# From indoor GIS maps to path planning for autonomous wheelchairs

Jérôme Guzzi and Gianni A. Di Caro

***Abstract*— This work focuses on how to compute trajectories for an autonomous wheelchair based on indoor GIS maps, in particular on IndoorGML maps, which set the standard in this context. Good wheelchair trajectories are safe and comfortable for the user and the people sharing the space with him, turn gently, are high legible, and smooth (at least $G^2$ continuos). We derive a navigation graph from a given IndoorGML map. We define and solve an optimization problem to find the desired path: given a succession of cells to traverse, the path corresponds to the best composite Bézier trajectory for the wheelchair. We discuss a related multi-objective path planning problem. Experimental results and an implementation on real robots show the planner performance.**

## I. Introduction

With the ageing of the population and the increasing need for assistive devices to support elderly's quality of life, robots are going to leave the research labs and begin to populate nursing homes, hospitals, and even houses. In particular, intelligent robotic wheelchairs will represent an important support for people with limited mobility to help maintaining their autonomy. Semi-autonomous wheelchairs will assist the driver when traversing difficult areas, such as crowded and cluttered spaces. Fully autonomous wheelchairs will serve people unable to drive and could be ordered from a remote location, like a taxi.

In this scenario, a set of fundamental questions arise. Which trajectories should a robotic wheelchair follow to be efficient but also safe and comfortable for the user? Which spatial knowledge is necessary to accomplish the task? How to gather it? Contrary to most indoor mobile robotics scenarios, we do not assume that the wheelchair uses an occupation map, built with local sensors. Instead, we consider that the wheelchair path planner relies on static floor maps, which may contain semantic information that facilitates the planning tasks.

In this paper, we address the above questions starting from the observation that indoor GIS maps contain all the (static) spatial information relevant to path planning and trajectory optimization in indoor spaces. In this respect, we adopt the new standard *IndoorGML* [1] to represent indoor maps, which we describe in Section III. Next we discuss how to derive the navigation graph for a wheelchair from the map. In Section IV, we describe a planner that compute the best trajectories for a wheelchair, taking into account the user comfort and the characteristics of indoor buildings. At this aim, composite Bézier curves are used. We define the related optimization problem and propose an algorithm to solve it approximately. Finally, in Section V, we address the multi-objective optimization problem that arises from the need to plan trajectories that are both comfortable for the user and short. A robot implementation (Section VI) and experimental results (Section VII) serves to evaluate, compare, and validate the different planners.

The paper's scope has three main limitations. First, we use, as a cost function for the optimization problem, a geometric feature that we expect to affect the user's comfort but that has not been validated yet through user studies. Nevertheless, the proposed system does not depend on the actual cost function being optimized and can be adapted to take into account additional geometric features. Second, for simplicity, we select the shortest route on the navigation graph and we focus on the best trajectory that traverse those spaces. In fact, humans pick routes by taking into consideration additional criteria, like when we prefer a longer but less crowded corridor. Lastly, the safety and comfort of a user depends also on the ergonomics and on the navigation behavior of the wheelchair, which should avoid dynamic obstacles that were not part of the map, like people, and modulate speed gently [2]. We expect to address these aspects within the multidisciplinary research project *Ageing without losing mobility and autonomy* [3] on technologies (ambient monitoring, user interface, assisted navigation, and control of (semi-) autonomous wheelchairs) to support the autonomous mobility and orientation of the elder, and, more in general, of the person with reduced mobility.

The paper main contribution is the formulation and solution of the trajectory optimization problem, and the presentation of a complete pipeline, based on IndoorGML, from data representation and gathering to robotic path planning.

## II. Related Work

Robots can extract semantic and topological information from occupancy grid maps built with their local sensors [4], [5], and robotic wheelchairs can take advantage of this information for path planning [6]. Path planning for wheelchairs on occupation grid maps uses cost functions that take into account the user's comfort, like lane preference and high visibility along the route [7].

Geographic information system (GIS) maps are commonly used by robots to navigate outdoors [8]. Recently, IndoorGML, a new standard for indoor maps that contains semantic, geometrical and topological information, has been

Authors are with the Dalle Molle Institute for Artificial Intelligence (IDSIA), USI-SUPSI, Manno-Lugano, Switzerland. E-mail: {jerome,gianni}@idsia.ch

This work was partially supported by the AAL Programme (ALMA project 2011-4-124).

approved [1]. IndoorGML maps partition the space into cells. In the context of (topological) path planning, cells are preferably convex. Convex cell decomposition of polygons is widely studied in robotics [9] as well in computational geometry [10] to achieve that goal.

Planning for smooth trajectories has been an important topic of research in robotics [11]. Robots controller have an easier task when following a smooth trajectory, which should ideally be $G^3$ continuos [12]. Splines and composite Bézier curves are families of smooth curves that researchers often look at for optimal smooth trajectories, for which the bend energy [13] – 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 [14]. Recent research has investigated the use of Bézier composite curves for planning trajectories along corridors [15], and across doors [16]. In this paper, we discuss how to plan an optimal trajectory that may traverse a whole building.

The wheelchair local navigation control may increase legibility by following pedestrian navigation rules [17], and favor comfort by reducing curvature [7] and modulating accelerations [2]. We focus instead on the contribution given by path planning.

Interestingly, drawing Bézier curves on a screen may also provide a human friendly interface for the remote control of a robotic wheelchair [18].

## III. Indoor spatial maps

The Open Geospatial Consortium has recently approved IndoorGML [1], a new standard that extends the Geographic Markup Language (GML) to describe indoor spaces and indoor navigation of heterogeneous agents [19]. The vision of the future is that an increasing number of buildings' floor plans will be stored in CAD formats like IFC, which contains rich geometrical and semantic information. Simple automatic tools will convert IFC data to IndoorGML maps. Users will visualize maps and get navigation instructions when entering inside a building, using IndoorGML to exchange spatial information.

An IndoorGML map is a *multi-layered graph of cells*. Each layer consists of a graph, where nodes are cells and edges are boundaries between cells. An edge may represent adjacency between cells or traversability across the common boundary. Additionally, IndoorGML maps contain interlayer edges to connect cells belonging to different layers.

In this paper, we assume that an IndoorGML floor map with at least one *geometric layer* is available. This layer contains all geometric information about the indoor spaces tagged by type: *navigable cell type*, like doors, stairs, rooms, or corridors, and *non navigable cell type*, like walls. Cells are geometrically described as polygons (possibly with holes) and cell boundaries as polylines. Doors, like all other spaces, are also described by cells[1].

[1]In IndoorGML terms, this is named *thick door model*.

A second layer, the *navigation layer*, is automatically derived from the geometric layer as following. We start by removing all cells that are non navigable by a wheelchair, like stairs. Next, we compute the non navigable area as the union of all walls, inflated by the wheelchair size plus a safety margin. For each navigable cell, we subtract the inflated non navigable area. We subdivide the resulting cells into convex cells (see Section III-A). If all navigable cells are convex, the trajectories are guaranteed to be feasible (see Section IV). Finally, for each pair of adjacent cells, an edge corresponding to the common boundary is added to the navigation graph. The resulting navigation layer represents the configuration space of the wheelchair as a *topological navigation graph* of cells, where routes are specified as a list of boundaries to cross.

### A. Cell decomposition for planning

The navigation layer should favor efficient path planning. In particular, to minimize computational time, it should contain a small number of cells. Moreover, trajectories are generally very constrained when crossing narrow passes (like along a corridor). Therefore we prefer to cut cells along short diagonals such that narrow passes are explicitly represented in the graph.

To keep the graph size small, the convexity constraint can be relaxed a little. We accept cells that are *quasi-convex*, that is cells for which the difference between them and their convex hull is very small. In any case, by adding a proper safety margin when inflating walls, trajectories will remain feasible even if passing through such quasi-convex cells.

The main idea of our cell decomposition algorithm is to sequentially subdivide non convex cells into quasi-convex cells by cutting them along the *shortest* diagonal that removes one of the non-convex vertices. An example of resulting navigation layer is illustrated in Fig 4.

## IV. Trajectory Planning

In this section, we assume that a topological path on the navigation graph is given that contains a list $(b_1, \ldots, b_n)$ of boundaries to pass while moving between the convex cells $(c_0, \ldots, c_n)$. Given start $(s)$ and end $(e)$ poses located in the first and last cells, we want to compute the best trajectory, for a wheelchair, that is contained inside the cells and crosses the boundaries in the right order.

### A. Cost as a variation of the path bend energy

Trajectories with small *curvature* $\kappa$ are beneficial for multiple reasons. The wheelchair can keep accelerations small, increasing in this way the comfort for a user sitting on it. For the same linear speed, the robotic wheelchair can reduce the angular speed, increasing performance (less slipping, less risk to lose localization). The paths may also look more natural, predictable and legible, which are important features when autonomous wheelchairs share the space with people.

Therefore we search for a trajectory $\gamma$ that minimizes the cost $C(\gamma)$, expressed as a variant of the *path bend energy*

that takes into account the variation of the curvature to reduce curvature changes:

$$C(\gamma) = \int_\gamma \left( \kappa(s)^2 + \left( \frac{\partial \kappa}{\partial s} \right)^2 \right) ds, \tag{1}$$

where $\gamma$ is parametrized by length $s$. Contrary to the original formulation of $C$ [16], the integral is not defined with respect to a specific parametrization, but as an intrinsic geometrical property of $\gamma$.

### B. Bézier composite curve

We search for the optimal $\gamma$ in the space of *$N$th order composite Bézier curves* [20], meaning that $\gamma$ is composed by successive Bézier curves of order $N$, one for each cell, smoothly joined at the cell boundaries. For each cell $c_i$, the curve segment $[0,1] \to c_i$ has the form

$$t \mapsto \sum_{j=0}^{N} P_j^{c_i} b_{j,N}(t), \tag{2}$$

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

$$b_{j,n}(t) = \binom{n}{j} t^j (1-t)^{n-j} \in [0,1], \text{ for } t \in [0,1] \tag{3}$$

and $P_j^{c_i} \in c_i$ 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 their use to approximate 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 also be contained. 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 join depends only on the adjacent $n$ control points:

$$\begin{aligned}
\gamma^c(0) &= P_0^c \\
\gamma^c(1) &= P_N^c \\
\dot{\gamma}^c(0) &= N\,(P_1^c - P_0^c) \\
\dot{\gamma}^c(1) &= N\left(P_N^c - P_{N-1}^c\right) \\
\kappa_{\gamma^c}(0) &= \frac{N-1}{N} \frac{(P_2^c - P_1^c) \times (P_1^c - P_0^c)}{\|P_1^c - P_0^c\|^3} \\
\kappa_{\gamma^c}(1) &= \frac{N-1}{N} \frac{\left(P_N^c - P_{N-1}^c\right) \times \left(P_{N-1}^c - P_{N-2}^c\right)}{\left\|P_N^c - P_{N-1}^c\right\|^3}.
\end{aligned} \tag{4}$$

$G^0$ geometric continuity (i.e. $C^0$ continuity for curves parametrized by length), requires that successive segments share the last, resp. first, control point. $G^1$ continuity needs that the three control points around the join to be collinear.

Furthermore, a trajectory, in order to be followed by a differential driven robot – like a typical wheelchair – should be $G^2$: the curvature (and consequently the angular speed) 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 also be necessary for a robot navigation controller to follow a curve [12]. 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.

In order to keep the number of degrees of freedom small, instead of searching in the space of 6th order Bézier curves, we first search for the optimal 4th order (cubic) Bézier composite curves. Then we transform the curve to a $G^2$ one (see Section IV-D).

*Parametrization:* We parametrize the $4n + 2$ degrees of freedom of the $3n + 2$ control points defining $\gamma$ as following. The curve passes through control points located at start $P_0^{c_0}$, end $P_3^{c_n}$, and on each boundary ($P_0^{c_i} = P_3^{c_{i-1}} \in b_i, i \in \{1, \ldots, n\}$). We linearly parametrize by $s^i \in [0,1]$ the position of $P_0^{c_i} = p_0(s^i; b_i)$ along $b_i$, where $s = 0, 1$ lie on the border. For the points $P_1^{c_0} = q_1(u_1^i; s)$ and $P_2^{c_n} = q_2(u_2^i; e)$ we use a polar parametrization with respect to $s$, respectively $e$, where $u_1^0, u_2^n \in [0,1]$ are radial distances divided by the cell length (the largest edge of the cell bounding box). Similarly, the points $P_{1,2}^{c_i}$, which control the derivatives at the cell boundaries, are polar parametrized with respect to $P_0^{c_i}$ with axis along $b_i$, i.e. $P_1^{c_i} = p_1(\alpha^i, u_1^i, s^i; b_i), P_2^{c_i} = p_2(\alpha^{i+1}, u_2^i, s^{i+1}; b_i)$. Figure 1 (left) illustrates the notation.

### C. Optimal trajectory

The parameters defining the optimal $\gamma = \gamma(\boldsymbol{u}, \boldsymbol{s}, \boldsymbol{\alpha})$ with respect to $C$ are obtained through the solution of the following non-linear constrained optimization problem:

$$\begin{aligned}
&\text{Minimize } C(\gamma(\boldsymbol{u}, \boldsymbol{s}, \boldsymbol{\alpha})) \\
&\text{Subject to} \\
&\qquad u_1^0, u_1^i, u_2^{i-1}, u_2^n, s^i, \alpha^i \in [0,1] \\
&\qquad p_1(\alpha^i, u_1^i, s^i; b_i) \in c_i \\
&\qquad p_2(\alpha^i, u_2^{i-1}, s^i; b_i) \in c_{i-1} \\
&\qquad q_1(u_1^0; s) \in c_0 \\
&\qquad q_2(u_2^n; e) \in c_n, \quad \forall i \in \{1, \ldots, n\},
\end{aligned} \tag{5}$$

which we solve numerically using constrained optimization by linear approximations (COBYLA solver) [21]. Since the solver works better for differentiable constraints, we reformulate all constraints of form $P_j^{c_i} \in c_i$ as:

$$P_j^{c_i} \in c_i \leftrightarrow d(P_j^{c_i}, \partial c_i) \geq 0, \tag{6}$$

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

Because of non linearity, the cost function may present several local minima. The COBYLA solver converges towards one of them, depending on the initial guess we provide. Therefore, a good initial estimate of the optimal parameters is needed. Figure 2) illustrates the convergence of two trajectories towards the local minima.

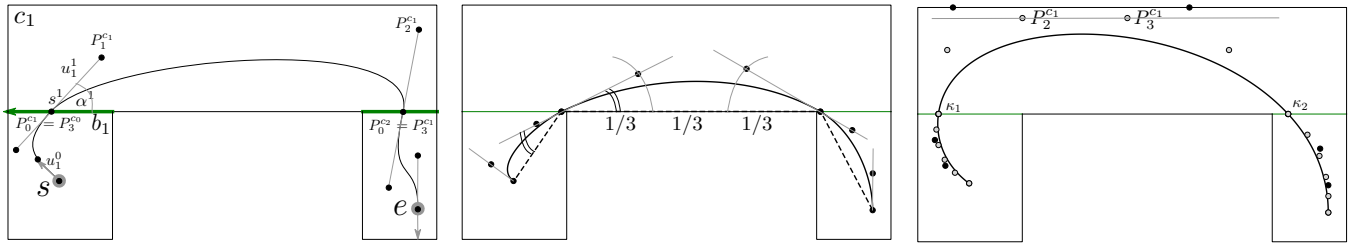


Fig. 1: Towards the generation of the optimal Bézier composite curve according to model 5. *Left:* start ($s$) and end ($e$) poses, and a topological path that traverses cells $(c_0, c_1, c_2)$ and crosses borders $(b_1, b_2)$, are given. The control points $P^c_{0,1,2,3}$ define the curve. They have four degrees of freedom at boundaries ($\alpha^1, s^1, u_1^1, u_2^0$ at $b_1$) and one DOF at start and end ($u_1^0$ at $s$). *Center:* The trajectory is initialized after computing the shortest path (dashed). *Right:* the $G^2$ trajectory is obtained by elevating the optimized 4th order trajectory (black control points) to 6th order (gray control points). The 6th order control points $P_2^{c_1}$ and $P_3^{c_1}$ are moved slightly along their connecting line to make the curvature continuos at $b_1$ and $b_2$.

*Initialization:* The initial guess is given by the following heuristics (see Figure 1 (center)), which gives satisfactory results even if the parameters are not further optimized. First, we find the shortest path (ignoring headings) from $s$ to $e$ that crosses $(b_1, \dots, b_n)$, which is a trajectory $\gamma_0$ made of line segments, one for each cell. The parameters $\boldsymbol{s}$ are chosen in order to place the points $P_0^{c_i}$ at the intersection of $\gamma_0$ with each boundary. The angles $\boldsymbol{\alpha}$ are initialized as a midway between the angles of the two incident segments of $\gamma_0$. $\boldsymbol{u}_{1,2}$ are set by placing the control points $P_2^i$ and $P_3^i$ at one third of the distance between $P_0^i$ and $P3^i$ (or on the cell borders if they would lie outside of the cell $c_i$).

### D. $G^2$ trajectory

4th order composite Bézier curves are not generally $G^2$ because of discontinuities in the curvature at the joins. The order of the segments should be increased at least to 6 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 4th order curve to a 6th order $G^2$ curve. The shape and cost $C(\gamma)$ will not be change significantly.

The degree of each segment is elevated to 6th order [20], (i.e., we find 6 control points that define the same Bézier curve). Then, for each join, we compute the curvatures $\kappa^i(0)$ and $\kappa^{i-1}(1)$ on the two sides (see Equation IV-B), which should be equal for the curve to be $G^2$. If they are different, we set the target curvature at boundary $b_i$ to

$$\kappa^i = \begin{cases} \sqrt{\kappa^i(0)\kappa^{i-1}(1)}, & \text{if } \kappa^i(0)\kappa^{i-1}(1) \geq 0 \\ 0, & \text{else} \end{cases} \tag{7}$$

and move the control point $P_2^{c_i}$ along the line passing by $P_2^{c_i}$ and $P_3^{c_i}$ as necessary to achieve curvature $\kappa^i$ (see Figure 1 (right)). Similarly, on the other side of the boundary, $P_3^{c_{i-1}}$ is moved along the line passing by $P_3^{c_{i-1}}$ and $P_2^{c_{i-1}}$.

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.

### E. Optimal trajectories for indoor buildings

*a) 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 passes perpendicularly in the middle when crossing doors. That is, $\alpha^i = \pi/2$ and $s^i = 1/2$ for all $i$ where $c_i$ or $c_{i-1}$ are doors.

This way of proceeding brings two advantages. First, legibility and predictability of the wheelchair trajectory near doors increase, where good negotiation of shared crossing with humans is critical. Second, the optimization problem is split into smaller problems, since the path $(b_1, \dots, b_n)$ can be subdivided into $m$ smaller chains

$$\big((b_1, \dots, b_{n_1}), (b_{n_1}, \dots, b_{n_2}), \dots, (b_{n_{m-1}}, \dots, b_{n_m})\big), \tag{8}$$

for which the trajectory is perpendicularly constrained into the middle point at $b_{n_i}, i = 1, \dots, m$. The optimization of one chain is *independent* from the other chains. We further impose the same constraints on cell boundaries that are very narrow, losing a little bit of freedom but reducing computational complexity. If all boundaries in a chain are constrained, like a sequence of doors, optimization is trivial and results in a straight line.

*b) Trajectories graph:* Contrary to start and end pose, 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 edge of a graph $G_{\text{opt}}$ with the nodes consisting of all doors and narrow passages.

Based on these insights, the performance of path computations at query time can be greatly speed up proceeding as following. Instead of optimizing the trajectory for the full topological path, the topological path is first subdivided into chains as described above. Then, the optimization is ran for the first and last chains (i.e., from $s$ to the first door and from the last door to $e$), while the optimized intermediate segments from $G_{\text{opt}}$ are retrieved from a pre-computed cache. In this

way, computational costs become almost independent of the number of boundaries crossed.

## V. Multi-objective path planning

In general, there may be multiple topological paths connecting two cells in the navigation layer. In this case, looking for the minimal curvature cost $C(\gamma)$ while neglecting other geometric features may be too limited. As a matter of fact, humans prefer *short* and less curved trajectories. Thus, we formulate a *multi-objective problem* that looks for trajectories of minimal length $L(\gamma)$ *and* cost $C(\gamma)$.

A dominated solution of a multi-objective problem is a solution for which there is at least another solution that is better with respect to every objective. No smart strategist would choose a dominated solution. Therefore, one general way to tackle multi-objective problem is to compute the set of non-dominated solutions, the *Pareto front* [22].

Computing the Pareto front of optimal trajectories between poses $s$ and $e$ is done as follows. For simplicity, let us assume that different trajectories have different costs. We will use Yen's single source k-shortest path algorithm [23] to compute the next best simple path on $G_{\text{opt}}$ between $s$ and $e$ with respect to a given (positive) edge cost.

First, we add to $G_{\text{opt}}$ all optimal trajectories from $s$ and to $e$ that originate from a node of $G_{\text{opt}}$ and do not cross any other node. Then, Yen's algorithm is applied, alternating between costs $L$ and $C$. At each step, if the returned solution is already part of the Pareto front, the iteration stops. Else, unless dominated, the solution is added to the Pareto front. The algorithm terminates because the number of simple paths is limited but does not need to check them all. In fact, it terminates as soon as it reaches a solution that is know to be better then the successive solutions with respect to both objectives.

If an approximation of the Pareto front is sufficient, we keep the query time limited by stopping the algorithm after a fixed number of iterations. In the case different trajectories have the same cost, the algorithm is slightly modified to stop only after all solutions with same costs are checked to be already part of the Pareto front.

## VI. Implementation

Floor maps are usually provided in CAD formats. Our custom software let us convert them to IndoorGML map and automatically add a navigation layer for wheelchairs, partitioned in navigable convex cells.

The planner has been implemented in Python making use of the following libraries: `scipy` for the COBYLA solver, `networkx` for the Yen's k-shortest path algorithm, and `shapely` for the computation of geometric constraints. The accuracy of the COBYLA algorithm can be tuned. We report in Section VII-A a test for the trade-off between accuracy and computational costs.

The robotic wheelchair is controlled through a ROS interface. The planner is integrated into `move_base`, the ROS package for mobile robots navigation, as a global planner plugin. Once optimized, the trajectory is sampled at fixed distances and the resulting list of poses is passed to the `move_base` local planner. The same ROS interface is used to control a TurtleBot robot.

## VII. Experiments

We report experimental results that evaluate the trajectory optimization performance and illustrate one example of solution for the multi-objective planner. All experiments are performed on a single core modern architecture CPU.

### A. Synthetic map

We begin by measuring the performance of the planner on a synthetic map, made by a succession of corners similar to a snake (see Figure 2). Two examples of trajectory optimization, shown on the top, converge slower (top right) and faster (top left) to the optimal solution. Figure 3 (left) reports how the computational cost grows almost quadratically with the length of the topological plan. The optimization solver can be tuned for faster or more accurate results. Figure 3 (right) illustrates the trade-off between computational cost and accuracy for paths of fixed length of 4 borders (i.e., with 18 degrees of freedom to optimize). For the experiments illustrated in the following, the planner is tuned for accuracy.

### B. 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 III by inflating walls by 40 cm and then decomposing the navigable space into convex cells. It contains about 750 node (cells) and 770 edges (cell's boundaries).

As detailed in Section IV-E, the planner initialization precomputes all trajectories between constrained boundaries, which form the edges of the graph $G_{\text{opt}}$. For our building, this requires about 3 minutes to compute 2000 trajectories. Figure 4 (left) displays the full map (top left) and an excerpt of the navigation layer together with trajectories in $G_{\text{opt}}$. Note how the curves originate perpendicularly 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 cost becomes almost linear in the length of the path (see Figure 4 (center)). On the other hand, at query time, we take advantage of $G_{\text{opt}}$ 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 second for almost all queries. This confirms that the planner is well suited for online planning for robots moving in real buildings.

### C. Multi-objective planning example

Figure 4 (right) displays the solution for a single multi-objective ($C$ vs $L$) planning problem between two poses in our building. For this case, the algorithm (Section V) terminates after three paths have been evaluated and return a Pareto front composed of two solutions. Only one additional

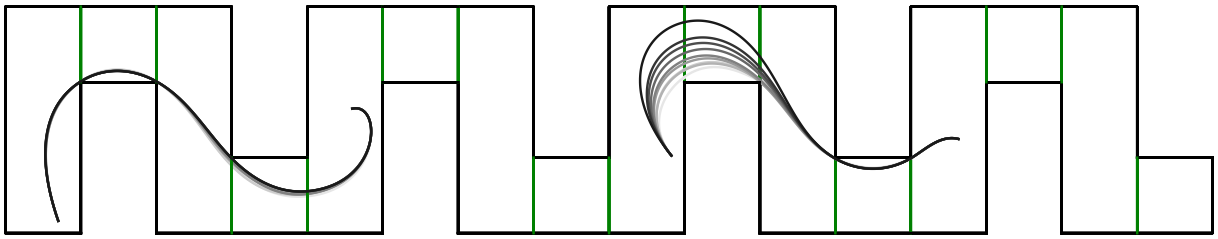

Fig. 2: Synthetic map with two examples of trajectory optimization. Darker colors mean further optimization (higher computational cost).

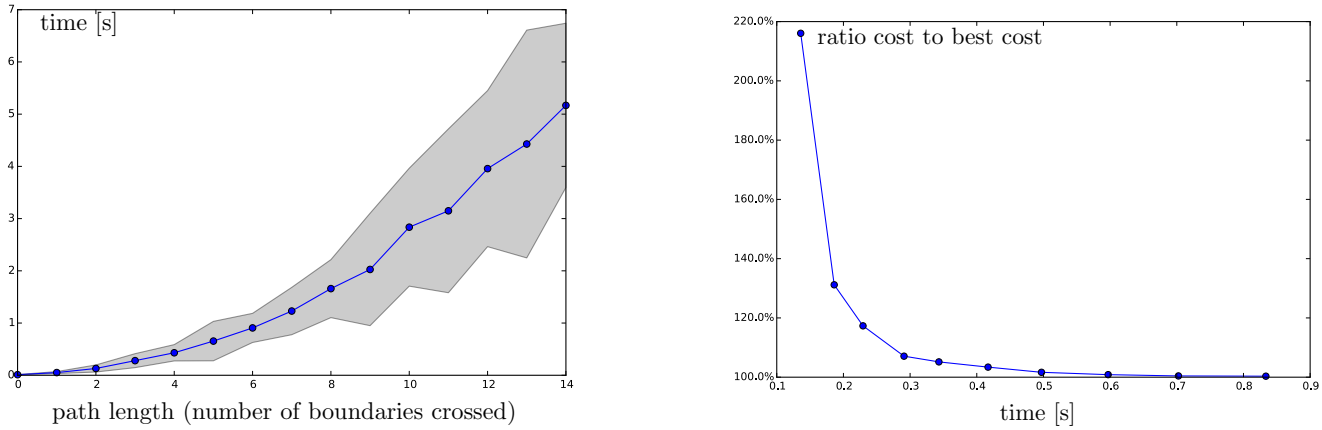


Fig. 3: Performance of trajectory on the synthetic map. *Left:* computational cost for optimizing trajectories between random poses (50 per length) of increasing (topological) length. Average $\pm$ one standard deviation shaded in grey. *Right*: trade-off between computational cost and curvature cost for 100 trajectories of length 4 between random poses.

dominated solution is computed. The planner give us the choice between the longer (left) and curvier (right) trajectory.

### D. Robot navigation

The robot localizes itself with a depth sensor, on the same IndoorGML map used by the planner. We configure the `move_base` local planner to keep its contribution to the final robot trajectory negligible.

Figure 5 and the accompanying video show some of the trajectories performed by the TurtleBot robot moving autonomously in our building. We compare our planner the default `move_base` global planners: `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 looks for smooth trajectories that turn gently, making use of the free space. They are easier to predict for a human observer and more similar to those produces by a skilled human driver.

## VIII. Conclusions

We discussed how to take advantage of indoor GIS maps to: (i) build a topological navigation graph, (ii) optimize a trajectory, (iii) perform multi-objective path planning to select short but comfortables trajectories for wheelchairs. We investigated the performance of the planner on synthetic maps. We showed that the planner is suitable for real time planning for robots in real world buildings.

After the first trials with TurtleBots, which we reported, we are currently carrying out experiments with actual robotic wheelchairs. We would like to evaluate the user's comfort and the acceptance of the robotic wheelchair by people sharing the space with it. The design of experiments that isolate the contribution of the path planner is challenging. For instance it requires that the controller, which let the wheelchair follow the planned trajectory while avoiding dynamical obstacles, work so reliable to be negligible.

Additionally, we focus our current research on more complex planning problems, where we take into account – in addition to the trajectory geometry – personal and social costs and risks to traverse spaces.

## References


[1] Open Geospatial Consortium Inc., "OGC IndoorGML v.1," http://www.opengeospatial.org/standards/indoorgml, 2014.

[2] 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.

[3] "Alma," http://www.alma-aal.org.

[4] Z. Liu, D. Chen, and G. von Wichert, "2D Semantic Mapping on Occupancy Grids." in *ROBOTIK*, 2012.

[5] A. Nüchter and J. Hertzberg, "Towards semantic maps for mobile robots." *Robotics and Autonomous Systems*, 2008.

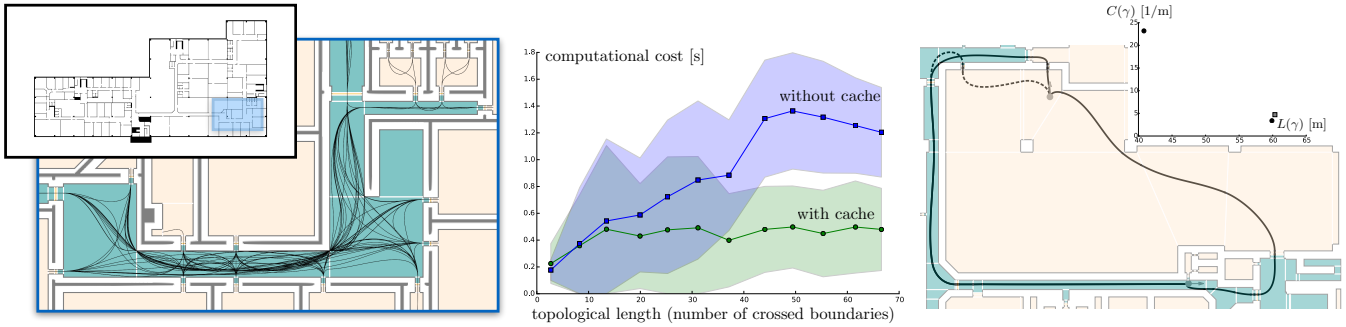


Fig. 4: *Left:* part of our building IndoorGML map with the optimal trajectories forming the edges of $G_{\text{opt}}$. Cells are colored by type (corridors green, rooms yellow, doors orange). *Center:* the average and standard deviation of the computational cost to optimize 1000 random trajectories vs. the topological length of the trajectory: full trajectory optimization at query time (squared blue markers) and optimization that uses pre-computed $G_{\text{opt}}$ (circular green markers). *Right:* multi-objective optimal paths between two poses (grey circles with arrow): Pareto optimal trajectories (solid lines) and the dominated solution (dashed line). *Top Right:* costs of the three trajectories – Pareto front (black circle), dominated solution (grey squares).

Fig. 5: *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 localisation and in actuation (center: our planner, right: `navfn` planner). The trajectories are featured in the accompanying video too.


[6] R. Li, L. Wei, D. Gu, H. Hu, and K. D. McDonald-Maier, "Multi-layered map based navigation and interaction for an intelligent wheelchair," in *ROBIO*, 2013.

[7] Y. Morales, A. Watanabe, and F. Ferreri, "Including human factors for planning comfortable paths," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, 2015.

[8] J. M. M. Tur, C. Zinggerling, and A. C. Murtra, "Geographical information systems for map based navigation in urban environments," *Robotics and Autonomous Systems*, 2009.

[9] J.-C. Latombe, "Exact Cell Decomposition," in *Robot Motion Planning*, 1991.

[10] B. Chazelle and D. P. Dobkin, "Decomposing a Polygon into its Convex Parts," *STOC*, 1979.

[11] S. Ön and A. Yazici, "A comparative study of smooth path planning for a mobile robot considering kinematic constraints," in *INISTA*, 2013.

[12] A. Piazzi, C. G. Lo Bianco, and M. Romano, "Smooth Path Generation for Wheeled Mobile Robots Using $\eta^3$-Splines," in *Motion Control*, 2010.

[13] H. Delingette, M. Hebert, and K. Ikeuchi, "Trajectory generation with curvature constraint based on energy minimization." *IROS*, 1991.

[14] Y.-J. Ho and J.-S. Liu, "Collision-free curvature-bounded smooth path planning using composite Bezier curve based on Voronoi diagram," in *CIRA*, 2009.

[15] 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.

[16] S. Wang, L. Chen, H. Hu, and K. McDonald-Maier, "Sensor-based dynamic trajectory planning for smooth door passing of intelligent wheelchairs," in *CEEC*, 2013.

[17] J. Guzzi, A. Giusti, L. M. Gambardella, G. Theraulaz, and G. A. Di Caro, "Human-friendly robot navigation in dynamic environments." in *ICRA*, 2013.

[18] 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 *IROS*, 2003.

[19] K. J. Li and J. Lee, "Indoor spatial awareness initiative and standard for indoor spatial data," in *IROS*, 2010.

[20] H. Prautzsch, W. Boehm, and M. Paluszny, *Bézier and B-Spline Techniques*. Springer, 2002.

[21] 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.

[22] J. Branke, K. Deb, K. Miettinen, and R. Słowiński, Eds., *Multiobjective Optimization*. Springer, 2008.

[23] J. Y. Yen, "Finding the K-Shortest Loopless Paths in a Network," *Management Science*, 1971.