Exact Cell Decomposition of Arrangements used for Path Planning in Robotics

Nora H. Sleumer
Institute of Theoretical Computer Science
Swiss Federal Institute of Technology
Zurich, Switzerland
sleumer@inf.ethz.ch

Nadine Tschichold-Gürman
Institute of Robotics
Swiss Federal Institute of Technology
Zurich, Switzerland
nadine@ifr.mavt.ethz.ch

September 29, 1999

Abstract

We present a practical algorithm for the automatic generation of a *map* that describes the operation environment of an indoor mobile service robot. The input is a CAD description of a building consisting of line segments that represent the walls. The algorithm is based on the exact cell decomposition obtained when these segments are extended to infinite lines, resulting in a line arrangement. The cells are represented by nodes in a connectivity graph. The map consists of the connectivity graph and additional environmental information that is calculated for each cell. The method takes into account both the path planning and position verification requirements of the robot and has been implemented.

1 Introduction

In the field of robotics, one of the main problems is robot motion planning. We consider the case of a car-like robot doing routine service operations, such as mail delivery, in an office building. The emphasis is on providing the robot and its programmer with a practical and robust basis for a path planning system that is calculated once and used regularly. For a one-body mobile robot moving in the plane, where the environment is composed of polygons, an *exact cell decomposition* that results in a *connectivity graph* is a well-known planning approach [Lat91], [HA92]. The free space of the robot is exactly partitioned into cells that are stored in a graph. The on-board global planner computes a path for the robot from this graph.

From a theoretical point of view, an exact cell decomposition with few cells can be computed the fastest. An environment described by n line segments can be sliced into $O(n)$ vertical strips in time $O(n \log n)$. From a mobile robot's point of view, however, minimizing the number of cells is not as important as having some leeway in the path to take, being able to verify its position frequently, and having an intuitive interface for its user.

We present a practical exact cell decomposition method, based on *line arrangements*, that fulfills these requirements for office-like buildings with straight walls. The algorithm is based on decomposing the part of the environment the robot can access into $O(n^2)$ cells that result when the n line segments defining the space (i.e. the walls) are extended to straight infinite lines. A connectivity graph based on the cell incidences is then generated. A *map* containing the graph and environmental information is used to compute the path, the navigation behavior, and the position verification data, as well as being an interface to the robot.

2 Problem Statement and Definitions

Let $\mathcal{A}$ be a single rigid robot moving in the two-dimensional Euclidean space $\mathbb{R}^2$whose environment consists of a set of non-intersecting polygons inside a bounding polygon. The input for the problem is a CAD representation of the building where the line segments represent those walls of the building that the robot can identify with its sensors. Given a point inside the bounding polygon, we define the *workspace* to be that part of the plane accessible from this point without crossing the boundary of a polygon. See Figure 1.

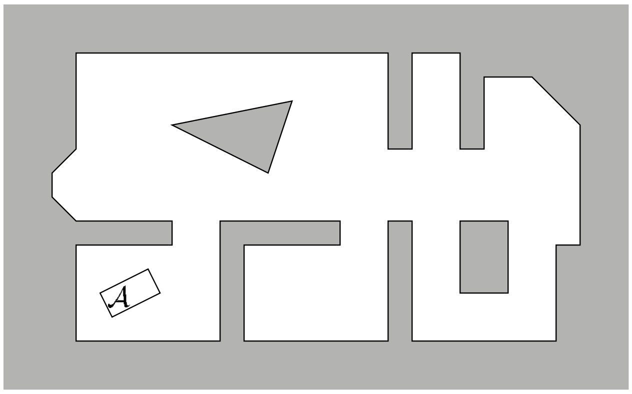

Figure 1: A CAD representation of a building defined by 39 line segments representing the walls. The robot $\mathcal{A}$ moves in the white workspace outlined by grey obstacles.

The robot $\mathcal{A}$ is represented as a two-dimensional polygon in the plane. A *configuration* q of $\mathcal{A}$ is a specification of the position and orientation of $\mathcal{A}$, denoted by $(x, y, \theta) \in \mathbb{R}^2 \times [0, 2\pi)$. The *configuration space* is the space of all the configurations of the robot. The *free space* is that subset of the configuration space where the robot does not intersect any of the polygons (or their interiors) and a *free configuration* is a point (x, y, θ) in this space. (If the robot consists of a point then its orientation is irrelevant.)

The problem is to create a data structure, called a *map*, which contains the connectivity graph and information about the workspace, so that, given the initial configuration q_{init} and a goal configuration q_{goal} of the robot, a route can be generated that the robot can follow. This *route* consists of a sequence of free configurations $q_{init}, q_1, \ldots, q_k, q_{goal}$, which we will call *checkpoints*. The actual *path* of the robot is an interpolation of these checkpoints and it is calculated by the robot's planning software. A *free path* is one which is entirely within the free space. In order to verify its position, the robot needs a map so it can compare the walls it can sense with a list of walls the it should be able to see from that position. See Figure 2.

We first present an exact cell decomposition method and algorithm that is suitable for planning the path of a free-flying point-sized robot moving in an exactly-defined environment consisting of a bounding outer polygon, possibly containing polygonal obstacles, such as that in Figure 1. The free space of the robot is partitioned into cells, which are stored in a graph, with

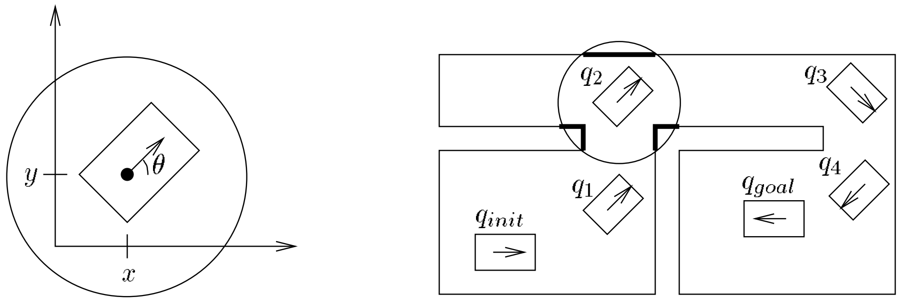

Figure 2: A rectangular robot with configuration (x, y, θ) and the circle representing its sensor range on the left. A route consisting of the configurations $q_{init}, q_1, q_2, q_3, q_4$ and q_{goal} on the right. The bold edges are the walls that the robot can sense when at checkpoint q_2.

adjacent nodes corresponding to adjacent cells. This graph is used by the robot's global path planner to get to its destination.

We based our assumptions of a robot's needs and capabilities on the mobile post system robot (MOPS), constructed at the Institute of Robotics at the ETH [VTG96]. It currently uses a graph calculated and optimized by hand. MOPS is a non-holonomic robot with two drive wheels that are located on the front end of the vehicle and a support castor at the other end. It can move forwards and backwards and is equipped with two scanned laser range-finders and 12 ultrasonic range finders. A global planner applies a graph search algorithm to calculate the route and generates a list of intermediate path segments to be traveled by the robot. It uses a position controller using classical control techniques to stay on the path and a situation based behavior selector to determine the behavior of the robot [KPVA95].

In the second part of the paper we look at the following issues that are important for service robots:

1. **Map Interface:** The programmer of the robot should be able to identify initial and goal positions easily on a routine basis. It should be possible to adapt the map if certain things in the building change. For example, a hallway could be under construction. Thus the map should be an intuitive and adaptable representation of the environment.
2. **Behavior Planning:** The map must contain environment information that enables the robot to figure out how to behave. If it is in a hallway it can move faster than in a room, for example.
3. **Path Planning:** There are many models of robots that can be used and additional constraints that must be taken into consideration when planning a path for the robot.

   (a) **Point Robot:** The free space of a point-sized robot is essentially the same as the workspace. It can move anywhere within the workspace and have any orientation.

   (b) **Non-point Robot:** The robot may be round or polygonal and not point-sized.

   (c) **Free-flying Robot:** A free-flying robot can move in any direction at any time, such as along a path defined by line segments.

   (d) **Non-holonomic Constraints:** A robot may be subject to *non-holonomic constraints*, meaning that, like a car, it cannot rotate and translate freely in the plane. A practical path must therefore be smooth, where the degree of smoothness depends on the turning radius of the robot.

(e) **Drift and Slippage:** Because of drift and slippage, no robot can follow the path exactly, when going around a corner, for example. Thus the path must be tolerant to small deviations.

(f) **Obstacle Avoidance:** There may be smaller obstacles that the robot can identify that are not part of the fixed environment, such as chairs. The robot should then be able to decide how to maneuver around such objects and get back on track.

4. **Position Verification:** Robots are inherently inexact. Unless a global positioning system is used, which is difficult indoors, the robot's position has to be deduced by dead reckoning, an inaccurate method on the long term. We assume the robot has sensors, such as laser range finders (LIDARs), and an algorithm that can match the information from the LIDARs, or other sensors, with the geometric data in the environment model and update its position. In order to be able to do this the map must contain information about which walls the robot can see from certain positions.
5. **Practical to Implement:** The algorithm must be simple, robust and fast enough.

We first consider other methods and how they deal with robot models and constraints. Then we describe our method and analyze it according to the above criteria.

3 Typical Methods for Path Planning

An exact cell decomposition is not the only approach for creating a connectivity graph. The *roadmap* and the *approximate cell decomposition* methods also produce graphs. The classic exact cell decomposition technique is the *vertical* or *trapezoidal decomposition*. A thorough description of robot motion planning methods can be found in Latombe's book [Lat91] and an extensive survey on motion planning has been written by Hwang and Ahuja [HA92]. We give a short overview of the three methods here.

3.1 Roadmap Methods

The roadmap approach to path generation consists of reducing the environmental information to a network of one-dimensional curves, called the *roadmap*. Once the roadmap has been constructed, a path can be calculated be connecting the initial and final configurations to the network and finding a path in the roadmap. Examples of roadmap methods are the visibility graph, Voronoi diagram, freeway net and silhouette graphs. In the *visibility graph*, the vertices of the polygons that can see each other are connected, as shown in Figure 3. The nodes of the graph represent the vertices of the polygon and an edge connects two nodes in the graph if their corresponding vertices are connected.

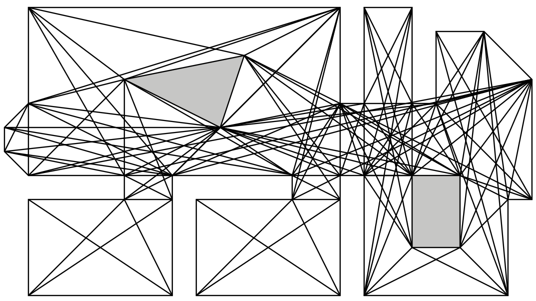

Figure 3: The visibility graph.

In general roadmap methods are fast and most of them are easy to implement, but they do not provide an intrinsic way of describing the environmental information.

3.2 Cell Decomposition Methods in General

A cell decomposition can be used for path planning in the following way:

In the preprocessing phase:

1. The free space of the polygonal two-dimensional configuration space is determined.
2. The free space is partitioned into a collection of cells.
3. A connectivity graph is constructed by connecting the cells that share a common boundary. (A hole in the bounding polygon corresponds to a cycle in the connectivity graph.)

In the on-line phase:

1. A sequence of cells, a *channel*, that the robot must traverse in order to go from the initial position to the goal position is obtained from the connectivity graph.
2. A free path is constructed from the channel.

If the robot is not a point and can turn in any direction then computing the free space is a major part of the calculation. Most methods assume a point-sized robot or a convex polygonal robot with a fixed orientation and increase the thickness of the wall by the width of the robot. The resulting free space as input. See Figure 4. The triangular robot has a fixed orientation and a reference point p. Given the workspace defined by the interior of the bold polygon, the free space of the robot, with respect to point p, is the white area within the bold line. However, it now becomes more difficult to describe the actual environment around a cell and most service robots do not have a fixed orientation.

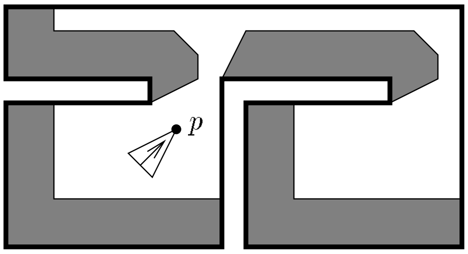

Figure 4: Robot with fixed orientation.

3.3 Approximate Cell Decomposition

In this approach the environment is divided into a fine grid where each cell in the grid contains a flag identifying it as free space or not. The resolution of the grid must be very high in order to capture every important detail, resulting in a graph with very many nodes. Thus path planning is not very efficient and the decomposition does not give rise to a natural way of describing the environment, but it is easy to implement. See Figure 5.

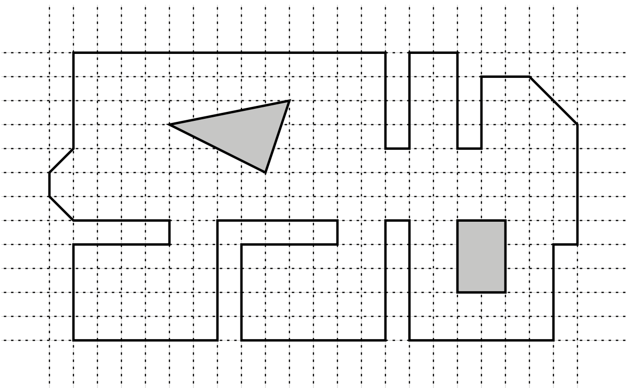

Figure 5: An approximate cell decomposition.

A quad-tree decomposition reduces the complexity of the graph. Recent advances have been also made by Thrun [Thr98] who has suggested integrating approximate and exact cell decomposition for constructing maps. The method starts with a grid-based map and builds a smaller topological map on top of it.

3.4 Exact Cell Decomposition

Exact cell decomposition methods decompose the free space into cells whose union is exactly the free space; a *partition* of the space is made. As stated in [Lat91], the cells generated by the decomposition should have the following two characteristics:

1. The geometry of each cell should be "simple" enough to make it easy to compute a path between any two robot configurations in the cell.
2. It should not be difficult to test the adjacency of any two cells and to find a path crossing the portion of boundary shared by two adjacent cells.

Much work has been done on decomposing polygons into convex regions in order to produce connectivity graphs that are useful for robot motion planning. First minimal convex decompositions of a polygonal region were investigated (see Figure 6). It was proved that the number of

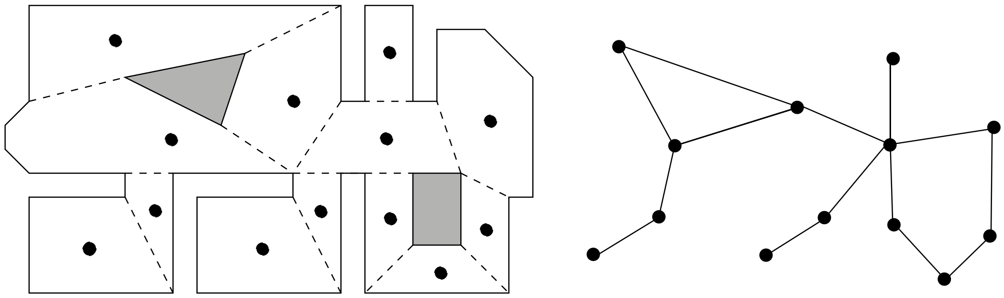

Figure 6: An exact cell decomposition (left) and its corresponding graph (right).

cells of an minimal decomposition is linear in the size of the input, $O(n)$, and that it can even be computed in time polynomial in the number of vertices n. However, Lingas [Lin82] has shown that holes in the configuration space make the problem NP-hard. So this method is neither practical nor fast. Therefore non-minimal decompositions were looked at.

The *vertical* or *trapezoidal decomposition* of Chazelle [Cha87] is the most well-known method. The number of cells is linear in the number of edges n and the decomposition can be obtained

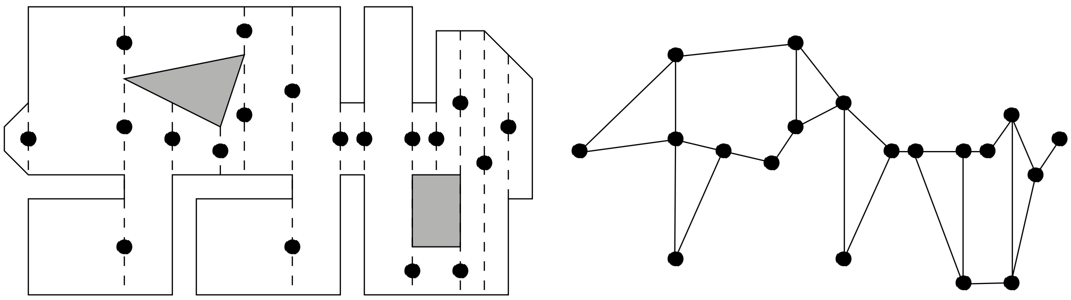

Figure 7: Vertical cell decomposition (left) and corresponding graph G' (right).

in time $O(n \log n)$. Representing the cells as nodes, a connectivity graph G can be derived. A more practical graph G' can be obtained by taking the midpoints of the segments separating adjacent cells as nodes (see Figure 7). The $O(n)$ edges in the graph are between nodes belonging to the same cell and are assigned a weight proportional to the length of the edge. A search for the shortest path in this weighted graph takes time $O(n \log n)$.

However, this method has some drawbacks. The cells are not very natural-looking. They tend to be long and skinny, which is not good for position verification. There are also not very many of them, which is good for theoretical optimality, but not in practice. The robot needs to be able to verify and adjust its position often and having a long hallway represented by a single node is not very useful.

None of the methods can solve the problem in its full generality. Some work has been done on finding a continuous motion from initial to goal position or proving that no such path exists. Schwartz and Sharir [SS83] present exact cell methods for decomposing the free space of a robot modeled as a polygon. The free space is three-dimensional, as it must take the orientation of the robot into consideration. This method is slow and difficult to implement. Avnaim, Boissonnat and Faverjon [ABF88] developed a practical method where only the boundary of the free space is decomposed. It is applicable to a robot that can translate and rotate freely in the plane. The disadvantage of these methods is that they do not present an intuitive user interface.

4 The Arrangement Method

When given a floor plan of a building and asked to divide the area into regions, an intuitive way to do so would be by extending the sides of the objects until they touch other objects. We can produce such a decomposition algorithmically by extending the line segments to lines, as depicted in Figure 8, and computing the cells of the resulting line arrangement.

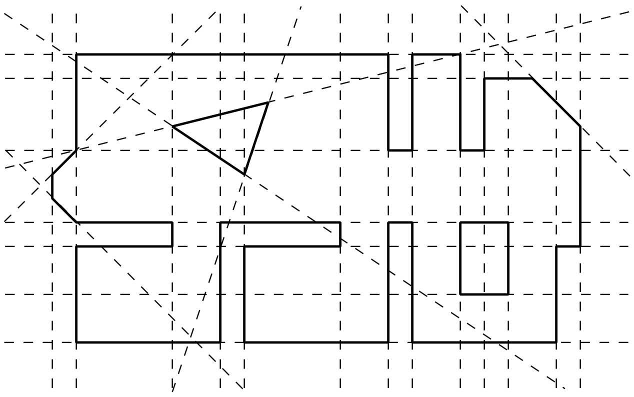

Figure 8: The 39 line segments (walls) result in an arrangement of 26 lines that decompose the free space into 76 cells.

An arrangement of lines partitions the plane into cells, edges and vertices. Arrangements have been studied extensively in computational geometry and one well-known fact is that, given a non-degenerate arrangement of n lines in two dimensions, the number of cells is $(n^2+n+2)/2 \in O(n^2)$ and the number of edges is n^2. The fact that the cells of an arrangement are convex, and that the union of a pair of adjacent cells is convex, is a useful property, as we shall see in the following section.

An incremental algorithm for computing all elements (cells, edges and vertices) of an arrangement has been described by Edelsbrunner in [Ede87]. It starts with two intersecting lines and adds one line after the other, updating the data structure with a cost of $O(n)$ per line. In total, this algorithm runs in time $O(n^2)$. If three lines cross at the same point, then symbolic perturbation must be used to create a cell with zero area, which must be eliminated in the connectivity graph produced for the robot. Ultimately we only want the cells with non-zero area that lie within the bounding polygon. Vertical lines must also be perturbed by rotating the whole plan. All the cells are enumerated, irrespective of the boundary of the free space.

We propose a simpler algorithm that only enumerates the cells inside the building and can handle degeneracies easily without creating extra cells. Our algorithm runs in time $O(n^3)$ and is well-suited for determining the arrangement decomposition within a polygonal area.

We first describe the algorithm for creating a map from which a path for a free-flying point-sized robot can easily be deduced. We cannot solve the general case for a non-point non-holonomic robot but we explain how environmental information can be introduced into the map to help the robot deal with the position localization and other path planning issues.

4.1 The Algorithm

The algorithm can be broken down into four steps:

1. **Extend the line segments representing the walls to lines:** The results of this step can be seen in Figure 8. Given n line segments, $O(n)$ lines will be created. If two line segments lie on the same line then only one copy is kept along with the information of what part of the line represents a wall.
2. **Compute the arrangement and store in an appropriate data structure:** This is a recursive procedure starting with any cell inside the building:
   (a) Walk around the cell boundary, keeping track of its geometry (the edges and vertices) and its neighbors. Each edge is either a wall or the border with a neighboring cell. See Figure 9.
   (b) Store the cells in an appropriate data structure, such as a balanced binary tree, with additional pointers to the neighboring cells. The sign vector, as described in [Sle99], can be interpreted as a binary number and used a the label for the cell.
   (c) Repeat (a) and (b) for the neighboring cells that have not been processed yet.
3. **Merge the cells that are split unnecessarily:** One ugly event that occurs in the arrangement decomposition is that cells may be unnecessarily split because of a line segment (wall) far away. Such a situation occurs with cells X and Y in Figure 9. The top edge of cell X results from the line induced by wall w_1, which is on the other side of wall w_2. Thus cells X and Y can be merged. For the non-wall boundary edges of each cell, check whether the corresponding wall segment is on the other side of another wall segment. If it is, then merge the cell with the corresponding neighbor. If the edge between the neighboring cells is a wall segment then no merging can be done. Even after the cell merging step, the union of two adjacent cells in the arrangement decomposition is convex.
4. **Calculate the environmental information for each cell:** We define the position of the checkpoint for each cell to be the center of gravity of the cell, which is guaranteed to be a point inside the cell. The edges of a cell usually correspond to a change in the

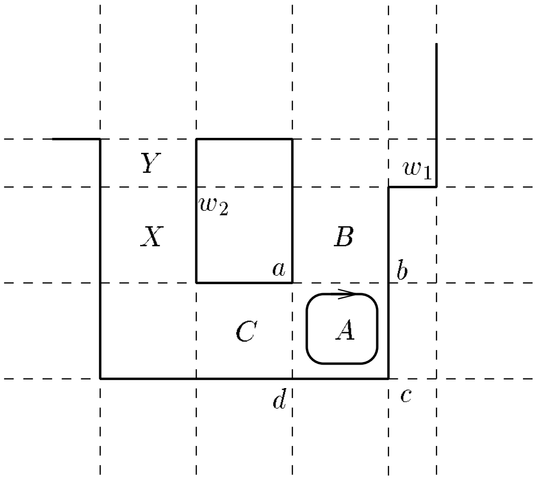

Figure 9: Cell A shares edge ab with cell B, edges bc and cd are a walls, and edge da is shared with cell C. Cells X and Y are adjacent and can be merged.

immediate environment, such as the corner of a wall. In order to store such information the edges of the cell and its neighbors are examined as to whether they are walls, and then those of the neighboring cells are examined and stored in the node. This generally suffices for position localization in buildings.

This algorithm is a two-dimensional variation of the one in [Sle99]. The result for the example building can be seen in Figure 10.

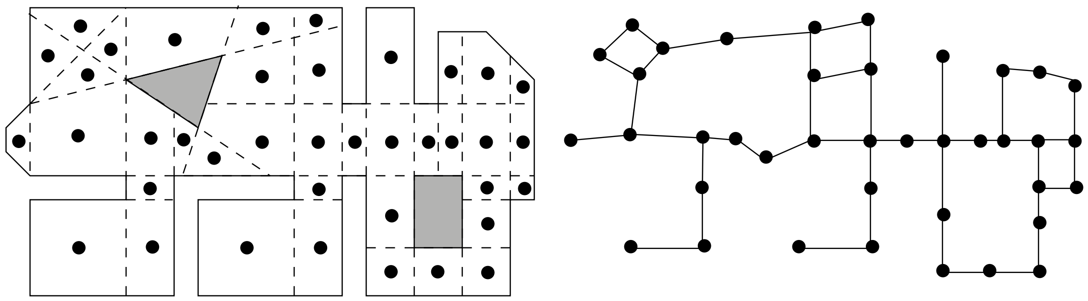

Figure 10: Arrangement decomposition with corresponding graph after cell merging. The number of cells inside the building has been reduced from 76 to 39.

4.2 Analysis of the Algorithm

Starting with n line segments, $O(n)$ lines are generated. The algorithm enumerates the $O(n^2)$ cells in the building in time $O(n^3)$, including the cell merging step, and constructs the connectivity graph along the way. We calculate the time required for each step and then determine the overall time complexity.

1. Extending the line segments to lines takes time $O(n)$.
2. The arrangement can be computed in time $O(n^3)$:
   (a) The time required for step (a) is the time needed to intersect each edge with all the $O(n)$ lines in the arrangement. This step is carried out exactly twice for each of the $O(n^2)$ edges in the arrangement so the total time complexity is $O(n^3)$. Degeneracies of three or more lines going through one point can easily be handled by local symbolic perturbation that does not increase the number of cells. Vertical or parallel lines do not require special treatment.
   (b) For maximum efficiency the cells should be stored in a balanced binary tree. If the $O(n^2)$ cells are stored in a balanced binary tree then each lookup will take time $O(\log n)$.
   (c) Each neighbor has to be looked up to see if it has already been calculated and pointers to these neighbors should be set. Each cell is looked up as often as it has neighbors so the total number of cell lookups equals the number of edges, $O(n^2)$. Thus the time needed is $O(n^2 \log n)$.
3. Merging the cells takes time $O(n^3)$. For each edge of each cell, intersect the line it is on with the $O(n)$ other lines and compare the intersection points with the wall segments on the line. Each of the $O(n^2)$ edges is analyzed once, resulting in time $O(n^3)$.
4. Calculating the additional environmental information takes time $O(n^2)$. On the average, a cell has about four edges since the number of cells is $(n^2+n+2)/2$, the number of edges is n^2 and each edge is incident to two cells. For bounded cells, the number of vertices is equal to the number of edges. Thus the total time needed to calculate the center of gravity of all the cells is $O(n^2)$ and the total time needed to inspect the cell edges and those of its neighbors is also $O(n^2)$.

The total time required is $O(n^3)$, where n is the number of line segments (walls). In practice, the number of cells within the bounding polygon seems to be linear in the number of edges, but pathological cases where this is not so can easily be constructed. Most buildings do not have large interior spaces where the cells can become arbitrarily complex.

The preprocessing step may take long, although only a few seconds for our real-world examples were needed, but the resulting connectivity graph is relatively small. Our algorithm is simple, is invariant of orientation of the lines, can handle degeneracies easily, produces the connectivity graph and environmental information directly, and runs in time polynomial in the number of line segments.

4.3 Map Interface

The map containing the connectivity graph and the environmental information provides a good interface for the user of the robot. The decomposition is intuitive, creating cells that are nicely shaped. Dynamic environment changes, such as doors that open and close, or hallways that are temporarily inaccessible, could also be modeled by setting a flag for each node declaring it to be open or not.

4.4 Behavior Planning

In the initial description of the environment, the line segments can be identified as being part of a hallway or office. The checkpoints can contain this information and the robot can adapt its behavior accordingly.

4.5 Path Planning

Given two points in the free space, a route can easily be obtained from the connectivity graph by taking the path in the graph with the fewest number of nodes. The distance can also be taken into consideration by calculating the distance between the checkpoints. The path that is calculated from this route depends on the model of robot that is used and the additional constraints that are taken into consideration.

1. **Free-flying Point Robot:** Since the union of two adjacent cells is convex, the point-sized robot can go from each point in one cell to each point in the neighboring cell without hitting a wall. So we could take any point in the cell to be a checkpoint. But since the robot is inherently inaccurate in following the path, the safest would be to go from center to center, where the center is the center of gravity of the cell.
2. **Drift and Slippage:** The fact that two adjacent cells together are convex, means that if the robot gets off course, then it can still aim directly for the next checkpoint without hitting a wall.
3. **Non-holonomic Constraints:** In order to produce smooth paths that go around corners, the robot can use a spline to calculate the path between the checkpoints. See Figure 11. For a specific robot, experiments would have to be made to determine how the

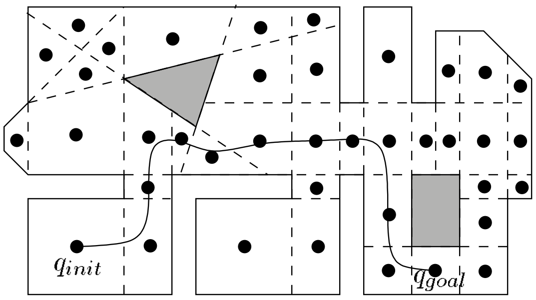

Figure 11: A path defined as an interpolated spline of the checkpoints from q_{init} to q_{goal}.

   spline should be computed. Weights and orientations could be attached to the checkpoints to influence the shape of the spline.
4. **Non-point Robots:** Since the route goes from cell center to cell center, it maximizes the clearance to the edges of the cell. The wall information in the map can also be used to adjust the route. This could become necessary when the robot is larger than the smallest cell. We cannot solve the problem in its full generality but the checkpoints of the map can be adapted for a specific robot.
5. **Course Adjustment:** If the robot encounters an undefined obstacle, such as a chair, the robot has enough information about the walls around it to calculate a new path.

4.6 Position Verification

Each checkpoint has a list of the nearest walls the robot can sense from that position. Comparing this geometric information with the information obtained by the robot's sensors allows it to verify its position. The robot must know its initial position and updates its position at each checkpoint. Since the lines are extensions of walls, most cells are adjacent to a critical point, such as a corner, on the robot's path. Corners are easy to detect and compare. Usually a wall corner is part of a cell or its neighbor. That is why it is generally enough to store the information about the walls of a cell and its neighbors.

5 Experimental Results

The algorithm has been implemented on a Sun workstation (Solaris on a Sparc Ultra 1) and on a PC (Linux on a Pentium 200) in the C programming language [Hee98]. It has been applied on the two-dimensional CAD plan of the complete ground floor of the CLA building at the ETH Zurich and a more realistic version containing only the hallways. See Table 1.

	# walls	# lines	# cells before merging	# cells after merging	time (s)
CLA complete	162	96	809	191	1.89
CLA hallways	64	41	126	50	0.12

Table 1: The number of cells for two floorplans.

The complete description contained 162 walls. These were extended to 96 lines which generated 809 cells. After the cell merging operation (step 3 of the algorithm), this number was reduced to 191, thus about 80% of the cells were eliminated. The whole algorithm for the generation of the connectivity graph and the environmental information took less than two seconds.

When the algorithm is used on a reduced plan, the resulting graph gets much smaller and simpler. In the specified case of the CLA building, the number of the walls was 64, which produced 50 cells, after pruning approximately 60 % of the cells during the merging step. The resulting decomposition can be seen in Figure 12.

6 Conclusions and Further Work

We have presented a new method for generating a map consisting of a connectivity graph and information about the walls of a building that represent the environment of a mobile robot. The input to the algorithm is the two-dimensional CAD model of a building. The algorithm is based on the exact cell decomposition obtained when the segments representing the walls of the building are extended to infinite lines. The connectivity graph and environmental information in the map can be used by the path planning, behavior scheduler and position verification modules of the robot, as well as being a flexible user interface for the robot's programmer. We believe that this is a very promising method for putting mobile robots in operation in new environments. The main advantages of this method are that the decomposition of the work space of the robot is intuitive and that the shape of the cells allows for robust and flexible path planning.

We would like to test this method using an actual robot to see which strategies should be used to derive a path from the list of checkpoints. Additional constraints can also be looked at.

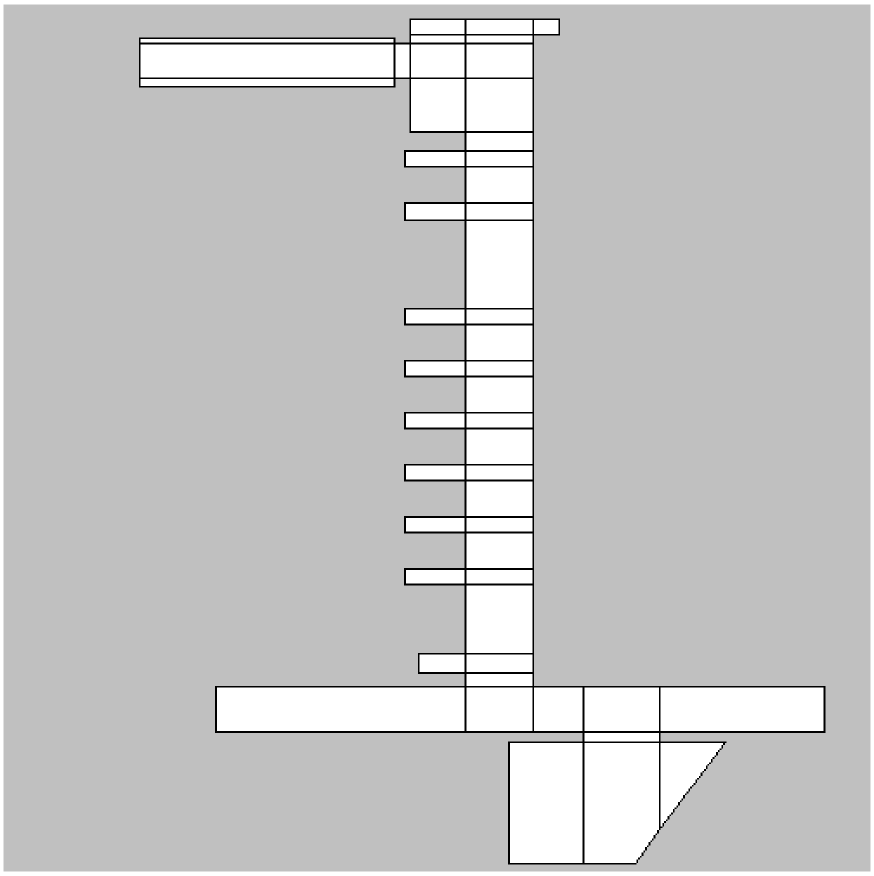

Figure 12: The arrangement decomposition of the hallways of the CLA building.

For example, in a building with large spaces it may be necessary for the robot to stay within sensor range of the walls. Such constraints could also be integrated into our method.

7 Acknowledgments

We would like to thank Frank Heer for implementing the method, and Jurg Nievergelt, Jürgen Richter-Gebert, Ambros Marzetta and Lutz Kettner for their constructive criticism and encouragement.

References

[ABF88] F. Avnaim, J.-D. Boissonnat, and B. Faverjon. A practical exact motion planning algorithm for polygonal objects amidst polygonal obstacles. In *Proc. IEEE Symp. on Robotics and Automation*, pages 1656–1661, 1988.

[Cha87] B. Chazelle. Approximation and decomposition of shapes. In J. T. Schwartz and C.-K. Yap, editors, *Advances in Robotics 1: Algorithmic and Geometric Aspects of Robotics*, pages 145–185. Lawrence Erlbaum Associates, Hillsdale, NJ, 1987.

[Ede87] H. Edelsbrunner. *Algorithms in Combinatorial Geometry*, volume 10 of *EATCS monographs on theoretical computer science*. Springer-Verlag, Berlin, 1987.

[HA92] J. K. Hwang and N. Ahuja. Gross motion planning – a survey. *ACM Computing Surveys*, 24(3):219–291, 1992.

[Hee98] F. Heer. Exact cell decomposition of arrangements used for path planning in robotics. Semesterarbeit, Institute of Theoretical Computer Science and Institute of Robotics, ETH Zurich, Switzerland, SS 1998.

[KPVA95] O. Kaiser, R. Pfiffner, S. Vestli, and A. Astolf. Positionsregelung für nichtholonome mobile Roboter. In *11. Fachgespräche über autonome mobile Systeme, Karlsruhe, Germany*, volume 11 of *Informatik Aktuell*, 1995. In German.

[Lat91] J.-C. Latombe. *Robot Motion Planning*. Kluwer Academic Publishers, Boston, 1991.

[Lin82] A. Lingas. The power of non-rectilinear holes. In *Proceedings of the 9th International Colloquium on Automata, Languages and Programming*, volume 140 of *Lecture Notes in Computer Science*, pages 369–383, 1982.

[Sle99] N. Sleumer. Output-sensitive cell enumeration in hyperplane arrangements. *Nordic Journal of Computation*, 6:137–147, 1999.

[SS83] J. T. Schwartz and M. Sharir. On the "piano movers" problem I: The case of a two-dimensional rigid polygonal body moving amidst poly-gonal barriers. *Communications on Pure and Applied Mathematics*, 36:345–398, 1983.

[Thr98] S. Thrun. Learning metric-topological maps for indoor mobile robot navigation. *Artificial Intelligence Journal*, 99(1):21–71, 1998.

[VTG96] S. J. Vestli and N. Tschichold-Gürman. MOPS, a system for mail distribution in office-type buildings. *Service Robot: An International Journal*, 2(2):29–35, 1996.