KINODYNAMIC ROBOT MOTION PLANNING BASED ON THE GENERALISED VORONOI DIAGRAM

Petr Švec*

Kinodynamic motion planning of an autonomous robot in an unknown or partially known indoor or outdoor environment is a challenging task, especially when the generated path must maintain the largest distance from surrounding obstacles and the robot's kinodynamic properties, its localisation, and uncertainty of the environment are also considered. A new motion planning technique, which is built on the generalised Voronoi diagram, for a robot with kinematic or dynamic constraints is proposed. The generalised Voronoi diagram serves this task effectively as it maintains the largest (the safest) possible distance from surrounding obstacles. Moreover, a novel approximation geometric algorithm, which embodies a trade-off between the efficiency of computation, implementation difficulty, and robustness, for computing this diagram is presented.

Key words : *kinodynamic motion planning, generalised Voronoi diagram, Fortune's plane sweep algorithm, global motion planning, real-time motion planning*

1. Introduction

Kinodynamic motion planning of an *autonomous robot* in an unknown or partially known indoor or outdoor environment is a challenging task, especially when the robot must maintain the safest distance from surrounding obstacles and its kinodynamic properties, localisation, and uncertainty of the environment are also considered (see Figure 1). The solution of the kinodynamic motion planning task involves the *global* and *real-time motion planning* between two given positions. See [12] for an overview of the basic motion planning methods.

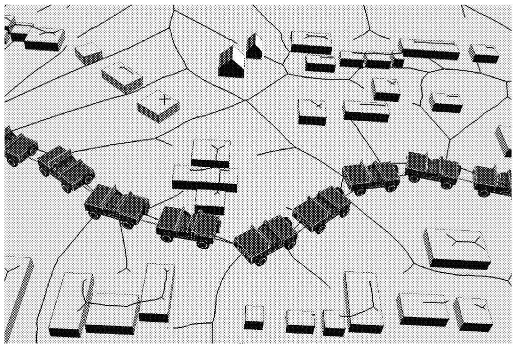

Fig. 1: Kinodynamic motion planning of a car-like robot based on the generalised Voronoi diagram

* P. Švec, Institute of Automation and Computer Science, Faculty of Mechanical Engineering, Brno University of Technology, Technická 2, 616 69 Brno, Czech Republic

The technique presented solves this task efficiently by means of the computational geometry [3]. The computational geometry emerged from the field of algorithms design and analysis in the late 1970s. Its success of the problems studied, practical and efficient solutions from the asymptotic time complexity point of view, and its huge range of application domains laid grounds for its future expansion into robotics.

One of the most useful structure in the computational geometry for robotics is the *Voronoi diagram* [3, 9] (see Figure 2). It has a lot of applications in various fields since it maintains the largest distance from surrounding generators. This property can be employed as a base for several motion planning tasks, where these generators are formed by obstacles. In addition to this, the *retraction property* of the Voronoi diagram ensures that the robot is always capable of transferring itself onto this diagram along a collisionless sight line, from which follows, that the Voronoi diagram entirely captures the continuity of the whole space as a topological graph [9]. The extension of this diagram for point, segment, or polygonal generators is called the *generalised Voronoi diagram* [9].

The usability of this diagram for the kinodynamic motion planning task is conditioned by an existence of efficient, robust, and practical algorithms for its computation. A novel approximation algorithm is presented together with its asymptotic time complexity. The added value of this algorithm lies in robustness and simplicity of its implementation and high computational efficiency in comparison to algorithms of the same class. This algorithm is further utilised in global or real-time motion planning methods, which exploit main properties of the Voronoi diagram. Furthermore, the generated path does not have to strictly maintain the largest distance from surrounding obstacles thus a path relaxation technique can be applied.

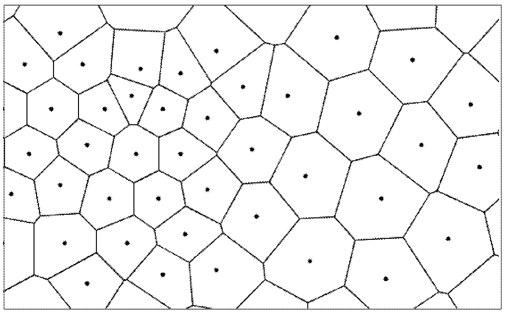

Fig.2: Voronoi diagram for point generators

2. Voronoi Diagram

The Voronoi diagram [9] (further VD, see Figure 2) represents a way of dividing a m-dimensional continuous space into a set of regions, where all locations in this space are associated with the closest isolated points (so-called generators).

A major reason for persisting success of Voronoi diagrams is that it can be generalised in a diversity of ways including 3D and higher dimensional varieties. There are a lot of variants (different types of generators and distance measures) of this diagram, which are covered in [6, 9].

Given a finite set $P = \{p_1, \dots, p_n\} \subset \mathbb{R}^2$, where $1 \leq n < \infty$, of n point generators in the plane. Let $\mathbf{x}_i$ and $\mathbf{x}_j$ be location vectors of p_i and p_j. Then, $\mathbf{x}_i \neq \mathbf{x}_j$ for $i \neq j$, $i, j \in \mathbb{N}_n$

($\mathbb{N}_n$ is a set of native numbers with the size n). All sites in the space are assigned to their nearest point generators from P with regard to the Euclidean distance. The result is a transformation of $\mathbb{R}^2$ into a set of regions

$$V(p_i) = \{\mathbf{x}| \parallel \mathbf{x} - \mathbf{x}_i \parallel \leq \parallel \mathbf{x} - \mathbf{x}_j \parallel \text{ for } \forall \mathbf{x} \in \mathbb{R}^2,\ i,j \in \mathbb{N}_n;\ j \neq i\} \qquad (1)$$

associated with generator p_i. The *ordinary Voronoi diagram* consists of regions $VD(P) = \{V(p_1), \dots, V(p_n)\}$ generated by a set of point generators P and all points laying on this diagram are ensured to have the largest possible distance from surrounding point generators.

Generalised Voronoi diagram

Given a set $O = \{o_1, \dots, o_n\} \subseteq \mathbb{R}^2 (1 \leq n < \infty)$ of n generators in the plane, their *generalised Voronoi diagram* [9] (further GVD) is a partition of the plane into regions, one for each generator, such that the region of generator $o_i \in O$ contains all locations of the plane that are closer to o_i than to any other generator $o_j \in O$. The generators in O can be a set of points, segments, polygons, areas, polyhedrons, etc.

3. Generalised Voronoi Diagram Computation

Challenging implementation of the Fortune plane sweep algorithm (Fortune, 1986) and other efficient methods for computing the *GVD* for segment generators leads further research to approximation algorithms, that represent a trade-off between speed of computation and implementation difficulty, and are easy made to be numerically robust. For most applications (like the robot motion planning) the computation of an approximated Voronoi diagram within a given precision is sufficient.

To evaluate an efficiency of an algorithm that carries out a computation over a set of input data, a standard approach, that is not susceptible to implementation or hardware, is needed. This approach evaluates the asymptotic behaviour of the time required by the algorithm with respect to the size of input data [9].

By using efficient techniques and data structures, the time complexity of the Voronoi diagram computation can equal to $O(n \log n)$. This is the case of the *Fortune plane sweep algorithm* (see [3, 5, 9, 12]), which is based on the fundamental *plane sweep technique* of the computational geometry.

The main idea of this algorithm is a shift of a horizontal line l (called a sweep line) from the top to the bottom part of a plane over all n generators $p \in P$, where $P = \{p_1, \dots, p_n\} \subseteq \mathbb{R}^2 (1 \leq n < \infty)$, while constructing the $VD(P)$. During this shift, a sequence of parabolic arcs (so-called a beach line) is maintained consisting of points, which have the same distance from the sweep line l and the generators of these parabolic arcs, as is illustrated in Figure 3. During the computation, the breakpoints between these parabolic arcs in the beach line trace the $VD(P)$, and the part of the plane above l does not affect the computation in comparison to the part of the plane below l.

Refer to [12] or [15] for a detailed description of this algorithm including description of its implementation and handling degenerated cases.

The *basic uniform approximation algorithm* uniformly approximates segment or polygonal generators by point generators first [9] and then applies the modified Fortune plane sweep technique to compute the *GVD* over this set [12]. Due to the uniform approximation

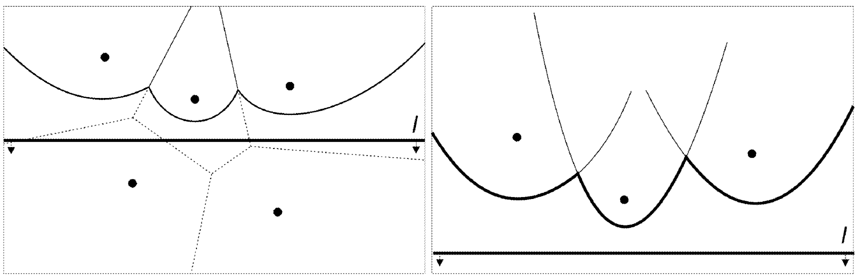

Fig.3: Beach line – a sequence of parabolic arcs

nature, this algorithm is very slow, thus a new approximation algorithm, which attempts to put approximation points only to places where needed, was proposed.

3.1. Approximation Algorithm with Fast Preprocessing

There is presented a new approximation algorithm for constructing the generalised Voronoi diagram for point, segment, or polygonal generators, which is a successor of the real-time non-uniform approximation algorithm as introduced in [14]. This algorithm is based on the Fortune plane sweep technique, which combines advantages of being optimal like the *divide-and-conquer algorithm* but avoiding the difficult merge step, and being relatively simple like the *incremental algorithm*, while representing a trade-off between a complexity of implementation and a speed of computation [3, 9].

The main idea of this algorithm is uniform or non-uniform approximation of every segment generator or series of segment generators by sequences of point generators with higher density in narrow corridors (see Figure 4 on the right). This approach attempts to efficiently detect edges of narrow corridors (segment generators), which are approximated by more point generators than others, thereby the computation is faster in comparison to uniform point distribution with the same precision for all generators.

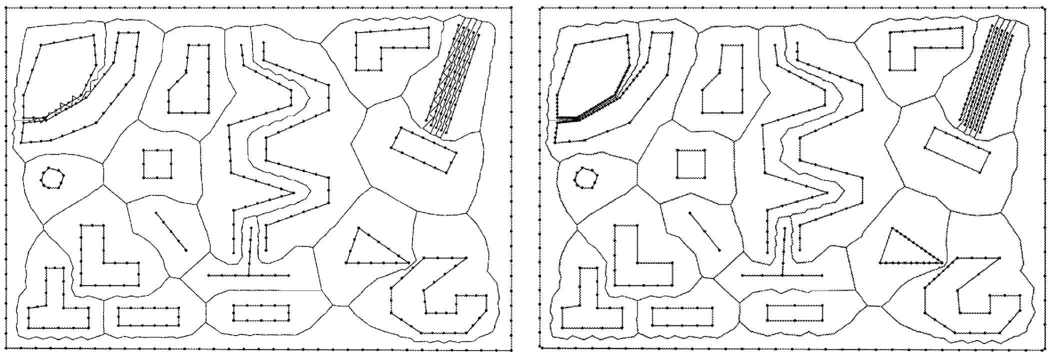

Fig.4: Result of the computation of the basic uniform approximation algorithm (left) and the result of the computation of the approximation algorithm with fast preprocessing (right)

Algorithm Outline

The computation is divided into two stages, the first – segment or polygonal generators fast preprocessing stage and the second – computing *GVD* using the Fortune plane sweep algorithm stage itself, refer to [12].

The preprocessing part includes a point approximation process, that is applied only when two neighbouring generators are close enough. It is based on the plane sweep technique to be able to detect the closest neighbouring segment generators in $O(n \log n)$ time complexity, where n is a number of segment generators. Having applied the neighbouring detection procedure, these segments are uniformly updated by new approximation point generators, which is followed by computing the *GVD*.

Having constructed the *GVD* for approximation point generators made in the preprocessing part, the redundant edges are deleted or not considered during further computation on this structure, see [12] for details.

Data Structures

The preprocessing part of the algorithm uses the following data structures:

1. **Priority queue *Q*** represented by a heap sorting its content according to y coordinates of input points [7].
2. **Status structure *T*** represented as AVL tree [3]. It serves for an efficient detection of neighbouring horizontal segments in $O(\log n)$ time complexity. It stores only top end points of segments detected on the plane sweep line (see Figure 5).
3. **Visibility structure *V*** serves for an efficient detection of neighbouring vertical segments in $O(\log n)$ time complexity. This structure belongs to the category of dimensional range searching data structures [3]. The input data is a set of segments $S \leftarrow \{s_1, s_2, \dots, s_n\}$ in one dimensional space. A query asks for a segment containing a given point or returns segments inside a one dimensional interval $[s : s']$. The main operations are adding a new segment and removing a set of segments within an interval.

   The searching task can be implemented using a balanced binary search AVL tree for efficiency. Each node in this tree stores a visibility segment for guiding the search. Each visibility segment contains a pointer to the original segment from which was generated.

   To find a segment containing a given point can be done as follows. Start in the root node and return the segment of this node if this segment contains the point, otherwise descend into its left or right child node. The decision which node to descend to depends on x coordinates of its segment end points and x coordinate of the given point. Repeat the whole process until a segment is found or return an empty set.

   To find a set of visibility segments within an interval is more complex. Traversing of the tree starts again in its root. While descending, x coordinates of both segment points in a node are being compared to x coordinates of the left and right points of the given input interval. It can result in six cases, refer to [12] for details.

Preprocessing Stage Outline

The algorithm starts by storing end points of all segment generators into the priority queue Q, where these end points are represented as events. During the process, these events are being dequeued from Q and handled according to their type.

If an event e_{p} represents the top point of its parent segment generator, neighbouring horizontal segment generators of this point are found in the status structure T (see Figure 5) and their point resolution is changed.

To detect neighbouring horizontal segment generators using the status structure T, traverse through this tree of top segment end points. The algorithm starts in the root node of T

and descends into its left or right child node according to the x coordinate of the event. During this descend, it maintains current left and right neighbouring segment generators. Once a leaf node is detected, the latest detected left and right neighbouring segment generators given by their end points are returned.

The point resolution of a given segment generator can be both, the uniform or the non-uniform point distribution. In case of using the uniform point distribution, the distance between points on a segment generator represents the value of resolution, and is given by the precision approximation factor P.

The next step is inserting the top point of the segment generator into the status structure T. This is followed by changing of resolution of segment generators immediately above currently processed segment generator. These segment generators are detected using the visibility structure V as described earlier. During this process, a few visibility segments disappear and maximum three new visibility segments in V are created.

If the event e_{p} represents a bottom point of its parent segment generator, the top point of this generator is removed from the status structure T. This is followed by the horizontal resolution update procedure.

Theorem 1. The asymptotic time complexity of the preprocessing part of the approximation algorithm with fast preprocessing is $O(n \log n)$ for n input segment generators (including segments of polygonal generators).

Theorem 2. The asymptotic time complexity of the approximation algorithm with fast preprocessing is $O(n \log n)$ for n input segment generators, where the resolution of approximation point generators on each segment generator is upper bounded by the precision approximation factor P.

Refer to [12] for a formal description of the algorithm above and proofs of the theorems.

3.2. Analysis of Experiments

According to accomplished experiments [12], there is a clear evidence, that the approximation algorithm with fast preprocessing is substantially faster than the basic uniform approximation algorithm, especially for large and detailed environments. The main reason for this is the amount of approximation point generators, which are put in a higher density into problematic places like narrow corridors to achieve sufficient precision. Not only the amount of approximation point generators is important but also their distribution. Based on these experiments, the less uniform distribution, the better efficiency of the Fortune plane sweep algorithm is. The reason for this is the amount of circle events being processed in the Fortune plane sweep algorithm, refer to [12].

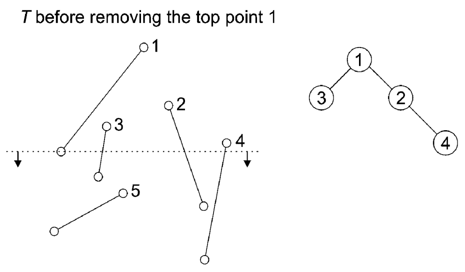

Fig.5: The status structure T

Even though the environment is very dense, the speed of the preprocessing part of the approximation algorithm with fast preprocessing is very high. Further, the amount of the time spent in this preprocessing part is negligible as follows from experiments [12]. The percentage from the whole time spent on computing of this part of the algorithm over a small amount of generators is actually dropping to an insignificant part as the amount of generators is raising.

In the case of the basic uniform approximation algorithm, the sufficient precision of approximation is not known in advance as in the case of the approximation algorithm with fast preprocessing. This makes a substantial practical difference between these two algorithms since the sufficient precision for the basic one must be determined in advance.

4. Motion Planning with Kinematic and Dynamic Constraints

The robot's *free configuration space* C_{free} [3] defines a range of possible positions, which the robot can achieve in its environment. All these positions are defined by its controllability. This controllability is given by the robot's kinematic model, which is consequently given by the robot's underlying mechanism. This kinematic model, to some extent, limits the robot's ability to move in a certain way. The robot dynamics is related to the robot's inertia and adds another additional constraints on C_{free} and trajectory due to mass and force considerations.

In robotics, when referring to the kinematic constraints, the term *holonomic* is used. By definition, a *holonomic robot* is a robot, that does not have any *nonholonomic* kinematic constraints [8] (for example, a wheel sliding constraint). Similarly, a *nonholonomic robot* is defined as a robot, that has one or more nonholonomic kinematic constraints [2,8,11]. The holonomic constraint reduces the size of C_{free} (or workspace W in case when the robot is represented by a point), whereas the non-holonomic constraint reduces the control space U with respect to the current configuration.

4.1. Kinematic models

A mathematical description of the robot's underlying mechanism can be described by its kinematic model. There are different kinematic models for different types of robots. Two kinematic models used in this work follow, other can be found in [1,2,4,8,11].

The kinematic model of a *unicycle robot* is defined using a set of differential equations,

$$\dot{x} = f_1(x,y,\theta,u_v,u_\omega)\ , \tag{2}$$

$$\dot{y} = f_2(x,y,\theta,u_v,u_\omega)\ , \tag{3}$$

$$\dot{\theta} = f_3(x,y,\theta,u_v,u_\omega) \tag{4}$$

as

$$\dot{x} = u_v \cos\theta\ , \tag{5}$$

$$\dot{y} = u_v \sin\theta\ , \tag{6}$$

$$\dot{\theta} = u_\omega \tag{7}$$

where x and y are the coordinates of the robot's position, θ is its orientation, and u_v and u_ω are the longitudinal and angular velocities, respectively. From the previous equations follow that the unicycle robot is defined as *omnidirectional* because it can move in any direction.

By adding an actively controlled steering angle u_δ, the *car-like model* (only Dubin's car [1] is considered) can be defined as,

$$\dot{x} = u_v \cos\theta \ , \tag{8}$$

$$\dot{y} = u_v \sin\theta \ , \tag{9}$$

$$\dot{\theta} = \frac{u_v}{L} \tan u_\delta \tag{10}$$

where L is the length of the robot, x and y are the coordinates of the middle point on the robot's rear axle, $u_v \geq 0$ is the longitudinal velocity, and $|u_\delta| \leq u_{\delta_{\max}}$ is the steering angle.

The total number of *control degrees of freedom* for the car-like model (two control degrees – actuation of the drive wheels and steering) is different in comparison to the number of its total degrees of freedom – coordinates x, y, and orientation parameter θ. For the car-like model, the first-order derivatives depend on each other, which means that the following nonholonomic constraint exists:

$$g(\dot{x}, \dot{y}, \dot{\theta}) = 0 \ . \tag{11}$$

However, there is no constraint in the form of

$$f(x, y, \theta) = 0 \ , \tag{12}$$

which implies, that the car-like model is only governed by the nonholonomic constraint.

4.2. Dynamic models

By adding two integrators in front of action variables u_v and u_ω in the unicycle kinematic model, a dynamic version of this model follows [8],

$$\dot{x} = s \cos\theta \ , \tag{13}$$

$$\dot{y} = s \sin\theta \ , \tag{14}$$

$$\dot{\theta} = \omega \ , \tag{15}$$

$$\dot{s} = u_{\mathrm{a}} \ , \tag{16}$$

$$\dot{\omega} = u_\alpha \tag{17}$$

where s and ω are new phase variables. The speed s is obtained by integration of acceleration u_{a} and the angular velocity ω is obtained by integration of angular acceleration u_α. Now the speed s and the unicycle orientation θ are continuous functions of time.

Similarly for the car-like model, a dynamic version follows,

$$\dot{x} = s \cos\theta \ , \tag{18}$$

$$\dot{y} = s \sin\theta \ , \tag{19}$$

$$\dot{\theta} = \frac{s}{L} \tan\delta \ , \tag{20}$$

$$\dot{s} = u_{\mathrm{a}} \ , \tag{21}$$

$$\dot{\delta} = \omega \ , \tag{22}$$

$$\dot{\omega} = u_\alpha \tag{23}$$

where s, δ, and ω are new phase variables. The speed s is obtained by integration of the acceleration u_{a}, the steering angle δ is obtained by integration of the steering velocity ω, which is given by integration of the steering acceleration u_α. Now the speed s is a continuous function of time and the steering angle δ is a C^1 smooth function of time.

4.3. Path following

A robot must be able to move correctly between two configurations on the constructed path. The nonholonomic constraint imposes that the curvature of a resulting path should have a maximum limit. For a robot in order to stay on the trajectory, it must exert effort to overcome the centrifugal acceleration, therefore a circular arc on the path should have a minimum curvature radius. To get a feasible version of this path under kinematic and dynamic constraints imposed by the robot, these straight angles should be approximated by circular arcs.

The task of the planner is to compute a trajectory through a state space (a configuration space or a phase space of this configuration space, see [8]) that connects initial and goal states while satisfying differential constraints. Let U be an action space, which is a bounded subset of $\mathbb{R}^m$. The planner computes an action trajectory $\tilde{u}$ defined as a function $\tilde{u} : t \rightarrow u$, where the time $t = \langle 0, \infty)$ and the control action $u \in U$.

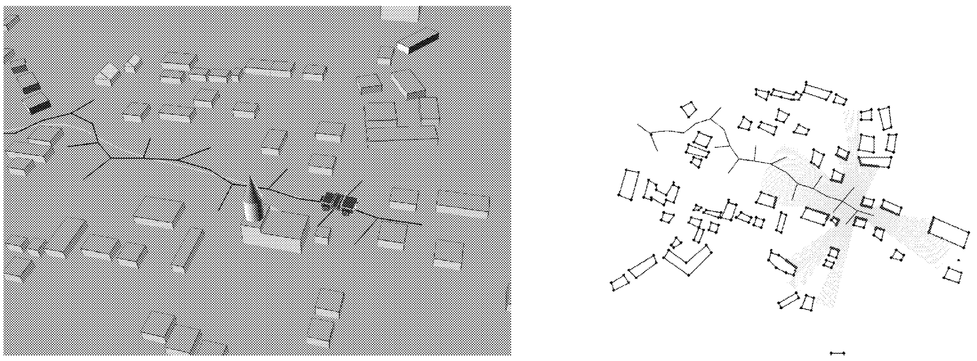

Fig.6: Path constructed by the kinematic controller of a robotic car (light gray path on the left)

The planner has to consider the robot's kinematic or dynamic constraints by either postprocessing the path (in the global motion planning) or directly including the kinematic and dynamic models into the path computation (in the real-time motion planning).

Global motion planning

In the global motion planning task, the *GVD* is used as a *roadmap* (see [12, 13]) in the workspace for a robot to be able to find a path between two given locations while preserving the largest (the safest) distance from surrounding obstacles. In this task, the postprocessing variant involves a decision whether the type of the path is suitable for the robot (due to its size or velocity), and if not, it attempts to find an another path and creates a new kinodynamic trajectory going through all key vertices on this path.

Real-time motion planning

In the real-time motion planning task, the robot must be able to autonomously detect surrounding obstacles, represent the raw sensor data as high-level features (segments), compute a local generalised Voronoi diagram, and proceed with a series of local motion planning tasks to achieve the final position by having only a direction to it or by having only its local referenced coordinates. This motion planning task is based on a sequence of local *GVD*s (see [12]), however, the robot's path-planning task is reduced to setting intermediate vertices (subgoals) lying on requested path and the local trajectories between these vertices are created according to the kinodynamic model. Clearly, the controller is able to adapt the path if dynamic changes of the environment occurs (it belongs to the feedback control category) and it can react to moving obstacles by recomputing a part of the *GVD* (or constructing a new local *GVD*).

The computation of a new global or local trajectory based on the kinodynamic model is handled by the *task controller*. The task controller generates a final smooth path based on spline curves (for examples cubic splines). The objective of the kinematic controller is to follow the path made up of a sequence of key vertices and segments.

An example of an applied car-like kinematic model on a generated global path laying on the generalised Voronoi diagram is shown in Figure 1. In this example, the key points on the found path during the computation are determined first, and then the path between these key points is interpolated with respect to a chosen kinematic model. The differential constraints are ignored in the planning process (construction of the GVD) first to be appropriately handled later on. This corresponds to executing the computed path as closely as possible using control techniques.

An example of an applied car-like kinodynamic model in the real-time motion planning is shown in Figure 6. In this example, differential constraints are considered directly in the planning process, which conforms to the natural motions of a mechanical system. Similarly, see Figure 7 for an omnidirectional robot moving in an office-like environment.

The change in the nonholonomic robot's orientation is coupled with the motion of the robot's origin that is why there is no need for planning of the robot's angular motion. Yet the change in the orientation of a holonomic robot is independent of the motion of the robot's origin. This orientation can be kept fixed or linearly interpolated, where the rotation angle θ is divided into n equal intervals and the equivalent rotation angle is

$$\theta_i = i\,\frac{\theta}{n}\ , \quad i = 1, 2, \ldots, n \tag{24}$$

where i is a time interval [10] (especially in cases when the orientations of the initial and final configurations are different).

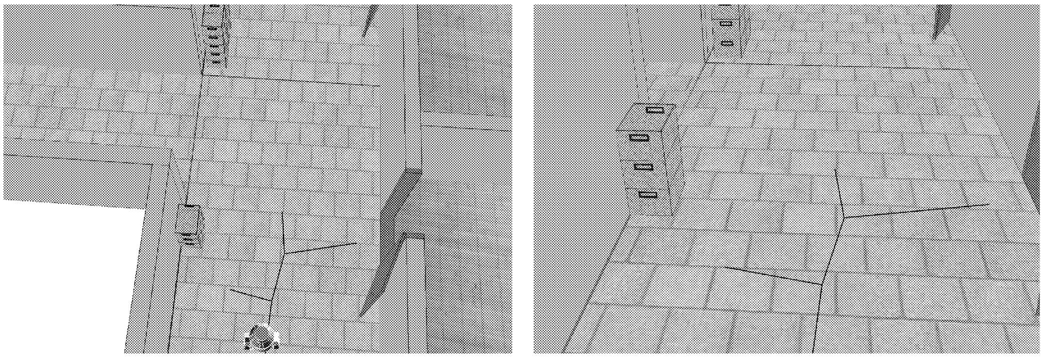
Fig.7: Real-time motion planning for a robot in an office-like environment

Even an omnidirectional robot can demand a smoothing postprocessing step not only in case of a path based on the global *GVD* (see Figure 7). This path has strictly the largest distance from surrounding obstacles. This property is not necessary for all parts of the path and can be easily relaxed (see Figure 8).

The corridor made along the path consists of maximal clearance circles of Voronoi vertices. This corridor can be used for constructing an arbitrary path, which suits a particular application regardless of the shape of the original path. Figure 8 shows this situation, where the original path strictly lays onto the generalised Voronoi diagram while the other one represents a path, which is short enough and still maintains some amount of clearance between the robot and obstacles. This path can be computed in $O(n)$ time complexity, where n is a number of path vertices.

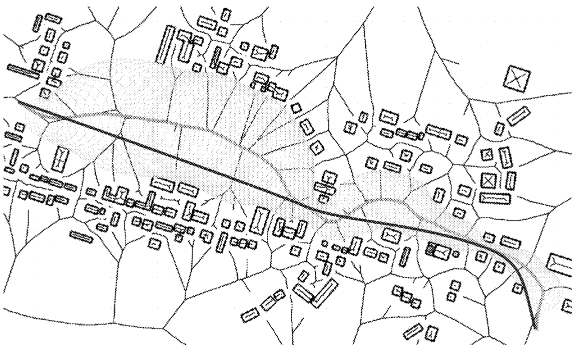

Fig.8: Corridor which is encoded into the path during the computation can be used to create different types of paths for different applications later on. There is shown a path strictly laying onto the generalised Voronoi diagram and the other one that is computed to be used in a relevant application

5. Conclusions

Considering kinodynamic properties of a nonholonomic robot is an inevitable part of the design of motion planning methods. In this work, the tasks solved include kinodynamic constraints in both the real-time (single-query) and global motion planning (multi-query) methods.

The main task of the kinodynamic robot motion planning is a collisionless transfer of a nonholonomic robot amid point, segment, or polygonal obstacles from an initial position to a final one, while satisfying the robot's kinematic and dynamic constraints. It is obvious that the rigidity of the generalised Voronoi diagram to maintain the largest or the safest distance from surrounding obstacles can be relaxed to accommodate the practicality of various motion planning tasks.

A novel approximation algorithm for constructing the generalised Voronoi diagram has been proposed. The Voronoi diagram is one of the most important structures in the computational geometry and is made by a topological graph, which has two main important properties of having the largest distance from surrounding generators (or obstacles) and an ability to represent the continuity of the whole space. These two properties directly satisfy a robot motion planning task, where the problem of finding the safest path amid obstacles is reduced to a problem of finding a path in a topological graph made by the generalised Voronoi diagram.

The new algorithm – the approximation algorithm with fast preprocessing is based on the Fortune plane sweep technique and attempts to detect narrow corridors in an environment that are consequently sampled with a higher density of approximation point generators. From this follows that the speed of computation of this algorithm is faster in comparison to the speed of computation of the basic uniform algorithm, which uses a uniform point approximation on every segment generator. The result of the approximation is that the parabolic arcs of the generalised Voronoi diagram (its edges) are substituted by sequences of straight segments.

The approximation algorithm with fast preprocessing directly falls into the family of geometric algorithms and it represents a trade-off between the speed of computation, robustness, and implementation difficulty while preserving the sufficient precision for most applications. The time complexity of this algorithm is $O(n \log n)$ for n input segment generators (or segments of polygonal generators) provided that the resolution of approximation point generators on every segment is upper bounded. This theorem was proved theoretically and in a suite of experiments [12].

References

[1] Anisi D.A.: Optimal Motion Control of a Ground Vehicle, Tech. rep., Swedish Defence Research Agency, System Technology Division, 2003, Stockholm.

[2] Braun T.: Embedded Robotics, Mobile Robot Design and Applications with Embedded Systems, Springer-Verlag, 2006, New York

[3] De Berg M., Van Kreveld M., Overmars M., Schwarzkopf O.: Computational Geometryš,: Algorithms and Applications, Springer-Verlag, 2000, Utrecht, Netherlands

[4] Egerstedt M.: Motion Planning and Control of Mobile Robots, PhD thesis, Department of Mathematics, Royal Institute of Technology, 2000, Stockholm

[5] Fortune S.: A Sweepline Algorithm for Voronoi Diagrams, In: Second Annual Symposium on Computational Geometry, New York 1986, pp. 313–322

[6] Fortune S.: Voronoi Diagrams and Delaunay Triangulations, In: Computing in Euclidean Geometry, Singapore 1995, vol. 4 of Lecture Notes Series on Computing, pp. 225–265

[7] Korsh J.F., Garrett L.J.: Data structures, algorithms, and program style using C, PWS Publishing Co., 1988, Boston, MA, USA

[8] Lavalle S.: Planning Algorithms, Cambridge University Press, 2006, Cambridge

[9] Okabe A., Boots B., Sugihara, K. Chiu S. N.: Spatial Tesselations : Concepts and Applications of Voronoi Diagrams, John Wiley & Sons, 2000, Chichester

[10] Selig J.M.: Introductory Robotics, Prentice-Hall, 1992, Hertfordshire

[11] Siegwart R., Nourbakhsh I.R.: Introduction to Autonomous Mobile Robots, 2004, Bradford Book

[12] Šeda M.: A Comparison of Roadmap and Cell Decomposition Methods in Robot Motion Planning, WSEAS Transactions on Systems and Control, 2, 2, (2007), pp. 101–108

[13] Švec P.: Using Methods of Computational Geometry in Robotics, PhD thesis, Institute of Automation and Computer Science, Faculty of Mechanical Engineering, Brno University of Technology, 2007, Brno

[14] Švec P.: A Construction of the 2D Generalized Voronoi Diagram, Part I: An Approximation Algorithm, In: Proceedings of the 12th International Conference on Soft Computing MENDEL 2006, Brno 2006, pp. 124–134

[15] Švec P.: A Construction of the 2D Generalized Voronoi Diagram, Part II: Some Issues of Implementation of Fortunes Plane Sweep Algorithm, In: Proceedings of the 12th International Conference on Soft Computing MENDEL 2006, Brno 2006, pp. 135–144

Received in editor's office: December 27, 2007
Approved for publishing: February 13, 2008