin and oriented along the edge (see Fig. 4.3:Top-right).

As an alternative to a grid graph, in order to take into account kinematic constraints for more complex robots, we can generate the graph using sampling methods such as Probabilistic Road Map (or its variations that have better spatial coverage, such as PRM* [76]).

Costs of a path

Similar to Section 1.3.5, we introduce a cost that measure the traversability of a path $\pi \subset E$, as the fraction of robots which would arrive at a destination if they would all follow the same path, i.e., the *survival* [125] $S(\pi)$ at the end of the path.

We assume that traversal probabilities of (non overlapping) patches are independent, or in other words, that the traversability along a path is Markovian. This allows us to compute the probability to traverse a path composed of edges $(e_1, \ldots, e_n)$ as $S(\pi) = \prod_{i=1}^{n} p(e_i)$, where $p(e_i)$ is the probability to traverse a single edge; then the path risk $\mathscr{R}(\pi) = -\log S(\pi)$ is additive, i.e., it is given by the sum of the edges' risk $-\log p(e_i)$.

The planning problem is framed as a two objective optimization path planning problem over $G = (N, E)$ with respect to length ($\mathscr{C}_1 = \text{length} \geq 0$) and and risk ($\mathscr{C}_2 = \mathscr{R} \geq 0$).

4.3.2 Approximated convex-hull of the Pareto front

As already discussed in Chapter 1.3.7, there are potentially a very large number of Pareto optimal paths. For navigation graphs derived from indoor maps, we could compute the whole set in a short time using Algorithm 4. Instead, for fine-grained outdoor maps, like the grid graph G, we cannot afford to compute the whole set. The general approach is instead to compute a subset of the Pareto set that covers enough the set of strategies that a ration agent would use [138].

We present one of these approaches, which restricts the choice to linear strategies. A linear strategy find the best path according to a weighted cost function

$$\mathscr{C}(k) = (1-k)\mathscr{C}_1 + k\mathscr{C}_2,$$

for some $k \in [0, 1]$. Their solutions $\pi(k)$ range from the shortest, but often non-traversable, path $\pi(0)$ to the safest, but longer, path $\pi(1)$ (see Figure 4.3). Solutions from the set of linear strategies are points on the convex hull of the Pareto front, and are denoted as *non-convex -dominated* [99].

In the worst case, the convex hull has the same size as the Pareto front, therefore we want to limit computational costs by searching a representative subset, an idea shared by many approaches. Cost are not normalized and solutions may be very unevenly spread in cost space, which is a common problem [32] that we overcome with a simple algorithm that is scale-invariant (under multiplication of costs by different factors), similarly to [33] but in our case limited to non-convex-dominated solutions.

Algorithm 5 applies a divide and conquer approach that iteratively subdivide a set of linear strategies with $[a, b]$ in $[a, k] \cup [k, b]$, where k is chosen such that optimal solutions for strategies a and b have the *same* cost, i.e., $\mathscr{C}(k)(\pi(a)) = \mathscr{C}(k)(\pi(b))$, as illustrated in Figure 4.2. The search starts from $a = 0$ and $b =$

1 and stops when solutions $\pi(a)$ and $\pi(b)$ have almost the same costs (by a factor $\epsilon > 0$). The algorithm returns (non-convex-dominated) solutions that corresponds to the knots of this partition.

The set returned by Algorithm 5 may still be too large for a strategic decision maker. Therefore we (optionally) prune all solutions that are almost-dominated [42] by a factor $\eta > 0$ (using brute-force comparison).

```
ApproximatedParetoCH($G, s, t, \mathscr{C}_1, \mathscr{C}_2, \epsilon$)
$a \leftarrow 0$
$b \leftarrow 1$
**return** PartialApproximatedParetoCH($G, s, t, \mathscr{C}_1, \mathscr{C}_2, a, b, \epsilon$))

PartialApproximatedParetoCH($G, s, t, \mathscr{C}_1, \mathscr{C}_2, a, b, \epsilon$)
$\pi_a \leftarrow$ ShortestPath($G, s, t, \mathscr{C}_1, \mathscr{C}_2, a$)
$\pi_b \leftarrow$ ShortestPath($G, s, t, \mathscr{C}_1, \mathscr{C}_2, b$)
/* If their costs are similar, stop the iteration */
**if** $\mathscr{C}_2(\pi_a) \leq (1+\epsilon)\mathscr{C}_2(\pi_b) \wedge \mathscr{C}_1(\pi_b) \leq (1+\epsilon)\mathscr{C}_1(\pi_a)$ **then**
    **return** $\{\pi_a, \pi_b\}$
**end**
/* $k$ is an optimal trade-off when both solutions have the
   same cost */
$k \leftarrow \frac{\mathscr{C}_1(\pi_a) - \mathscr{C}_1(\pi_b)}{\mathscr{C}_2(\pi_b) - \mathscr{C}_2(\pi_a) - \mathscr{C}_1(\pi_b) + \mathscr{C}_1(\pi_a)}$
$\Pi_a \leftarrow$ PartialApproximatedParetoCH($G, s, t, \mathscr{C}_1, \mathscr{C}_2, a, k$)
$\Pi_b \leftarrow$ PartialApproximatedParetoCH($G, s, t, \mathscr{C}_1, \mathscr{C}_2, k, b$)
**return** $\Pi_a \cup \Pi_b$

ShortestPath($G, s, t, \mathscr{C}_1, \mathscr{C}_2, k$)
/* Results are cached to avoid re-computations */
**return** the shortest path between $s$ and $t$ according to cost $\mathscr{C}(k)$
```

Algorithm 5: Approximation ($\epsilon > 0$) of the convex hull of the Pareto front of all paths between s and t on G w.r.t. positive cost functions $\mathscr{C}_1$ and $\mathscr{C}_2$.

Example on a real map. Figure 4.3 illustrates solutions of the multi-objective problem on a real map where we applied our estimators. The shortest path (red) is Pareto-optimal, but has a very low traversal probability. The path with the lowest risk (green) is also Pareto-optimal, but may be unnecessarily long; between the two extremes, a potentially very large set of Pareto-optimal paths exist, spanning the trade-off between risk and length.

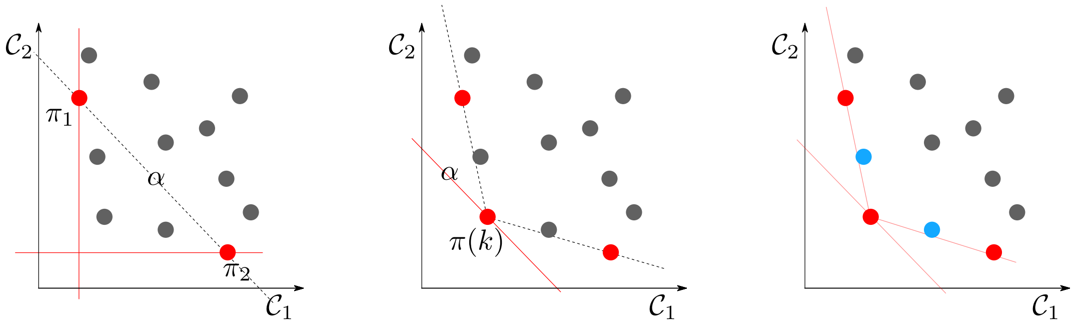


Figure 4.2. An illustrative application of Algorithm 5 in cost space. *Left*: the algorithm starts by computing $\pi(0)$ and $\pi(1)$ and k (depicted as a line with slope $\alpha = \frac{k}{k-1}$ passing through their costs). *Center*: a new optimal solution $\pi(k)$ is computed that lies on a line with slope α and minimal intercept. *Right*: the algorithm is applied recursively on the left and on the right of $\pi(k)$ until no more (significantly different) solutions are found; the algorithm returns non-dominated solutions (red), ignoring any convex-dominated solution (blue) of the Pareto set.

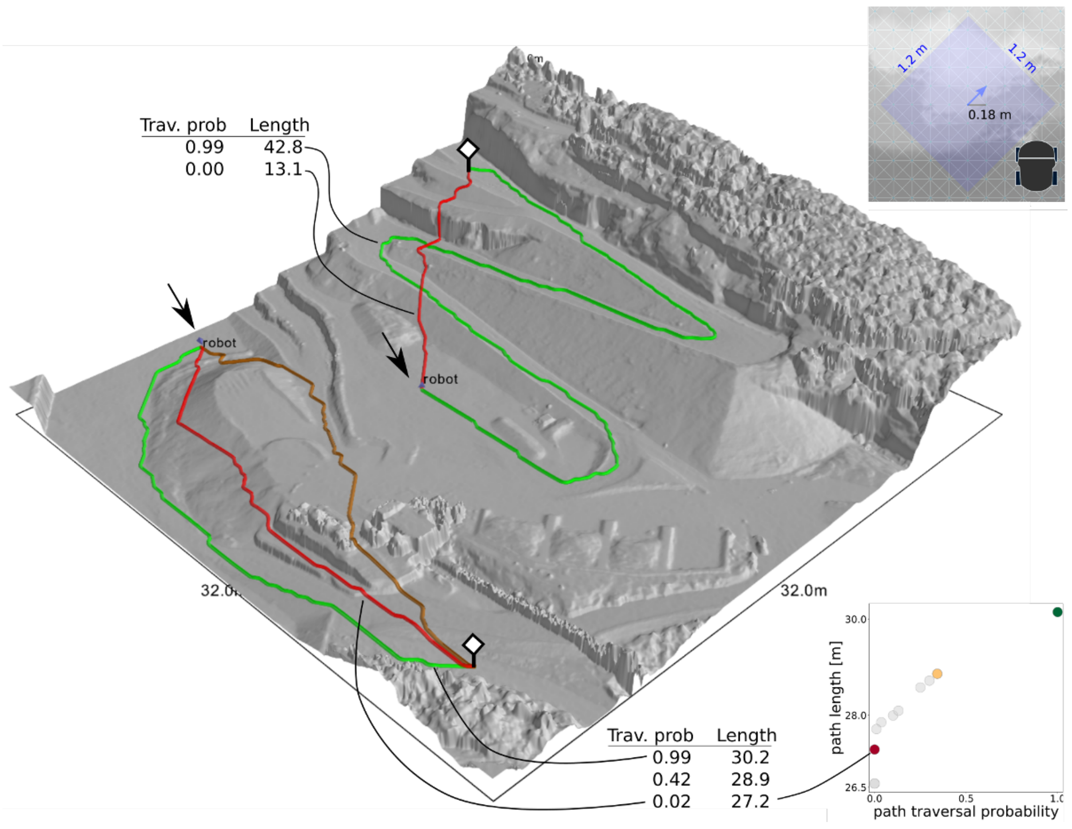
4.3.3 Experiments

We tested our approach on an outdoor grass slope. First we used a Tango device, hand-held at 1 m from the ground pointing downwards, to build a height-map map of the area. Then we compute the navigation graph applying the traversability classifier in any node of N to estimate its risk. The map is about 10 x 10 m large and the planning graph has 29929 nodes and 234800 edges.

Planner performance

We randomly draw 1000 pairs of connected source and target nodes on N, we run Algorithm 5 and measure for different ϵ: (1) the computation cost, (2) the number of computed solutions.

Results Figure 4.4 illustrates the results. Contrary to the indoor maps tested in Section 1.4, there is a smaller dependency of the computational cost on the distance between source on target. As expected, the computational cost decreases for larger ϵ because the search terminates earlier, resulting in a smaller set of solutions. Pruning (almost dominated) solutions significantly reduce their number even for very small η as this eliminates small variations, which may be beneficial to improve robustness with respect to the classifier's noise. On this map, running the planner with $\epsilon \approx 1$ and $\eta \approx 0.1$ results, in average, in 2-4 solutions computed in few seconds.



Trav. prob	Length
0.99	42.8
0.00	13.1

Trav. prob	Length
0.99	30.2
0.42	28.9
0.02	27.2

Figure 4.3. Left: a selection of Pareto optimal paths on a map of a quarry for two pairs of source (arrow) and target (white square) locations. The paths are colored by estimated traversal probability $S(\pi)$ from red (non-traversable) to green (surely traversable). Corresponding values for traversal probability and length of each trajectory are shown in the side tables. *Right top*: a portion of the planning graph on the quarry map. Nodes are placed on a regular grid at 18 cm. The blue edge's traversal probability is estimated by applying the classifier on a 1.2 m × 1.2 m patch (light blue) centered at the edge origin and directed along the edge. Robot's silhouette is shown for size comparison. *Right bottom*: The trade-off between path length and traversability for Pareto optimal paths of the bottom source-target location (note that on the horizontal axis we are plotting $S(\pi) = e^{-\mathscr{R}(\pi)}$ and not $\mathscr{R}(\pi)$, therefore the solutions are not on the convex hull). Colored dots correspond to paths drawn on the left.

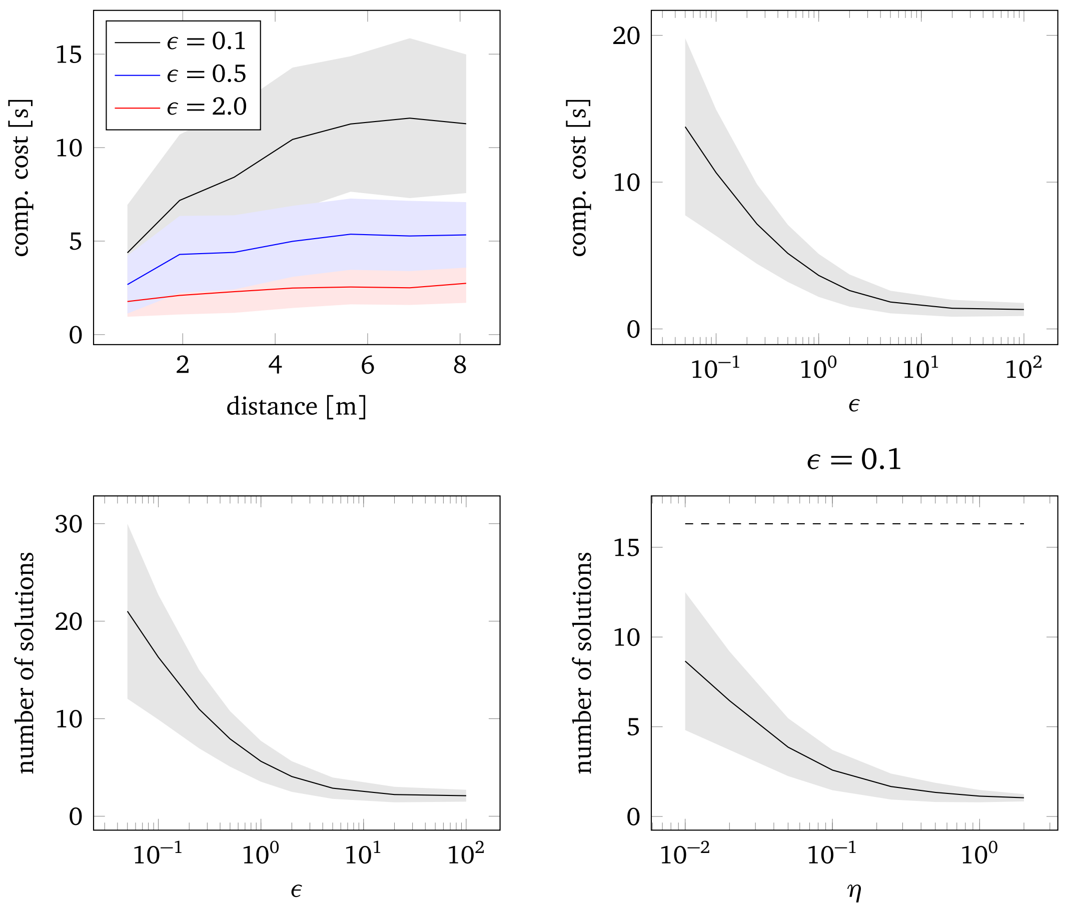


Figure 4.4. Evaluation of the risk-aware planner on 1000 random instances on a 10 m x 10 m outdoor terrain. We report the average over all instances; shaded areas represent ±1 standard deviations. *Top Left*: Computational cost for three value of ϵ versus the distance between source and target. *Top Right*: Computational cost versus tolerance ϵ. *Bottom Left*: number of solutions versus tolerance ϵ. *Bottom Right*: number of solutions after pruning any solution almost-dominated by factor η from solutions computed using $\epsilon = 0.1$ (dashed line).

Real robot experiments

Pioneer 3-AT As a partial test, we have attempted to follow the safest solution found by Algorithm 5 with the real robot. Figure 4.5 illustrates some of the results when the robot starts from the blue silhouette and targets the white markers.

The safest way for the robot to reach the top area (square mark) is to follow the smooth side-walk ramp uphill, avoiding the grass slope which has an irregular shaped terrain. This path is estimated as certainly traversable: we verified this is in fact a traversable path by tele-operating the robot through it.

The maximal-traversability path that reaches the circle mark involves first reaching the square mark uphill on the sidewalk, then heading down on the grassy slope for a short distance: even though it is long, this path is in fact the most rational to reach such point, because traversing the grass slope uphill or transversally is challenging.

The star mark lies on a difficult to reach area in the middle of the grass slope. The location is not reachable from the sidewalk above it, because that area of the sidewalk is flanked by a small step that is correctly estimated to be not traversable. The maximal-traversability path, instead, accesses the grass from a point left to start, then proceeds uphill avoiding obstacles and excessively steep or rugged areas; the path has a traversal probability of 0.21, and we were unable to successfully teleoperate the robot through it because it was blocked by a bump.

The *reachability* map illustrates which parts of the terrain the robot can *reach* from its current pose. For a given target location, it is defined by the maximal traversal probability among all paths from source to target.

NCCR Robotics Part of the work presented in this thesis was developed in the context on NCCR ([Swiss] National Centres of Competence in Research) Robotics. Figure 4.6 illustrates a demonstration performed in collaboration with the other research partners (ETHZ, UNI-ZH and EPFL): a drone (not depicted) built a map that we used to compute safe, short trajectories for a legged robot.

4.4 The impact of the estimator quality on the navigation policy

In this section we analyze the impact of the traversability estimator on the Canadian traveller problem.

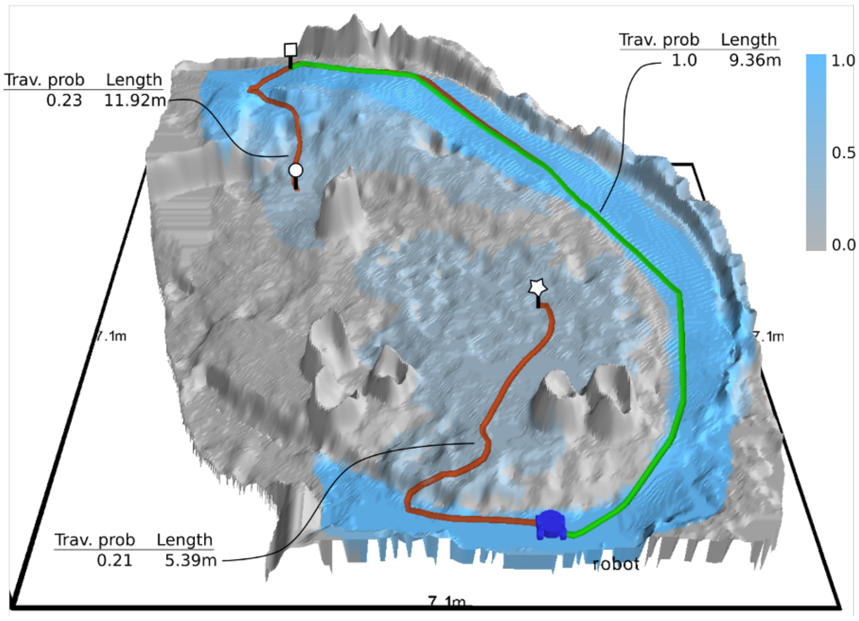


Figure 4.5. Paths of maximal traversability from the robot's initial pose (blue silhouette) to three different goals in the Slope map. Paths are colored according to their traversal probability, from red (low) to green (high). The blue overlay represents the *reachability* map from the robot's initial pose: blue (certainly reachable) to gray (certainly not reachable).

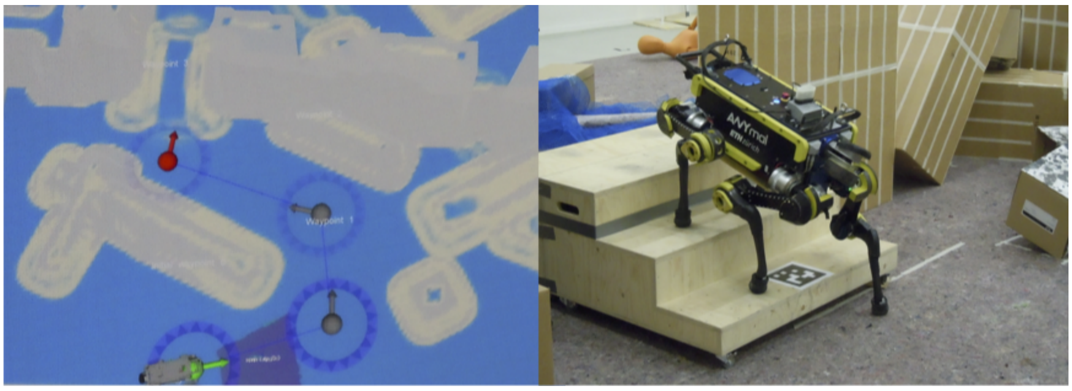


Figure 4.6. ANYmal [71] robot from ETHZ follows one of the solutions, computed by our planner, which was selected by an operator during a search & rescue simulation for NCCR Robotics in November 2017. *Left*: reachability map. *Right*: the robot passing over stairs located above the red marker in the left picture.

4.4.1 Problem formulation

We are given a graph $G = (N, E)$ and an agent that moves on G. At the beginning, the true traversability $r : H \to \{0, 1\}$ (0: not traversable, 1: traversable) of some edges in $H \subseteq E$ is not known to the agent. A non-ideal classifier estimates the traversal probability as $q : H \to [0, 1]$. The agent uses knowledge of q to navigate between a starting node $s \in N$ and a target node $t \in N$ according to navigation policy π. We assume that s and t are connected.

The problem is framed as a POMDP, whose states (n, k) are given by a node $n \in N$ and by the current knowledge $k : H \to \Omega = \{0, 1, \text{unknown}\}$ of the true values of traversability of H. A navigation policy $\pi : N \times \Omega^{|H|} \to N$ selects the next node the agent will travel to according to its state. When the agent reaches a new node n, it observes the true value of some edges $H_n \subseteq H$ and sets $k(e) = r(e)$, $\forall e \in H_n$. In the original CTP, H_n is the set of edges incident to n; here we allow more general mapping between n and H_n depending on the type of maps.

The cost of a policy is defined as the cost of the trajectory from s to t that it generates. The optimal policy π_{opt} is defined as the policy with the lowest expected cost.

If the traversability estimation q is exact, it associates a 100% traversal probability to hidden edges that are in fact traversable, and a 0% probability to edges that are not; then we can expect that π_{opt} will lead the agent to follow the minimum-cost path.

However, in the following we assume q to be inexact, i.e., it may assign non-zero probabilities to edges that are in fact non-traversable. Then, the optimal policy may lead the agent to reach a node where an edge, that would be followed next, is revealed to be non-traversable, and thereafter backtrack to follow a different path. Similarly, if a traversable hidden edge on the minimum-cost path is assigned a probability less than 100%, the optimal policy may prefer a longer path that minimizes the risk to backtrack.

We are interested in how the cost of a policy depends on the quality of the estimations, which we assume are generated by a binary classifier.

4.4.2 Binary traversability classifier

We model q as a stochastic function that assigns scores to hidden edges according to a probability distribution that depends only on the true traversability of the edge. To keep the model simple, we assume that the distribution is symmetric if we exchange the classes; this models the assumption that the classifier works equally well for traversable and not traversable edges

$$p(q|r=0) = p(1-q|r=1) = B_{\alpha,\beta}(q). \quad (4.1)$$

$B_{\alpha,\beta}$ represents the family of Beta distributions, which is well suited to model binary classifiers [15, 21, 83, 115].

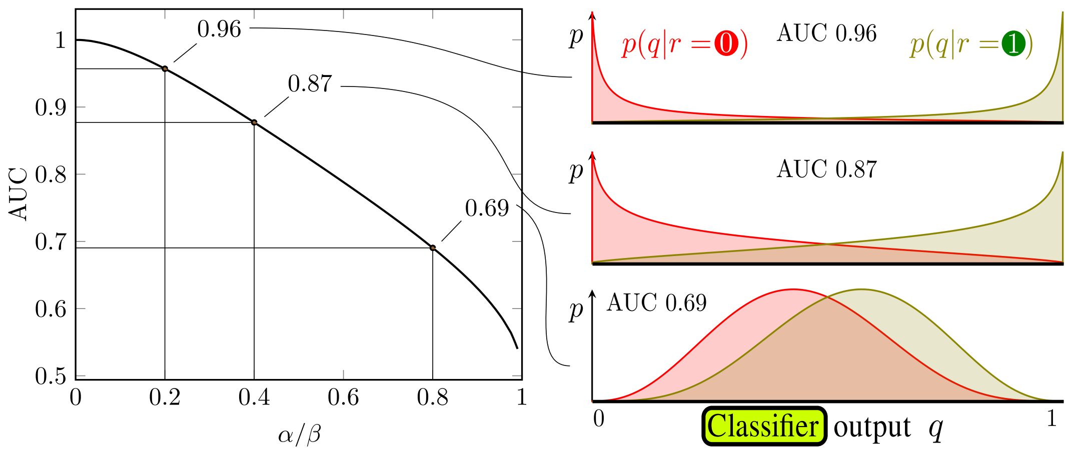


Figure 4.7. *Left*: relation between α, β and AUC for a calibrated classifier ($\beta = \alpha+1$). *Right*: probability distribution of the classifier output $q(e)$ applied to a non-traversable (red) or traversable (green) edge e; for a strong (*top*), medium-quality (*middle*), weak (*bottom*) classifier. Note that the strong classifier returns polarized (close to 0.0 or 1.0) outputs, whereas the weak classifier is uncertain and aware of it (outputs are close to 0.5)

We limit our analysis to *calibrated* classifiers (i.e., $p(r=1|q) = q$), which is the case if $\beta = \alpha+1$: this implies that the score returned by a calibrated classifier can be interpreted directly as a probability by the agent. Intuitively, this means that if we collect many hidden edges for which the classifier returned a given probability q, a fraction close to q of them will in fact be traversable. Several techniques have been developed [43, 157] to calibrate off-the-shelf classifiers, so in the following we focus on calibrated classifiers only.

A calibrated classifier may or may not be accurate. Following the best practices in Machine Learning [15], we measure a classifier's quality via its Area Under the [ROC] Curve (AUC). AUC values range from 0.5 (for a classifier that returns random or constant answers) to 1.0 (for an ideal classifier). Let e_0 be a random non-traversable edge ($r(e_0) = 0$), and e_1 be a random traversable edge ($r(e_1) = 1$). The AUC value $q_{\text{auc}}$of classifier q can be intuitively interpreted as the probability that $q(e_1) > q(e_0)$. The classifier returning exact answers (1.0 for traversable edges, 0.0 for non-traversable edges) is calibrated and has $q_{\text{auc}}= 1.0$.[1] A classifier returning always 0.5 is also calibrated (on a balanced dataset) but has $q_{\text{auc}}= 0.5$, which means that its answers are not informative.

For any $0.5 \leq q_{\text{auc}} \leq 1.0$, there is a single choice of the pair $\alpha(q_{\text{auc}}), \beta(q_{\text{auc}})$ that yields a calibrated classifier (see Figure 4.7).

4.4.3 Optimal and baseline policies

The optimal policy π_{opt} is defined as the policy with the lowest *expected* cost, and is computed using Value Iteration (see [44] for a reference to the algorithm).

We compare π_{opt} with a family of baseline policies $\pi(\tau)$, parametrized by a threshold $\tau \in [0, 1]$. A policy $\pi(\tau)$ is defined as follows:

- in any state (n, k), we consider the subgraph composed of all and only the edges which are either known to be traversable, or unknown with $q(e) \geq \tau$ (this excludes the edges known to be non-traversable and unknown edges with $q(e) < \tau$);

- if at least one path to the target exists in such subgraph, the shortest is computed and the first of its edges is traversed; else, the path with the highest probability of being traversable is computed (regardless of cost) on the full graph, and its first edge is traversed.

For any τ, $\pi(\tau)$ is guaranteed to eventually lead an agent to its target t as long as t is reachable from the source node s. $\pi(\tau)$ defines *reactive* policies which decide which edge to traverse next by making hard assumptions on the traversability of all unobserved edges, but revise these decisions as soon as new edges are observed.

$\pi(0)$ is a baseline *optimistic* policy that strives for the shortest path, ignoring classifier estimations and assuming all unobserved edges are traversable.

$\pi(1)$ is a baseline *pessimistic* policy: it assumes that hidden edges are not traversable unless observed to be traversable. In the (common) case in which this does not yield a path to the target, this policy always chooses the action which proceeds along the path with highest traversal probability, ignoring edge costs.

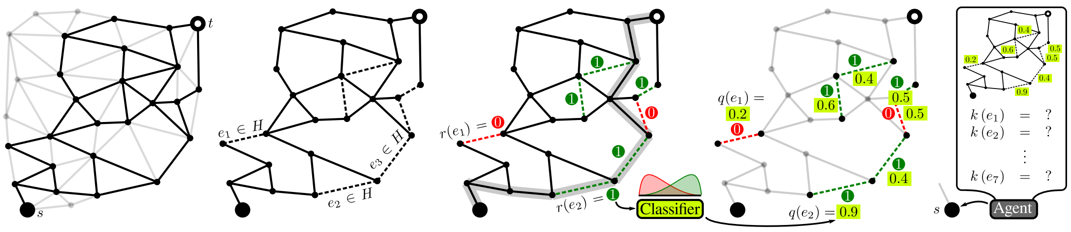


Figure 4.8. Instance generation with a random graph, from left to right: generation of nodes and edges; random choice of edges in H; realization of each edge as traversable (dark green) or non-traversable (red) (ensuring connectivity between s and t); realization of the traversability estimates (light green) according to classifier model; the agent at the beginning of the simulation knows the graph and q, but not r.

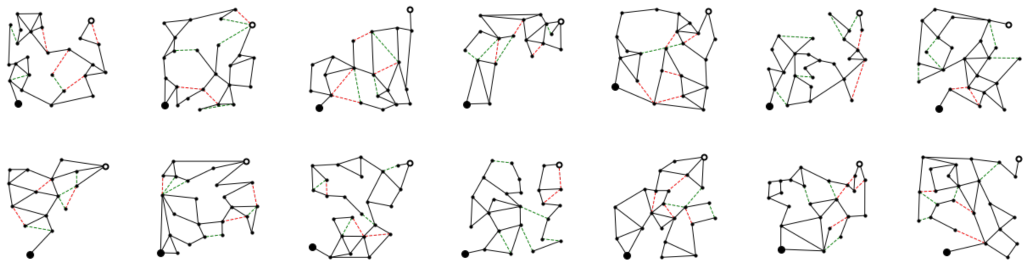


Figure 4.9. 14 out of 100K random graphs and realizations with 7 hidden edges (with an average of 23 nodes and 35 edges). On each graph, we apply classifiers of different quality to generate more than 1M planning instances. For every instance we collect the true cost of all policies.

4.4.4 Experimental setup

In the following, we generate a large amount of planning problem instances, and compare the performance of different policies.

One instance is defined as follows (see right part of Figure 4.8).

- We consider: a graph $G = (N, E)$, in which each edge has an associated cost; a pair of source and target nodes $s, t \in N$; a set H of hidden edges $H \subseteq E$.
- We generate one realization $r : H \to \{0, 1\}$ of the hidden edges true traversability, unknown to the agent; it assigns a binary traversability value (not traversable or traversable) to each hidden edge. Each $r(e)$ is *independently* generated following a Bernoulli(0.5) distribution; if in the resulting graph t is not reachable from s, a new realization is drawn.
- We generate one realization $q : H \to [0, 1]$ of the traversability probabilities, which are known to the agent. q is generated according to r by a classifier with a given AUC value q_{auc}. In particular, we sample the classifier output from $q(e) \sim B_{\alpha(q_{\text{auc}}),\beta(q_{\text{auc}})}$ if $r(e) = 0$ or from $q(e) \sim B_{\beta(q_{\text{auc}}),\alpha(q_{\text{auc}})}$ if $r(e) = 1$.

Once an instance is defined, we compute the optimal policy π_{opt}, simulate an agent following it in the realization r, and measure the cost of the resulting trajectory. We do the same with baseline policies $\pi(\tau)$ for different values of τ.

We do not report these costs directly; instead, we are interested in the ratio $c(\pi) \geq 1$, called *competitive ratio*, between such costs and the cost of the minimal-cost path in G (which can be computed given r).

In each of the experiments below, we analyze the effect of a different parameter (q_{auc}, $|H|$, τ) and report statistics about competitive ratios of each policy computed over many instances.

Random graphs

We generate random graphs as illustrated in the first three illustrations (from left to right) of Figure 4.8 by: (1) drawing 30 points uniformly between [0,1]; (2) connecting the points using a Delaunay triangulation; (3) selecting s and t at the bottom-left and top-right corner respectively; (4) randomly deleting half of the

[1] q_{auc}= 1.0 does not imply calibration: the classifier returning 0.9 for all traversable edges and 0.1 for non-traversable edges is *not* calibrated but has q_{auc}= 1.0 and perfect accuracy.

edges without disconnecting s and t; (5) randomly selecting edges in H; and (6) randomly selecting one feasible realization r.

We use simple strategies to generate interesting planning instances. For example: in (4) we prune parts of the graph that no policy would visit (e.g., leave nodes other than s and t and bridges that do not separate s and t); in (5) we force that at least one hidden edge separates s and t along the minimal-cost path but we avoid picking hidden edges that would anyway need to be passed (e.g., bridges between s and t).

For each experiment, we generate planning instances corresponding to 100K random graphs and realizations r (see Figure 4.9).

Indoor map

We use a floor map of a real building (see Chapter 1) where an agent estimates if doors are open (traversable) or closed (non-traversable). The navigation graph (Section 1.3.4), depicted in Figure 4.10:Left, is composed of local trajectories derived from the building geometry and has 187 nodes and 236 edges; s and t are located in two rooms at the opposite side of the building.

The set H of hidden edges contains all 9 doors that may be traversed when traveling from s to t. We run simulations over all ($2^9 = 512$) realizations r, generating 100 instances for each classifier q_{auc}value (for total 500K instances).

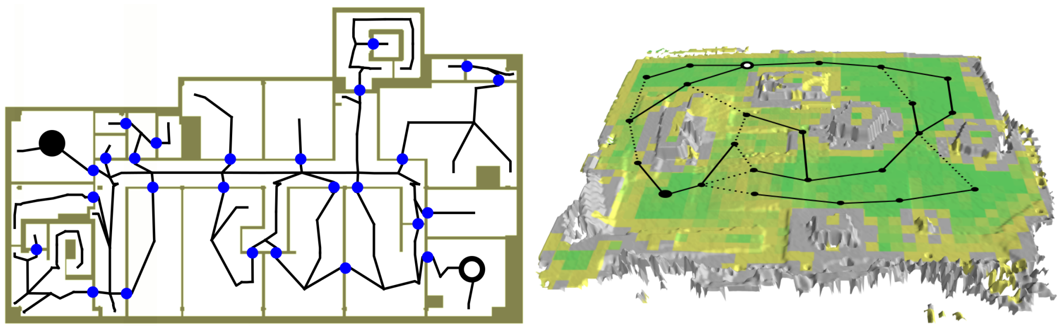


Figure 4.10. Left: Indoor floor map; navigation graph with doors that may be locked (blue circles), source (black circle) and target (white circle) nodes. *Right*: Rugged terrain map acquired by 3D reconstruction with an UAV; classifier outputs for traversable (green), non-traversable (gray) and uncertain (yellow) terrain patches; corresponding navigation graph between source (black circle) and target (white circle) locations with known (solid lines) and hidden edges (dashed lines).

Rugged terrain map

Figure 4.10:Right shows a 3D mapping from the ETH-ASL traversability dataset [149]: an experimental scenario with several obstacles, such as bumps, ramps, holes, boxes and slippery surfaces. The traversability classifier presented in Section 4.3 estimates whether the robot will be able to traverse a patch of terrain. We draw the navigation graph by hand with 24 nodes and 30 edges. We take into account traversability estimations to label edges as traversable, not traversable or with uncertain traversability. We identify a total of 8 uncertain edges that correspond to challenging terrain such as ramps, boxes edges or high bumps and are modeled as hidden edges in H. As above, we use a single graph from which we generate all ($2^8 = 256$) realizations r, generating 100 instances for each classifier AUC value (for a total 250K instances).

4.4.5 Experimental Results

In this Section, we present and discuss the experimental results. First, in Section 4.4.5, we analyze the performance of policies on random graphs with 7 hidden edges. Then, we look at results on random graphs with different numbers of hidden edges and on real-world maps.

Random graph with $|H| = 7$

Baseline policies Figure 4.11:Top-Left illustrates the effect of different values of τ on the average performance of baseline policies $\pi(\tau)$. As expected, the performance of the optimistic policy $\pi(0)$, which ignores classifier outputs, does not depend on the classifier quality; moreover it has the lowest average performance. Policies with $\tau \approx 0.6$ perform best regardless of classifier quality. For low quality classifier, the performance penalty for being too pessimistic (high τ) is progressively reduced, to the point that, for extremely low quality classifiers, the pessimistic policy $\pi(1)$ offers a comparable performance. More precisely, from a smooth u-shaped function for high q_{auc}, the dependency of the competitive ratio on τ becomes, as we approach q_{auc}= 0.5, a step-like function that just discriminates if $\tau > 0.5$.

For all other experiments, we only report results for $\pi(0)$ and $\pi(1)$, which have opposite distinct characteristics and are clearly interpretable, and for the nearly optimal choice of $\tau = 1/2$. In fact, $\pi(1/2)$ has also a simple interpretation: it considers as traversable any edge that has a higher probability to be traversable.

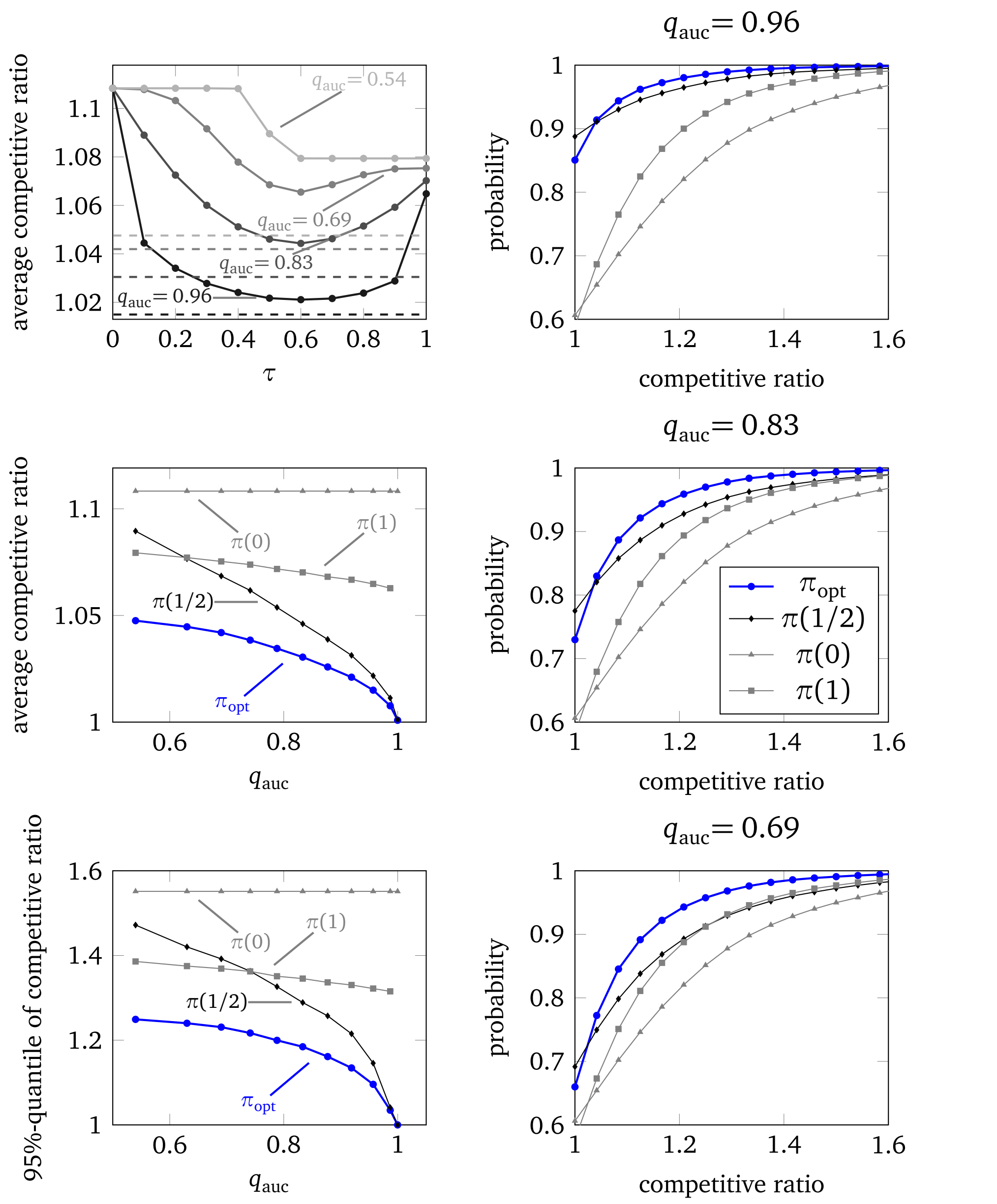


Figure 4.11. Competitive ratio of policies on 100K random graphs with 7 hidden edges. *Left-Top*: average as a function of τ, with high-quality (black), medium-quality (dark gray) and low-quality (light gray) classifiers; performance of π_{opt} is represented by dashed lines of the same color. *Left-Middle*: average versus q_{auc}. *Left-Bottom*: 95%-quantile versus q_{auc}. *Right*: the cumulative distribution of competitive ratios for high quality classifier (*top*), medium quality (*middle*), and low quality (*bottom*).

Impact of classifier quality on competitive ratio distribution Figure 4.11 reports mean, 95%-quantile and cumulative distribution of competitive ratios versus classifier quality. As expected, the competitive ratios are close to 1 when the classifier is very accurate ($q_{\text{auc}} \approx 1$) and grows as the classifier quality decreases. The optimal policy π_{opt} is almost always better than any baseline policy. The only exception is the policy $\pi(1/2)$, which has a higher probability of minimal cost ($c = 1$) (Fig. 4.11:Right). Not surprisingly, π_{opt} has the lowest mean cost.

For accurate classifiers, policy $\pi(1/2)$ is on par with the optimal policy, but performs significantly worse for inaccurate classifiers. In fact, for inaccurate classifiers, the gap between π_{opt} and $\pi(1/2)$ increases while the gap between π_{opt} and $\pi(0)$ or $\pi(1)$ decreases.

In general, the impact of the classifier quality on the average competitive ratio is limited ($c(\pi) \leq 1.11$). This is also due to the "forgiveness" of the planning problem (CTP) we are considering, where [rational] policies try at worst few different routes. For reference, the highest competitive ratio sample over all planning instances and policies is 5.23. When we consider the 95%-quantile, the impact is more significant, but the relations between the four policies remain similar.

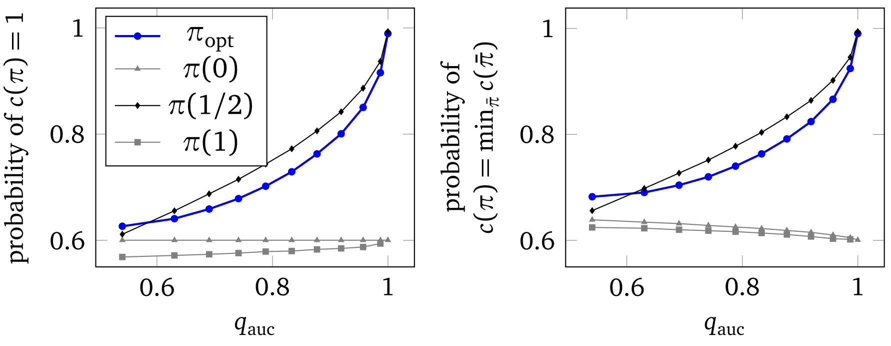


Figure 4.12. Left: probability that a policy follows the shortest path. *Right*: probability that a policy has at most a cost equal to any other policy in $\{\pi_{\text{opt}}, \pi(0), \pi(1/2), \pi(1)\}$.

Policies comparison on planning instances On many random instances, policies have minimal costs (see Figure 4.12:Left and first row of Table 4.1). As expected, the higher the classifier quality, the more probable is for a policy to follow the shortest path. This probability remains very significant (around 60%) even for low quality classifiers; the corresponding planning instances are trivial. We note that the gap between all four policies fades for low quality classifiers, while for high quality classifiers π_{opt} and $\pi(1/2)$ finds the shortest path much more often than $\pi(0)$ or $\pi(1)$. Interestingly, $\pi(1/2)$ follows the shortest path with higher probability than π_{opt} except when the classifier quality is extremely low or high.

Figure 4.12:Right illustrates the probability that a policy ranks first (possibly tied) among other policies on a random instance. This represents the fraction of samples for which the agent will not regret (in the aftermath) to having followed a particular policy. For a good part of these instances, the agent actually follows the shortest path and the impact of the classifier quality is similar to the one illustrated before.

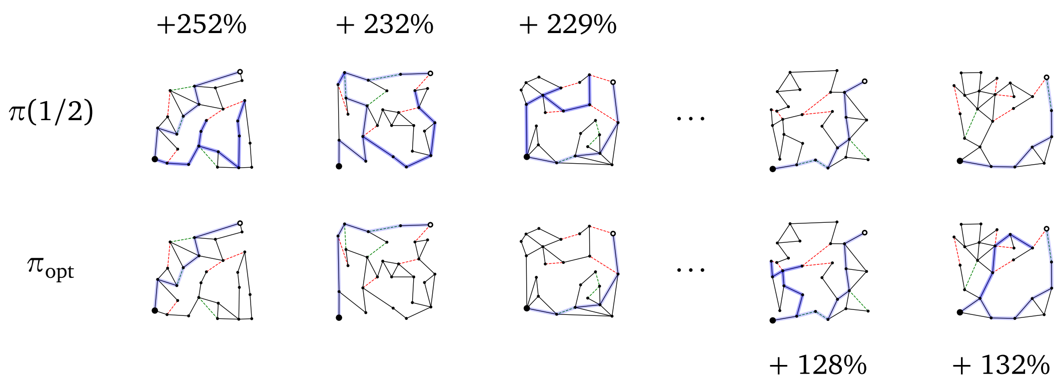


Figure 4.13. Planning instance on random graphs with 7 hidden edges for $q_{\text{auc}} = 0.83$, ordered (from left to right) by the ratio between the costs of the policies $\pi(1/2)$ and π_{opt}. On the left, three instances for which $\pi(1/2)$ has a cost that is 252%, 232%, and 229% higher. On the right, two instances for which π_{opt} is worse and pay a penalty of 128%, and 232%. The trajectory generated by the policies is drawn as a semi-transparent blue line; when the robot discovers non-traversable edges, it may need to back-track, which leads to darker segments.

Figure 4.13 lists planning instances for $q_{\text{auc}} = 0.83$ ordered by the penalty that the agent will pay for following $\pi(1/2)$ instead of π_{opt}. On the left, there are sample instances on which the cost of $\pi(1/2)$ is more than 200% higher. On the right — after many instances in the middle where the policies have the same cost — we reach instances where the cost of π_{opt} is more than 100% higher. As discussed before, there are actually more instances for which π_{opt} is worse, yet the average cost over all instance is nonetheless smaller for π_{opt}. Table 4.1 contains the details of this comparison for different classifiers: the probability to pay at most a given penalty for following one or the other policy. We note that

Table 4.1. Cost of policies π_{opt} and $\pi(1/2)$ on random graphs with 7 hidden edges using classifiers of different quality. Top two rows: percentage of instances for which the two policies follow the *same* path (first row: shortest path, second row: a non minimal path). Middle section: percentage of instances for which π_{opt} has higher, and significantly higher ($+10\%$, $+50\%$, $+100\%$), cost compared to $\pi(1/2)$. Lower section: Percentage of instances for which $\pi(1/2)$ has higher, and significantly higher ($+10\%$, $+50\%$, $+100\%$), cost compared to π_{opt}.

q_{auc}	0.999	0.987	0.957	0.919	0.877	0.833	0.788	0.741	0.690	0.630	0.540
$c(\pi_{\text{opt}}) = c(\pi(1/2)) = 1$	86.21%	78.11%	71.07%	65.35%	60.98%	57.17%	53.65%	50.56%	47.77%	44.59%	40.82%
$c(\pi_{\text{opt}}) = c(\pi(1/2)) \neq 1$	13.06%	15.56%	17.49%	19.21%	20.36%	21.24%	21.92%	22.33%	22.65%	22.57%	22.13%
$c(\pi_{\text{opt}}) > c(\pi(1/2))$	0.54%	4.25%	7.59%	9.83%	11.51%	12.96%	14.19%	15.18%	16.00%	16.73%	16.95%
$c(\pi_{\text{opt}})/c(\pi(1/2)) > 110\%$	0.07%	0.65%	1.37%	1.91%	2.52%	2.99%	3.58%	4.07%	4.65%	5.47%	6.26%
$c(\pi_{\text{opt}})/c(\pi(1/2)) > 150\%$	0.00%	0.02%	0.05%	0.08%	0.10%	0.10%	0.14%	0.18%	0.22%	0.26%	0.34%
$c(\pi_{\text{opt}})/c(\pi(1/2)) > 200\%$	0.00%	0.00%	0.00%	0.01%	0.01%	0.01%	0.02%	0.02%	0.02%	0.03%	0.04%
$c(\pi(1/2)) > c(\pi_{\text{opt}})$	0.19%	2.09%	3.86%	5.61%	7.15%	8.63%	10.24%	11.93%	13.58%	16.11%	20.10%
$c(\pi(1/2))/c(\pi_{\text{opt}}) > 110\%$	0.16%	1.66%	3.03%	4.44%	5.64%	6.73%	7.97%	9.37%	10.62%	12.55%	15.68%
$c(\pi(1/2))/c(\pi_{\text{opt}}) > 150\%$	0.03%	0.30%	0.64%	0.92%	1.15%	1.37%	1.68%	1.96%	2.29%	2.66%	3.31%
$c(\pi(1/2))/c(\pi_{\text{opt}}) > 200\%$	0.00%	0.04%	0.09%	0.14%	0.19%	0.23%	0.30%	0.39%	0.39%	0.51%	0.66%

whereas the probability for π_{opt} of being worse is higher, the probability to pay at most a penalty of 10%, 50% or 100%, is significantly smaller; this difference becomes more significant for classifiers of lesser quality.

Random graph with $1 \leq |H| \leq 9$

We report in Figure 4.14 how the competitive ratio increases as a function of the number of hidden edges $|H|$. The impact of the classifier quality on the policies' performance growths with planning complexity (higher $|H|$), yet the relative performance between policies remains similar.

We also report the measured mean computational cost for computing the optimal policy π_{opt} by Value Iteration using a single modern CPU core. The baseline policies $\pi(\tau)$ have instead negligible computation costs that scale polynomially with respect to $|H|$.

Real-world graphs

We compare the results on random graphs with experiments on two real maps, where we study the impact of classifier quality on the policies' performance. Note that although we sample over all possible realizations (and many classifications), these two maps represent just two graph instances and, as already discussed, the policies performance has a large variability over different graphs.

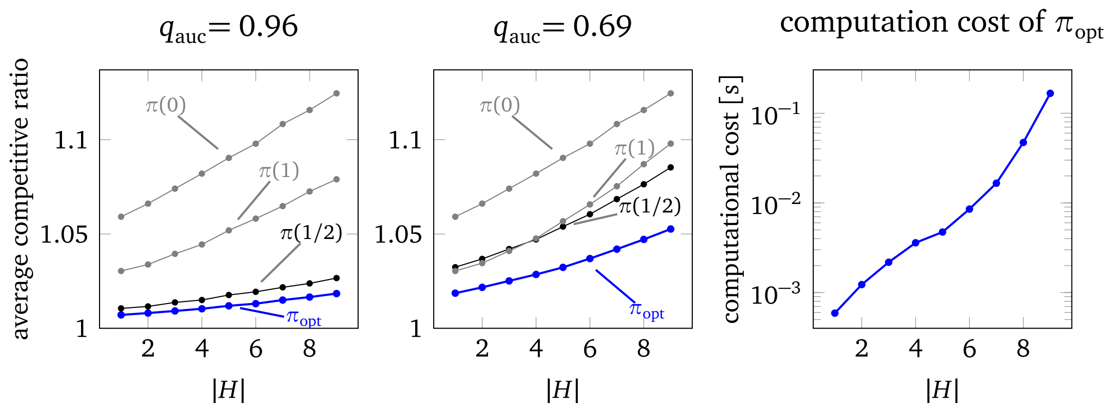


Figure 4.14. *Left*: Competitive ratio of different policies on random graphs as a function of $|H|$, with a high-quality (left) and a low-quality (center) classifier. *Right*: Mean computational cost for computing a single instance of π_{opt}.

Indoor map Figure 4.15:Top summarizes the policies' performance on map of a real building, where hidden edges correspond to doors (that may be open or closed). The impact of estimation quality on the optimal policy is comparable to the previous results. Interestingly, on this particular map, baseline policy $\pi(0)$ perform much better than on the average random graphs.

Rugged terrain map We report a similar experiment for the Rugged terrain map in Figure 4.15:Bottom. The average competitive ratios for this graph are high when compared to the randomly-generated graph ($\geq$ 90%-quantile), denoting

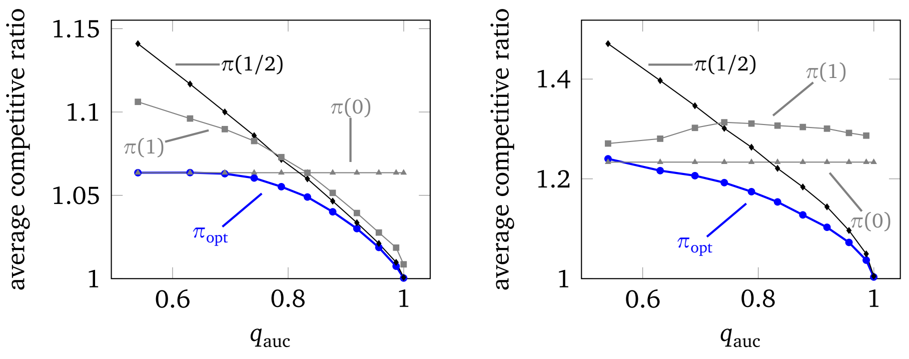


Figure 4.15. Competitive ratio of policies as a function of classifier quality on real world maps. *Left*: Indoor floor map. *Right*: Rugged terrain map.

an harder than average planning instance. Also in this case, the performance of $\pi(0)$ with respect to π_{opt} is much better than on average graphs.

4.4.6 Discussion

The overall impact of uncertainty on the optimal policy is relatively small: even with very weak classifiers ($q_{\text{auc}} = 0.54$), the average path cost increases by only about 5% with respect to the shortest path; however, this is very dependent on the specific map, as shown in the real-world scenario in Figure 4.15:Right where the competitive ratio with a weak classifier exceeds 1.2.

The baseline strategies yield a worse performance than the optimal policy, but such penalty is small in most cases. The best baseline reactive policies are moderately pessimistic ($\tau \approx 0.6$), which is particularly preferable when uncertainty is high.

Computing the optimal policy has a much higher cost but gives better performance, in particular when uncertainty increases (due to lower classifier quality and/or more hidden edges), or more importance is given to the worst case than to an average case. This is also very dependent on the graph as shown by the many instances where a reactive policy had better performance.

4.5 Conclusions and Perspectives

In this section we discussed different interpretation of traversal probabilities and we focused on two of them, which lead to the risk-aware path planning problem and the Canadian traveller problem.

Risk-aware path planning To introduce the first problem, we briefly illustrated how machine learning and simulation techniques on synthetically generated maps are used to train a rough terrain traversability estimator that associates a terrain patch to the probability that a robot will successfully traverse it. Then, we discussed how to make use of these estimations to formulate one-shot, risk-aware path planning problems, i.e., a multi-objective problems to maximize path traversability and minimize path length. We introduced an algorithm to approximate the set of non-convex-dominated solutions. Experiments on real maps showed that this method produces few strategic choices and requires low computational cost, at least on relatively small maps. We also presented two application with real robots (one wheeled and one legged) that highlight the generality of the approach.

Ongoing work targets a comparison with other techniques to approximate the Pareto front (or its convex hull) based on Genetic Algorithms.

Canadian traveller problem In the second part, we allowed the robot to locally refine the estimations during the execution of its navigation policy. We presented results from a large-scale simulation campaign to evaluate the effects of two sources of uncertainty (inaccuracy of the classifier and number of hidden edges) on the cost of paths obtained by the optimal policy and a family of baseline reactive strategies. The penalty from bad estimations is small enough, on the graphs we considered, that we may conclude that we don't need accurate estimations. Yet, in some situations, robots may have other additional constraints (e.g. they should arrive before a given deadline) which requires different objectives than the one we considered in this study.

In the future, we would like to investigate the impact of the graph itself on the policy performance. Which graphs give rise to the most challenging planning instances? Are the distributions we used to generate graph samples representative of real-world maps? Results on the real-world maps indicate that the distribution may indeed be not enough representative. We are also interested to study the impact of the classifier calibration on the policy costs because real-world classifiers are typically not perfectly calibrated.

Chapter 5

Conclusion

5.1 Summary

We have presented a pipeline for path planning in robotics that accounts for many issues that arise in real-world problems; in particular, we have discussed solutions to the following issues: (1) representing complex, hierarchical spatial information that is relevant to path planning; (2) accounting for the indirect effects of the robot moving in the environment; (3) computing coarse-grained high-level routes; (4) refining those routes to precise geometrical trajectories with desirable properties; (5) and, finally, follow the trajectories while navigating among robots and other obstacles. We have examined the impact of partial information, introduced multi-objective path planning problems and illustrated real world applications.

Spatial model and high-level path planning

In Chapter 1, we presented an abstract spatial representation that is inspired by the way people reason about indoor spaces and that is well suited for robotics applications in environments shared with humans. We derived a high-level navigation graph from an IndoorGML map, a recently approved GIS standard for indoor spaces; we introduced an algorithm to decompose the robot's configuration space in a set of quasi-convex cells and derive a convenient planning graph.

We modeled generic costs associated to routes and argued that three costs — effort, safety risk, and social impact — describe a wide range of scenarios with humans and/or robots. We defined the associated multi-objective path planning problem, which we solved with an exact algorithm, based on the observation that the solution space for typical indoor planning graphs is small. We measured the performance of the multi-objective planner performance on a real building map with different cost configurations.

The model is general; it was successfully applied to real world scenarios in two different research projects, in which we tested our solutions on real robots: (online) path planning services for autonomous wheelchairs in nursing homes and coordination of a fleet of drones in an urban environment.

We developed a pipeline — from extracting maps out of floor plans, to updating spatial attributes, to computing individual plans — which serves as a proof of concept for the model.

Trajectory planning

In Chapter 2, we focused on the trajectory of mobile robots, and in particular of autonomous wheelchairs, in the absence of dynamic obstacles. We discussed how high-level routes are refined to geometrical curves that are smooth, legible and comfortable to the user when executed by a wheelchair. We identified a proxy cost that discriminates gently turning paths — which we use to formulate an optimization problem — and discussed alternative costs.

We searched for solutions in the space of composite Bézier curves that traverse convex cells and are constrained while passing through narrow passes. At least 6-th order Bézier curves would be needed to get optimal curves with continuous curvature; yet we showed a method for relaxing the curvature constraints and speed up the search using 4-th order Bézier curves. We introduced an heuristic to fix the free parameters, which is used as an initial guess by the optimization solver. The planning complexity is dramatically reduced by splitting the curves into independently optimized chains and pre-computing most of them.

We introduced a further link with Chapter 1 when presenting another multi-objective problem: the search for short and gently-turning trajectories. We concluded with tests performed on a real mobile robots; the trajectories followed by the robots look legible and similar to the trajectory followed when a skilled person is teleoperating the robot. We provide an implementation that works with generic trajectory costs, optional constraints and supports different search spaces (2-th, 4-th and 6-th order composite Bézier curves), and different third-party numerical solvers.

Local navigation In Chapter 3, we finally consider dynamic aspects: how the robots, given a trajectory to follow or a point to reach, should move among people and/or other robots. We focused on reactive navigation rules that should produce safe (no collisions), efficient (short and fast), legible and predictable trajectories, which we argued are the key attributes for the robot to be perceived as human-friendly. We achieved smooth human-robot co-navigation by letting robots follow the same navigation algorithm as pedestrians. We adapted the policy to address robotic peculiarities, like differential-drive kinematics, safety, and social constraints.

We compared the resulting policy with two state-of-the-art algorithms for reactive robot navigation in a variety of scenarios; its performance, in simulation and with real robots, compared favorably in terms of efficiency and safety Moreover, the human-like policy let macroscopic (crowd) behaviors emerge, like the formation of lanes, which are expected by humans co-navigating along with the robots, and lead to increased legibility, predictability and efficiency in mixed groups.

We measured the impact of partial information, observing that limited sensing ability delays the emergence of macroscopic behaviors. In our context, robots implicitly communicate by sensing the position and velocity of each other: by hindering this communication we prevented efficient coordination.

We analyzed the impact of navigation behaviors on the cost of a trajectory optimized in Chapter 2 and concluded that, compared to alternative behaviors, human-like navigation leads to a more accurate following of the prescribed path, and to a smaller (albeit still large) increase of jerk.

Traversability estimates

In Chapter 4, we introduced the topic of uncertain traversability estimates, which are typical of outdoor robotic scenarios. We discussed how different failures of traversing a patch of terrain give rise to different path planning problems.

We briefly presented how to train a classifier that generates traversability estimates of a patch of terrain from examples collected in simulation over synthetically generated terrains. We use these estimates as an inspiration to define a multi-objective path planning problem; we introduce a simple algorithm that approximates the Pareto front (which is potentially very large) by returning a small selection of optimal paths (in terms of minimal risk and minimal length); we presented results of the planner performance on a real map and investigated the trade-off between computational cost and the number of optimal solutions found.

This solution is well-suited for an ongoing research project in search and rescue robotics, because it offers the human operator few choices of trajectories to select from when controlling a semi-autonomous robot moving over difficult terrain.

Then, we measured how the quality of the traversability estimate impacts path planning. We considered a scenario where the robot is able to locally refine (using its sensors) a rough estimate available at planning time; we modeled this as an instance of the Canadian traveller problem whose the goal is to compute navigation policies that are resilient to local changes in the graph such as the deletion of an incident edge. We introduced a class of heuristic policies and compared their performance with the optimal policy. We showed that heuristics perform comparably well when applied to an average graph, but that their performance degrades when considering worst-case metrics. The optimal policy, which has a much higher computational cost, helps with dealing with more uncertain predictions (larger uncertainty and/or more edges that are uncertain).

Discussion

The thesis deals with a complex pipeline that links spatial information to robot actions; in particular we highlight the interplay between the components of such pipeline: understanding this interplay is required to design a generic but realistic and usable model.

Through the research, we have encountered three recurrent challenges — hierarchical structures, partial information and multiple objectives — and tackled them from different points of view.

We validated most of the proposed models and algorithms either on real robots or in simulation on real-world scenarios, both indoor and outdoor. Testing the feasibility of ideas in different contexts was an expensive but necessary step to identify relevant, focused research questions.

5.2 Looking forward

We are currently working on several open research questions related to the topics discussed in this thesis.

One line of research deals with measuring, through extensive experiments and user studies, the legibility, predictability, and (human) comfort of the robot behaviors (Chapter 3); for human-like navigation, we are designing a user study to be performed in an immersive simulation to overcome limitations of robot hardware and guide the design of future real-robot experiments.

For the spatial representation (Chapter 1), we plan to model at least one other scenario, in addition to the nursing home, to validate the description of social spaces in terms of micro and macro attributes.

For outdoor navigation (Chapter 4), we are actively working on more accurate models for traversability estimators and their application to path planning in real-world contexts, also considering complex, legged robots such as ANYmal [71] and K-Rock [100] developed by other partners of the NCCR Robotics Consortium.

On a longer term, we are interested in the connections between all components of a robot controller. For example, for the pipeline we have been discussing in this thesis, we may be able to: (1) generalize our approach to learn the mapping between environment state and control outcome, e.g., to predict critical density for a group of robot to form a jam from the geometry of the space; (2) learn to predict the geometry of optimal trajectory from high-level routes (something we currently do with an heuristic); (3) gather feedback during plan execution and use it to refine map attributes (through online learning).

More generally, our goal is to preserve important information when components interact and communicate; an interaction could consist, for example, of a path planner that passes a target trajectory to a controller, which in turn provides feedback about how well it is following the path. In particular, components should preserve, when possible, information about uncertainty.

Inspiration from and comparison with the real world is fundamental for robotics. Like theoretical physicists are ultimately confronted with the outcome of experiments, roboticists should ground their ultimate source of knowledge in real robots performing real tasks in the world. We should implement our models and bring them out there. This means overcoming technical issues that occasionally make robotics research unpleasant. Yet, these issues, which may be at first perceived as mere implementation details, sometimes hide important lessons. The difficulty lies in noticing them.

Appendix A

Publications

Path and trajectory planning in indoor spaces We discuss spatial representations and path planning in a workshop paper [12]. We present an optimal trajectory planner for wheelchairs in a conference paper [11]. We describe the prominent use case of supporting users with restrained mobility in nursing homes in two project deliverables [9, 10].

Human-friendly local navigation We introduce the algorithm in a conference paper [16] and compared it with other collision avoidance algorithms in another conference paper [17]. We discuss groups behaviors emerging from local navigation rules in a third conference paper [15]. In another conference paper, we present a machine learning approach to let a robot track people using low-lying sensors, a very useful capability when navigating among people [8]. Finally, in two conference papers we discuss how affective states can modulate navigation and increase coordination [13, 14].

Planning with traversability estimations We present a machine learning approach to navigate along forest trails in a journal paper [7]. We apply a similar technique to estimate terrain traversability for ground robots in a conference paper [3]. In a journal paper, we discuss how uncertain traversability estimations lead to a risk-aware, multi-objective path planning problem [4]. We discuss the impact of traversability estimation on the cost of navigation policies in a workshop paper [19]. A longer version is currently under review for a conference [20].

Other publications Publications [1, 2, 5, 6, 18, 21] are not directly related to the content of this thesis, although they discuss mobile robotics and/or multi-agent planning.

Conference papers

[2] J. Banfi, J. Guzzi, A. Giusti, L. Gambardella, and G. A. Di Caro. “Fair multi-target tracking in cooperative multi-robot systems”. In: *International Conference on Robotics and Automation (ICRA)*. IEEE. 2015, pp. 5411–5418.

[3] R. O. Chavez-Garcia, J. Guzzi, L. M. Gambardella, and A. Giusti. “Image Classification for Ground Traversability Estimation in Robotics”. In: *Proceedings of the 18th International Conference on Advanced Concepts for Intelligent Vision Systems (ACIVS)*. Springer, 2017, pp. 325–336.

[6] F. Ghiringhelli, J. Guzzi, G. A. Di Caro, V. Caglioti, L. M. Gambardella, and A. Giusti. “Interactive augmented reality for understanding and analyzing multi-robot systems”. In: *International Conference on Intelligent Robots and Systems (IROS)*. IEEE/RSJ. 2014, pp. 1195–1201.

[8] A. P. Gritti, O. Tarabini, J. Guzzi, G. A. Di Caro, V. Caglioti, L. M. Gambardella, and A. Giusti. “Kinect-based people detection and tracking from small-footprint ground robots.” In: *International Conference on Intelligent Robots and Systems (IROS)*. IEEE/RSJ. 2014.

[11] J. Guzzi and G. A. Di Caro. “From indoor GIS maps to path planning for autonomous wheelchairs”. In: *International Conference on Intelligent Robots and Systems (IROS)*. IEEE/RSJ. Oct. 2016, pp. 4773–4779.

[12] J. Guzzi and G. A. Di Caro. “Towards supporting elderly’s orientation, mobility, and autonomy”. In: *Workshop on Improving the quality of life in the elderly using robotic assistive technology: benefits, limitations, and challenges - International Conference on Social Robotics (ICSR)*. Oct. 2015.

[13] J. Guzzi, A. Giusti, G. A. Di Caro, and L. M. Gambardella. “A Model of Artificial Emotions for Behavior-Modulation and Implicit Coordination in Multi-robot Systems”. In: *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*. 2018. to appear.

[14] J. Guzzi, A. Giusti, G. A. Di Caro, and L. M. Gambardella. “Artificial Emotions as Dynamic Modulators of Individual and Group Behavior in Multi-robot System”. In: *Proceedings of International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*. 2018. to appear.

[15] J. Guzzi, A. Giusti, L. M. Gambardella, and G. A. Di Caro. “Bioinspired obstacle avoidance algorithms for robot swarms”. In: *Proceedings of the International Conference on Bio-Inspired Models of Network, Information, and Computing Systems (BIONETICS)*. Springer, 2014, pp. 120–134.

[16] J. Guzzi, A. Giusti, L. M. Gambardella, G. Theraulaz, and G. A. Di Caro. "Human-friendly robot navigation in dynamic environments." In: *Proceedings of the International Conference on Robotics and Automation (ICRA)*. May 2013, pp. 423–430.

[17] J. Guzzi, A. Giusti, L. M. Gambardella, and G. A. Di Caro. "Local reactive robot navigation: A comparison between reciprocal velocity obstacle variants and human-like behavior". In: *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2013, pp. 2622–2629.

[18] J. Guzzi, A. Giusti, G. A. Di Caro, and L. M. Gambardella. "Mighty Thymio for Higher-Level Robotics Education". In: *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence (AAAI)*. AAAI. 2018. to appear.

[19] J. Guzzi, R. O. Chavez-Garcia, L. M. Gambardella, and A. Giusti. "On the Impact of Uncertainty on Path Planning". In: *Federated AI for Robotics Workshop (FAIR)*. 2018. to appear.

[20] J. Guzzi, R. O. Chavez-Garcia, L. M. Gambardella, and A. Giusti. "On the Impact of Uncertainty on Path Planning". In: *Conference on Robot Learning (CoRL)*. 2018. under review.

[21] S. Toniolo, J. Guzzi, A. Giusti, and L. M. Gambardella. "Learning an Image-based Obstacle Detector with Automatic Acquisition of Training Data". In: *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence (AAAI-18 Demo Track)*. 2018.

One publication is currently under review.

Journal articles

[1] J. Banfi, J. Guzzi, F. Amigoni, E. F. Flushing, A. Giusti, L. M. Gambardella, and G. A. Di Caro. "An Integer Linear Programming Model for Fair Multitarget Tracking in Cooperative Multirobot Systems". In: *Autonomous Robots* (Apr. 2018).

[4] R. O. Chavez-Garcia, J. Guzzi, L. M. Gambardella, and A. Giusti. "Learning Ground Traversability from Simulations". In: *IEEE Robotics and Automation Letters* 3.3 (July 2018), pp. 1695–1702.

[5] M. Dorigo, D. Floreano, L. Gambardella, F. Mondada, S. Nolfi, T. Baaboura, M. Birattari, M. Bonani, M. Brambilla, A. Brutschy, D. Burnier, A. Campo, A. Christensen, A. Decugniere, G. Di Caro, F. Ducatelle, E. Ferrante, A. Forster, J. Martinez Gonzales, J. Guzzi, V. Longchamp, S. Magnenat, N. Mathews, M. Montes de Oca, R. O'Grady, C. Pinciroli, G. Pini, P. Retornaz, J. Roberts, V. Sperati, T. Stirling, A. Stranieri, T. Stutzle, V. Trianni, E. Tuci, A. Turgut, and F. Vaussard. "Swarmanoid: A Novel Concept for the Study of Heterogeneous Robotic Swarms". In: *Robotics Automation Magazine, IEEE* 20.4 (Dec. 2013), pp. 60–71.

[7] A. Giusti, J. Guzzi, D. C. Ciresan, F.-L. He, J. P. Rodriguez, F. Fontana, M. Faessler, C. Forster, J. Schmidhuber, G. A. Di Caro, D. Scaramuzza, and L. M. Gambardella. "A Machine Learning Approach to Visual Perception of Forest Trails for Mobile Robots". In: *IEEE Robotics and Automation Letters* (2016), pp. 661–667.

Technical Reports

[9] J. Guzzi. *Ageing without Losing Mobility and Autonomy (ALMA). Deliverable D4.1 - Distributed information gathering and management*. Tech. rep. IDSIA, 2013.

[10] J. Guzzi. *Ageing without Losing Mobility and Autonomy (ALMA). Deliverable D4.3 - Path planning services*. Tech. rep. IDSIA, 2016.

Appendix B

Software Releases

We have developed software tools to experiment with the ideas presented in the text and apply them in research projects; all software listed below is released online as open source libraries.

Rich indoor maps

Core We release an Python implementation [7] of the core spatial model introduced in Chapter 1 with generic cost attributes and single/multi-objective planners that use these cost functions. The software stack consists of: (1) a SketchUp plugin to convert between CAD floor plans and IndoorGML maps; (2) a library to import/export/edit IndoorGML maps and their spatial attributes, and compute navigation graphs for different kind of agents; (3) a GUI where users inspect and edit maps; (4) a suite of topological path planners that compute the best paths according to multiple costs; (5) a geometrical planner that compute the best trajectories in multiple search spaces (composite Bézier curves of 2-th, 4-th or 6-th), according to different costs (bend energy, bending cost, maximal curvature, length, ...) and with variable smoothness requirements (G^0, G^1 or G^2).

ALMA We release an implementation of the web server developed in the ALMA project [1] that provides real time mapping and planning services to users in nursing homes. The server implements an instance of the core model tailored to social spaces, elderly users and users on wheelchairs. The server exposes resources through an HTTP interface to different type of clients: people can visualize and edit the maps through an HTML interface; ambient sensors provide real-time updates about the environment through a JSON interface; users get directions, with a description of difficulties they may experience along the route, through the same JSON interface, according to their characteristics and the current state of the environment.

DFW We also release the implementation [2] of the central planner and of the distributed nodes (running on light poles) that control a fleet of drones in a smart city. Similarly to the ALMA implementation, the library extends the core model with specific attributes and planners. Moreover, it contains a global traffic coordinator that manages the exclusive locks that drones need to enter a cell. We also provide the drone controller, build on top of ROS and ArduPilot, which interfaces with UWB sensors to localize the drone in the flyway.

Optimal trajectory planner

The core spatial model [7] described above contains an implementation of the trajectory planner discussed in Chapter 2. The planner is implemented in Python and the following core libraries: `scipy` and `pyOpt` for the COBYLA solver, `networkx` for the Yen's k-shortest path algorithm, and `shapely` for the computation of geometric constraints. We also provide a ROS interface [5] with the planning server as a plugin for `move_base`, which is the main ROS package for mobile robotic navigation. We used this interface to control a TurtleBot robot in Section 2.4.4.

Multi-agent navigation

Multi-agent simular We release a multi-agent simulator [4] with a custom (simplified) 2D physics engine to experiment with different navigation algorithm for heterogeneous groups of agent, developed in ObjC. The simulator can run on a cluster of computers to perform large scale experiments but can also be used to visualize the behavior of up to 1000 agents in real time on a personal computer.

A 3D immersive interface allows people to experience (virtual) interaction with robots of comparable size and velocity.

Human-friendly navigation We provide a C++ library that implements the human-like navigation behavior and its ROS wrapper [3]. The library has been used to control multiple types of robot systems: large swarm of little-wheeled robots, flying drones, and larger wheeled robots interacting with people.

Traversability estimations

We release two smaller pieces of software: a library [6] reproduces the experiments of Section 4.4.4, i.e., generates random maps and efficiently computes the costs of navigation policies; a library [8] provides a ROS package that implements a risk-aware multi-objective path planner for the NCCR Robotics search & rescue challenge.

Online repositories

[1] J. Guzzi. *ALMA planning*. `https://github.com/jeguzzi/ri_alma`. 2018.

[2] J. Guzzi. *DFW planning*. `https://github.com/jeguzzi/ri_dfw`. 2018.

[3] J. Guzzi. *Human-like navigation*. `https://github.com/jeguzzi/hl_navigation`. 2018.

[4] J. Guzzi. *Multi-agent navigation simulator*. `https://github.com/jeguzzi/man_sim`. 2018.

[5] J. Guzzi. *Optimal trajectory planner interface*. `https://github.com/jeguzzi/ri_robot`. 2018.

[6] J. Guzzi. *Resilient path planning*. `https://github.com/jeguzzi/resilience`. 2018.

[7] J. Guzzi. *Rich maps planning*. `https://github.com/jeguzzi/ri_planning`. 2018.

[8] J. Guzzi. *Risk-aware planner*. `https://github.com/jeguzzi/risk_aware_planning`. 2018.

Bibliography

[1] F. Ahmed and K. Deb. “Multi-objective optimal path planning using elitist non-dominated sorting genetic algorithms”. In: *Soft Computing* 17.7 (2013), pp. 1283–1299.

[2] F. Ahmed and K. Deb. “Multi-objective path planning using spline representation”. In: *Proceedings of the IEEE International Conference on Robotics and Biomimetics (ROBIO)*. IEEE. 2011, pp. 1047–1052.

[3] *ALMA*. `http://www.alma-aal.org`.

[4] J. Alonso-Mora et al. “Collision avoidance for multiple agents with joint utility maximization”. In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. May 2013, pp. 2833–2838.

[5] J. Alonso-Mora et al. “Optimal Reciprocal Collision Avoidance for Multiple Non-Holonomic Robots”. In: *Proceedings of the International Symposium on Distributed Autonomous Robotic Systems (DARS)*. 2010, pp. 1–14.

[6] R. Alterovitz, T. Siméon, and K. Y. Goldberg. “The Stochastic Motion Roadmap - A Sampling Framework for Planning with Markov Motion Uncertainty.” In: *Robotics - Science and Systems* (2007).

[7] A. Andersson. “Multi-objective optimisation of ship routes”. MA thesis. ABB Corporate Research, Chalmers University of Technology, 2015.

[8] A. Bar-Noy and B. Schieber. “The Canadian Traveller Problem.” In: *ACM-SIAM symposium on Discrete algorithms*. 1991.

[9] T. Becker, C. Nagel, and T. H. Kolbe. “A Multilayered Space-Event Model for Navigation in Indoor Spaces”. In: *3D Geo-Information Sciences*. Springer Berlin Heidelberg, 2009, pp. 61–77.

[10] R. Benayoun et al. “Linear programming with multiple objective functions: Step method (stem)”. In: *Mathematical Programming* 1.1 (Dec. 1971), pp. 366–375.

[11] J. van den Berg and M. Overmars. “Planning Time-Minimal Safe Paths Amidst Unpredictably Moving Obstacles”. In: *The International Journal of Robotics Research* 27.11-12 (2008), pp. 1274–1294.

[12] J. van den Berg, D. Manocha, and M. Lin. “Reciprocal velocity obstacles for real-time multi-agent navigation”. In: *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)*. 2008, pp. 1928–1935.

[13] J. van den Berg et al. “Reciprocal collision avoidance with acceleration-velocity obstacles”. In: *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)* (2011), pp. 3475–3482.

[14] J. van den Berg et al. “Reciprocal n-body collision avoidance”. In: *Proceedings of the International Symposium Robotics Research (ISRR)*. 2011, pp. 3–19.

[15] C. M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

[16] Z. Bnaya, A. Felner, and S. E. Shimony. “Canadian Traveler Problem with Remote Sensing.” In: *International Joint Conferences on Artificial Intelligence*. 2009.

[17] Z. Bnaya et al. “Repeated-task Canadian Traveler Problem”. In: *AI Communications* 28.3 (2015), pp. 453–477.

[18] M. Bonani et al. “The marXbot, a miniature mobile robot opening new perspectives for the collective-robotic research”. In: *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2010, pp. 4187–4193.

[19] S. Bozkurt, A. Yazici, and K. Keskin. “A multicriteria route planning approach considering driver preferences”. In: *Proceedings of the IEEE International Conference on Vehicular Electronics and Safety (ICVES)*. IEEE. 2012, pp. 324–328.

[20] J. Branke et al., eds. *Multiobjective Optimization*. Springer, 2008.

[21] K. H. Brodersen et al. “The Balanced Accuracy and Its Posterior Distribution.” In: *International Conference on Pattern Recognition*. 2010.

[22] R. A. Brooks. “A robust layered control system for a mobile robot”. In: *IEEE Journal on Robotics and Automation* 2.1 (Mar. 1986), pp. 14–23.

[23] R. A. Brooks. “Elephants don’t play chess”. In: *Robotics and Autonomous Systems* 6.1 (1990), pp. 3–15.

[24] G. Brown. “Mapping Spatial Attributes in Survey Research for Natural Resource Management: Methods and Applications”. In: *Society & Natural Resources* 18.1 (Dec. 2004), pp. 17–39.

[25] R. O. Chavez-Garcia et al. “Image Classification for Ground Traversability Estimation in Robotics”. In: *Proceedings of the 18th International Conference on Advanced Concepts for Intelligent Vision Systems (ACIVS)*. Springer, 2017, pp. 325–336.

[26] R. O. Chavez-Garcia et al. “Learning Ground Traversability from Simulations”. In: *IEEE Robotics and Automation Letters* 3.3 (July 2018), pp. 1695–1702.

[27] B. Chazelle and D. P. Dobkin. “Decomposing a Polygon into its Convex Parts”. In: *Proceedings of the annual ACM symposium on Theory of computing (STOC)*. 1979.

[28] J.-W. Choi, R. Curry, and G. Elkaim. “Piecewise Bézier Curves Path Planning with Continuous Curvature Constraint for Autonomous Driving”. In: *Machine Learning and Systems Engineering*. 2010.

[29] E. Clementini, P. Di Felice, and P. van Oosterom. “A small set of formal topological relationships suitable for end-user interaction”. In: *Advances in Spatial Databases*. Springer, June 1993, pp. 277–295.

[30] E. G. Collins Jr et al. “Human-aware robot motion planning with velocity constraints”. In: *Proceedings of the International Symposium on Collaborative Technologies and Systems*. 2008, pp. 490–497.

[31] J. Current and M. Marsh. “Multiobjective transportation network design and routing problems: Taxonomy and annotation”. In: *European Journal of Operational Research* 65.1 (1993), pp. 4–19.

[32] I. Das and J. E. Dennis. “A closer look at drawbacks of minimizing weighted sums of objectives for Pareto set generation in multicriteria optimization problems”. In: *Structural optimization* 14.1 (Aug. 1997), pp. 63–69.

[33] I. Das and J. E. Dennis. “Normal-Boundary Intersection: A New Method for Generating the Pareto Surface in Nonlinear Multicriteria Optimization Problems”. In: *SIAM Journal on Optimization* 8.3 (1998), pp. 631–657.

[34] M. Davoodi et al. “Multi-objective path planning in discrete space”. In: *Applied Soft Computing* 13.1 (2013), pp. 709–720.

[35] K. Deb. “Multi-objective optimization”. In: *Search methodologies*. Springer, 2014, pp. 403–449.

[36] H. Delingette, M. Hebert, and K. Ikeuchi. “Trajectory generation with curvature constraint based on energy minimization”. In: *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (1991).

[37] D. Dey et al. “Gauss meets Canadian traveler: shortest-path problems with correlated natural dynamics.” In: *Proceedings of the International Conference on Autonomous Agents and Multi-agent Systems (AAMAS)*. 2014, pp. 1101–1108.

[38] M. Dorigo et al. “Swarmanoid: A Novel Concept for the Study of Heterogeneous Robotic Swarms”. In: *Robotics Automation Magazine, IEEE* 20.4 (Dec. 2013), pp. 60–71.

[39] R. M. Downs and D. Stea. *Cognitive Maps and Spatial Behaviour: Process and Products*. John Wiley & Sons, 2011.

[40] A. Dragan, K. Lee, and S. Srinivasa. “Legibility and predictability of robot motion”. In: *Proceedings of the ACM/IEEE International Conference on Human-Robot Interaction (HRI)*. Mar. 2013, pp. 301–308.

[41] P. Eyerich, T. Keller, and M. Helmert. “High-Quality Policies for the Canadian Traveler’s Problem”. In: *Third Annual Symposium on Combinatorial Search* (2010).

[42] M. Farrow and M. Goldstein. “Almost-Pareto Decision Sets in Imprecise Utility Hierarchies”. In: *Journal of Statistical Theory and Practice* 3.1 (2009), pp. 137–155.

[43] T. Fawcett and A. Niculescu-Mizil. “PAV and the ROC convex hull”. In: *Machine Learning* 68.1 (2007), pp. 97–106.

[44] D. Ferguson, A. Stentz, and S. Thrun. “PAO* for planning with hidden state”. In: *Proceedings of the International Conference on Robotics and Automation (ICRA)*. IEEE, 2004, 2840–2847 Vol.3.

[45] T. Flash and N. Hogan. “The coordination of arm movements: an experimentally confirmed mathematical model”. In: *Journal of neuroscience* 5.7 (1985), pp. 1688–1703.

[46] D. Fried et al. “Complexity of Canadian traveler problem variants”. In: *Theoretical Computer Science* 487 (2013), pp. 1–16.

[47] C. Fulgenzi, A. Spalanzani, and C. Laugier. “Probabilistic motion planning among moving obstacles following typical motion patterns”. In: *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2009, pp. 4027–4033.

[48] B. Giles-Corti and R. J. Donovan. “Relative Influences of Individual, Social Environmental, and Physical Environmental Correlates of Walking”. In: *American Journal of Public Health* 93.9 (Sept. 2003), pp. 1583–1589.

[49] B. Giles-Corti et al. “Increasing walking”. In: *American Journal of Preventive Medicine* 28.2 (Feb. 2005), pp. 169–176.

[50] M. A. Goodrich and A. C. Schultz. “Human-robot interaction: a survey”. In: *Foundations and TA Model of Artificial Emotions for Behavior-Modulationrends in Human-Computer Interaction* 1.3 (2007), pp. 203–275.

[51] S. Greco, J. Figueira, and M. Ehrgott. “Multiple criteria decision analysis”. In: *Springer's International series* (2005).

[52] S. Gulati and B. Kuipers. “High performance control for graceful motion of an intelligent wheelchair”. In: *2008 IEEE International Conference on Robotics and Automation (ICRA)*. 2008.

[53] J. Guzzi. *Ageing without Losing Mobility and Autonomy (ALMA). Deliverable D4.1 - Distributed information gathering and management*. Tech. rep. IDSIA, 2013.

[54] J. Guzzi. *Ageing without Losing Mobility and Autonomy (ALMA). Deliverable D4.3 - Path planning services*. Tech. rep. IDSIA, 2016.

[55] J. Guzzi and G. A. Di Caro. “From indoor GIS maps to path planning for autonomous wheelchairs”. In: *International Conference on Intelligent Robots and Systems (IROS)*. IEEE/RSJ. Oct. 2016, pp. 4773–4779.

[56] J. Guzzi and G. A. Di Caro. “Towards supporting elderly's orientation, mobility, and autonomy”. In: *Workshop on Improving the quality of life in the elderly using robotic assistive technology: benefits, limitations, and challenges - International Conference on Social Robotics (ICSR)*. Oct. 2015.

[57] J. Guzzi et al. “A Model of Artificial Emotions for Behavior-Modulation and Implicit Coordination in Multi-robot Systems”. In: *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*. 2018. to appear.

[58] J. Guzzi et al. “Artificial Emotions as Dynamic Modulators of Individual and Group Behavior in Multi-robot System”. In: *Proceedings of International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*. 2018. to appear.

[59] J. Guzzi et al. “Bioinspired obstacle avoidance algorithms for robot swarms”. In: *Proceedings of the International Conference on Bio-Inspired Models of Network, Information, and Computing Systems (BIONETICS)*. Springer, 2014, pp. 120–134.

[60] J. Guzzi et al. “Human-friendly robot navigation in dynamic environments.” In: *Proceedings of the International Conference on Robotics and Automation (ICRA)*. May 2013, pp. 423–430.

[61] J. Guzzi et al. “Local reactive robot navigation: A comparison between reciprocal velocity obstacle variants and human-like behavior”. In: *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2013, pp. 2622–2629.

[62] J. Guzzi et al. “On the Impact of Uncertainty on Path Planning”. In: *Federated AI for Robotics Workshop (FAIR)*. 2018. to appear.

[63] E. T. Hall. “A system for the notation of proxemic behavior”. In: *American Anthropologist* 65.5 (1963), pp. 1003–1026.

[64] L. He and J. van den Berg. “Meso-scale planning for multi-agent navigation”. In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. May 2013, pp. 2839–2844.

[65] D. Helbing, I. Farkas, and T. Vicsek. “Simulating dynamical features of escape panic.” In: *Nature* 407.6803 (2000), pp. 487–490.

[66] D. Helbing et al. “Self-organizing pedestrian movement”. In: *Environment and planning B* 28.3 (2001), pp. 361–384.

[67] D. Hennes et al. “Multi-robot collision avoidance with localization uncertainty”. In: *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*. 2012, pp. 4–8.

[68] Y.-J. Ho and J.-S. Liu. “Collision-free curvature-bounded smooth path planning using composite Bezier curve based on Voronoi diagram”. In: *Proceedings of the IEEE International Symposium on Computational Intelligence in Robotics and Automation (CIRA)*. 2009.

[69] *HRVO Library*. `http://gamma.cs.unc.edu/HRVO`.

[70] Q. Huang and H. Wang. *Fundamental study of jerk: evaluation of shift quality and ride comfort*. Tech. rep. SAE, 2004.

[71] M. Hutter et al. *Anymal-a highly mobile and dynamic quadrupedal robot*. Tech. rep. Robotic Systems Lab, ETH Zurich Zurich Switzerland, 2016.

[72] J. Hwang, T. Park, and W. Hwang. "The effects of overall robot shape on the emotions invoked in users and the perceived personalities of robot". In: *Applied ergonomics* 44.3 (2013), pp. 459–471.

[73] J.-H. Hwang, R. C. Arkin, and D.-S. Kwon. "Mobile robots at your fingertip: Bezier curve on-line trajectory generation for supervisory control". In: *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2003.

[74] H. Jun and Z. Qingbao. "Multi-objective Mobile Robot Path Planning Based on Improved Genetic Algorithm". In: *Proceedings of the International Conference on Intelligent Computation Technology and Automation*. Vol. 2. May 2010, pp. 752–756.

[75] H. Kanoh and K. Hara. "Hybrid Genetic Algorithm for Dynamic Multi-objective Route Planning with Predicted Traffic in a Real-world Road Network". In: *Proceedings of the Annual Conference on Genetic and Evolutionary Computation*. GECCO '08. ACM, 2008, pp. 657–664.

[76] S. Karaman and E. Frazzoli. "Sampling-based algorithms for optimal motion planning". In: *The International Journal of Robotics Research* 30.7 (June 2011), pp. 846–894.

[77] R. Kirby, R. Simmons, and J. Forlizzi. "COMPANION: A constraint-optimizing method for person-acceptable navigation". In: *Proceedings of the IEEE International Symposium on Robot and Human Interactive Communication*. 2009, pp. 607–612.

[78] B. Kluge and E. Prassler. "Recursive agent modeling with probabilistic velocity obstacles for mobile robot navigation among humans". In: *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Vol. 1. 2003, pp. 376–380.

[79] R. Knepper and D. Rus. "Pedestrian-inspired sampling-based multi-robot collision avoidance". In: *Proceedings of the International Symposium on Robot and Human Interactive Communication*. 2012.

[80] T. Kruse et al. "Legible robot navigation in the proximity of moving humans". In: *Proceedings of IEEE Workshop on Advanced Robotics and its Social Impacts (ARSO)*. 2012, pp. 83–88.

[81] M. Kuderer, H. Kretzschmar, and W. Burgard. "Teaching mobile robots to cooperatively navigate in populated environments". In: *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Nov. 2013, pp. 3138–3143.

[82] M. Kuderer et al. “Feature-based prediction of trajectories for socially compliant navigation”. In: *Proceedings of Robotics: Science and Systems*. 2012.

[83] M. Kull, T. S. Filho, and P. Flach. “Beta calibration: a well-founded and easily implemented improvement on logistic calibration for binary classifiers”. In: *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*. Vol. 54. PMLR, Apr. 2017, pp. 623–631.

[84] K. J. Kyriakopoulos and G. N. Saridis. “Minimum jerk path generation”. In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. Vol. 1. Apr. 1988, pp. 364–369.

[85] C.-P. Lam et al. “Human-centered robot navigation – Toward a harmoniously coexisting multi-human and multi-robot environment”. In: *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2010, pp. 1813–1818.

[86] G. B. Lamont, J. N. Slear, and K. Melendez. “UAV swarm mission planning and routing using multi-objective evolutionary algorithms”. In: *Proceedings of the IEEE Symposium on Computational Intelligence in Multicriteria Decision Making*. IEEE. 2007, pp. 10–20.

[87] J.-C. Latombe. “Exact Cell Decomposition”. In: *Robot Motion Planning*. Springer, 1991, pp. 200–247.

[88] S. LaValle. *Planning algorithms*. Cambridge University Press, 2006.

[89] K. J. Li and J. Lee. “Indoor spatial awareness initiative and standard for indoor spatial data”. In: *Proceedings of the Workshop on Standardization for Service Robot, IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2010.

[90] C. Lichtenthäler et al. “Increasing perceived value between human and robots – Measuring legibility in human aware navigation”. In: *Proceedings of the IEEE Workshop on Advanced Robotics and its Social Impacts (ARSO)*. 2012, pp. 89–94.

[91] C. Lichtenthäler and A. Kirsch. “Goal-predictability vs. Trajectory-predictability: Which Legibility Factor Counts”. In: *Proceedings of the ACM/IEEE International Conference on Human-robot Interaction (HRI)*. ACM, 2014, pp. 228–229.

[92] Z. Liu, D. Chen, and G. von Wichert. “2D Semantic Mapping on Occupancy Grids.” In: *Proceedings of the German Conference on Robotics (ROBOTIK)*. May 2012, pp. 1–6.

[93] D. Lu and W. Smart. “Towards more efficient navigation for robots and humans”. In: *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Nov. 2013, pp. 1707–1713.

[94] M. Luber et al. “People tracking with human motion predictions from social forces”. In: *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)*. 2010, pp. 464–469.

[95] M. Luperto, A. Quattrini Li, and F. Amigoni. “A System for Building Semantic Maps of Indoor Environments Exploiting the Concept of Building Typology”. In: *RoboCup 2013: Robot World Cup XVII*. Springer Berlin Heidelberg, 2014, pp. 504–515.

[96] E. Machuca and L. Mandow. “Multiobjective heuristic search in road maps”. In: *Expert Systems with Applications* 39.7 (2012), pp. 6435–6445.

[97] E. Machuca et al. “An empirical comparison of some multiobjective graph search algorithms”. In: *Proceedings of the Annual Conference on Artificial Intelligence AAAI*. Springer. 2010, pp. 238–245.

[98] L. Mandow and J. P. de la Cruz. “Comparison of heuristics in multiobjective A* search”. In: *Proceedings of the Conference of the Spanish Association for Artificial Intelligence*. Springer. 2005, pp. 180–189.

[99] R. Marler and J. Arora. “Survey of multi-objective optimization methods for engineering”. In: *Structural and Multidisciplinary Optimization* 26.6 (Apr. 2004), pp. 369–395.

[100] K. Melo, T. Horvat, and A. J. Ijspeert. “K-Rock, a Bio-robot Outside the Lab, Back In Nature”. In: *Proceedings of the International Symposium on Adaptive Motion of Animals and Machines (AMAM)*. 2017.

[101] Y. Morales, A. Watanabe, and F. Ferreri. “Including human factors for planning comfortable paths”. In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. 2015.

[102] A. I. Mouaddib. “Multi-objective decision-theoretic path planning”. In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. 2014.

[103] M. Moussaïd, D. Helbing, and G. Theraulaz. “How simple rules determine pedestrian behavior and crowd disasters.” In: *Proceedings of the National Academy of Sciences of the United States of America* 108.17 (2011), pp. 6884–6888.

[104] M. Moussaïd et al. “Experimental study of the behavioural mechanisms underlying self-organization in human crowds.” In: *Proceedings of the Royal Society B: Biological sciences*. Vol. 276. 1668. Aug. 2009, pp. 2755–2762.

[105] M. Moussaïd et al. “Traffic instabilities in self-organized pedestrian crowds.” In: *PLoS Computational Biology* 8.3 (2012), e1002442. arXiv: `1203.5267`.

[106] S. C. Narula and V. Vassilev. “An interactive algorithm for solving multiple objective integer linear programming problems”. In: *European Journal of Operational Research* 79.3 (1994), pp. 443–450.

[107] E. Nikolova and D. R. Karger. “Route Planning under Uncertainty - The Canadian Traveller Problem.” In: *Proceedings of the Annual Conference on Artificial Intelligence (AAAI)* (2008).

[108] A. Nüchter and J. Hertzberg. “Towards semantic maps for mobile robots.” In: *Robotics and Autonomous Systems* (2008).

[109] OGC. *IndoorGML core schema*. `http://schemas.opengis.net/indoorgml/1.0/indoorgmlcore.xsd`. 2016.

[110] OGC. *IndoorGML navigation schema*. `http://schemas.opengis.net/indoorgml/1.0/indoorgmlnavi.xsd`. 2016.

[111] S. Ön and A. Yazici. “A comparative study of smooth path planning for a mobile robot considering kinematic constraints”. In: *Proceedigns of the International Symposium on Innovations in Intelligent Systems and Applications (INISTA)*. 2013.

[112] Open Geospatial Consortium Inc. *OGC IndoorGML v.1*. `http://www.opengeospatial.org/standards/indoorgml`. 2014.

[113] C. H. Papadimitriou and M. Yannakakis. “Shortest paths without a map”. In: *Theoretical Computer Science* (1991).

[114] A. Papadopoulos, L. Bascetta, and G. Ferretti. “Generation of human walking paths”. In: *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Nov. 2013, pp. 1676–1681.

[115] W. J. Park and R. M. Kil. “Pattern Classification With Class Probability Output Network.” In: *IEEE Trans. Neural Networks* (2009).

[116] A. Piazzi, C. G. Lo Bianco, and M. Romano. “Smooth Path Generation for Wheeled Mobile Robots Using η^3-Splines”. In: *Motion Control*. 2010.

[117] G. H. Polychronopoulos and J. N. Tsitsiklis. “Stochastic shortest path problems with recourse.” In: *Networks* (1996).

[118] M. J. D. Powell. “A Direct Search Optimization Method That Models the Objective and Constraint Functions by Linear Interpolation”. In: *Advances in Optimization and Numerical Analysis*. 1994.

[119] H. Prautzsch, W. Boehm, and M. Paluszny. *Bézier and B-Spline Techniques*. Springer, 2002.

[120] P. Raja and S. Pugazhenthi. “Optimal path planning of mobile robots: A review”. In: *International Journal of Physical Sciences* 7.9 (2012), pp. 1314–1320.

[121] P. Ratsamee et al. “Social navigation model based on human intention analysis using face orientation”. In: *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2013, pp. 1682–1687.

[122] J. Rios-Martinez et al. “Navigating between people: a stochastic optimization approach”. In: *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)*. Vol. 231855. 2012, pp. 2880–2885.

[123] F. Rubio et al. “Industrial robot efficient trajectory generation without collision through the evolution of the optimal trajectory”. In: *Robotics and Autonomous Systems* 86 (2016), pp. 106–112.

[124] M. Rufli, J. Alonso-Mora, and R. Siegwart. “Reciprocal Collision Avoidance With Motion Continuity Constraints”. In: *IEEE Transactions on Robotics* 29.4 (Aug. 2013), pp. 899–912.

[125] J. Rupert G Miller. *Survival Analysis*. John Wiley & Sons, 2011.

[126] *RVO2 Library*. `http://gamma.cs.unc.edu/RVO2`.

[127] O. F. Sahin and V. Aksakalli. “A Comparison of Penalty and Rollout-Based Algorithms for the Canadian Traveler Problem”. In: *International Journal of Machine Learling and Computing* 5.4 (2015), pp. 319–324.

[128] L. Scandolo and T. Fraichard. “An anthropomorphic navigation scheme for dynamic scenarios”. In: *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)*. 2011, pp. 809–814.

[129] P. R. Schrater, D. C. Knill, and E. P. Simoncelli. “Mechanisms of visual motion detection.” In: *Nature Neuroscience* 3.1 (2000), pp. 64–68.

[130] H. Seki, T. Sugimoto, and S. Tadakuma. “Novel driving control of power assisted wheelchair based on minimum jerk trajectory”. In: *IEEJ Transactions on Electronics, Information and Systems* 125 (2005), pp. 1133–1139.

[131] Z. Shillert and P. Fiorini. "Motion planning in dynamic environments using velocity obstacles". In: *The International Journal of Robotics Research* 17.7 (1998), pp. 760–772.

[132] E. A. Sisbot et al. "Navigation in the presence of humans". In: *Proceedings of IEEE-RAS International Conference on Humanoid Robots*. 2005, pp. 181–188.

[133] E. A. Sisbot et al. "A mobile robot that performs human acceptable motions". In: *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2006, pp. 1811–1816.

[134] N. H. Sleumer and N. T. Gurman. *Exact Cell Decomposition of Arrangements used for Path Planning in Robotics*. Tech. rep. ETHZ, 199.

[135] J. Snape et al. "The hybrid reciprocal velocity obstacle". In: *IEEE Transactions on Robotics* 27.4 (2011), pp. 696–706.

[136] J. Snape et al. "Smooth and collision-free navigation for multiple robots under differential-drive constraints". In: *Proceedings of the International Conference on Intelligent Robots and Systems (IROS)*. 2010, pp. 4584–4589.

[137] A. Stentz. "Optimal and efficient path planning for partially-known environments". In: *Proceedings of the International Conference on Robotics and Automation (ICRA)*. IEEE. 1994, pp. 3310–3317.

[138] R. E. Steuer. *Multiple criteria optimization: theory, computation, and application*. Wiley, 1986.

[139] B. S. Stewart and C. C. White III. "Multiobjective A*". In: *Journal of the ACM* 38.4 (Oct. 1991), pp. 775–814.

[140] M. Svenstrup, T. Bak, and H. J. Andersen. "Trajectory planning for robots in dynamic human environments". In: *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2010, pp. 4293–4298.

[141] Y. Tamura, T. Fukuzawa, and H. Asama. "Smooth collision avoidance in human-robot coexisting environment". In: *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2010, pp. 3887–3892.

[142] P. W. Thorndyke and B. Hayes-Roth. "Differences in spatial knowledge acquired from maps and navigation". In: *Cognitive Psychology* 14.4 (Oct. 1982), pp. 560–589.

[143] P. Trautman et al. “Robot navigation in dense human crowds: the case for cooperation”. In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. May 2013, pp. 2153–2160.

[144] P. Trautman and A. Krause. “Unfreezing the robot: navigation in dense, interacting crowds”. In: *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2010, pp. 797–803.

[145] J. M. M. Tur, C. Zinggerling, and A. C. Murtra. “Geographical information systems for map based navigation in urban environments”. In: *Robotics and Autonomous Systems* 57.9 (2009), pp. 922–930.

[146] A. Turner et al. “From Isovists to Visibility Graphs: A Methodology for the Analysis of Architectural Space”. In: *Environment and Planning B: Planning and Design* 28.1 (Feb. 2001), pp. 103–121.

[147] S. Wang et al. “Sensor-based dynamic trajectory planning for smooth door passing of intelligent wheelchairs”. In: *Proceedings of the Computer Science and Electronic Engineering Conference (CEEC)*. 2013, pp. 7–12.

[148] C. Weinrich et al. “Prediction of human collision avoidance behavior by lifelong learning for socially compliant robot navigation”. In: *Proceedings of the International Conference on Robotics and Automation (ICRA)*. May 2013, pp. 376–381.

[149] M. Wermelinger et al. “Navigation planning for legged robots in challenging terrain”. In: *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2016, pp. 2153–0866.

[150] D. Wilkie, J. van den Berg, and D. Manocha. “Generalized velocity obstacles”. In: *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2009, pp. 5573–5578.

[151] A. Wu and J. P. How. “Guaranteed infinite horizon avoidance of unpredictable, dynamically constrained obstacles”. In: *Autonomous Robots* 32.3 (2012), pp. 227–242.

[152] K. Yamori. “Going with the flow: Micro–macro dynamics in the macrobehavioral patterns of pedestrian crowds.” In: *Psychological review* 105.3 (1998), pp. 530–557.

[153] X. Yang et al. “Route Selection for Railway Passengers: A Multi-objective Model and Optimization Algorithm”. In: *Journal of Transportation Systems Engineering and Information Technology* 13.5 (2013), pp. 72–100.

[154] W. Yanyang, W. Tietao, and Q. Xiangju. "Study of multi-objective fuzzy optimization for path planning". In: *Chinese Journal of Aeronautics* 25.1 (2012), pp. 51–56.

[155] A. B. E. Yasuaki, M. Yoshiki, and Y. Abe. "Collision avoidance method for multiple autonomous mobile agents by implicit cooperation". In: *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Vol. 3. 2001, pp. 1207–1212.

[156] J. Y. Yen. "Finding the K-Shortest Loopless Paths in a Network". In: *Management Science* 17.11 (1971), pp. 712–716.

[157] B. Zadrozny and C. Elkan. "Transforming classifier scores into accurate multiclass probability estimates". In: *International Conference on Knowledge Discovery and Data Mining*. ACM. 2002, pp. 694–699.

[158] H. Zender et al. "Conceptual spatial representations for indoor mobile robots". In: *Robotics and Autonomous Systems* 56.6 (June 2008), pp. 493–502.

[159] Y. Zhang, D.-w. Gong, and J.-h. Zhang. "Robot path planning in uncertain environment using multi-objective particle swarm optimization". In: *Neurocomputing* 103 (2013), pp. 172–185.

[160] B. D. Ziebart et al. "Planning-based prediction for pedestrians". In: *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2009, pp. 3931–3936.

[161] E. Zitzler, K. Deb, and L. Thiele. "Comparison of multiobjective evolutionary algorithms: Empirical results". In: *Evolutionary computation* 8.2 (2000), pp. 173–195.