Robotica (2018) volume 36, pp. 971–993. © Cambridge University Press 2018
doi:10.1017/S0263574718000164

Clearance-driven motion planning for mobile robots with differential constraints

Evis Plaku, Erion Plaku* and Patricio Simari

Department of Electrical Engineering and Computer Science,
The Catholic University of America, Washington, DC 20064, USA.
E-mails: 74plaku@cua.edu, simari@cua.edu

(Accepted February 3, 2018. First published online: February 27, 2018)

SUMMARY

This paper presents an approach that integrates the geometric notion of *clearance* (distance to the closest obstacle) into sampling-based motion planning to enable a robot to safely navigate in challenging environments. To reach the goal destination, the robot must obey geometric and differential constraints that arise from the underlying motion dynamics and the characteristics of the environment. To produce safe paths, the proposed approach expands a motion tree of collision-free and dynamically feasible motions while maintaining locally maximal clearance. In distinction from related work, rather than explicitly constructing the medial axis, the proposed approach imposes a grid or a triangular tessellation over the free space and uses the clearance information to construct a weighted graph where edges that connect regions with low clearance have high cost. Minimum-cost paths over this graph produce high-clearance routes that tend to follow the medial axis without requiring its explicit construction. A key aspect of the proposed approach is a route-following component which efficiently expands the motion tree to closely follow such high-clearance routes. When expansion along the current route becomes difficult, edges in the tessellation are penalized in order to promote motion-tree expansions along alternative high-clearance routes to the goal. Experiments using vehicle models with second-order dynamics demonstrate that the robot is able to successfully navigate in complex environments. Comparisons to the state-of-the-art show computational speedups of one or more orders of magnitude.

KEYWORDS: Motion Planning; Mobile Robots; Navigation; Path Planning; Robot Dynamics.

1. Introduction

A fundamental property of autonomous robots is their ability to navigate complex, obstacle-rich environments without collision. Motion planning plays an essential role in an extensive number of applications in robotics, including self-driving cars, search-and-rescue missions, space exploration, surgical interventions, and computer-aided design.[6]

Though widely studied, planning the trajectory of an autonomous robot is notoriously difficult. To reach a given destination, the robot must overcome the challenges that arise from navigating amidst numerous obstacles and narrow passages. Motions must obey the physical dynamics of the robot, which impose constraints on velocity, acceleration, direction, and turning radius. A successful planner must produce physically plausible motions that obey these geometric and differential constraints. This is challenging since motion planning is PSPACE-complete[37] when considering only geometric constraints and becomes undecidable when coupled with differential constraints.[3] Additionally, many applications require not only generating feasible trajectories, but also ensuring that these trajectories stay sufficiently far from the obstacles. In fact, maintaining clearance from obstacles while navigating in unstructured environments is vital to ensuring safety.

To address these challenges, we rely on geometry processing to extract semantically meaningful information from complex environments in order to facilitate robot navigation. We propose an

* Corresponding author. E-mail: plaku@cua.edu

https://doi.org/10.1017/S0263574718000164 Published online by Cambridge University Press

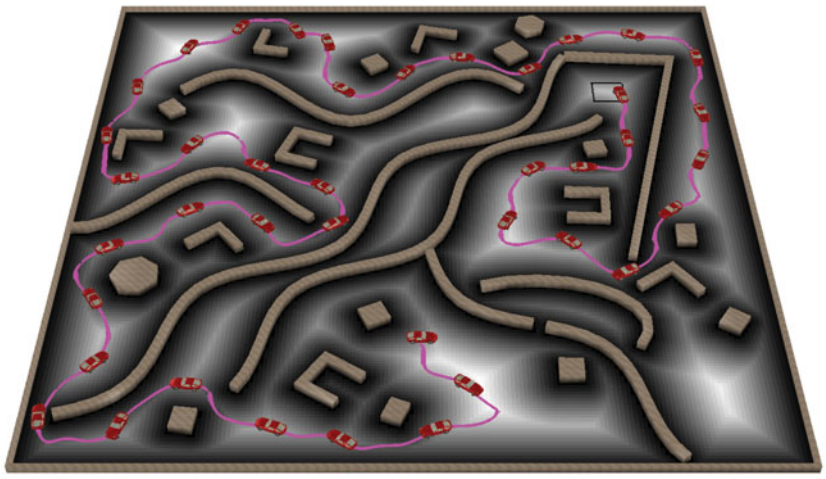


Fig. 1. An example of a motion-planning problem solved by our approach where the robot has to navigate from its start position to the goal region (black rectangle) while avoiding collisions with the obstacles (brown). The gray-scale value of the faces in the terrain mesh is proportional to their distance to the nearest obstacle. The solution trajectory is shown in magenta overlaid with snapshots of the robot. Videos of solutions obtained by our approach on this and other scenes can be found at https://goo.gl/3eqBMv.

approach that combines geometry processing with sampling-based motion planning. We leverage the notion of *clearance*, defined for any point in the navigable space as the distance from it to the nearest obstacle boundary (Fig. 1). Intuitively, discouraging the robot from approaching obstacles will promote the generation of safe trajectories that maximize clearance.

Maintaining locally maximal clearance effectively implies that the robot should remain on or near the medial axis of the navigable terrain. Traditionally, as discussed in the next section, state-of-the-art algorithms have explicitly computed the medial axis in order to achieve high-clearance routes. However, explicitly constructing an exact medial axis over general geometry is difficult and expensive, particularly if we would like the method to potentially generalize to higher dimensions. Fortunately, for motion planning, we propose that such an explicit construction is not necessary. Instead, we rely on the distance field of the obstacle boundaries over the navigable terrain, the ridges of which coincide with the medial axis. We construct the distance field over a triangular or grid-based tessellation. This is inexpensive to compute and provides an abstraction which we use to effectively guide sampling-based motion planning.

To account for the dynamics, the proposed approach leverages from sampling-based motion planning the notion of expanding a motion tree whose branches define collision-free and dynamically feasible motions. We also leverage from prior work, such as `GUST`[35] and its precursor `Syclop`,[36] the idea of using discrete search over a tessellation of the environment to guide the expansion of the motion tree. While prior work relied on shortest paths to guide the motion-tree expansion, the premise of our approach is that significant improvements in runtime and solution quality can be obtained if the motion-tree expansion is guided by high-clearance routes. To achieve this premise, we first contribute an efficient method to compute high-clearance routes to guide the motion-tree expansion. This contribution is not so much in the method as a stand-alone component – as there is a considerable body of work that focuses on computing high-clearance paths – but in its suitability to effectively guide the motion-tree expansion and incorporate feedback information to enable the discovery of alternative high-clearance routes when expansion along the main route becomes difficult due to constraints imposed by the obstacles and the robot dynamics. To make this possible, we use the distance field over the tessellation to construct a weighted graph where edges that connect regions with low clearance have high cost. Minimum-cost paths over this graph produce high-clearance routes that follow the medial axis without requiring its explicit construction. Additionally, we further enhance the edge cost definition with the use of a robust cost estimator, which results in diminishing returns once a base level of clearance is attained, thus reducing computational cost while maintaining route quality.

https://doi.org/10.1017/S0263574718000164 Published online by Cambridge University Press

The second major contribution is an efficient method to expand the motion tree so that it closely follows high-clearance routes. This is made possible by converting the route into a sequence of waypoints, grouping the nodes in the motion tree according to the waypoints, and attempting to expand from one group to the next in order to reach the waypoints in succession while remaining close to the route. Several technical challenges arise such as determining from which group to expand, effectively expanding from one group to the next, remaining close to the route, and handling situations where expansions fail due to constraints imposed by the obstacles and the robot dynamics.

Note that the route computation and route following work in tandem. When progress along a route becomes difficult due to constraints imposed by the obstacles and the dynamics, the approach penalizes the regions where the motion-tree expansion failed in order to discover alternative high-clearance routes to the goal. The route computation and route following are repeatedly invoked until a solution is found.

Experiments are performed using two robot models (a snake and a vehicle) operating in complex unstructured environments where the robot has to wiggle its way through many obstacles and narrow passages in order to reach the goal. Comparisons to state-of-the-art motion planners show that our planner, which leverages this geometric feature, is faster by one to two orders of magnitude while also generating solutions with significantly higher clearances. We also note that it is possible to make GUST work with our clearance-based tessellation graph. In fact, we did so for the experiments. Modifying GUST to use our route-following behavior would require significant effort and a complete redesign that would intrinsically change the nature of this algorithm so that it would no longer be GUST. Experiments showed that GUST is faster when using our clearance-based tessellation graph than when using its original, Euclidean-distance, tessellation graph. However, the modified version of GUST is still significantly slower, by an order of magnitude, than our approach. These results further demonstrate the significance of the route-following behavior in our approach.

2. Related Work

Extensive research has focused on motion-planning methods capable of finding trajectories that have high-clearance from obstacles and the medial axis is widely used toward that end.

At a high level, algorithms for computing the medial axis can be classified broadly into Voronoi-based methods and distance-field methods. Voronoi-based methods approximate the medial axis using a subset of the vertices and edges of the obstacles' Voronoi cells that extend into the navigable space.[1,13] Another method computes generalized Voronoi diagrams in two and three dimensions using graphics hardware by dividing the space into regular cells and computing for each cell the closest primitive and the distance to that primitive.[18] Distance-field approaches search a decomposition of the free space for cells that are intersected by the medial axis.[14,42] Unfortunately, in either case, computing an explicit representation of the medial axis, especially for navigable spaces in higher dimensions, is highly nontrivial and can only be approximated.[10]

In the above approaches, the distance to the medial axis serves as a complementary proxy for clearance, i.e., the smaller the distance to the medial axis, the larger the clearance. In contrast, we perform a direct evaluation of clearance as the distance to the nearest obstacle. This geometric query can be efficiently evaluated on myriad geometric representations and sidesteps the explicit construction of the medial axis. In addition, as we will see in Section 5.8, this direct evaluation can be convolved with a robust estimator of cost to model the notion of diminishing returns once sufficient clearance has been obtained.

Medial Axis Probabilistic Road Map (MAPRM)[44] constructs a roadmap containing nodes that lie in close proximity to the medial axis. By biasing randomly-sampled configurations toward the medial axis, it maps narrow passages more efficiently than uniform random sampling.[29] Other work provides extensions for the case of free-flying rigid bodies moving in three dimensions[45] and flexible objects.[15] A general framework for using the workspace medial axis in PRM planners has also been developed.[19] Further improvements allow for uniform sampling.[46] However, this family of approaches focuses on motion planning in configuration space, which does not take dynamics into account. In contrast, our method considers motion dynamics, thus ensuring that we generate physically plausible motions that can be transferred to real-world applications.

https://doi.org/10.1017/S0263574718000164 Published online by Cambridge University Press

A more related approach, Medial Axis Rapidly-Exploring Random Tree (MARRT)[11] extends RRT[28] by expanding a motion tree along the medial axis of the free space. At each iteration, the tree is extended from the nearest vertex to a random configuration. While RRT uses linear interpolation to connect the two configurations, MARRT uses bisection to move the intermediate configurations onto the medial axis. This enables MARRT to avoid cluttering near obstacles, which is an issue for RRT. Another method, Transition-based RRT (TRRT),[12] plans paths in continuous cost spaces by combining RRT with stochastic optimization methods which use transition tests to accept or reject a new potential state. fRRT is an informed RRT variant, which uses costs as in A* to bias sampling toward low-cost regions. Deformable RRT[17] seeks to optimize the sample location in order to improve the quality of the solutions. In other work, Gaussian mixture models or Gaussian mixture fields have been integrated with RRT to learn models of collision and collision-free regions which are then used to improve sampling.[20,31,33] To improve the expansion in RRT, numerical methods have also been used which approximate the reachable state space[34] or use any-angle path biasing.[32] To speed up motion planning in dynamic environments, diffusion maps have been proposed which efficiently represent pairwise cost-to-go potentials.[5] RRT and its variants, however, due to the random sampling and the nearest-neighbor heuristic, have difficulty expanding the motion tree in cluttered environments and through narrow passages, especially when the dynamics are nonlinear.

Other approaches use decompositions to facilitate motion planning. PDST[24] creates a subdivision, KPIECE[8] relies on multi-layered grids, and Syclop[36] and GUST[35,43] use discrete search over a workspace triangulation. Another approach introduces a cell decomposition and meshing-based methodology to represent environments for navigation and extract paths with clearance information. The local clearance triangulations are computed by refinement operations on a Constrained Delaunay Triangulation of the input obstacle set.[21] These methods, however, do not focus on computing safe trajectories, and might cause the robot to move in close proximity to obstacles. In contrast, our approach relies on high-clearance routes to guide the motion-tree expansion.

Other improvements made by our approach include procedures for selecting regions based on clearance values, following safer routes, and updating route edge costs to reflect the progress made. Comparisons to the state-of-the-art show that our planner generates solution trajectories with significantly higher clearance while also being one to two orders of magnitude faster.

3. Preliminaries

This section defines the motion-planning problem and provides background information on sampling-based motion planning.

3.1. Problem formulation

Let $\mathcal{O} = \{\mathcal{O}_1, \ldots, \mathcal{O}_m\}$, $\mathcal{G}$, and $\mathcal{W}$ denote the obstacles, goal region, and bounding box, respectively, of the environment. The obstacles and the goal lie inside $\mathcal{W}$ and are pairwise disjoint. The mobile robot is represented by its geometric shape, state space $\mathcal{S}$, control space $\mathcal{U}$, and motion equations f. $\mathcal{S}$ consists of a set of variables that describe the state, such as position, orientation, steering angle, and velocity. $\mathcal{U}$ defines the control inputs that can be applied to the mobile robot, such as setting the acceleration and turning rate of the steering wheel. The motion equations are expressed as a set of differential equations

$$\dot{s} = f(s, u), \tag{1}$$

which describe how the state $s \in \mathcal{S}$ changes when applying the control $u \in \mathcal{U}$. The resulting motion is computed by a function

$$s_{\text{new}} \leftarrow \text{SIMULATE}(s, u, f, dt), \tag{2}$$

where s_{new} is the new state obtained after applying u to s and integrating f (using Runge–Kutta) for one time step dt.

To facilitate the presentation, Fig. 2 shows the car and the snake models used in the experiments. The car state $s = (x, y, \theta, \psi, v)$ defines the position (x, y), orientation θ, steering angle ψ, and

https://doi.org/10.1017/S0263574718000164 Published online by Cambridge University Press



Fig. 2. The car and snake models used in the experiments.

velocity v. The car is controlled by setting the acceleration u_{acc} and the steering rate u_ω. The motion equations f are defined as

$$\dot{x} = v\cos(\theta)\cos(\psi), \dot{y} = v\sin(\theta)\cos(\psi), \tag{3}$$

$$\dot{\theta} = v\sin(\psi)/L, \dot{v} = u_{\text{acc}}, \dot{\psi} = u_\omega, \tag{4}$$

where L is the distance between the back and the front wheels.

The snake is modeled as a car pulling several trailers.[26] The state $s = (x, y, \theta, \psi, v, \theta_1, \ldots, \theta_N)$ defines the position (x, y), orientation θ, steering angle ψ, and velocity v of the head link and the orientation θ_i of each of the N trailer links. Motion equations are extended to include the changes that occur to each of the trailer links as

$$\dot{\theta}_i = \frac{v}{H}(\sin(\theta_{i-1}) - \sin(\theta_0))\prod_{j=1}^{i-1}\cos(\theta_{j-1} - \theta_j), \tag{5}$$

where $\theta_0 = \theta$. The model can be made to resemble a snake, as shown in Fig. 2, by setting the hitch distance H between the links to a small value ($H = 0.01$ in the experiments).

A state $s \in \mathcal{S}$ is considered valid if the mobile robot lies collision-free inside $\mathcal{W}$ when placed according to the position and orientation specified by s. Proximity-query package[25] is used to efficiently implement $\text{COLLISION} : \mathcal{S} \to \{\texttt{true}, \texttt{false}\}$.

A motion trajectory $\zeta : \{1, \ldots, \ell\} \to \mathcal{S}$ is defined by starting at a state $s \in \mathcal{S}$ and applying a sequence of controls $\langle u_1, \ldots, u_{\ell-1}\rangle$ in succession; i.e., $\zeta(1) = s$ and $\forall i \in \{2, \ldots, \ell\}$:

$$\zeta(i) = \text{SIMULATE}(\zeta(i-1), u_{i-1}, f, dt). \tag{6}$$

The motion-planning problem for a mobile robot whose motions are expressed by a set of differential equations is defined as follows.

Definition 1 *Motion-Planning Problem.* Given

- the environment in terms of its bounding box $\mathcal{W}$, obstacles $\mathcal{O} = \{\mathcal{O}_1, \ldots, \mathcal{O}_m\}$, and goal region $\mathcal{G}$,
- a model of a mobile robot in terms of its geometric shape, state space $\mathcal{S}$, control space $\mathcal{U}$, motion equations f, and time step dt,
- and an initial state $s_{\text{init}} \in \mathcal{S}$

compute a sequence of controls $\langle u_1, \ldots, u_{\ell-1}\rangle$ such that the trajectory $\zeta : \{1, \ldots, \ell\} \to \mathcal{S}$ obtained by starting at s_{init} and applying the controls in succession, as defined in Eq. (6), avoids collisions and reaches $\mathcal{G}$; i.e.,

$$\forall i \in \{1, \ldots, \ell\} : \text{COLLISION}(\zeta(i)) = \texttt{false} \text{ and } \text{POS}(\zeta(\ell)) \in \mathcal{G},$$

where $\text{POS}(\zeta(\ell))$ denotes the position component of $\zeta(\ell)$.

3.2. Motion tree

When dealing with dynamics, sampling-based motion planners search for a solution by expanding a motion tree $\mathcal{T}$ with collision-free and dynamically feasible motion trajectories as branches. The

https://doi.org/10.1017/S0263574718000164 Published online by Cambridge University Press

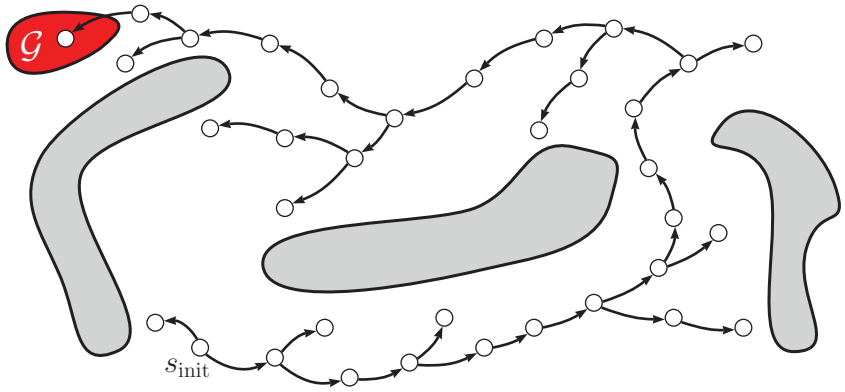


Fig. 3. Nodes in the motion tree represent collision-free states and edges denote collision-free and dynamically feasible motions. Obstacles are shown in gray. The goal region, $\mathcal{G}$, is shown in red.

motion tree $\mathcal{T}$ is represented as a directed graph $\mathcal{T} = (V_{\mathcal{T}}, E_{\mathcal{T}})$. Each node $v \in V_{\mathcal{T}}$ is associated with a collision-free state, denoted by $\text{STATE}(v)$. Each edge $(v, v') \in E_{\mathcal{T}}$ is associated with a collision-free motion from $\text{STATE}(v)$ to $\text{STATE}(v')$. Such motion is obtained by applying a control $u \in \mathcal{U}$ to $\text{STATE}(v)$ and integrating the motion equations f for one time step dt; i.e., $\text{STATE}(v') = \text{SIMULATE}(\text{STATE}(v), u, f, dt)$.

The motion tree $\mathcal{T}$ initially contains only the root node, which is associated with the initial state; i.e., $V_{\mathcal{T}} = \{v_{\text{init}}\}$, $E_{\mathcal{T}} = \emptyset$, and $\text{STATE}(v_{\text{init}}) = s_{\text{init}}$. The motion tree $\mathcal{T}$ is incrementally expanded by adding new nodes and edges. A branch is generated from a node $v \in V_{\mathcal{T}}$ by applying controls and integrating the motion equations f for several time steps. Each state obtained after an integration step, when collision free, is added to $\mathcal{T}$ with the previous state as its parent. The branch generation from v stops as soon as a collision is encountered. This process of selecting a node and generating a branch is repeated until a solution is found or a runtime limit is reached. A solution is found when a new node v_{new} that has reached the goal, i.e., $\text{POS}(\text{STATE}(v_{\text{new}})) \in \mathcal{G}$, is added to $\mathcal{T}$. The solution corresponds to the trajectory $\zeta_{\mathcal{T}}(v_{\text{new}})$, which is obtained by concatenating the motions associated with the edges that connect the root of $\mathcal{T}$ to v_{new}. Figure 3 provides an illustration.

4. Method

The proposed approach uses high-clearance routes over a discrete tessellation of the environment to effectively guide the motion-tree expansion. A schematic illustration and pseudocode are shown in Fig. 4 and Algorithm 1. The tessellation and the computation of the high-clearance routes are discussed first, followed by a description of the overall motion planner. Runtime analysis is presented at the end.

4.1. Discrete tessellation as an abstract layer

The environment is tessellated into a finite set of regions $\mathcal{R} = \{\mathcal{R}_1, \ldots, \mathcal{R}_n\}$, which are disjoint except at the boundaries. The tessellation map is represented as an undirected weighted graph $\mathcal{D} = (V_{\mathcal{D}}, E_{\mathcal{D}}, \text{COST}_{\mathcal{D}})$. The vertex set is defined as $V_{\mathcal{D}} = \{v_{\mathcal{R}_1}, \ldots, v_{\mathcal{R}_n}, v_{\mathcal{G}}\}$, where $v_{\mathcal{R}_i}$ corresponds to the region $\mathcal{R}_i$ and $v_{\mathcal{G}}$ corresponds to the goal $\mathcal{G}$. An edge $(v_{\mathcal{R}_i}, v_{\mathcal{R}_j})$ is added to $E_{\mathcal{D}}$ for every pair of neighboring regions $\mathcal{R}_i, \mathcal{R}_j \in \mathcal{R}$. An edge $(v_{\mathcal{R}_i}, v_{\mathcal{G}})$ is added to $E_{\mathcal{D}}$ for every $\mathcal{R}_i \in \mathcal{R}$ that is not disjoint from $\mathcal{G}$. The function $\text{COST}_{\mathcal{D}} : E_{\mathcal{D}} \to \mathbb{R}^{\geq 0}$ defines the cost for each edge.

The approach can work with different tessellation types including triangulations and grids; in either case, only regions that share a boundary are considered to be neighbors. The grid generation is straightforward, while for the triangulation case, we use the Triangle package[39,40] which efficiently generates high-quality triangulations based on a desired maximum triangle area.

A function $\text{LOCATEREGION}_{\mathcal{D}} : \mathcal{W} \to \mathcal{R} \cup \{\texttt{null}\}$ is used to find the region $\mathcal{R}_i \in \mathcal{R}$ that contains a given point $p \in \mathcal{W}$. When p is not inside any of the regions, $\text{LOCATEREGION}_{\mathcal{D}}$ returns `null`. $\text{LOCATEREGION}_{\mathcal{D}}(p)$ runs in constant time for grid tessellations since the ith coordinate of the grid cell containing p can be computed as $\lfloor \text{dims}[i](p[i] - \min[i])/(\max[i] - \min[i]) \rfloor$, where

https://doi.org/10.1017/S0263574718000164 Published online by Cambridge University Press

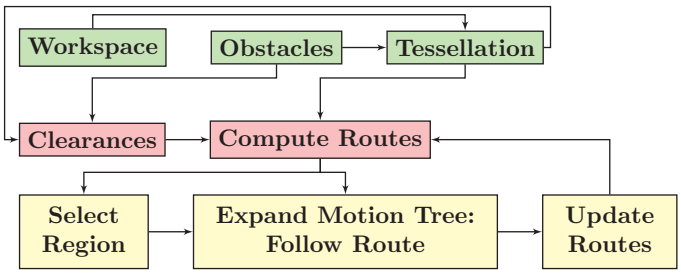


Fig. 4. Schematic illustration of the proposed approach.

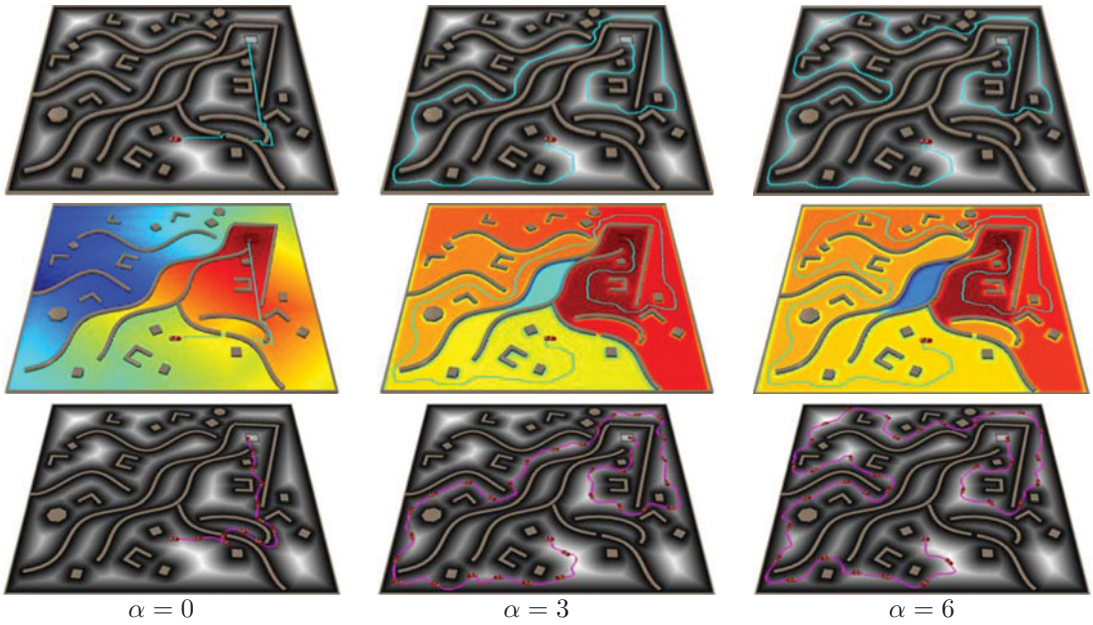


Fig. 5. (Top) Figures in the top row show how the solution route (cyan) changes when increasing the clearance exponent α, which is used in the definition of the cost for the edges of the tessellation (Eq. (7)). The regions in the tessellation are shown in a gray scale based on their distance to the nearest obstacle. (middle) Figures in the middle row show how the minimum cost routes to the goal change as a function of α. The regions in the tessellation are color-coded based on the minimum cost route to the goal. The jet color scheme is used, where the spectrum varies from blue (high cost) to red (low cost). (Bottom) Figures in the bottom row show the collision-free and dynamically feasible simulated execution over the generated solution route (magenta). Snapshots of the robot are overlaid along the solution trajectory.

dims[i], min[i], max[i] denote the number of cells, minimum, and maximum values along the ith grid dimension. In the case of triangulations, LocateRegion$_{\mathcal{D}}$ runs in logarithmic time by using a hierarchy of triangles, where each triangle keeps track of the triangles it intersects in the next level of the hierarchy.[2,23] A simpler implementation that works well in practice for LocateRegion$_{\mathcal{D}}$ in triangulations, which we use in this paper, is to impose a grid and store in each grid cell the list of triangles that intersect the grid cell boundary, are inside the grid cell, or contain the grid cell. This is done once during a precomputation stage. Then, LocateRegion$_{\mathcal{D}}(p)$ first locates the grid cell that contains p and then iterates over the list of the triangles associated with the grid cell to find the triangle that contains p, returning `null` if the list of the triangles is empty. This tends to run in constant time in practice when using a fine-grained grid since each grid cell tends to intersect only a constant number of triangles.

4.2. High-clearance routes to the goal

The approach uses high-clearance routes to guide the motion-tree expansion to the goal. Such routes are computed by searching the tessellation $\mathcal{D}$. To ensure that preference is given to edges with high

https://doi.org/10.1017/S0263574718000164 Published online by Cambridge University Press

clearance, the edge cost is defined as

$$\text{COST}_{\mathcal{D}}(\mathcal{R}_i, \mathcal{R}_j) = \frac{||\text{CENTROID}(\mathcal{R}_i) - \text{CENTROID}(\mathcal{R}_j)||}{(\min\{\text{CLEAR}(\mathcal{R}_i), \text{CLEAR}(\mathcal{R}_j)\})^{\alpha}}, \tag{7}$$

where $\text{CENTROID}(\mathcal{R}_i)$ denotes the mean position of the points in $\mathcal{R}_i$, $||\cdot||$ denotes the Euclidean distance, $\alpha \in \mathbb{R}^{\geq 0}$, and $\text{CLEAR}(\mathcal{R}_i)$ denotes the minimum distance separating $\text{CENTROID}(\mathcal{R}_i)$ from the obstacles, i.e.,

$$\text{CLEAR}(\mathcal{R}_i) = \min\{||\text{CENTROID}(\mathcal{R}_i) - o|| : o \in \mathcal{O}_1 \cup \ldots \cup \mathcal{O}_m\}. \tag{8}$$

The minimum-cost route to the goal, denoted by $\text{ROUTE}_{\mathcal{D}}(\mathcal{R}_i)$, is computed for each $\mathcal{R}_i \in \mathcal{R}$. This is achieved by a single call to Dijkstra's shortest-path algorithm using $v_{\mathcal{G}}$ as the source. The exponent α serves to tune the importance of the clearance when computing the routes. Setting α to a high value leads to high-clearance routes that closely follow the medial axis, as shown in Fig. 5. In our experiments, we vary α between 0 and 6. See Section 5.4 for more details and results.

$\text{CLEAR}(\mathcal{R}_i)$ can be computed by using point-polygon distances or point-mesh distances when all obstacles are represented as one mesh. Even approximate computations can be used since they tend to assign high costs to edges with low clearances and low costs to edges with high clearances. In the experiments, in addition to the point-mesh distances, we use a brushfire search[6] to efficiently approximate $\text{CLEAR}(\mathcal{R}_i)$ for each $\mathcal{R}_i \in \mathcal{R}$. The brushfire search starts by assigning a value of 1 and inserting into a queue all the regions in $\mathcal{R}$ that share a boundary with an obstacle or partially intersect an obstacle (which could happen in the case of grid-based tessellations). As in breadth-first search, a region $\mathcal{R}_i$ is removed from the front of the queue. Neighbors of $\mathcal{R}_i$ for which the clearance has not yet been defined are assigned the value $\text{CLEAR}(\mathcal{R}_i) + 1$, and are inserted at the end of the queue. The brushfire search, as the experiments show, works well for fine-grained tessellations.

4.3. Motion planning guided by high-clearance routes

The overall search for a collision-free and dynamically feasible motion trajectory to the goal is driven by procedures to (i) select a region $\mathcal{R}_i \in \mathcal{R}$ using $\text{ROUTE}_{\mathcal{D}}(\mathcal{R}_i)$ to give preference to those regions associated with high-clearance routes to the goal, (ii) expand the motion tree $\mathcal{T}$ so that it follows $\text{ROUTE}_{\mathcal{D}}(\mathcal{R}_i)$ as closely as possible, and (iii) penalize edges in the tessellation where the motion-tree expansion failed in order to discover alternative high-clearance routes to the goal. After initializing $\mathcal{T}$, the approach invokes these procedures iteratively until a solution is found or a runtime limit is reached. Pseudocode and a schematic illustration are shown in Algorithm 1 and Fig. 4, respectively. More details are provided below.

4.3.1. Region selection based on route costs. A node $v \in V_{\mathcal{T}}$ reaches $\mathcal{R}_i \in \mathcal{R}$ when $\text{POS}(\text{STATE}(v)) \in \mathcal{R}_i$. The set of nodes in the motion tree $\mathcal{T}$ that have reached $\mathcal{R}_i$ is then defined as

$$\text{NODES}_{\mathcal{T}}(\mathcal{R}_i) = \{v : v \in V_{\mathcal{T}} \text{ and } \text{POS}(\text{STATE}(v)) \in \mathcal{R}_i\}. \tag{9}$$

The region $\mathcal{R}_i \in \mathcal{R}$ with the maximum weight among those that have been reached by the motion tree $\mathcal{T}$ is selected for expansion. The weight, denoted by $w(\mathcal{R}_i)$, is defined as

$$w(\mathcal{R}_i) = \beta^{\text{SEL}(\mathcal{R}_i)}/\text{COST}(\text{ROUTE}_{\mathcal{D}}(\mathcal{R}_i)), \tag{10}$$

where $0 < \beta < 1$ and $\text{SEL}(\mathcal{R}_i)$ denotes the number of times $\mathcal{R}_i$ has been selected for expansion in previous iterations of the approach. Note that $\text{COST}(\text{ROUTE}_{\mathcal{D}}(\mathcal{R}_i))$ is defined as the sum of the costs associated with the edges in $\text{ROUTE}_{\mathcal{D}}(\mathcal{R}_i)$.

This definition ensures that $w(\mathcal{R}_i)$ is high when $\text{ROUTE}_{\mathcal{D}}(\mathcal{R}_i)$ has high clearance (since it will have low cost). By giving preference to regions associated with high-clearance routes, the motion-tree expansion is more likely to quickly reach the goal since the robot will have more room to maneuver as it follows the route. The term $\beta^{\text{SEL}(\mathcal{R}_i)}$ with $0 < \beta < 1$ serves as a penalty to avoid selecting $\mathcal{R}_i$ indefinitely. In fact, repeated selections of $\mathcal{R}_i$ will sufficiently reduce $w(\mathcal{R}_i)$ so that some other region $\mathcal{R}_j$ will eventually have a higher weight and be selected for expansion. This is essential to ensure the

https://doi.org/10.1017/S0263574718000164 Published online by Cambridge University Press

Input: $\mathcal{W}$: environment bounding box; $\mathcal{O}$: obstacles; $\mathcal{G}$: goal region; $\mathcal{S}$: state space; $\mathcal{U}$: action space; f: motion equations; dt: time step; s_{init}: initial state; $t_{\max}$: upper bound on runtime
Output: collision-free and dynamically feasible trajectory from s_{init} to $\mathcal{G}$ or `null` if no solution is found

$\mathcal{R} = \{\mathcal{R}_1, \ldots, \mathcal{R}_n\} \leftarrow$ TESSELLATION($\mathcal{W}, \mathcal{O}$) *§ 4.1*
$\mathcal{C} = \langle$CLEAR($\mathcal{R}_1$), …, CLEAR($\mathcal{R}_n$)$\rangle \leftarrow$ CLEARANCES($\mathcal{R}, \mathcal{W}, \mathcal{O}$) *§ 4.2*
$\mathcal{D} = (V_{\mathcal{D}}, E_{\mathcal{D}}, \text{COST}_{\mathcal{D}}) \leftarrow$ TESSELLATIONGRAPH($\mathcal{R}, \mathcal{G}, \mathcal{C}$) *§ 4.2*
$\langle$ROUTE$_{\mathcal{D}}(\mathcal{R}_1)$, …, ROUTE$_{\mathcal{D}}(\mathcal{R}_n)\rangle \leftarrow$ COMPUTEROUTES($\mathcal{D}$) *§ 4.2*
$\mathcal{T} \leftarrow$ CREATEMOTIONTREE(s_{init}) *§ 3.2*
while TIME $< t_{\max}$ **do**
  $\mathcal{R}_i \leftarrow$ SELECTREGION($\mathcal{R}$) *// based on route cost, § 4.3.1*
  status $\leftarrow$ FOLLOW(ROUTE$_{\mathcal{D}}(\mathcal{R}_i)$) *§ 4.3.2*
  if v added such that POS(STATE(v)) $\in \mathcal{G}$ **then return** $\zeta_{\mathcal{T}}(v)$
  if status = `no_progress` **then** *§ 4.3.3*
    $\langle$ROUTE$_{\mathcal{D}}(\mathcal{R}_1)$, …, ROUTE$_{\mathcal{D}}(\mathcal{R}_n)\rangle \leftarrow$ RECOMPUTEROUTES($\mathcal{D}$)
return `null`

function FOLLOW(ROUTE$_{\mathcal{D}}(\mathcal{R}_i)$) *§ 4.3.2*
$\langle \mathcal{X}_1, \ldots, \mathcal{X}_r \rangle \leftarrow$ ROUTE$_{\mathcal{D}}(\mathcal{R}_i)$
$\rho = \langle p_1, \ldots, p_\ell \rangle \leftarrow \langle$CENTROID($\mathcal{X}_1$), …, CENTROID($\mathcal{X}_r$)$\rangle$
$\Gamma_1 \leftarrow$ NODES$_{\mathcal{T}}(\mathcal{X}_1)$ *// Eq. (9)*
$\Gamma \leftarrow \{\Gamma_1\}$
while p_r has not been reached **do**
  $\Gamma_j \leftarrow$ select from Γ
  $v \leftarrow$ select from Γ_j
  reached $\leftarrow$ `false`
  $p_{\text{target}} \leftarrow$ SAMPLETARGET($p_{j+1}, d_{\text{follow}}, d_{\text{reach}}$)
  nrSteps $\leftarrow$ MAXNRSTEPS(STATE(v), p_{target})
  for 1 … nrSteps **and** $||\text{POS}(\text{STATE}(v)) - p_{\text{target}}|| > d_{\text{reach}}$ **and** reached = `false` **do**
    $u \leftarrow$ CONTROLLER(STATE(v), p_{target})
    $s_{\text{new}} \leftarrow$ SIMULATE(STATE(v), u, f, dt)
    if DIST(POLYLINE(ρ), POS(s_{new})) $\le d_{\text{follow}}$ **or** COLLISION(s_{new}) = `true` **then break** for loop
    $v_{\text{new}} \leftarrow$ ADDNEWVERTEX($\mathcal{T}, s_{\text{new}}, a, v$)
    if $||\text{POS}(s_{\text{new}}) - p_{j+1}|| \le d_{\text{reach}}$ **then**
      mark p_{j+1} as reached
      reached $\leftarrow$ `true`
      failed $\leftarrow 0$
      if $\Gamma_{j+1} \notin \Gamma$ **then** create Γ_{j+1} and add it to Γ
      add v_{new} to Γ_{j+1}
    else
      add v_{new} to Γ_j
      $v \leftarrow v_{\text{new}}$
  if reached = `false` **then**
    INCREASE(COST$_{\mathcal{D}}(\mathcal{X}, \mathcal{X}_{j+1})$)
    failed $\leftarrow$ failed + 1
  if several failures **then return** `no_progress`
return `progress`

Algorithm 1: Pseudocode for the proposed approach

probabilistic completeness of the approach and to avoid becoming trapped when expansions from a region repeatedly fail due to constraints imposed by the obstacles and the robot dynamics.

4.3.2. Route following. After selecting $\mathcal{R}_i \in \mathcal{R}$, the approach seeks to expand $\mathcal{T}$ so that it follows ROUTE$_{\mathcal{D}}(\mathcal{R}_i)$. Let $\langle \mathcal{X}_1, \ldots, \mathcal{X}_r \rangle$ denote the sequence of regions defined by ROUTE$_{\mathcal{D}}(\mathcal{R}_i)$ (with $\mathcal{X}_1 = \mathcal{R}_i$ and $\mathcal{X}_r = \mathcal{G}$). First, ROUTE$_{\mathcal{D}}(\mathcal{R}_i)$ is converted into a sequence of waypoints, denoted by $\rho = \langle p_1, \ldots, p_r \rangle$, where $p_j =$ CENTROID($\mathcal{X}_j$). Let POLYLINE(ρ) denote the polyline obtained by connecting the waypoints with segments. While following ROUTE$_{\mathcal{D}}(\mathcal{R}_i)$, the motion-tree expansion is allowed to deviate at most a distance d_{follow} away from POLYLINE(ρ), where d_{follow} is a user-defined parameter. This gives the planner some flexibility since exact following may not be possible due to constraints imposed by the obstacles and the dynamics.

Route following seeks to reach the waypoints in succession. A node $v \in V_{\mathcal{T}}$ is considered to have reached the waypoint p_j if $||\text{POS}(\text{STATE}(v)) - p_j|| \le d_{\text{reach}}$, where $0 < d_{\text{reach}} \le d_{\text{follow}}$ is a user-defined parameter. The purpose of this parameter is again to provide flexibility since reaching p_j exactly may

https://doi.org/10.1017/S0263574718000164 Published online by Cambridge University Press

not be possible due to the constraints imposed by the obstacles and the dynamics. Experimental results with various settings of these parameters are discussed in Section 5.6 and illustrated in Fig. 11.

Route following starts by creating a set Γ_1 which contains all the nodes in $\mathcal{T}$ that have reached $\mathcal{X}_1$; i.e., $\Gamma_1 = \text{NODES}_{\mathcal{T}}(\mathcal{X}_1)$. The procedure for reaching the waypoint p_{j+1} from Γ_j is as follows. A target point, p_{target}, is sampled near p_{j+1} (Algorithm 1-FOLLOW:9). Specifically, p_{target} is generated with probability b (set to 0.1 in the experiments) uniformly at random inside $\text{CIRCLE}(p_{j+1}, d_{\text{reach}})$ and with probability $1 - b$ inside $\text{CIRCLE}(p_{j+1}, d_{\text{follow}})$. Sampling p_{target} near p_{j+1} enables the planner to generate different trajectories when attempting to reach p_{j+1} from v and to reach other points near p_{j+1} from which it may be easier to reach p_{j+1}. The bias b provides the greedy aspect where the planner seeks to extend a motion-tree branch from $\text{STATE}(v)$ to a point that is within d_{reach} distance from p_{j+1}.

A collision-free and dynamically feasible trajectory is expanded from $\text{STATE}(v)$ toward p_{target} by applying controls and integrating the motion equations f for several steps. The maximum number of steps is set to $2||\text{POS}(\text{STATE}(v)) - p_{\text{target}}||/(dt * \text{SPEED}(\text{STATE}(v)))$ to reflect the distance that should be traveled and the current speed. The multiplication factor 2 is used to allow more steps in case the robot moves more slowly than expected. A proportional-integral-derivative (PID) controller is used to select controls that steer the mobile robot from $\text{STATE}(v)$ toward p_{target} (Algorithm 1-FOLLOW:12). For a car or a snake model, a PID controller selects controls that turn the wheels and then moves straight toward p_{target}. If the new state, i.e., $s_{\text{new}} \leftarrow \text{SIMULATE}(\text{STATE}(v), u, f, dt)$ (Algorithm 1-FOLLOW:13), obtained after each simulation step is in collision, then the trajectory generation stops. It also stops when s_{new} is more than a distance d_{follow} away from $\text{POLYLINE}(\rho)$ since the objective is to follow the route closely (Algorithm 1-FOLLOW:14). In our experiments, we use a value of d_{follow} between 1 and 5, the results for which are illustrated in Fig. 11. If s_{new} is not in collision and s_{new} remains within d_{follow} distance from $\text{POLYLINE}(\rho)$, then a new node v_{new} is added to $\mathcal{T}$ with s_{new} as its state and v as its parent (Algorithm 1-FOLLOW:15). If s_{new} reaches p_{j+1}, then v_{new} is added to Γ_{j+1} and the trajectory generation stops successfully (Algorithm 1-FOLLOW:16–21). Otherwise, v_{new} is added to Γ_j and the motion-tree expansion continues from v_{new}, i.e., $v \leftarrow v_{\text{new}}$ (Algorithm 1-FOLLOW:24).

When expansions from Γ_j toward p_{j+1} fail, it may be necessary to generate new nodes in Γ_j that would allow successful expansions. This can be achieved by going back to Γ_{j-1} and reaching Γ_j again. For this reason, the route following maintains a set $\Gamma = \{\Gamma_1, \ldots, \Gamma_k\}$, where k is the index in the sequence ρ of the last waypoint that has been reached. A weight is defined for each Γ_j as

$$w(\Gamma_j) = \gamma^{j/|\rho|} \beta^{\text{SEL}(\Gamma_j)}, \tag{11}$$

where $\gamma > 1$. In this way, $w(\Gamma_j)$ is large when j is close to the end of the sequence. As before, $\beta^{\text{SEL}(\Gamma_j)}$ is a penalty factor to ensure that Γ_j will not be selected indefinitely.

Route following proceeds iteratively by selecting Γ_j with the maximum weight from Γ and expanding toward p_{j+1}. This procedure gives priority to expansions toward the end of the sequence. When these expansions fail, the procedure allows for expansions from earlier parts, which is necessary to ensure progression.

4.3.3. Updating routes. When the route expansion fails to progress from Γ_j to p_{j+1}, $\text{COST}_{\mathcal{D}}(\mathcal{X}_j, \mathcal{X}_{j+1})$ is increased (multiplied by 1.1 in the experiments). If the failures persist, the current route is abandoned, and $\text{ROUTE}_{\mathcal{D}}(\mathcal{R}_i)$ is recomputed for each $\mathcal{R}_i \in \mathcal{R}$, as described in Section 4.2. As a result of the cost increases, edges along which the expansion failed are less likely to be part of the recomputed routes. This allows the approach to follow alternative routes when progress along the current route becomes difficult.

4.4. Runtime analysis

The tessellation is computed in $O(n \log n)$ time when using a triangulation,[40] where n denotes the number of triangles, and in $O(1)$ time when imposing a grid. The tessellation graph is sparse with $|E| \in O(n)$ since each triangle or grid cell has only a constant number of neighbors. As a result, region clearances are computed in $O(n)$ time when using the brushfire search. The minimum-cost high-clearance routes over $\mathcal{D}$ are computed in $O(|E| \log n) \in O(n \log n)$ time.

Motion planning (Algorithm 1:6–11) is dominated by calls to $\text{FOLLOW}(\text{ROUTE}_{\mathcal{D}}(\mathcal{R}_i))$. Let n_{MP} denote the number of times the while loop in FOLLOW (line 5) is entered over the entire execution of

https://doi.org/10.1017/S0263574718000164 Published online by Cambridge University Press

the overall algorithm. When the while loop is entered for the ith time, let n_i denote the number of nodes that are added to the motion tree as a result of executing lines 8–18 in FOLLOW. This means that SIMULATE and COLLISION were called at most $n_i + 1$ times since the last call could have resulted in an invalid state that was not added to $\mathcal{T}$. Counting all the nodes that are added to the motion tree yields $|V_{\mathcal{T}}|$. Counting the calls that resulted in an invalid state is at most n_{MP}. Therefore, the total number of times SIMULATE and COLLISION are called is at most $n_{\text{MP}} + |V_{\mathcal{T}}|$. SIMULATE runs in $O(d)$ time, where d is the number of state variables, since it uses Runge–Kutta to integrate f. Sweep-and-prune algorithms[25,41] reduce the time for collision checking to $O(n_{\text{polys}} \log n_{\text{polys}})$, where n_{polys} is the total number of vertices for the polygons representing the obstacles and the mobile robot. Putting it all together, the overall runtime is

$$O(n \log n) + O(n_{\text{MP}} + |V_{\mathcal{T}}|)(d + n_{\text{polys}} \log n_{\text{polys}}). \tag{12}$$

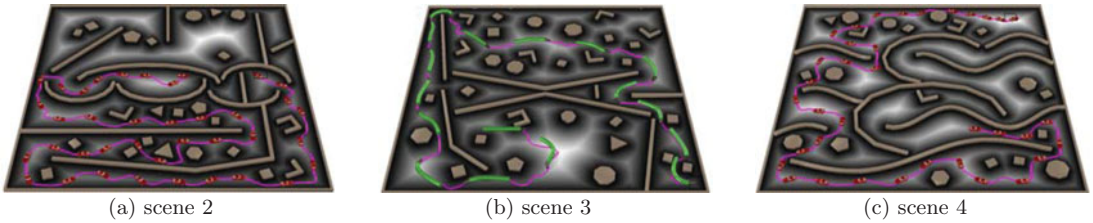
Bounding the number of iterations, n_{MP}, remains an open problem. In fact, motion planning with differential constraints is undecidable.[3] As is the case with other sampling-based motion planners, our approach offers probabilistic completeness, which guarantees that a solution, when it exists, will eventually be found. The probabilistic completeness follows from the analysis in ref. [35] which can be applied to sampling-based motion planners that use tessellations. Specifically, it suffices to show that the motion tree will be expanded infinitely often from each region (when the algorithm runs infinitely). That is, for any region $\mathcal{R}_i \in \mathcal{R}$ and any iteration count j there exists an integer $k > j$ such that SELECTREGION($\mathcal{R}$) selects $\mathcal{R}_i$ during the kth iteration. The iteration count refers to the number of times SELECTREGION($\mathcal{R}$) has been invoked since the beginning of the execution. In our approach, this lemma is guaranteed by the weight associated with each region (Eq. (10)) and the penalty factor β applied to the weight after each selection. Suppose to the contrary that $\mathcal{R}_i$ is never selected after the jth iteration. Since the weight of the region being selected decreases by a factor of β after each selection, its weight will become less than $w(\mathcal{R}_i)$ after being selected a certain number of times. Since one region must be selected during each iteration, after a finite number of iterations, the weight of every other region will become less than $w(\mathcal{R}_i)$. Therefore, $\mathcal{R}_i$ must be selected. This shows that our approach guarantees that each region will be selected infinitely often for expansion, so the probabilistic completeness follows. Also, note that when our approach finds a solution, it is guaranteed to be collision-free and dynamically feasible since only collision-free and dynamically feasible trajectories are added as branches to the motion tree $\mathcal{T}$.

5. Experiments and Results

Experiments are conducted in simulation using complex environments, as shown in Figs. 1 and 6, where a mobile robot, modeled as a car and (separately) a snake (Section 3), has to pass through several narrow passages and navigate amidst numerous obstacles in order to reach the goal. Navigation is made even more challenging by the differential constraints imposed by the dynamics of the mobile robot. Experiments are also conducted by varying the obstacle density (Section 5.9). Specifically, we created three series of environments: wavelines, random obstacles, and mazes, and varied the number of wavelines, percentage of area covered by random obstacles, and maze dimensions. Figure 16 provides some examples. Experiments also evaluate the impact of the obstacle-clearance heuristic, clearance computation (mesh distance or brushfire), tessellation granularity, and type (triangulation or grid) on the overall performance of the approach.

The approach is compared to several state-of-the-art motion planners, namely `RRT`,[28] `MARRT`,[11] `fRRT`,[22] `KPIECE`,[8] `GUST`,[35] and a modified version of `GUST` that uses the clearance-based cost (Eq. (7)) for the edges of the tessellation instead of the Euclidean distance. The `RRT` implementation uses the connect version with goal bias as recommended in the literature.[26,27] `MARRT` uses bisection to move sampled points toward the medial axis. `fRRT` is an informed `RRT` variant that uses costs as in A* to bias sampling toward low-cost regions. The implementation of `fRRT` follows the details provided in the technical report.[22] For `RRT` and its variants, efficient data structures are used for the computation of the nearest neighbors.[4,30] The implementation of `KPIECE`[8] follows the OMPL library.[9] The implementation of `GUST` has all the features enabled, including abandoning shortest paths when

https://doi.org/10.1017/S0263574718000164 Published online by Cambridge University Press



(a) scene 2 (b) scene 3 (c) scene 4

Fig. 6. Environments used for the experiments (scene 1 is shown in Fig. 1).

no progress is made. We also modified GUST to use our clearance-based cost function for the edges in the tessellation (Eq. (7)).

5.1. Environments and problem instances

Due to the probabilistic nature of sampling-based motion planning, the performance is measured using 60 different problem instances for each scene. Each instance is generated by placing the robot and the goal at random positions near the locations shown in Figs. 1 and 6. Each scene has several narrow passages, each sufficiently wide for the robot to pass through. The short paths to the goal are more difficult to follow as they pass through the narrower passages.

The performance of a method for a given scene and robot model is measured by running it on each of the 60 problem instances. Results report the mean runtime and several metrics on the solution trajectory, namely its length, and average and minimum clearance from the obstacles. The mean is calculated after dropping the five best and worst results to avoid the influence of outliers. The metrics for a trajectory $\zeta : \{1, \ldots, \ell\} \rightarrow \mathcal{S}$ are defined as

$$\text{DIST}(\zeta) = \sum_{i=1}^{\ell-1} ||\text{POS}(\zeta(i)) - \text{POS}(\zeta(i+1))|| \tag{13}$$

$$\text{MEANCLEAR}(\zeta) = \frac{1}{\ell} \sum_{i=1}^{\ell} \text{CLEAR}(\text{POS}(\zeta(i))) \tag{14}$$

$$\text{MINCLEAR}(\zeta) = \min \bigcup_{i=1}^{\ell} \{\text{CLEAR}(\text{POS}(\zeta(i)))\}. \tag{15}$$

The experiments were run on an Intel Core i7 (CPU: 1.7GHz, RAM: 4GB) using Ubuntu 15.10 and g++-4.8.4.

Unless otherwise indicated, the following default parameter values are used in the experiments: $\alpha = 6$ (Eq. (7)); $\beta = 0.95$ (Eq. (10)); $d_{\text{reach}} = 1.6 \times \text{robotWidth}$, $d_{\text{follow}} = 3 \times \text{robotWidth}$ (Section 4.3.2), where robotWidth is the width of the mobile robot; workspace dimensions: 80×80 units; car dimensions: 1.5×3.4; snake link dimensions: 1.15×1.7; number of snake links: six; default tessellation corresponds to a triangulation obtained by Triangle[40] when setting the maximum triangle area to $1/32\%$ of the overall workspace area.

5.2. Runtime comparisons

The results in Fig. 7 show that our approach is significantly faster, by one to two orders of magnitude, than RRT, MARRT, fRRT, KPIECE, GUST, and modified GUST. In RRT, the expansion from the nearest vertex to a random sample often leads to areas blocked by obstacles, making it difficult to reach the goal. This is more pronounced in obstacle-rich environments characterized by multiple narrow passages, such as those used in the experiments, causing RRT to time out. MARRT seeks to push each intermediate state toward the medial axis. Although such expansion avoids cluttering near obstacles, it still has difficulty quickly expanding toward the goal. Consider, for example, Scene 3 shown in Fig. 6. The expansion in MARRT will clutter around the medial axis of the area confined by the curved objects. Due to the random sampling, the probability of expanding along the narrow passages is quite low. This causes MARRT to time out, since a solution to the problem instances used in the experiments

https://doi.org/10.1017/S0263574718000164 Published online by Cambridge University Press

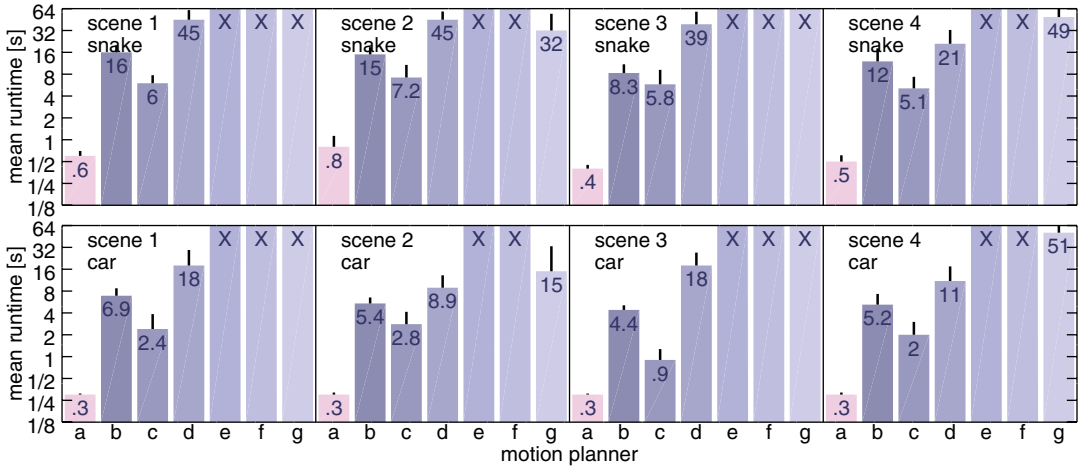


Fig. 7. Runtime comparisons: (a) our approach (b) GUST[35] (c) GUST modified to use our clearance-based edge cost, as defined in Eq. (7) (d) KPIECE[8] (e) RRT[28] (f) MARRT[11] (g) fRRT.[22] The obstacle-clearance exponent α for the clearance-based edge cost (Eq. (7)) was set to six for both our approach and modified GUST. Entries marked with X denote failure to solve the problem instances (64 s for each run). Due to significant differences in runtime, logscale is used for the y-axis with the label showing the actual value rather than its logarithm. The vertical segment on top of each bar indicates the standard deviation. Runtime includes everything, from reading the input file to reporting that a solution is found. For clarity, the runtime numbers shown in the graph are rounded to the nearest integers for values larger than 10 s and to one decimal point for values less than 10 s.

can be obtained only by passing through several specific passages. fRRT, as an informed version of RRT, improves upon RRT by biasing the expansion more toward the goal. Even though fRRT is able to solve some problems where RRT timed out, fRRT still suffers from the use of the nearest-neighbor heuristic, which often leads to failed expansions from areas blocked by obstacles. KPIECE covers the space uniformly but lacks direction toward the goal, so often requires significant runtime to find a solution.

GUST uses shortest paths in the tessellation to guide the expansion. This usually results in considerable improvements over other motion planners. In obstacle-rich environments, however, the shortest paths tend to guide the expansion closer to obstacles. Moreover, following the shortest paths in our environments requires the mobile robot to pass through the narrower passages, making it more difficult to find a solution. GUST eventually abandons the shortest paths but such explorations considerably increase the runtime.

In contrast, our approach uses high-clearance routes to guide the motion-tree expansion. Following such routes facilitates the expansion since the robot has more room to maneuver. This results in speedups of one to two orders of magnitude while also significantly improving the clearance as discussed in the following section.

To further test the importance of the route following in our approach, we modified GUST to use the clearance-based cost (Eq. (7)) instead of the Euclidean distance for the edges of the tessellation. This allows GUST to prefer expansions from regions associated with higher clearance. The modified GUST was faster than the original GUST, but still about an order of magnitude slower than our approach. These results demonstrate that the efficiency of our approach derives not only from using a clearance-based cost but more critically from being able to quickly follow high-clearance routes and adapt the routes based on the difficulties that it encounters during the motion-tree expansion.

5.3. Clearance and solution-length comparisons

Figure 8 summarizes the results on various metrics related to the solution trajectory such as its length and clearance from the obstacles. The figure shows the effect size, which provides a statistical measure on the magnitude of the differences of the solution length and clearance when comparing our approach to other motion planners. The effect size is measured as

$$\text{Hedges'}g = (\bar{\mu}_1 - \bar{\mu}_2)/\sigma_p, \tag{16}$$

https://doi.org/10.1017/S0263574718000164 Published online by Cambridge University Press

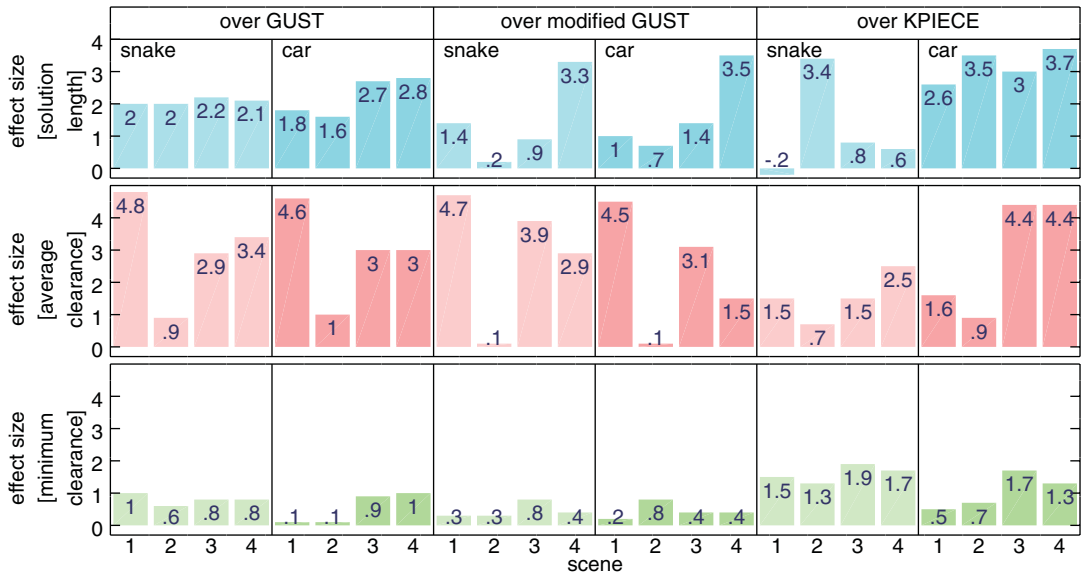


Fig. 8. Statistical effect size on the solution length, average clearance, and minimum clearance (Section 5.1, Eqs. (13)–(15)) of our approach over the other planners measured as Hedges' g (Eq. (16)). For the solution length, a positive value indicates that our approach found shorter solutions than GUST, modified GUST, or KPIECE. For the average and minimum clearances, a positive value indicates that our approach found solutions with larger clearances. Effect sizes in $[-0.2, 0.2]$ are typically classified as "small," around 0.5 as "medium," and above 0.8 as "large."[7,38]

where $\bar{\mu}_1$ and $\bar{\mu}_2$ denote the mean of the two datasets and σ_p denotes the pooled standard deviation.[38] Intuitively, the effect size expresses how many standard deviations separate the two means.

Figure 8 shows that our approach offers statistically significant improvements with large effect sizes. By using high-clearance routes to guide the motion-tree expansion, the approach is able to generate solution trajectories with high clearance. The approach is also able to generate shorter solutions than GUST, modified GUST, or KPIECE (RRT, MARRT, and fRRT were not used in the comparisons since RRT and MARRT timed out, while fRRT was able to solve only some of the problem instances). KPIECE relies on a uniform random exploration, which generally leads to long solutions. Even though GUST uses shortest paths as guide, due to the characteristics of the environments used in the experiments, GUST is unable to find solutions along those shortest paths. Eventually GUST abandons the shortest paths in favor of more random explorations, which enable it to discover alternative paths to the goal at the expense of increasing the solution length. Modified GUST expands more from regions with higher clearance. As the robot has more room to maneuver, modified GUST is less likely to abandon the guide in favor of more random explorations. As a result, modified GUST ends up producing solutions shorter than or comparable in length to GUST. However, our approach is able to produce even shorter solutions since it avoids random expansions by closely following high-clearance routes.

Even though our approach found shorter solutions than GUST, modified GUST, or KPIECE, they are still significantly longer than the trajectories that pass through the narrower passages. In fact, as shown in Figs. 1 and 6, the solutions generated by our approach take the long routes to the goal since such routes increase the clearance from the obstacles.

5.4. Impact of the high-clearance routes

Figure 9 shows the results when varying the obstacle-clearance exponent (α) used in the definition of the cost for the tessellation edges (Eq. (7)). When the exponent has a high value, the motion-tree expansion is guided by routes with high clearance. A value of zero removes the clearance from consideration and, in effect, uses shortest paths to guide the motion-tree expansion. This increases the runtime since it is more difficult to follow the shortest routes than the high-clearance routes. The improved runtime over GUST even when the exponent is zero (compare Figs. 7 and 9) is due to the other improvements offered by our approach such as the route-following component. As the value of the exponent α is increased, the runtime of our approach improves significantly.

https://doi.org/10.1017/S0263574718000164 Published online by Cambridge University Press

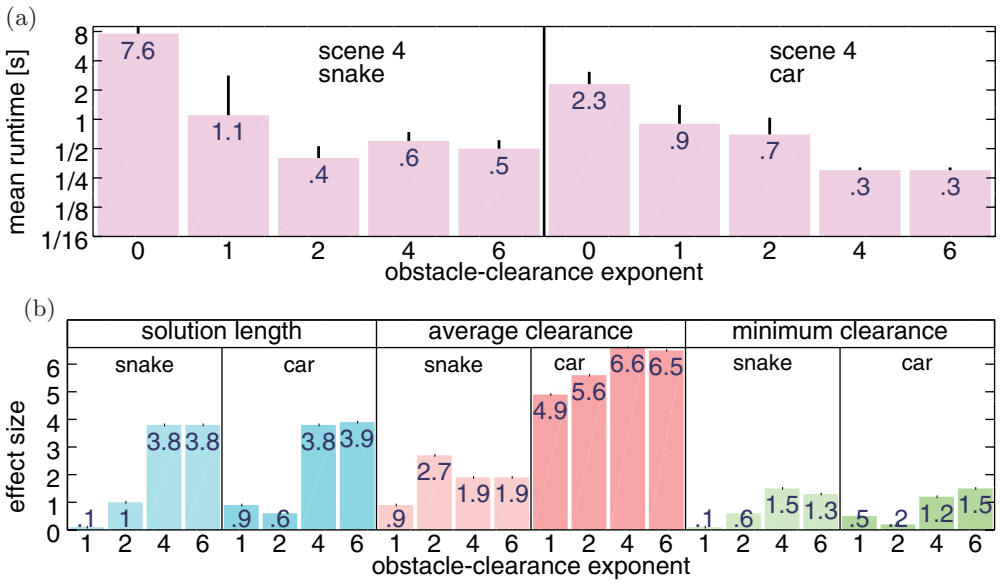


Fig. 9. (a) Runtime results and (b) Statistical effect size, measured as Hedge's g (Eq. (16)), on the solution length, average clearance, and minimum clearance when varying the obstacle-clearance exponent α in the tessellation edge cost (Eq. (7)). Runtime includes everything, from reading the input file to reporting that a solution is found. (b) The effect sizes are calculated with respect to the case when the exponent is zero. For the solution length, when the exponent has the value i, a positive value for the effect size indicates that the approach found longer solutions than when the exponent was zero. For the average and minimum clearances, a positive value indicates that the approach found solutions with higher clearances when using i than zero. Results are shown for Scene 4. Similar results are obtained for the other scenes.

Figure 9 also shows the effect size on the solution length, average clearance, and minimum clearance. Setting $\alpha = 0$ leads to shorter solution trajectories but with significantly smaller clearances. As α increases, the solutions become longer and the clearances become higher.

5.5. Impact of the tessellation and method to compute clearances

Figure 10 shows the results when using triangulations or grid-based tessellations. The results indicate that the approach works well for both types. Comparisons when varying the granularity of the tessellation show that the approach works well for a wide range. A coarse-grained tessellation is fast to compute and search over but cannot be directly used to obtain reliable estimates of clearance. A coarse-grained tessellation results in poor approximations of the medial axis. As a result, the runtime of the overall approach increases since the motion-tree expansion is not led effectively along high-clearance routes. As the tessellation becomes more fine-grained, it leads to better approximations of the medial axis but increases the graph-search time used for the computation of the high-clearance routes. Our recommendation is to use tessellations that are neither too fine-grained nor too coarse-grained. Since our approach works well for a wide range of granularities, it makes it easier to find a tessellation that will work well when running our approach on a new problem.

Experiments were also conducted by varying the method used to compute the region clearances. The results indicate that the overall approach works well in both cases; i.e., whether using the mesh-distance computation or the brushfire search. The brushfire search is better suited for tessellations where the regions have similar sizes. The mesh-distance approach has a higher preprocessing cost but provides more accurate approximations and can be used with nonuniform tessellations. When the tessellation is fine-grained, the overall approach is faster when using the brushfire search. As the granularity increases, the overall approach becomes faster when using the mesh-distance computations.

5.6. Impact of other parameters

As shown in Fig. 11, the approach works well for a wide range of parameter values. Recall that the weight of a region is reduced by a factor of β (Section 4.3.1) each time it is selected for expansion in order to avoid potentially selecting the same region indefinitely. As β increases, the region selection

https://doi.org/10.1017/S0263574718000164 Published online by Cambridge University Press

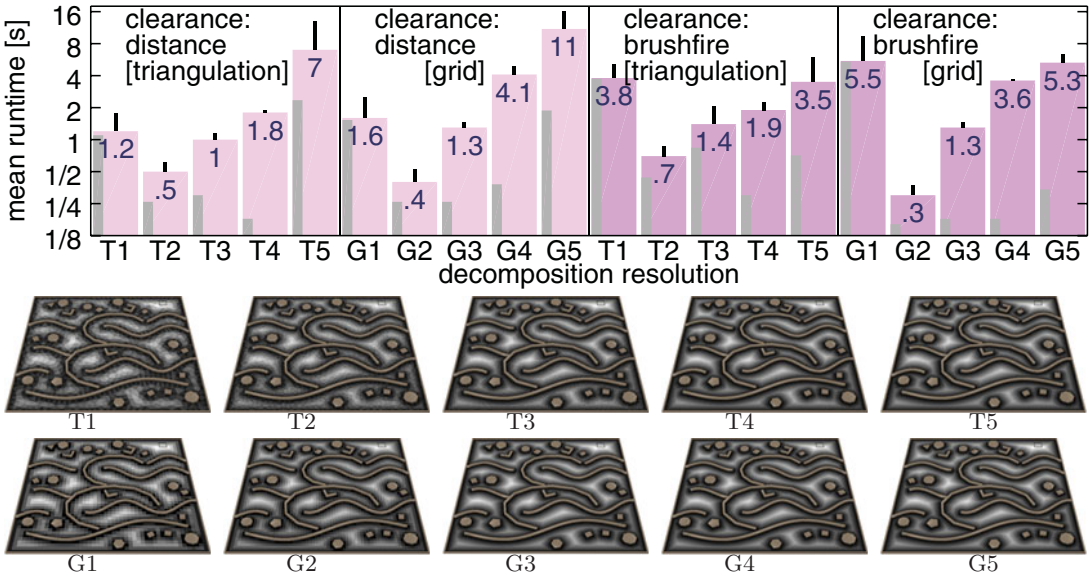


Fig. 10. Runtime results when varying the resolution and the type of the tessellation. The triangulations T1, T2, T3, T4, T5 are obtained by setting the maximum triangle area in the triangulation to 1/16%, 1/32%, 1/64%, 1/128%, 1/256%, respectively, of the overall workspace area. The grid tessellations G1, G2, G3, G4, G5 are obtained by setting the grid resolution to 48×48, 64×64, 128×128, 192×192, 256×256, respectively. Results are shown for both the distance and the brushfire method of computing the region clearances from the obstacles. The thin gray bar on the left side indicates the mean preprocessing time (Algorithm 1:1–4), which corresponds to creating the tessellation and computing the high-clearance routes. Note that the overall time reported by the wide bars includes the preprocessing time; in fact, it includes everything from reading the input file to reporting that a solution is found (Algorithm 1:1–12). Results are shown for Scene 4 and the snake model. Similar results are obtained for the other scenes as well as the car model.

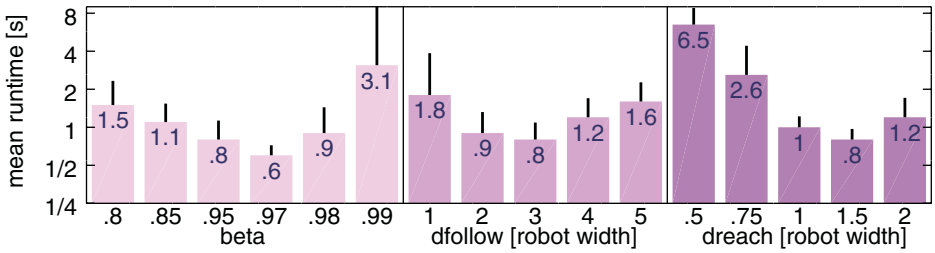


Fig. 11. Runtime results when varying (a) β : selection penalty (Section 4.3.1) (b) d_{follow}: route width (Section 4.3.2) (c) d_{reach}: waypoint radius (Section 4.3.2). The parameters d_{follow} and d_{reach} are expressed in terms of the robot's width. Results are shown for Scene 2 and the snake model. Similar results are obtained for the other scenes as well as the car model.

becomes greedier. This often reduces the runtime, but when β approaches 1, the approach may be caught in a cycle in which it repeatedly expands a single region. We recommend setting $\beta = 0.95$ initially when considering a new problem and using the interval $\beta \in [0.85, 0.96]$ for fine tuning.

Figure 11 also shows the results when varying the parameters d_{follow} and d_{reach} that determine the closeness with which the tessellation route from the selected region is followed (Section 4.3.2). Setting these parameters to small values increases the runtime of the approach since, due to dynamics and obstacles, it becomes more difficult for the robot to follow the selected route. Large values are likely to cause considerable deviations from the selected route, which make the motion-tree expansion more difficult and time consuming. The approach, however, as shown in Fig. 11, works well for a wide range of values. When working with a new problem, our recommendation is to set d_{follow} and d_{reach} about 2–5 times and 1–2 times the width of the mobile robot, respectively.

https://doi.org/10.1017/S0263574718000164 Published online by Cambridge University Press

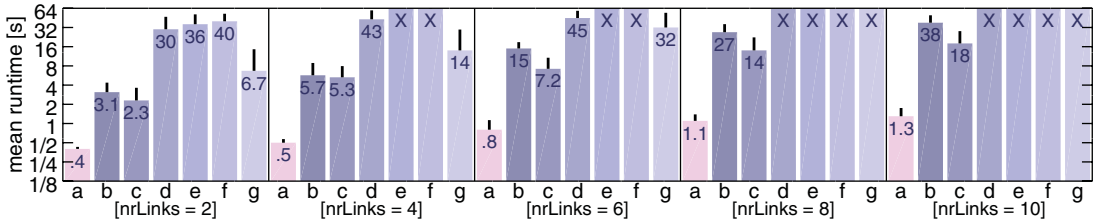


Fig. 12. Runtime results for Scene 2 when varying the number of snake links: (a) our approach (b) GUST[35] (c) GUST modified to use our clearance-based edge cost, as defined in Eq. (7) (d) KPIECE[8] (e) RRT[28] (f) MARRT[11] (g) fRRT.[22] Similar results are obtained for the other scenes.

5.7. Impact of the state dimension

Figure 12 shows the runtime results as a function of the number of snake links. Each additional link increases the state dimension by one, so the experiments test the impact of the state dimensionality. The results show that, as expected, the runtime increases with the number of links. For our approach, the increase in runtime is moderate, solving even problems with 10 snake links (state dimension is 15) in 1 s. This is due to the efficacy of using high-clearance routes to guide the motion-tree expansion. The results show that our approach remains significantly faster (by one to two orders of magnitude) than GUST, modified GUST, KPIECE, RRT, MARRT, and fRRT.

5.8. Impact of the tessellation edge cost

We also ran experiments using a more robust edge-cost definition based on the Geman–McLure M-estimator.[16] Specifically, we changed the edge cost in Eq. (7) to

$$\text{ROBUST-COST}_{\mathcal{D}}(\mathcal{R}_i, \mathcal{R}_j) = \frac{||\text{CENTROID}(\mathcal{R}_i) - \text{CENTROID}(\mathcal{R}_j)||}{(\min\{\text{gm}(\mathcal{R}_i), \text{gm}(\mathcal{R}_j)\})^{\alpha}}, \tag{17}$$

where

$$\text{gm}(\mathcal{R}_i) = \frac{(\text{CLEAR}(\mathcal{R}_i))^2}{\sigma^2 + (\text{CLEAR}(\mathcal{R}_i))^2}. \tag{18}$$

The parameter σ expresses the notion of sufficient clearance from the obstacles. In this way, rather than seeking to generate routes that maximize the clearance from the obstacles, we seek to generate routes that maintain a certain clearance from the obstacles, yielding diminishing returns for farther distances. Such routes will still facilitate motion planning while potentially leading to shorter solutions. Figure 13 provides some examples.

The results in Fig. 14 show that the runtime increases when σ is small since the routes tend to get close to obstacles and even pass through narrow passages that could have been avoided by following longer routes with higher clearance. When σ is large, the approach works well, completing in runtimes similar to those obtained when Geman–McLure is not used. The benefit, however, of using the Geman–McLure edge cost is that it tends to generate shorter routes with sufficient clearance. This enables the approach to obtain shorter solutions, as shown by the results in Fig. 14.

Note that α serves a different purpose than σ. As discussed in Sections 4.2 and 5.4, setting α to large values leads to high-clearance routes that avoid short paths that go through narrow passages, as is the case for the scenes used in our experiments. Such routes cannot be obtained by keeping α fixed and just changing σ. In fact, for a fixed value of α, varying σ may not be sufficient to force the minimum-cost path to be substantially different from the shortest path, as Fig. 15 shows. Therefore, when considering a new problem, our recommendation is to set α to a large value (so that path clearance has precedence over length) and then set $\sigma \in [1 \times \text{robotWidth}, 3 \times \text{robotWidth}]$.

5.9. Impact of the density of obstacles

Experiments were also conducted by varying the obstacle density. Specifically, we created three series of environments: wavelines, random obstacles, and mazes. For each series, we varied the number of wavelines, percentage of area covered by random obstacles, and maze dimensions.

https://doi.org/10.1017/S0263574718000164 Published online by Cambridge University Press

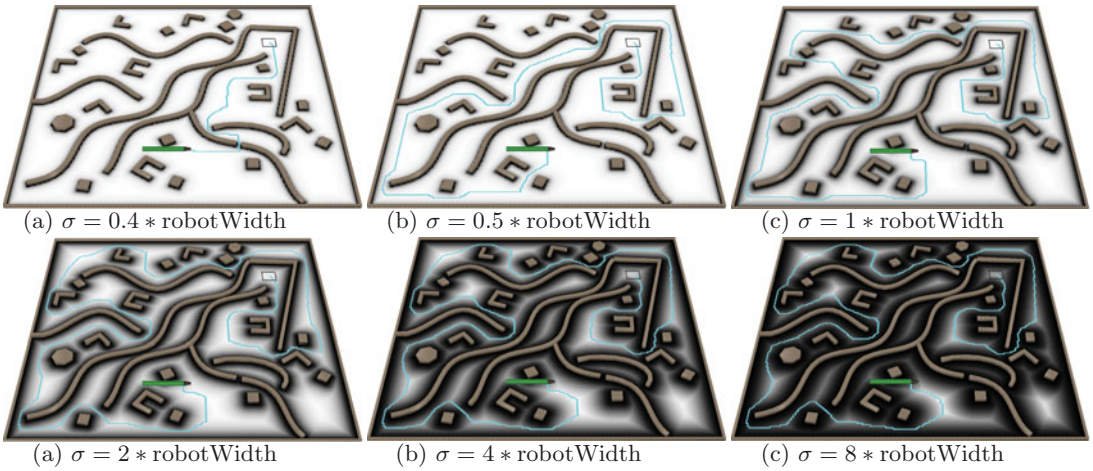


Fig. 13. Main route when using the modified edge cost based on the Geman–McLure M-estimator (Eq. (17)) for various values of σ, expressed in terms of the width of the robot . Note that the white area (indicating more maneuverability) increases as σ becomes smaller, since a smaller clearance is desired. See also Fig. 5 for the main route when using the original edge cost defined in Eq. (7).

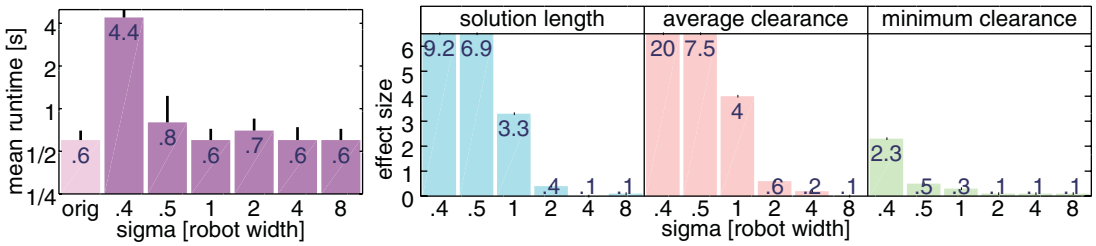


Fig. 14. Runtime results and statistical effect size, measured as Hedges' g (Eq. (16)), on the solution length, average clearance, and minimum clearance (Section 5.1, Eqs. (13)–(15)) when varying σ in the definition of the robust edge cost based on the Geman–McLure M-estimator (Eq. (17)). In the runtime graph, the bar labeled "orig" refers to our approach using the original edge cost, as defined in Eq. (7). In the effect-size graph, a positive value for the solution length indicates that our approach found shorter solutions when using the edge cost based on the Geman–McLure M-estimator (Eq. (17)) Geman–McLure based edge cost. For the average and minimum clearances, a positive value indicates that our approach found solutions with larger clearances when using the original edge cost (Eq. (7)). Results are shown for Scene 1 and the snake model. Similar results are obtained for the other scenes as well as the car model.

The first series, as shown in Fig. 16(a), consists of a number of wavelines placed horizontally, each having some gaps to allow the robot to move. To obtain statistically significant results, for a given number of wavelines, we generated 30 scenes. Each scene is constructed by first reserving some space at the bottom and at the top for the initial robot placement and goal placement, and then placing the wavelines equidistantly along the y-axis in the remaining space. Each waveline corresponds to a sinusoidal curve, specifically

$$y + \text{amplitude} \times \sin(\text{angle} \times (x - \text{x}_{\min})/(\text{x}_{\max} - \text{x}_{\min})), \tag{19}$$

where $\text{x}_{\min}$ and $\text{x}_{\max}$ are the bounds along the x-axis. To make the wavelines different from each other, the amplitude and angle are set at random from $[0.5, 3]$ and $[2\pi, 3\pi]$, respectively. A number of gaps, selected at random from $\{2, 3, 4\}$, are added along each waveline. Half the gaps have the size set at random from $[1.15, 2.5]$ and the other half from $[4.0, 6.0]$, so that each waveline has gaps that are difficult to pass through and others that are not as difficult. As the number of wavelines increases, the problem becomes considerably harder since the robot has to wiggle its way through the passages from one waveline to the next.

https://doi.org/10.1017/S0263574718000164 Published online by Cambridge University Press

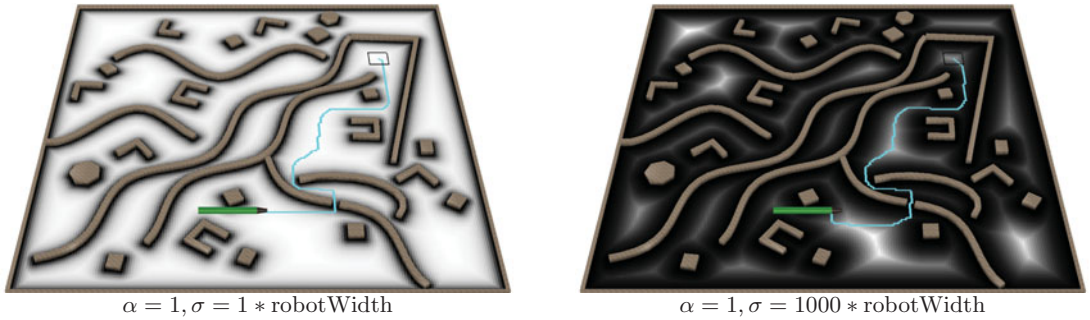


Fig. 15. Main route when keeping α fixed and varying σ in the Geman–McLure robust edge cost (Eq. (17)). For a fixed α, it is not possible to vary σ to make the route avoid the short path that goes through the narrow passage.

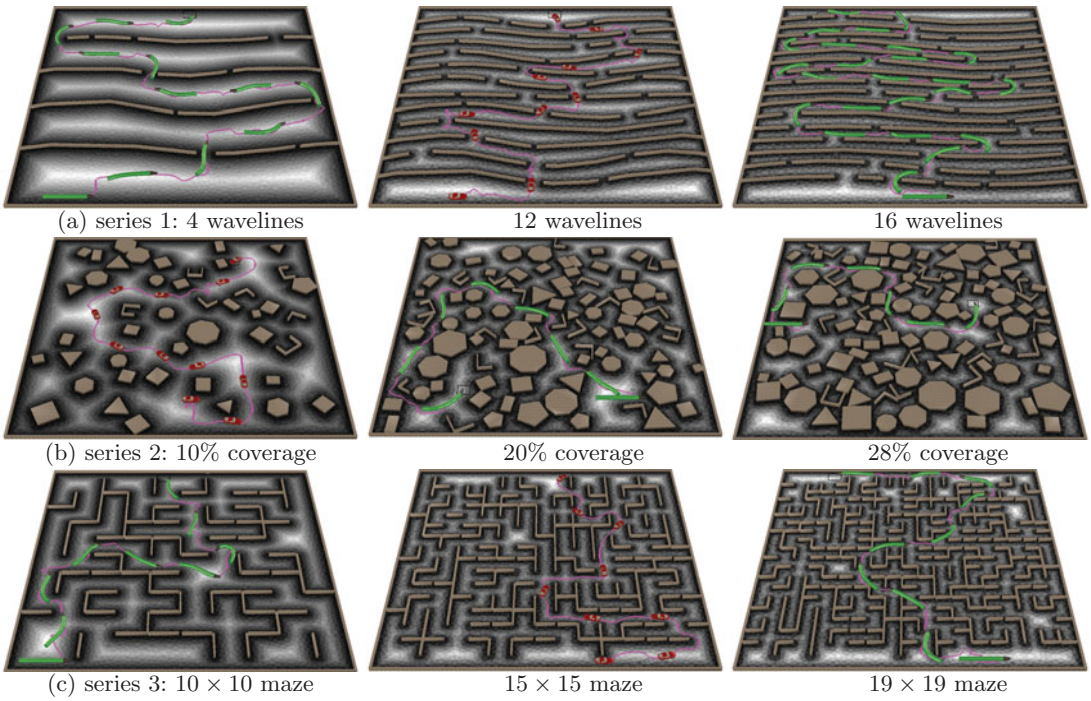


Fig. 16. Examples of scenes from (a) wavelines, (b) random obstacles, and (c) maze series, as described in Section 5.9.

The second series, as shown in Fig. 16(b), consists of a number of obstacles placed randomly throughout the environment. We vary the percentage of the area occupied by the obstacles. For a given percentage, as before, we generate 30 random scenes.

The third series, as shown in Fig. 16(c), consists of maze environments, where we vary the maze dimensions. For a given dimension, 30 different mazes are created using Kruskal's algorithm. For each maze, 5% of the walls, selected at random, are removed in order to allow more than one way to reach the goal.

For the waveline, random obstacles, and maze series, experiments were conducted with up to 16 wavelines, 28% obstacle coverage, and 19 × 19 mazes, respectively. Larger values resulted in scenes where the robot, due to its physical dimensions, did not have enough room to reach the goal.

Figure 17 shows the runtime results when comparing our approach to other sampling-based motion planners over the waveline, random obstacles, and maze series. The results indicate that our approach is significantly faster. As the environments become more cluttered, the other approaches have difficulty finding solutions. By relying on high-clearance routes, our approach is able to quickly find solutions, offering speedups of one to two orders of magnitude over the other motion planners. When the

https://doi.org/10.1017/S0263574718000164 Published online by Cambridge University Press

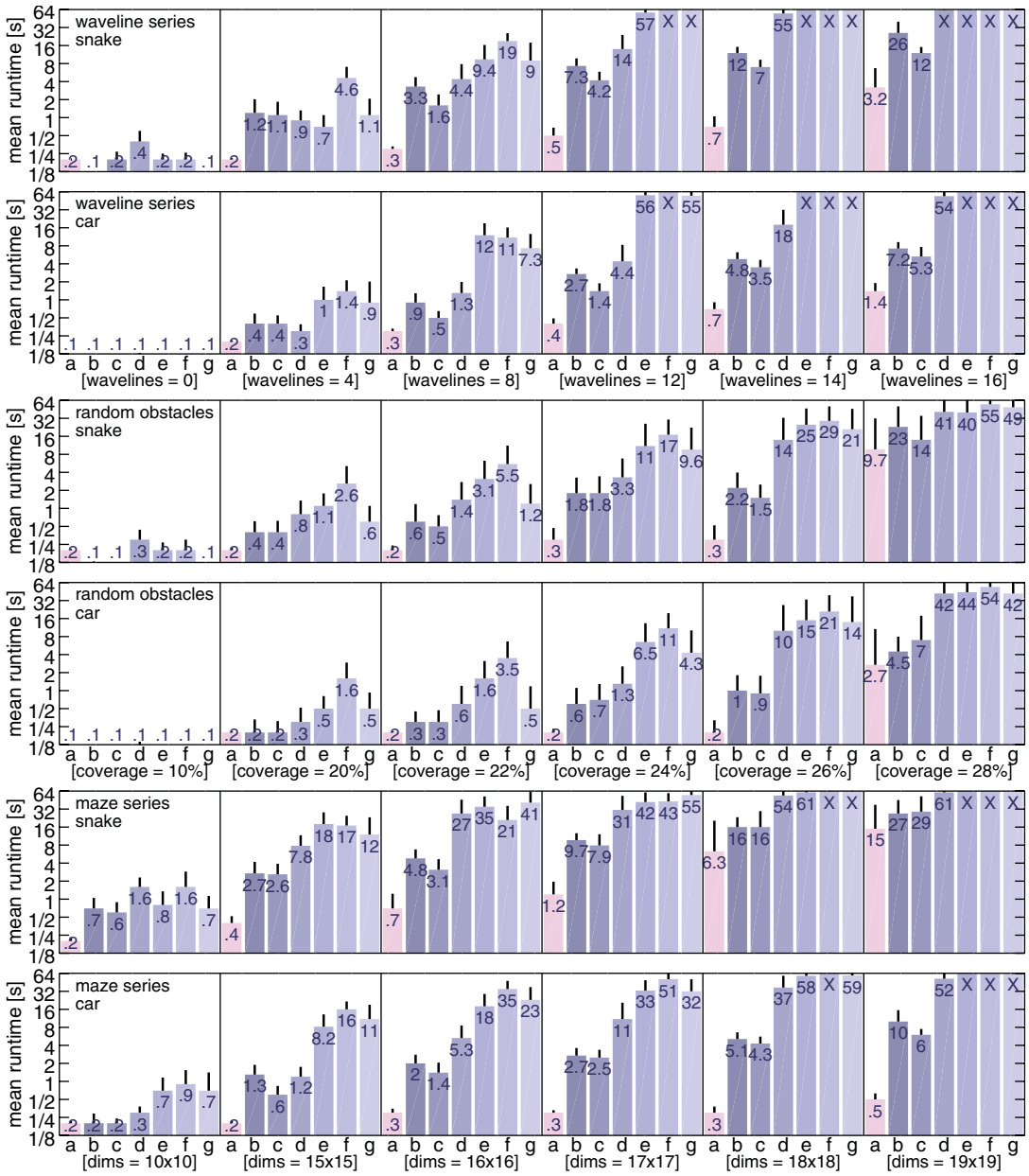


Fig. 17. Runtime results for the waveline, random obstacles, and maze series: (a) our approach (b) GUST[35] (c) GUST modified to use our clearance-based edge cost, as defined in Eq. (7) (d) KPIECE[8] (e) RRT[28] (f) MARRT[11] (g) fRRT.[22] Runtime includes everything, from reading the input file to reporting that a solution is found. Entries marked with X denote failure to solve the problem instances (64 s for each run). The obstacle-clearance exponent α was set to six for both our approach and modified GUST.

environments become too dense, RRT, MARRT, fRRT time out, while KPIECE gets close to the runtime limit. In the most dense environments, even though the benefits of using clearance information are less pronounced since all routes have low clearance, our approach still maintains its competitive advantage over GUST (about 1.6–20 times faster) and over modified GUST (about 1.4–12 times faster).

Figure 18 shows that our approach offers statistically significant improvements with large effect sizes with respect to the clearance from the obstacles of the computed solution trajectories. This results from expanding the motion tree along high-clearance routes. In many cases, our approach also produces shorter solution trajectories. Our approach closely follows the routes, while other planners, including GUST and modified GUST, tend to explore in random directions, which considerably increases the length of the solution trajectory.

https://doi.org/10.1017/S0263574718000164 Published online by Cambridge University Press

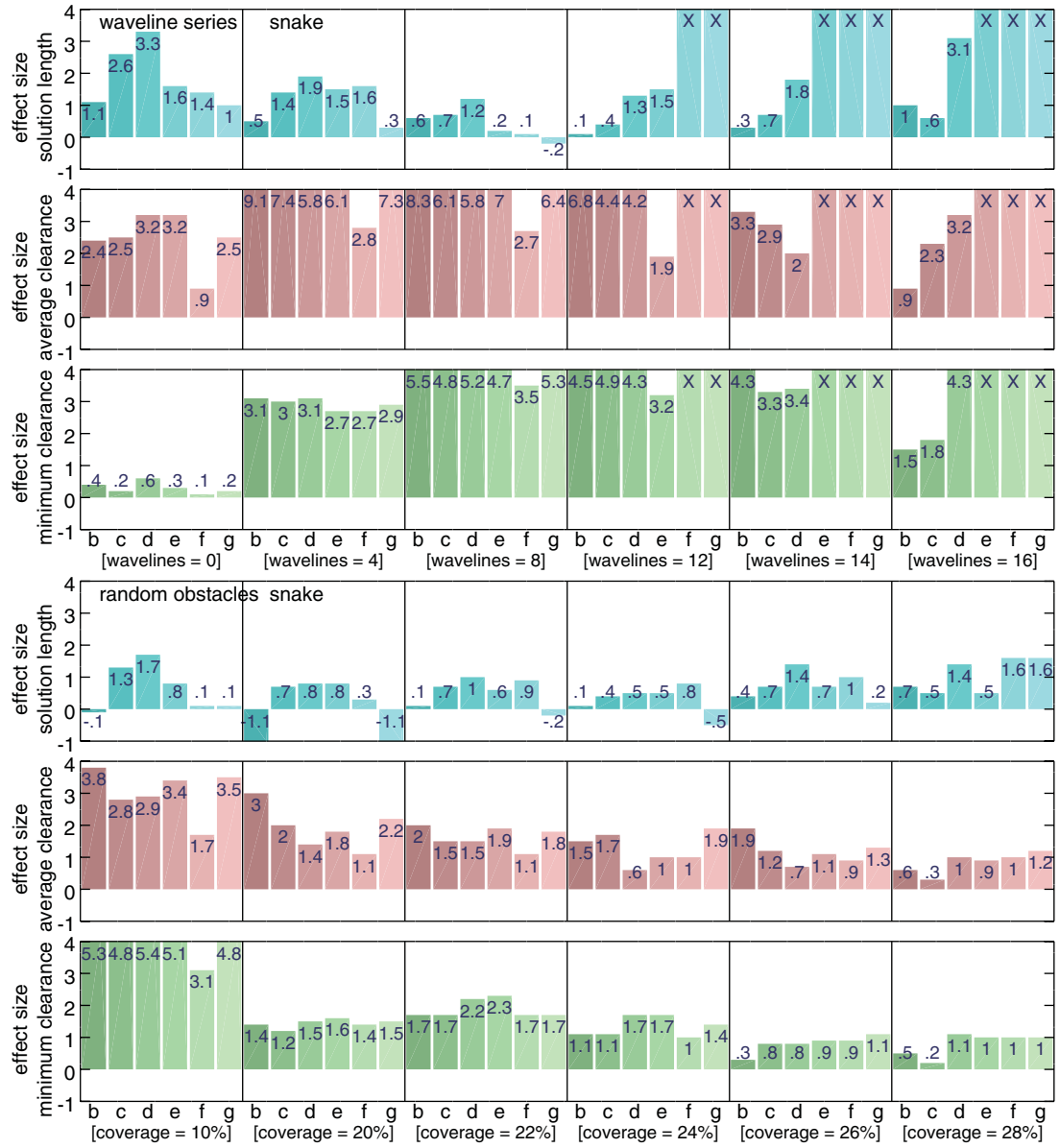


Fig. 18. Statistical effect size, measured as Hedges' *g* (Eq. (16)), on the solution length, average clearance, and minimum clearance (Section 5.1, Eqs. (13)–(15)) of our approach over the other planners: (b) over GUST[35] (c) over modified GUST, which uses our clearance-based edge cost, as defined in Eq. (7) (d) over KPIECE[8] (e) over RRT[28] (f) over MARRT[11] (g) over fRRT.[22] For the solution length, a positive value indicates that our approach found shorter solutions. For the average and minimum clearances, a positive value indicates that our approach found solutions with larger clearances. Entries marked with X denote failure to solve the problem instances. Results are shown for the waveline and random obstacles series with the snake robot model. Similar results were obtained for the maze series as well as the car model.

6. Discussion

This paper presented an approach that combines geometry processing with sampling-based motion planning to enable an autonomous mobile robot to navigate safely and efficiently in complex environments. A key aspect of this work was the use of high-clearance routes over a discrete tessellation of the environment to guide the sampling-based expansion of a motion tree. Experimental results, using a car and snake-like model operating in challenging environments, demonstrate the effectiveness of this approach in producing high-clearance paths, while performing orders of magnitude faster than state-of-the-art methods.

https://doi.org/10.1017/S0263574718000164 Published online by Cambridge University Press

This work opens up several venues for further research. In particular, the proposed approach could benefit from introducing local shape descriptors, in addition to clearance, in order to provide information about narrow passages, hard turns, or cluttered areas, which the motion planner can take into account during the motion-tree expansion. Another direction is to more deeply investigate the trade off between the speed of generating a successful trajectory and the quality of the path. The long-term goal of this research is to rely on geometry processing techniques to extract semantically meaningful information from complex environments in order to facilitate navigation of autonomous robots.

Acknowledgments
The work of Erion Plaku is supported by NSF IIS-1548406.

Supplementary Material
To view supplementary material for this article, please visit https://doi.org/10.1017/S0263574718000164

References

1. N. Amenta, S. Choi and R. K. Kolluri, "The power crust, unions of balls, and the medial axis transform," *Comput. Geom.* **19**, 127–153 (2001).
2. M. de Berg, O. Cheong, M. van Kreveld and M. H. Overmars, *Computational Geometry: Algorithms and Applications* (Springer-Verlag, Santa Clara, CA, 2008).
3. M. S. Branicky, "Universal computation and other capabilities of continuous and hybrid systems," *Theor. Comput. Sci.* **138**(1), 67–100 (1995).
4. S. Brin, "Near Neighbor Search in Large Metric Spaces," *Proceedings of the 21th International Conference on Very Large Data Bases*, Zurich, Switzerland (1995) pp. 574–584.
5. Y. F. Chen, S. Y. Liu, M. Liu, J. Miller and J. P. How, "Motion Planning with Diffusion Maps," *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, Deajeon, South Korea (2016) pp. 1423–1430.
6. H. Choset, K. M. Lynch, S. Hutchinson, G. Kantor, W. Burgard, L. E. Kavraki and S. Thrun, *Principles of Robot Motion: Theory, Algorithms, and Implementations* (MIT Press, Cambridge, MA, 2005).
7. J. Cohen, "A power primer," *Psychol. Bull.* **112**(1), 155 (1992).
8. I. A. Şucan and L.E. Kavraki, "A sampling-based tree planner for systems with complex dynamics," *IEEE Trans. Robot.* **28**, 116–131 (2012).
9. I. A. Şucan, M. Moll and L. E. Kavraki, "The open motion planning library," *IEEE Robot. Autom. Mag.* **19**(4), 72–82 (2012). Available at: http://ompl.kavrakilab.org
10. T. Culver, J. Keyser and D. Manocha, "Exact computation of the medial axis of a polyhedron," *Comput. Aided Geom. Des.* **21**(1), 65–98 (2004).
11. J. Denny, E. Greco, S. Thomas and N. M. Amato, "MARRT: Medial Axis Biased Rapidly-Exploring Random Trees," *Proceedings of the IEEE International Conference on Robotics and Automation*, Hong Kong, China (2014) pp. 90–97.
12. D. Devaurs, T. Simeon and J. Cortés, "Enhancing the Transition-Based RRT to Deal with Complex Cost Spaces," *Proceedings of the IEEE International Conference on Robotics and Automation*, Karlsruhe, Germany (2013) pp. 4120–4125.
13. T. K. Dey and W. Zhao, "Approximating the medial axis from the voronoi diagram with a convergence guarantee," *Algorithmica* **38**(1), 179–200 (2003)
14. M. Etzion and A. Rappoport, "Computing the Voronoi Diagram of a 3-d Polyhedron by Separate Computation of its Symbolic and Geometric Parts," *Proceedings of Symposium on Solid Modeling and Applications*, New York, NY (1999) pp. 167–178.
15. L. J. Guibas, C. Holleman and L. E. Kavraki, "A Probabilistic Roadmap Planner for Flexible Objects with a Workspace Medial-Axis-Based Sampling Approach," *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, Kyongju, South Korea (1999) pp. 254–260.
16. F. R. Hampel, E. M. Ronchetti, P. J. Rousseeuw and W.A. Stahel, *Robust Statistics: The Approach Bon Influence Functions*, Vol. 114 (John Wiley & Sons, New York, NY, 2011).
17. F. Hauer and P. Tsiotras, "Deformable Rapidly-Exploring Random Trees," *Proceedings of the Robotics: Science and Systems Conference*, Boston, MA (2017) p. P08.
18. K. E. Hoff III, J. Keyser, M. Lin, D. Manocha and T. Culver, "Fast Computation Of Generalized Voronoi Diagrams Using Graphics Hardware," *Proceedings of the Conference on Computer Graphics and Interactive Techniques*, New York, NY (1999) pp. 277–286.
19. C. Holleman and L. E. Kavraki, "A Framework for Using the Workspace Medial Axis in PRM Planners," *Proceedings of the IEEE International Conference on Robotics and Automation*, San Francisco, CA (2000) pp. 1408–1413.

https://doi.org/10.1017/S0263574718000164 Published online by Cambridge University Press

20. J. Huh, B. Lee and D. D. Lee, "Adaptive Motion Planning with High-Dimensional Mixture Models," *Proceedings of the IEEE International Conference on Robotics and Automation*, Singapore (2017) pp. 3740–3747.
21. M. Kallmann, "Dynamic and robust local clearance triangulations," *ACM Trans. Graph.* **33**(5), 161:1–161:17 (2014).
22. S. Kiesel, E. Burns and W. Ruml, "Abstraction-Guided Sampling for Motion Planning," *Proceedings of the Symposium on Combinatorial Search* Niagara Falls, Canada (2012) pp. 162–163. Also as UNH CS Technical Report 12-01.
23. D. G. Kirkpatrick, "Optimal search in planar subdivisions," *SIAM J. Comput.* **12**(1), 28–35 (1983).
24. A. M. Ladd and L. E. Kavraki, "Motion Planning in the Presence of Drift, Underactuation and Discrete System Changes," *Proceedings of the Robotics: Science and Systems Conference*, Boston, MA (2005) pp. 233–241.
25. E. Larsen, S. Gottschalk, M. C. Lin and D. Manocha, Fast Proximity Queries with Swept Sphere Volumes. *Technical Report* (Department of Computer Science, University of North Carolina, Chapel Hill, NC, 1999).
26. S. M. LaValle, *Planning Algorithms* (Cambridge University Press, Cambridge, MA, 2006).
27. S. M. LaValle, "Motion planning: The essentials," *IEEE Robot. Autom. Mag.* **18**(1), 79–89 (2011).
28. S. M. LaValle and J. J. Kuffner, "Randomized kinodynamic planning," *Int. J. Robot. Res.* **20**(5), 378–400 (2001).
29. J. M. Lien, S. L.Thomas and N. M. Amato, "A General Framework for Sampling on the Medial Axis of the Free Space," *Proceedings of the IEEE International Conference on Robotics and Automation*, Taipei, Taiwan (2003) pp. 4439–4444.
30. M. Muja and D. G. Lowe, "Scalable nearest neighbor algorithms for high dimensional data," *IEEE Trans. Pattern Anal. Mach. Intell.* **36**, 2227–2240 (2014).
31. M. Mukadam, X. Yan and B. Boots, "Gaussian Process Motion Planning," *Proceedings of the IEEE International Conference on Robotics and Automation*, Stockholm, Sweden (2016) pp. 9–15.
32. L. Palmieri, S. Koenig and K. O. Arras, "RRT-Based Nonholonomic Motion Planning Using Any-Angle Path Biasing," *Proceedings of the IEEE International Conference on Robotics and Automation*, Stockholm, Sweden (2016) pp. 2775–2781.
33. L. Palmieri, T. Kucner, M. Magnusson, A. Lilienthal and K. Arras, "Kinodynamic Motion Planning on Gaussian Mixture Fields," *Proceedings of the IEEE International Conference on Robotics and Automation*, Singapore (2017) pp. 6176–6181.
34. S. D. Pendleton, W. Liu, H. Andersen, Y. H. Eng, E. Frazzoli, D. Rus and M. H. Ang, "Numerical approach to reachability-guided sampling-based motion planning under differential constraints," *IEEE Robot. Autom. Lett.* **2**(3), 1232–1239 (2017).
35. E. Plaku, "Region-guided and sampling-based tree search for motion planning with dynamics," *IEEE Trans. Robot.* **31**, 723–735 (2015).
36. E. Plaku, L. E. Kavraki and M. Y. Vardi, "Motion planning with dynamics by a synergistic combination of layers of planning," *IEEE Trans. Robot.* **26**(3), 469–482 (2010).
37. J. Reif, "Complexity of the Mover's Problem and Generalizations," *Proceedings of the IEEE Symposium on Foundations of Computer Science*, San Juan, Puerto Rico (1979) pp. 421–427.
38. R. Rosenthal, H. Cooper and L. Hedges, "Parametric Measures of Effect Size," **In**: *The Handbook of Research Synthesis* (H. Cooper and L. V. Hedges, eds.) (Russell Sage Foundation, New York, 1994) pp. 231–244.
39. J. R. Shewchuk, "Triangle: Engineering a 2d Quality Mesh Generator and Delaunay Triangulator," **In**: *Applied Computational Geometry: Towards Geometric Engineering*, Lecture Notes in Computer Science, (M. C. Lin and D. Manocha, eds.) vol. 1148 (Springer, Berlin, Heidelbergm, 1996) pp. 203–222. Code available at https://www.cs.cmu.edu/∼quake/triangle.html
40. J. R. Shewchuk, "Delaunay refinement algorithms for triangular mesh generation," *Comput. Geom.: Theory Appl.* **22**, 21–74 (2002).
41. D. J. Tracy, S. R. Buss and B.M. Woods, "Efficient Large-Scale Sweep and Prune Methods with AABB Insertion and Removal," *Proceedings of the IEEE Conference on Virtual Reality*, Lafayette, LA (2009) pp. 191–198.
42. J. Vleugels and M. Overmars, "Approximating generalized Voronoi diagrams in any dimension," *Graph. Models Image Process.* (Utrecht University, Utrecht, Netherlands, 1995).
43. A. Wells and E. Plaku, "Adaptive Sampling-Based Motion Planning for Mobile Robots with Differential Constraints," **In**: *Towards Autonomous Robotic Systems*, Lecture Notes in Computer Science, (C. Dixon and K. Tuyls, eds.) vol. 9287 (Springer, Cham, 2015) pp. 283–295.
44. S. A. Wilmarth, N. M. Amato and P. F. Stiller, "MAPRM: A Probabilistic Roadmap Planner with Sampling on the Medial Axis of the Free Space," *Proceedings of the IEEE International Conference on Robotics and Automation*, Detroit, MI (1999) pp. 1024–1031.
45. S. A. Wilmarth, N. M. Amato and P. F. Stiller, "Motion Planning for a Rigid Body Using Random Networks on the Medial Axis of the Free Space," *Proceedings of the Symposium on Computational Geometry*, New York, NY (1999) pp. 173–180.
46. H. C. Yeh, J. Denny, A. Lindsey, S. L. Thomas and N. M. Amato, "UMAPRM: Uniformly Sampling the Medial Axis," *Proceedings of the International Conference on Robotics and Automation*, Hong Kong, China (2014) pp. 5798–5803.

https://doi.org/10.1017/S0263574718000164 Published online by Cambridge University Press