Robust topological skeleton extraction from occupancy grids for mobile robot navigation

Fabrice Mayran de Chamisso[1*] Laurent Soulier[1] Michaël Aupetit[2]

[1] CEA, LIST, F-91191 Gif-sur-Yvette, France
[2] Qatar Computing Research Institute

laurent.soulier@cea.fr

May 19, 2016

Résumé

Le calcul d'une simplification topologique telle que le squelette topologique à partir d'une grille d'occupation décrivant les obstacles à proximité est l'un des facteurs-clés pour permettre la navigation de robots mobiles dans des environnements de grande taille. Cet article présente une méthode robuste d'extraction du squelette topologique à partir d'une carte vectorielle Euclidienne. Nous proposons des critères simples et efficaces calculatoirement pour réduire le nombre d'arêtes parasites du squelette en supprimant celles qu'un robot mobile ne pourrait physiquement traverser. L'emploi de plusieurs étapes de regroupement de pixels en arêtes et sommets permet ensuite d'extraire une représentation sous forme de graphe de la topologie de l'environnement. Cette approche utilise uniquement de l'arithmétique entière, sans faire appel à des fonctions trigonométriques ou transcendantes. La complexité temporelle de l'extraction est en pratique linéaire en le nombre de pixels présents dans la grille d'occupation.

Mots Clés

extraction de topologie, squelette topologique, carte vectorielle Euclidienne, robot mobile, grille d'occupation, navigation, SLAM topologique

Abstract

Computing a topological simplification such as the topological skeleton from an occupancy grid describing nearby obstacles is one of the key factors in achieving efficient mobile robot navigation in large-scale environments. We propose a robust process to extract the topological skeleton from a vectorial Euclidean distance map. In order to reduce the amount of parasitic edges due to noise in the occupancy grid, we propose simple and computationally efficient criteria to detect and remove edges that a mobile robot would not be able to physically traverse. Multiple clustering steps are then used to define edges and vertices and extract a graph representation of the skeleton. The approach only uses integer arithmetics and does not require trigonometry or the use of transcendent functions. On real-world data, skeleton extraction runs close to linear time complexity in the number of pixels in the grid.

Keywords

topology extraction, topological skeleton, vectorial Euclidean distance map, mobile robot, occupancy grid, navigation, topological SLAM

1 Introduction - motivations

A mobile robot is an autonomous system which has to physically navigate inside an environment, with or without a map of said environment. Some of the basic tasks a mobile robot has to perform include localizing itself within the environment - eventually while drawing a map of said environment, for a task named Simultaneous Localization and Mapping or SLAM [4, 13]-, planning its path from some origin to some destination, moving along the planned path and avoiding obstacles. This last task is critical for safety when interacting with humans.

We consider the problem of a mobile robot which acquired a local map of its environment in the form of an occupancy grid, which is a discretized representation of the probability density that space around the robot is occupied [14, 29]. Each pixel of the grid encodes as a log-odd the probability that space at its position is occupied (log-odd above zero) or empty (log-odd under zero).

Recent SLAM algorithms use topological or hybrid metrical-topological mapping [3, 6, 8, 21, 25, 28] where the map is a set of vertices describing discrete places and a set of edges describing the transformations between vertices. While being more robust to large-scale uncertainty and odometric drift than dense representations, topological representations require vertices to be robustly defined from sensor data in the sense that multiple observations of

*Funding: this work was supported by a DGA-MRIS scholarship

a place will lead to the same vertices. The reliability of vertex detection is usually not considered [8] or left aside as a secondary issue [21]. Reliability is however discussed in depth in [5].
In this paper, we suppose that a robot wants to extract the topology of the environment around the origin of the grid $(0, 0)$. In most cases, this position is the robot's own (known or estimated) position. The robot is only interested in paths it can use to navigate, that is paths that:

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

We propose a lightweight topology extraction method "from pixels to graph" using a topological skeleton computed on a local occupancy grid through the *vectorial Euclidean distance map* (vEDM) [12]. The vEDM contains for each empty pixel of the grid a vector pointing to the closest occupied pixel. We use the vEDM since it can be computed using *integer arithmetics*, resulting in faster computations especially on processors without floating point units. The vEDM may also be reused to perform obstacle avoidance using virtual repulsive forces [7].
In addition to the topology extraction method, we propose criteria to simplify the skeleton by removing spurious edges which the robot would not be able to *physically traverse*. Compared to existing approaches, this removal does not make assumptions on yet unknown space and is only parametrized by the robot's size, making it both more intuitive from a robotics point of view and more *robust to occlusions* and noise. Finally, the approach achieves *close to linear time complexity* in the number of pixels in the grid on real-world datasets, exceeding 100 frames per second on square grids of size 300×300 pixels.

2 State of the art of topology extraction

The topological skeleton, sometimes called 'Generalized Voronoï Diagram' or GVG [2, 10] is a convenient topological representation of an environment which has already been used for SLAM [10, 28]. Edges of the skeleton are the set of points equidistant from two or more obstacles while vertices are the set of points where edges meet or end. Computing the skeleton from a discrete representation such as an occupancy grid is traditionally done through homotopic thinning, wavefront propagation or distance map calculation. The vast state of the art of these approaches is reviewed in [2]. Briefly, (distance ordered) homotopic thinning, though easily parallelizable, leads to hard-to-exploit skeletons. Moreover, the skeleton simplification techniques such as only connecting the center of maximal balls [26] lead to skeletons unstable with grid updates since some centers may appear or disappear. Wavefront propagation techniques need complex models to describe the wavefront. Finally, distance maps [1, 12, 24] need to be computed as a first step and exploiting them requires reconstruction of ridge lines which is a process sensitive to noise and may lead to 'holes' in the skeleton, an issue addressed by Li and Vossepoel [22]. A comprehensive review of existing distance transform algorithms can be found in [17] and [15]. Alternatively to a distance transform, an Euclidean feature transform [18] can be used to diagnose the ridges directly. This does not however address the issue of noise.
One of the consequences of noise in the grid is the presence of spurious edges on the skeleton. Simplification of the skeleton is a known issue discussed notably in [2, 10, 26] and [16]. Examples without simplification of the skeleton can be found in [22]. They show that a simplification is indeed necessary for robot navigation. In order to remove spurious edges, Choset et al. [10] proposed to discard edges terminating near a wall (dead ends). While perfectly valid in static situations, this approach becomes unstable when the grid gets updated: sometimes, the bottom of a dead end can't be seen from the current origin of the grid. If the robot moves back and forth, the edge may keep appearing and disappearing, causing navigation algorithms to get stuck in an infinite loop. The same issue affects the "gateway" approach of Beeson et al. [5] which distinguishes edges terminating in free space as possible exits and only consider shortest paths leading to the exits as valid. Indeed, the "exits" may actually be unseen dead ends. Attali [2], Malandain et al. [23] and Couprie et al. [11] proposed criteria based on bissector angles (angles between the vectors pointing to the two obstacles equidistant from an edge) to remove spurious edges. While these criteria do not make assumptions on dead ends, they require trigonometric computations which are expensive and require floating-point representations. Note that bissector angles can easily be recovered from the vEDM.

3 Our approach

3.1 Notations

Suppose that the dimensions of the grid are (w, h). We write L_1 ("Manhattan") distances using simple bars $|.|$ and Euclidean distances using double bars $||.||$. We consider that the skeleton has to be computed only within range $r \leq min(w, h)$ of the origin (Euclidean distance used for range), represented by a dark red dashed circle on Figure 1. The skeleton is computed in the Euclidean space. We use the term "pixel cluster" as a set of pixels where each pixel is connected to the cluster horizontally, vertically or diagonally. Finally, we suppose that the center of the robot is not allowed to get closer to an obstacle than some fixed dis-

tance D. While D may take into account both the robot's size and a minimum allowed distance from any part of the robot to any obstacle, we call it "robot size" for simplicity.

3.2 Overview of our approach

The following steps are required in our approach to compute the GVG or Extended GVG [5] from an occupancy grid around a given origin and extract it in a graph format:

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

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

3. From the origin, radially propagate a wavefront (Figure 1). Propagation stops on a specific pixel if it is occupied or if the pixel is too far from the origin (or outside the grid). For the most part, propagation only reaches line of sight pixels. Step 3 is an optimization specific to robot navigation and could be ignored by marking all occupied pixels directly in one pass over the image (see Figure 4), for a new step 3'.

4. Using the occupied pixels detected at step 3/3', compute a vEDM (see Figure 2), that is associate each pixel to a vector pointing to the closest occupied pixel.

5. (Figure 2) For each pixel with a vector computed at step 4, test whether or not it belongs to an acceptable edge (where 'acceptable' will be properly defined later).

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

7. For each remaining 'edge' pixel, detect whether or not it is a vertex, where a vertex is defined as the intersection of two edges or as the end point of an edge (Figure 3).

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

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

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

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

3.3 Computing a vector map (steps 3 and 4)

Initially, the vector field is set to $\mathbf{V_{xy}} = (\infty, \infty)$ or to values bigger than $\mathbf{V_{xy}} = (w, h)$ for every pixel (x, y) in the grid. During step 3, pixels are considered in an order defined by a growing circle using the L_1 distance, as sketched on figure 1. Each time an empty pixel is considered, its empty neighbors are marked to be considered during the next pass if they have not been considered already and if they are not further than r from the origin (Euclidean distance). This expansion phase stops if no vertex was marked during a pass. Each time an occupied vertex is marked, its associated vector in the vector field is set to $(0; 0)$. This way of proceeding tends to confine the search for the skeleton to the zone actually visible by the robot, or in other words, the *interior* of the environment, shown in green on figure 1. It avoids computing the part of the skeleton that lies in zones not accessible from the origin of the grid without traversing occupied space. The added computational burden of step 3 is balanced by the simplification of steps 4 to 10. The interest of step 3 is clear on the figure: the vector field has mostly been calculated *inside* the building, even though cracks (places where the log-odd approaches 0) in the wall on the bottom-left corner led to some calculations being carried outside the building. Figure 4 shows the actual decrease in computation time due to step 3.

We suppose that at least one occupied pixel was reached during step 3 (otherwise, the skeleton is empty). Step four computes a field of vectors to the closest occupied pixel, which is very like distance map calculation but vectors contain both distance and direction. Linear complexity algorithms exist to compute Euclidean distance maps, such as that of [9] based on incremental construction of the Voronoï cells or that of [19] based on building an intermediate map of closest obstacles along the vertical axis. A review of modern parallel approaches to Euclidean distance map calculation is given in [24]. The difficulty of constructing an Euclidean distance (or vectorial) map comes from the fact that the boundary of Voronoï cells (a cell is defined by the set of pixels closer to one occupied pixel, the *seed*, then to any other occupied pixel) is not restricted to the boundary of pixels and the local width of a cell may be arbitrarily small.

Instead of an exact algorithm, we chose to use the approximate algorithm of Danielsson [12] to compute the vEDM. This algorithm propagates vectors from pixels to their neighbors, where occupied pixels start with a vector

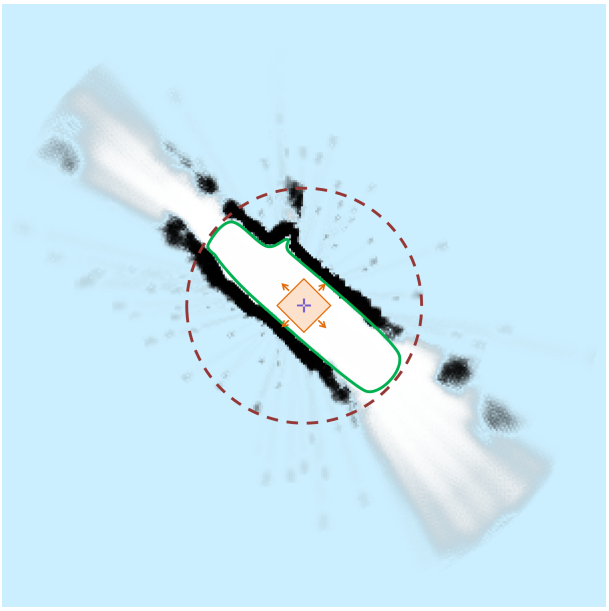

Figure 1: The role of *step 3*: discriminating an interior and an exterior part in the environment. Maximum detection range in dark red (with dashes). Occupied pixels in black, free pixels in white. Shades of grey and light blue represent space whose occupancy state is unknown so far (light blue is completely unknown). Expansion using the L_1 distance in orange. Resulting visible zone in green.

of $(0, 0)$. With an eight-neighbor propagation scheme, the algorithm was demonstrated in [12] to be completely accurate for a pixel when there exists a path from the closest occupied pixel to it within the Voronoï cell. Otherwise, the maximum error of the algorithm is 0.076 pixels. There are extremely few cases where the algorithm is not exact. This algorithm does not require dynamical allocation or sorting and does not use a stack, as opposed to Breu et al. [9], Hirata et al. [19] and later approaches. Moreover, the algorithm of Danielsson has all pixels of the same Voronoï cell belong to the same cluster, which alleviates the issue of isolated 'hot' pixels. Parallelization of the computation is also possible, even on embedded platforms such as FPGAs.
Both steps 3 and 4 are efficient computationally wise since they can be coded with integer arithmetics, without dynamical allocation, divisions, square roots and trigonometry).
At the end of step 4, the vEDM is obtained as a vector field $\mathbf{V}$ such as that of figure 2.

3.4 Detecting and pruning edges (step 5)

During step 5, each pixel of the image expanded at step 4 is tested to see if it belongs to an edge of the skeleton.
Let $J(\mathbf{V_{xy}}) = (\mathbf{V_{(x+1)y}} - \mathbf{V_{xy}}, \mathbf{V_{x(y+1)}} - \mathbf{V_{xy}})$ be the Jacobian of the vector field at point (x, y). We propose an edge detection criterion based on the maximum norm of the Jacobian along the x and y directions, $M = max(||\mathbf{V_{(x+1)y}} - \mathbf{V_{xy}}||, ||\mathbf{V_{x(y+1)}} - \mathbf{V_{xy}}||$. If $M < 2D$, then the walls the edge is supposed to go between are too close for the robot to pass, so the edge is discarded. Zhang et al. [30] seem to use this same criterion of minimal distance between the two obstacles. If $M \geq 2D$, the edge is kept and a local tangent vector to it is computed as $\frac{\mathbf{V_{(x+1)y}}+\mathbf{V_{xy}}}{||\mathbf{V_{(x+1)y}}+\mathbf{V_{xy}}||}$ if $||\mathbf{V_{(x+1)y}} - \mathbf{V_{xy}}|| >= ||\mathbf{V_{x(y+1)}} - \mathbf{V_{xy}}||$ or $\frac{\mathbf{V_{x(y+1)}}+\mathbf{V_{xy}}}{||\mathbf{V_{x(y+1)}}+\mathbf{V_{xy}}||}$ if $||\mathbf{V_{(x+1)y}} - \mathbf{V_{xy}}|| < ||\mathbf{V_{x(y+1)}} - \mathbf{V_{xy}}||$. The tangent vector is perpendicular to the difference of adjacent vectors, in the direction where the norm of the Jacobian is maximal. Note that the skeleton is computed with an offset of $(0.5, 0.5)$ due to the biased Jacobian.
We found a further refinement for mobile robot navigation: check that the pixel pointed to by the average of adjacent vectors in the x or y direction (subsequently called average pixel) is far enough from an obstacle, where far enough means the robot should be able to navigate up to this point without hitting an obstacle. Formally, suppose that $||\mathbf{V_{xy}} - \mathbf{V_{(x+1)y}}|| \geq ||\mathbf{V_{xy}} - \mathbf{V_{x(y+1)}}||$ (the other case is handled by permuting x and y). Then, let $(x', y') = \frac{(x+(x+1),y+y)+\mathbf{V_{xy}}+\mathbf{V_{(x+1)y}}}{2} = (x + 0.5, y) + \frac{\mathbf{V_{xy}}+\mathbf{V_{(x+1)y}}}{2}$. If $M' = ||\mathbf{V_{x'y'}}|| < D$, then it is likely that there is no actual path between the walls and that the presumed edge is a shallow dead end. The method is not exact since it considers that the edge remains straight when going towards an obstacle (the average of both vectors is tangent to the edge at the current pixel, so the approximation is first-order). However, experiments in real conditions proved it to be quite reliable even though there is no metric in literature to describe the quality of the extracted skeleton. Figure 2 shows the two edge validation criteria $M \geq 2D$ and $M' \geq D$.
It is easy to implement the Extended GVG [5] with a (vectorial) Euclidean distance map. The difference between simple and extended GVGs is that in the latter, no edge is allowed to lie more than a distance L away from the closest obstacle and there must be edges at distance L from any obstacle. With our approach, for each pixel at coordinates (x, y):

- If $||V_{xy}|| > L$, then the pixel at (x, y) can't belong to an Extended GVG edge.
- If $||V_{xy}|| \leq L$ and if one of the eight neighbors (x', y') of (x, y) defined by $(x' - x)^2 + (y' - y)^2 \in 1, 2$ verifies $||V_{x'y'}|| > L$, then the pixel at (x, y) necessarily belongs to an Extended GVG edge.
- Otherwise, the two edge validation criteria $M \geq 2D$ and $M' \geq D$ are used like for the regular GVG computation.

The Extended GVG is particularly useful in environments whose dimensions are much larger than the maximum range of the robot's distance sensors, such as outdoor environments. Within our implementation, the transition from

Extended GVG to GVG is equivalent to changing L from a finite value to positive infinity.

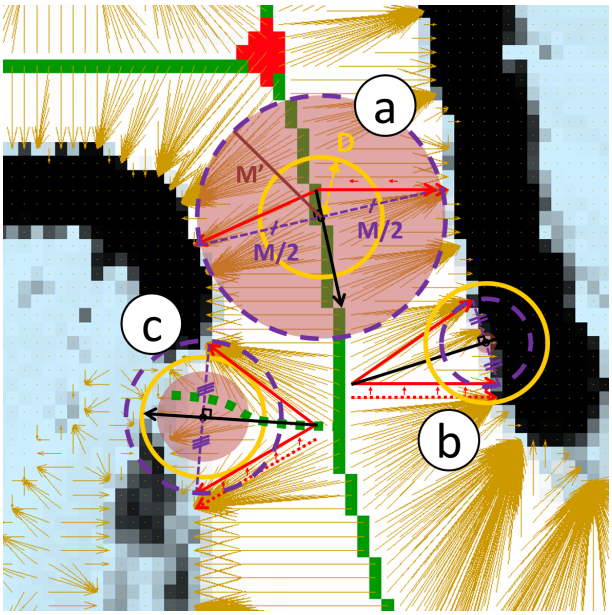

Figure 2: Removal of parasitic edges (*step 5*): in configuration (a), $M/2$ (dashed purple circle) and M' (red disc) are high enough for the edge to be kept. In configuration (b), $M/2 < D$, so the pixel does not belong to an edge. In configuration (c), $M/2 \geq D$ but $M' < D$, invalidating the edge hypothesis in dotted green. Black arrows are the local edge tangents, computed as average of both vectors and scaled arbitrarily.

3.5 Detecting vertices (steps 7 and 8)

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

For each edge pixel, a test is performed to determine if it is a vertex. This test sketched on Figure 3 consists in checking how many edges cross a closed contour around the pixel. A pixel marked as edge is a vertex if and only if the contour crosses exactly one or strictly more than two edges. As opposed to the method proposed in [23] or to simple point characterization [20], contours are not restricted to 4- or 8-neighborhoods, which is mandatory to detect a vertex in case (d') of Figure 3.

This contour-based vertex detection does only work well if edges are locally one or two pixels wide, which represents the immense majority of cases. In order to handle edges locally wider than two pixels (which may happen when multiple individual edges get too close to each other and form one single apparent edge), if a pixel has more than 7 out of 8 neighbors belonging to edges, it is automatically labeled as vertex, as well as said 7 or 8 neighbors. This refinement is almost never necessary.

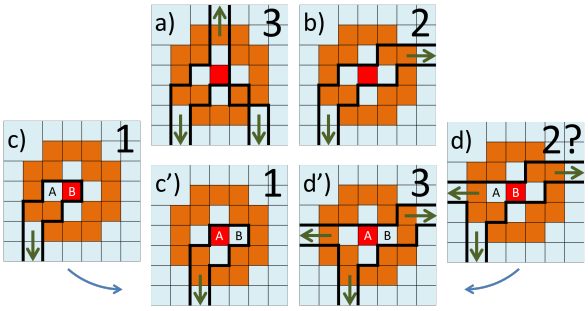

Figure 3: Vertex detection process (*step 7*): an edge pixel is a vertex if and only if one or strictly more than two edges cross the orange contour. Errors may occur on one pixel but are likely not to occur on neighboring pixels (in case (d),2 edges are detected on pixel B while on neighboring pixel A, there are actually 3 edges -case (d')-), and it is sufficient that one pixel be marked as vertex. Multiple neighboring pixels can be marked as vertex (pixels A and B for cases (c) and (c')), which calls for a clustering process.

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

1. Two different clusters must be disconnected regarding 8-way connection.
2. An edge must always link exactly two vertex clusters. This means that two edges can't "touch" each other at any place but a vertex (using 8-way connectivity).

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

Finally, each one of the original vertex pixels marks a zone of radius 1 pixel (L_1 distance) around itself as belonging to the cluster it belongs to (because anything that would be within this range would belong to the cluster). 1 is actually the integral part of $3/2$ (3 being the radius of the clustering filter, which can also be seen as the minimum inter-vertex distance). The center of the cluster is also marked as vertex, as well as the zone of 1 pixel around it, because it is conceptually impossible that the center of the cluster does not belong to the cluster.

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

3.6 Graph extraction (steps 9 and 10)

If the second rule of vertex clustering is verified, then edges always stop at a vertex cluster. Thus, each edge segment links exactly two vertex clusters. Edge segments are found using a simple clustering algorithm. Then, edges with less than one extremal vertex get pruned, which notably eliminates the rare case where an edge links a vertex to itself. Vertices with exactly two edges are possible due to vertex clustering, as shown on figure 6. Path segments made of chains of such vertices can be simplified to a single edge between both extremal vertices. Skeletons shown on all figures of this paper were all extracted without topological errors in a graph format such as that shown on Figure 6.

3.7 Implementation

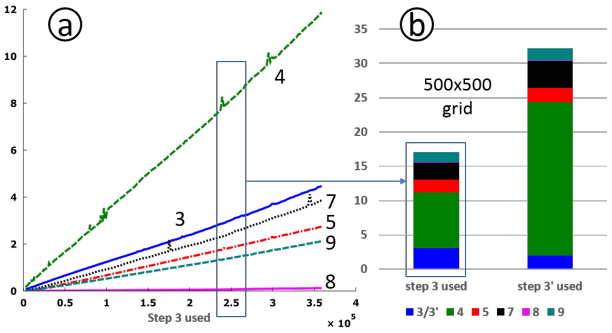

Figure 4: (a) Execution speed (ms) of the steps of our method as a function of the size of the grid (in square pixels) on a 2.7 GHz core. (b) Step 3 can be replaced by step 3' where all occupied pixels are direcly marked as Voronoï seeds, which increases the total execution time. These graphs show the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

The above series of algorithms was implemented in C using only integer arithmetics. Dynamical allocation was only used for clustering and graph extraction. The actual execution time of the whole computation is on the order of 6 milliseconds for a grid of 300×300 pixels. Without parallelization or further optimizations, extrapolation (from Figure 4) leads to typical execution times of a few seconds for a grid whose dimensions reach 10 000 pixels. This is the order of magnitude obtained for CPUs by Man et al. [24] in their approach to Euclidean distance mapping using CPUs and GPUs. However, comparing both approaches is biased since on the one hand, Man et al. recover an exact Euclidean distance map while we use an approximate vEDM but on the other hand, Man et al. don't run the edge and vertex detection (steps 5 to 8). We observed that steps 6+ also run near linear complexity (Figure 4), which was not obvious given that these steps include naively implemented clustering which may operate in quadratic complexity in the number of pixels to cluster [27]. The worst case for

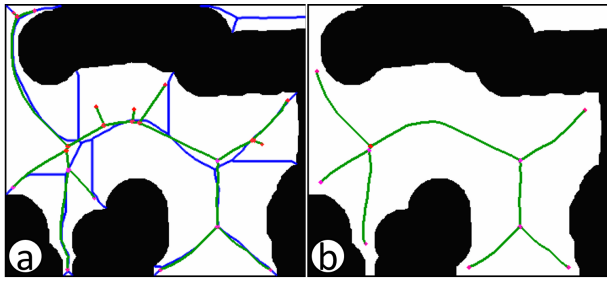

Figure 5: (a) Comparing our approach (green edges) to that of Garrido et al. [16] (blue edges) on a dataset of [16] reveals that the skeleton of Garrido et al. is not a true Voronoï diagram where all points are equidistant to two obstacles. (b): the robot size S was increased, simplifying the skeleton due to the two edge validation criteria $M \geq 2D$ and $M' \geq D$. Step 3' was used in all cases.

clustering is when all edges first form independent clusters ($r/2$ vertical edges of length up to r pixels, separated by one pixel) and then join at the bottom of the image. This is far from an usual situation where there are only a handful of edges and clusters, so that managing clusters takes negligible time compared to scanning the image.

We compared our skeleton computation approach (steps 4 to 6) to the works of Garrido et al. [16] on Figure 5 and to that of Beeson et al. [5] on Figure 6. Remarkably, there is almost no visible difference for a well chosen robot size ($D = 5$ pixels) with the works of Beeson, validating our approach. Beeson et al. don't describe how they computed the skeleton (whether a distance map was used or not and how the ridges were detected), so we can't compare the efficiency of the algorithms.

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

4 Conclusion and future work

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

- We use a vectorial Euclidean distance map to compute the skeleton. This map can be reused to achieve

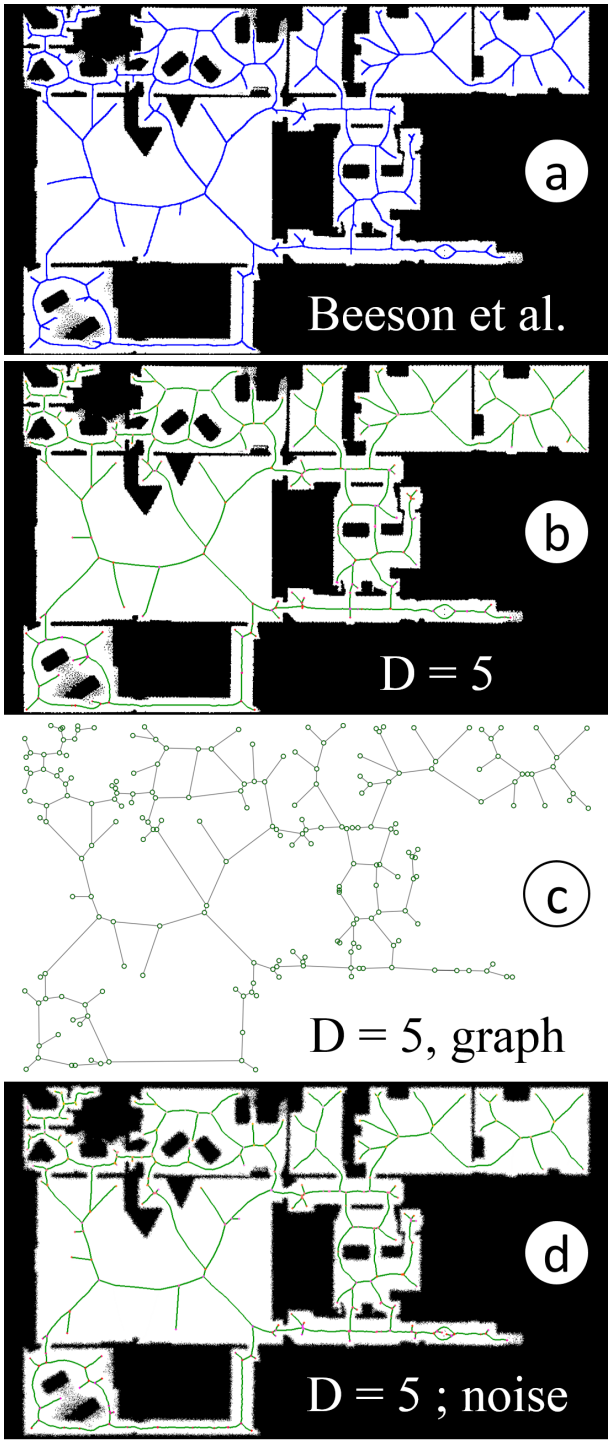

Figure 6: Our approach (b) gives similar results to that of Beeson et al. [5] (a) on a dataset of [5]. Step 3' was used in order to compute the skeleton on the whole image. Image (c) shows the graph extracted from (b), preserving two-edge vertices. For image (d), before computing the skeleton, each pixel of the original image was offset by a random number up to $10 = 2D$ pixels in the x and y directions. This simulated noise, whose amplitude is comparable to the size of the robot, causes only minor modifications of the skeleton, confirming robustness of the method.

obstacle avoidance.

- Spurious edges are removed from the skeleton using criteria parametrized only by the robot's size (the same size being used for obstacle avoidance). These criteria do not make assumptions on whether edges lead to dead ends or are exits leading to free space. The approach is thus more robust to occlusions and point-of-view differences.
- Our vertex detection and clustering algorithms allow robust extraction of the simplified skeleton in a graph format for use in topological mapping.
- Detection of the skeleton (steps 1 to 5) runs in linear complexity. Later steps (steps 6 to 10) either take negligible time or run near linear complexity on real-world data. The total execution time (around 6 ms for a 300×300 pixel grid) is compatible with robotics experiments.

We are currently testing the robustness of our approach as place extraction technique for hybrid metrical/topological SLAM within a framework such as ATLAS [8] operating in extremely large scale highly cyclic environments such as the road network of a city.

References

[1] C. Arcelli and G. Sanniti di Baja. A one-pass two-operation process to detect the skeletal pixels on the 4-distance transform. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 11(4):411–414, April 1989.

[2] D. Attali. *2D and 3D skeletons and Voronoi graphs*. Thesis, Université Joseph-Fourier - Grenoble I, 1995.

[3] T. Bailey. *Mobile Robot Localisation and Mapping in Extensive Outdoor Environments*. PhD thesis, Australian Centre for Field Robotics, University of Sydney, August 2002.

[4] T. Bailey and H. Durrant-Whyte. Simultaneous localization and mapping (SLAM): part ii. *Robotics Automation Magazine, IEEE*, 13(3):108–117, 2006.

[5] P. Beeson, N. K. Jong, and B. Kuipers. Towards autonomous topological place detection using the extended voronoi graph. In *Proceedings of the IEEE International Conference on Robotics and Automaton (ICRA-05)*, 2005.

[6] P. Beeson, J. Modayil, and B. Kuipers. Factoring the mapping problem: Mobile robot map-building in the hybrid spatial semantic hierarchy. *The International Journal of Robotics Research*, 29(4):428–459, 2010.

[7] J. Borenstein and Y. Koren. Real-time obstacle avoidance for fast mobile robots. *Systems, Man and Cybernetics, IEEE Transactions on*, 19(5):1179–1187, 1989.

[8] M. Bosse, P. Newman, J. Leonard, and S. Teller. Simultaneous localization and map building in large-scale cyclic environments using the atlas framework. *International Journal of Robotics Research*, 23(12): 1113–1139, December 2004.

[9] H. Breu, J. Gil, D. Kirkpatrick, and M. Werman. Linear time euclidean distance transform algorithms. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 17(5):529–533, May 1995.

[10] H. Choset and K. Nagatani. Topological Simultaneous Localization And Mapping (SLAM): toward exact localization without explicit localization. *Robotics and Automation, IEEE Transactions on*, 17 (2):125–137, 2001.

[11] M. Couprie, D. Coeurjolly, and R. Zrour. Discrete bisector function and euclidean skeleton in 2d and 3d. *Image and Vision Computing*, 25(10):1543 – 1556, 2007.

[12] P.-E. Danielsson. Euclidean distance mapping. *Computer Graphics and Image Processing*, 14:227–248, Feb. 1980.

[13] H. Durrant-Whyte and T. Bailey. Simultaneous localization and mapping: part i. *Robotics Automation Magazine, IEEE*, 13(2):99–110, 2006.

[14] A. Elfes. Using occupancy grids for mobile robot perception and navigation. *Computer*, 22(6):46–57, 1989.

[15] R. Fabbri, L. D. F. Costa, J. C. Torelli, and O. M. Bruno. 2d euclidean distance transform algorithms: A comparative survey. *ACM Comput. Surv.*, 40(1): 2:1–2:44, Feb. 2008.

[16] S. Garrido, L. Moreno, M. Abderrahim, and F. Martin. Path planning for mobile robot navigation using voronoi diagram and fast marching. In *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on*, pages 2376–2381, Oct 2006.

[17] G. Grevera. Distance transform algorithms and their implementation and evaluation. In *Deformable Models*, Topics in Biomedical Engineering. International Book Series, pages 33–60. Springer New York, 2007.

[18] W. H. Hesselink. A linear-time algorithm for euclidean feature transform sets. *Information Processing Letters*, 102(5):181 – 186, 2007.

[19] T. Hirata. A unified linear-time algorithm for computing distance maps. *Inf. Process. Lett.*, 58(3):129–133, May 1996.

[20] G. Klette. *Computer Analysis of Images and Patterns: 10th International Conference, CAIP 2003, Groningen, The Netherlands, August 25-27, 2003. Proceedings*, chapter Simple Points in 2D and 3D Binary Images, pages 57–64. Springer Berlin Heidelberg, Berlin, Heidelberg, 2003.

[21] B. Kuipers, J. Modayil, P. Beeson, M. MacMahon, and F. Savelli. Local metrical and global topological maps in the hybrid spatial semantic hierarchy. In *Robotics and Automation, 2004. Proceedings. ICRA '04. 2004 IEEE International Conference on*, volume 5, pages 4845–4851 Vol.5, April 2004.

[22] H. Li and A. M. Vossepoel. Generation of the euclidean skeleton from the vector distance map by a bisector decision rule. In *Computer Vision and Pattern Recognition, 1998. Proceedings. 1998 IEEE Computer Society Conference on*, pages 66–71, Jun 1998.

[23] G. Malandain and S. Fernndez-Vidal. Euclidean skeletons. *Image and Vision Computing*, 16(5):317 – 327, 1998.

[24] D. Man, K. Uda, H. Ueyama, Y. Ito, and K. Nakano. Implementations of parallel computation of euclidean distance map in multicore processors and gpus. In *Networking and Computing (ICNC), 2010 First International Conference on*, pages 120–127, Nov 2010.

[25] M. Nitsche, P. de Cristoforis, M. Kulich, and K. Kosnar. Hybrid mapping for autonomous mobile robot exploration. In *Intelligent Data Acquisition and Advanced Computing Systems (IDAACS), 2011 IEEE 6th International Conference on*, volume 1, pages 299–304, Sept 2011.

[26] C. Pudney. Distance-ordered homotopic thinning: A skeletonization algorithm for 3d digital images. *Computer Vision and Image Understanding*, 72(3):404 – 413, 1998.

[27] R. Sibson. Slink: an optimally efficient algorithm for the single-link cluster method. *The Computer Journal (British Computer Society)*, 16(1):30–34, 1973.

[28] S. Thrun. Learning metric-topological maps for indoor mobile robot navigation. *Artificial Intelligence*, 99(1):21 – 71, 1998.

[29] S. Thrun. Learning occupancy grids with forward models. In *Intelligent Robots and Systems, 2001. Proceedings. 2001 IEEE/RSJ International Conference on*, volume 3, pages 1676–1681 vol.3, 2001.

[30] Q. Zhang, D. Whitney, F. Shkurti, and I. Rekleitis. Ear-based exploration on hybrid metric/topological maps. In *Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on*, pages 3081–3088, Sept 2014.