Robotica (2013) volume 31, pp. 1327–1335. © Cambridge University Press 2013
doi:10.1017/S0263574713000489

A real-time motion planning algorithm for a hyper-redundant set of mechanisms

Nir Shvalb†*, Boaz Ben Moshe‡ and Oded Medina§

†*Mechanical Engineering, Ariel University, Ariel, Israel*
‡*Computer Science, Ariel University, Ariel, Israel*
§*Industrial Engineering, Ariel University, Ariel, Israel*

(Accepted May 1, 2013. First published online: June 11, 2013)

SUMMARY
We introduce a novel probabilistic algorithm (CPRM) for real-time motion planning in the configuration space $\mathcal{C}$. Our algorithm differs from a probabilistic road map (PRM) algorithm in the motion between a pair of anchoring points (local planner) which takes place on the boundary of the obstacle subspace $\mathcal{O}$. We define a varying potential field f on $\partial\mathcal{O}$ as a Morse function and follow $\vec{\nabla} f$. We then exemplify our algorithm on a redundant worm climbing robot with n degrees of freedom and compare our algorithm running results with those of the PRM.

KEYWORDS: Motion planning; Path planning; Redundant manipulators; Open chain; Serial manipulator design and kinematics; Legged robots; Climbing robots.

1. Introduction

Hyper-redundant robots are characterized by considerable kinematic redundancy (see refs. [1, 2]). Hyper-redundant robots can be divided into the following groups:

(1) *Robot swarms*. These are coordinated multi-robot systems which consist of large numbers of mostly simple physical robots. Each swarm member may be restricted to motion on a given graph or may be free to translate in space. Potential applications for swarm robots are mainly for sensing tasks, where miniaturization is essential. A swarm, with each member possessing few degrees of freedom (DOFs), can easily give rise to an entity with hundreds of DOFs (see ref. [3]).
(2) *Snake robots*. These biomorphic hyper-redundant robots resemble a snake. Their ability to change the shape of their bodies allows them to perform a wide range of behaviors, such as climbing stairs or tree trunks. Their extra joints enhance maneuverability within tight obstacle fields, suggesting applications in congested environments (e.g. disaster relief[4] medical applications[5]). Rigid-link hyper-redundant designs imitate the biological backbone and may possess many DOFs (for example consider ref. [6], featuring 4 DOFs, and the surgical snake-like robot[7]).
(3) *Humanoid robots*. These robots resemble the human body and may also present with dozens of DOFs (see ref. [8]).

* Corresponding author. E-mail: odedmedina@gmail.com

Due to their great deal of redundancy it is easier to control these mechanisms by a predefined set of motions rather than calculate their options in real time. However, if they are to be deployed near an obstacle lade zone, though mechanically fit, they would need a real-time computing algorithm for problem solving on the go. In this paper, we present our algorithm for this problem and exemplify our algorithm on such a robot.

Definition 1.1. In a mechanism, configuration space $\mathcal{C}$ is the set of all its possible configurations.

Definition 1.2. In a given mechanism, its free configuration space $\mathcal{C}_{\text{free}}$ and a pair of configurations $p_s, p_g \in \mathcal{C}_{\text{free}}$, *motion planning* is an algorithm that finds a path $\gamma \in \mathcal{C}_{\text{free}}$ from p_s to p_g. An *optimal path* is one that minimizes a weight function(s) defined on $\mathcal{C}_{\text{free}}$.

It is also possible to set $\mathcal{W}$—a certain submanifold of $\mathcal{C}_{\text{free}}$ to be the submanifold on which motion planning will take place (i.e. instead of $\mathcal{C}_{\text{free}}$) see ref. [9]. Note that in order to take advantage of the redundancy (overcoming obstacles) for hyper-redundant robots, we require that $\mathcal{W} = \mathcal{C}_{\text{free}}$.

Various algorithms exist for motion planning.[10] Grid-based approaches overlay a grid on top of the configuration space, and assume that each configuration is identified with a grid point. At each grid point, the robot is allowed to move to adjacent points as long as the line between them is entirely within $\mathcal{C}_{\text{free}}$. This discretizes the set of actions, and search algorithms, such as $A^\star$ and Dijkstra, are used to find a path from the beginning point to the goal. Still, the above-mentioned example of hyper-redundant robots shows that these give rise to astronomical configuration spaces and hence are not applicable for real-time computation. The following algorithms are commonly used for real-time computations: (1) *Potential field algorithms* place focal points of repulsion and attraction within the configuration space, such that the resultant field reflects the values of a certain weight function. A path may then be constructed from the point of origin, subject to the direction of the potential field, to the destination. Such potential field algorithms are efficient, but fall prey to local minima (see ref. [11]), and when applying such algorithms to a obstacle congested environment, local minima are abundant. Yet, potential field algorithms may be used as a *local planner*.[12] (2) *Sampling-based algorithms* represent the configuration

https://doi.org/10.1017/S0263574713000489 Published online by Cambridge University Press

space with a *roadmap* of sampled configurations. A basic algorithm samples N configurations in $\mathcal{C}$, and retains those in $\mathcal{C}_{\text{free}}$ to use as milestones. A roadmap is then constructed, connecting all milestone pairs p and q if the line segment pq is entirely within $\mathcal{C}_{\text{free}}$. These algorithms work well for high-dimensional configuration spaces, because unlike combinatorial algorithms, their running time is not (explicitly) exponentially dependent on the dimension of $\mathcal{C}$, and in general they are substantially easier to implement. They are probabilistically complete, i.e. the probability that they will produce a solution approaches 1 as runtime increases.[13] However, they are not applicable for real-time computations when considering the hyper-redundant case (see ref. [14] for exactly the same problem as exemplified here). This paper addresses the problem of real-time motion planning for the hyper-redundant robot by combining the two algorithms.

Remark 1.3. In order to tackle the elapsed time problem, one may alternatively attempt not to optimize the motion planning algorithms, but to simplify the configuration space. In other words, it is possible to reduce the configuration space by severe compression (see Medina et al.[15]) so that a robot's embedded motion planning system will be capable of storing and accessing an otherwise immense data file, far beyond the system's memory capacity. This issue, however, is beyond the scope of this paper.

2. The Mechanical System

We shall exemplify our algorithm on a planar climbing worm robot, although our algorithm is applicable for any order of redundancy and is also applicable for many other mechanism topologies.

The robot at hand (see Fig. 1) was designed to climb metal trusses using magnetic grasping. We designed an open-loop actuated nine-link robot inspired by an inchworm. Such a worm moves forward by fixing its rear end, and extending its body to grip the front end. Furthermore, inchworms advance mainly in a planar manner, although it may also switch its motion planes when tackled with persistent obstacles. The robot has nine links with an articulated 1-DOF joint at each pair. We denote the ith axis direction vector by $\hat{\omega}_i$. All joints $\hat{\omega}_i$, for $i = 2, \ldots, 7$, are made parallel while $\hat{\omega}_2$ and $\hat{\omega}_8$ joints are perpendicular to all other joints and are confined to the plane of motion (see Fig. 1). In other words, fixing $\hat{\omega}_1$ and $\hat{\omega}_8$ results in a planar stepping movement, while $\hat{\omega}_1$ and $\hat{\omega}_8$ joints are used for rotating the plane in which the motion takes place.

Each end-link can be attached perpendicularly to the climbing surface by a magnet. The robot joints are actuated by HSR 5990 gt HighTec servo motors with a range of 180° and a 30 kg/cm torque when actuated by 7.4 V.

A step can be taken in two different manners: An *Inchworm step*, i.e. a heel-to-toe stepping motion (see Fig. 3) and a *slinky step* where the robot's rear end passes its front end (see Fig. 2). These enable avoiding obstacles, while the robot is extended up to 71% of its total length, by using the former technique to approach the obstacle (fine steps) and the latter to get past it.

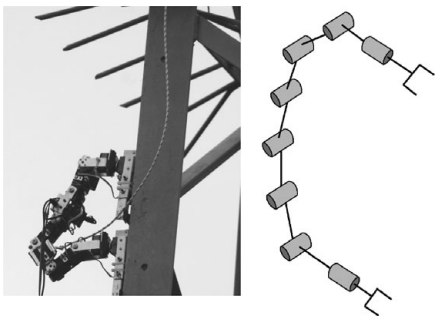


Fig. 1. The inchworm-inspired climbing robot and its schematic design.

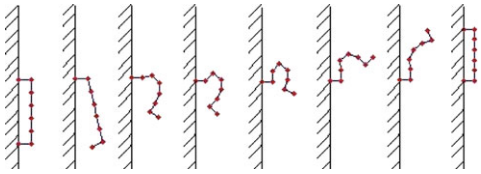


Fig. 2. (Colour online) "Slinky" maneuver simulation results.

3. Problem Statement

The configuration space is, by definition, an eight-dimensional torus $\mathbb{T}^8 = \mathbb{S}^1 \times \cdots \times \mathbb{S}^1$. Nevertheless, for practical purposes, one should consider the actuators' limitations. Thus, we will restrict our configuration space to $-\pi/2 < \theta_i < \pi/2$ for all $i = 1, \ldots, 8$, making $\mathcal{C} = \mathbb{I}^8$ the eight-dimensional cube. Furthermore, since we assume that all stepping motions take place on one plane, we may dismiss the first and last joints, thus reducing the configuration space to the six-dimensional cube $\mathbb{I}^6$. We denote the ith joint position by

$$(\mathbf{x}_i, \mathbf{y}_i) = \left(\sum_{k=1}^{i} L_k \cos \sum_{m=1}^{k} \theta_m, \quad \sum_{k=1}^{i} L_k \sin \sum_{m=1}^{k} \theta_m \right),$$

where L_k is the length of the kth link and θ_k is the angle between links k and $k-1$.

Summing up, we regard the configuration space as a torus $\mathbb{T}^6$ with an obstacle $\mathcal{O}_\theta$ so that $\mathcal{C} = \mathbb{I}^6$, where $\mathcal{O}_\theta$ is defined exactly by the *angles limitation* equations defined above. Obviously, we need to avoid *obstacles in the robot's work*

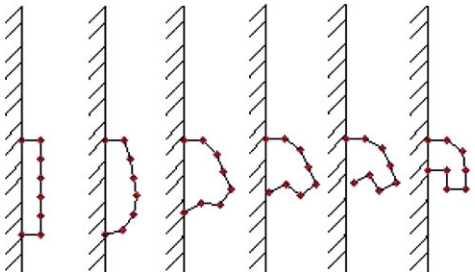


Fig. 3. (Colour online) "Heel-to-toe" maneuver simulation results.

https://doi.org/10.1017/S0263574713000489 Published online by Cambridge University Press

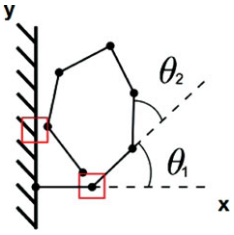


Fig. 4. (Colour online) Collision between second node and seventh node.

space as well. Such obstacles are formulated as a set of halfplane (in the work space) obstacles denoted by $\mathcal{O}_X$ and $\mathcal{O}_Y$. Thus, a point obstacle $(\mathbf{x}_0, \mathbf{y}_0)$ in the work space will be avoided by keeping a "safe distance" from a Cartesian rectangle surrounding it:

$$\|\mathbf{x}_i - \mathbf{x}_0\| > \epsilon \ , \|\mathbf{y}_i - \mathbf{y}_0\| > \epsilon \ , \forall i = 1, \ldots, n+1. \quad (1)$$

In order to avoid *self-collisions*, i.e., disallow configurations where the distance between any two joints is less than $\epsilon > 0$ (see Fig. 4), we define a new obstacle and denote it by $\mathcal{O}_{col}$ defined as

$$\|\mathbf{x}_i - \mathbf{x}_j\| > \epsilon \ , \|\mathbf{y}_i - \mathbf{y}_j\| > \epsilon \ , \forall i \neq j. \quad (2)$$

Remark 3.1. For a given configuration $c \in \mathcal{C}$ we denote the traveling distance (in the work space) of a given joint i, due to a resolution-sized step in joint j by $s(c)_{i,j}$. For each configuration, we define the maximal step size as $s := \max_{i,j,c}\{s(c)_{i,j}\}$. Obviously to avoid collisions, ϵ should be greater then s; for simplicity, we choose the same value for self-collisions and obstacles. For the climbing robot, this happens for the distal joint when the entire robot is aligned and the base joint is actuated.

Note that under our angle limitations, equation (2) is reduced to the set of all pairs i, j, where $\|i - j\| \geq 4$.

Torque limitation is to be considered as well. We denote such an obstacle by $\mathcal{O}_T$, which is defined by the configurations where the maximal torque exerted on one or more of the actuators exceeds a given torque T_0. The torque (resulting at the base joint 1) is given by

$$T = mg \sum_{k=1}^{n-1} \mathbf{x}_k + \frac{mg\mathbf{x}_n}{2}, \quad (3)$$

where n is the number of DOF ($n = 6$ in our case) and m is the mass of a single link (here, all links are assumed to be homogeneous).

Remark 3.2. As a benchmark for the shortest path possible, the following approach was taken. The torque T was calculated for all n-dimensional grid points $(\theta_1, \theta_2, \ldots, \theta_n) = \mathcal{N}^n$, where $\mathcal{N} = \{-\pi/2, -\pi/2 + \Delta\theta, \ldots, \pi/2\}$ and $\Delta\theta$ is the grid resolution taken here as $\pi/6$. Fixing $n = 6$ the configuration space volume exceeds 5×10^6 voxels, and the maximal number of neighboring configurations for each configuration is $\sum_{k=1}^{6}(C_6^k) = 728$ ($C_m^n = \frac{m!}{n!(m-n)!}$). This approach forces one to make do with low resolutions, which, in turn, may result in large translations in the work space. Therefore, this approach is not applicable for congestive obstacle zones.

Nevertheless, we conducted several tests considering only 12 arcs $\gamma_{i,j}$ for each configuration i (connecting it to its adjacent configuration j), and reducing actuation to one motor at a time. In addition, we considered a maximum endurable torque such that if $\gamma_{i,j} > T_{\max}$ we set $\gamma_{i,j}$ to infinity. We then implemented Dijkstra's algorithm in order to extract the shortest path connecting a given origin and goal configurations. Calculating $\mathcal{C}_{\text{free}}$ took over an hour on a dual-core PENTIUM IV platform, which makes this naive approach non-applicable for real-time calculations, but still qualitatively demonstrates reasonable paths to compare our results with.

4. CPRM Motion Planning Scheme

As indicated above, we would like to combine both potential field algorithms with sampling-based algorithms. Our algorithm differs from a probabilistic road map (PRM) algorithm in the motion between a pair of anchoring points which takes place on the boundary of the obstacle subspace $\mathcal{O}$. We define a varying potential field f on $\partial\mathcal{O}$ as a Morse function and follow $\vec{\nabla} f$. Such an approach makes it possible for two non-mutually visible points to be connected and thus decreases runtime. We define a potential field as a Morse function in $\mathcal{C}$ restricted to $\partial\mathcal{O}$. Still, a situation at which $\vec{\nabla} f = 0$ on a "large" subspace is not welcome, since in such a place one would be disoriented. In order to accommodate such a property, we use a restricted Morse function as our potential field.

Definition 4.1. Given M a compact n-dimensional smooth manifold and a smooth map $f : M \to \mathbb{R}$, we call $p \in M$ a *critical point* of f if $\partial f/\partial x_i = 0$ for all local coordinates x_i of M at p. A critical point is said to be *non-degenerate* if the Hessian matrix $H = [\partial^2 f/\partial x_i \partial x_j]_{i,j=1}^n$ is non-singular at p. A *Morse function* of M is a smooth function f with only non-degenerate critical points (compare with ref. [16]).

Fact 4.2 (Ni et al.[17])**.** Surprisingly, it turns out that almost every smooth function $f : M \to \mathbb{R}$ is Morse. Furthermore, if $g : M \to \mathbb{R}$ is a smooth function, then there exists a Morse function $f : M \to \mathbb{R}$ arbitrarily close to g.

Since we differentially follow $\vec{\nabla} f$, it is essential that f be a Morse function in order to avoid situations in which critical points are non-isolated (degenerated critical points).

4.1. The algorithm

As described above, our motion planning algorithm is a crawling motion planner, i.e., whenever a configuration space obstacle is met, motion is continued by crawling on its boundary, while motion in the ambient space is a linear one (see Fig. 5). Since all the obstacles are "convex" in a certain way as we discuss in Proposition 4.4, this method is sufficient for most situations. Still, there may be situations in which we

https://doi.org/10.1017/S0263574713000489 Published online by Cambridge University Press

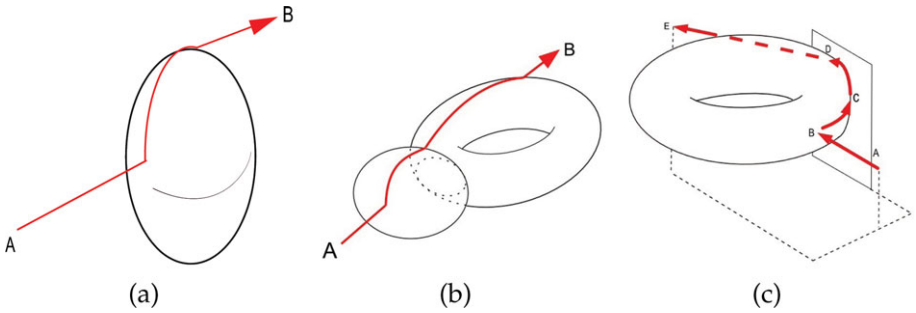


Fig. 5. (Colour online) A schematic view of the algorithm (a) crawling on the obstacle's surface, (b) crawling on two obstacles, (c) crawling on an obstacle $\mathcal{O}$ that intersects with an angle-limitation boundary $\mathcal{O}_\theta$.

will need to traverse n or more obstacles at once, a situation which could cause our algorithm to fail. In order to tackle this problem, we use a probabilistic method and continue our crawling scheme by redefining f. The algorithm works well due to Fact 4.2 (alternatively, f can be corrected if slightly perturbed).

We divide the configuration space into two subspaces—the free space C_{free} and the obstacle space $\mathcal{O} = \mathcal{O}_{\text{col}} \cup \mathcal{O}_T \cup_i \mathcal{O}_{x_i} \cup_j \mathcal{O}_{y_j} \cup \mathcal{O}_\theta$. We define two configuration points $p_s, p_g \in \mathcal{C}_{\text{free}}$ as the origin and goal configurations. Motion in C_{free} is carried out by following the $\vec{l} = p_g - p_s$ direction. Therefore:

Definition 4.3. In the free configuration space, f is defined as the Euclidean distance in $\mathcal{C}$ from current configuration p_c to p_g. Note that $\vec{\nabla} f$ is simply $\vec{l} = p_g - p_c$ in $\mathcal{C}_{free}$.

When a path intersects $\mathcal{O}$, motion is continued on $\mathcal{O}$ boundary following the direction of $\vec{\nabla}_{\mathcal{O}} f$ using the following: (1) calculate $\vec{\nabla}\mathcal{O}$ (see Section 3), (2) calculate the null space K to all $\vec{\nabla}\mathcal{O}$, and (3) project $\vec{l}$ on K and proceed motion at the projection direction. This is carried out until $\vec{l}$ leaves $\mathcal{O}$, that is, until $\vec{l}$ points out of $\mathcal{O}$.

Let us denote $\vec{n}_i$ as the normal to the ith obstacle calculated in the intersection configuration (see Fig. 5a) When reaching a configuration which satisfies more than one obstacle, we may ignore $\mathcal{O}_i$ for which $\vec{l} \cdot \vec{n}_i > 0$. In configurations where all $\mathcal{O}_i$ maintain $\vec{l} \cdot \vec{n}_i < 0$ (see Fig. 5c), motion takes place at the null space of all $\vec{n}_i$.

Algorithm 1: Morse crawling pseudocode.

Data Inputs p_s,p_e, $\mathcal{O}_i$, $\mathcal{O}_j$, $\mathcal{C} = [\frac{-pi}{2}, \frac{\pi}{2}]^n$
Output A path from p_s to p_e avoiding $\mathcal{O}_i$, $\mathcal{O}_j$.
set $p_c = p_s$
Perturb f if it satisfies equation (6) or (9).
While $f > 0$ do:
  If $p_c \notin \partial\mathcal{O}_1$, $p_c \notin \partial\mathcal{O}_2$
    move linearly toward p_e reducing f.
  If $p_c \in \partial\mathcal{O}_i$ but $p_c \notin \partial\mathcal{O}_j$
    Follow $\vec{\nabla}_{\partial\mathcal{O}_i} f$.
    If $p_c \in \partial\mathcal{O}_j$ as well
      Move in perpendicularly to $\vec{\nabla}_{\mathcal{O}_j} f$ and $\vec{\nabla}_{\mathcal{O}_i} f$ reducing f.

4.2. *Completeness of Algorithm 1*

In order for f to be an efficient potential field, one would like to have f as Morse, preferably with no critical points on $\partial\mathcal{O}$. We will now prove that this is the case.

Proposition 4.4. The critical points of almost all potential field functions f (as defined above) are not on $\mathcal{O}_x$, $\mathcal{O}_y$, $\mathcal{O}_{col}$, $\mathcal{O}_T$ nor on $\mathcal{O}_\theta$.

Proof. We shall prove that for any given vector $\nabla f = \vec{l}$, there are no normal vectors to $\partial\mathcal{O} \subset \mathcal{C}$ which are equal to it (up to sign). For brevity, we denote the set $\{\theta_1, \theta_2, \ldots, \theta_n\}$ by Θ and the sum $\sum_{j=1}^{k} \theta_j$ by Θ_k. Recall that the configuration space is defined in the orthogonal space bounded by $\theta_i = [\frac{-\pi}{2}, \frac{\pi}{2}]$, for all $i = 1, 2, ..., n$. An obstacle $\mathcal{O}_x$ (and $\mathcal{O}_T$) is implicitly defined by

$$h_x(\Theta) = \sum_{i=1}^{n} a_i \cos(\Theta_i) - x_0. \tag{4}$$

By differentiating, we get $\frac{\partial h_x}{\partial \theta_i} = -\sum_{k=1}^{n} a_k \sin(\Theta_k)$. We compare $\vec{\nabla} h_x = \vec{l}$ to find points on $\partial\mathcal{O}_x$ whose normal equals the path direction $\vec{l}$. Solving (4) we get

$$\begin{matrix} \Theta_m = 2\pi(k_m + 1) - \sin^{-1}\left(\frac{l_{m+1} - l_m}{a_m}\right), \\ 2\pi k_m + \sin^{-1}\left(\frac{l_{m+1} - l_m}{a_m}\right) \end{matrix} \tag{5}$$

where $l_{n+1} = 0$ and $|k_m| \leq \left\lfloor \frac{m+1}{4} \right\rfloor$. Assigning (5) to (4), we get the relation

$$\sum_{m=1}^{n} \sqrt{a_m^2 - (l_{m+1} - l_m)^2} = x_0. \tag{6}$$

This completes the proof for $\mathcal{O}_x$. Clearly, almost all $\vec{l}$ will not satisfy this, and even if it does we can and shall perturb f to avoid the appearance of such critical points.

For $\mathcal{O}_y$ which is defined as a sum of sines rather than cosines, the proof is identical. $\mathcal{O}_{\text{col}}$ where the j_1th joint and

https://doi.org/10.1017/S0263574713000489 Published online by Cambridge University Press

the j_2th joint (assume $j_2 > j_1$) collision are defined by

$$h_x(\Theta) = \sum_{i=j_1}^{j_2} a_i \cos(\Theta_i), \tag{7}$$

$$h_y(\Theta) = \sum_{i=j_1}^{j_2} a_i \sin(\Theta_i). \tag{8}$$

By differentiating, equating to $\vec{l}$ and assigning to (7), we get

$$\sum_{m=j_1}^{j_2} \sqrt{a_m^2 - (l_{m+1} - l_m)^2} = 0. \tag{9}$$

Again, we can and shall perturb f to avoid equality. □

Remark 4.5. A basic result of Morse theory states that almost all functions are Morse. More precisely, since $\mathcal{O}$ is an immersed submanifold in $\mathcal{C} \in \mathbb{R}^n$ and $f : \mathcal{C} \times \mathcal{O} \to \mathbb{R}$, which is given by $f(o, c) = \|o - c\|^2$, then for almost all $c \in \mathcal{C}$ the function $f_c = f(\cdot, c) : \mathcal{O} \to \mathbb{R}$ is a Morse function on $\mathcal{O}$ (see Milnor,[18] p. 36). So actually any collision function in the work space which defines a smooth manifold in $\mathcal{C}$ (e.g. point–point, point–segment, segment–segment, and plane–point) together with a function defined as a distance between current configuration (on $\mathcal{O}$) and the target configuration will most likely end up to be Morse.

Consider a configuration $c \in \mathcal{C}$ which satisfies multiple manifolds $i \in I$ such that $\vec{n}_i \cdot \vec{l} > 0$ for all $i \in I$ (here, n_i is the normal to the ith manifold at c). In this case, motion is confined to the $n - \|I\| \geq 0$ dimensional subspace. When equality is satisfied, our algorithm fails due to the local minima. Since we consider a total of more than 14 obstacles (one torque obstacle, $m_v \geq 2$ vertical (in the work space) obstacles, $m_h \geq 1$ horizontal obstacles and 10 self-collision obstacles) local minima happen quite often.

As for non-regular obstacles (i.e. those that exhibit undefined derivatives in some point subset Σ), we apply the same rational, that is, setting two anchoring points to be not-connected if the path between them crosses Σ.

4.3. Road map

To overcome the local minima problem, we apply the probabilistic road map scheme: we randomly add m points in $\mathcal{C}_{\text{free}}$ in addition to p_s, p_g and examine pairwise connectivity. The resulting connectivity graph is weighted by the number of steps needed to transverse node pairs. Note that two nodes may be found to be connected using CPRM while for PRM they will not. In other words, the CPRM algorithm reduces the number of nodes needed for the solution. So, for cases where just few nodes are needed, the choice of the algorithm makes no real difference. Here, we used Dijkstra's algorithm to calculate the roadmap's shortest path. Nevertheless, in cases where many nodes are needed, we believe that a DFS algorithm or some other uninformed search algorithm would be more efficient in the sense of elapsed time.

Algorithm 2: Crawling probabilistic road map.

Data Inputs p_s,p_g, m, $\{\mathcal{O}_i\}_1^{k>1}$, $\mathcal{C} = [\frac{-pi}{2}, \frac{\pi}{2}]^n$
Output The shortest path from p_s to p_g avoiding all $\mathcal{O}_i$
Generate m random points $p_r \in \mathcal{C}_{\text{free}}$
While shortest path is not obtained, do:
  for all p_i, p_j, $i \neq j$, do:
  check connectivity between p_i and p_j (using algorithm 1).
    If points are connected:
      grade the connected arc as the mean of the
      weight function on the connecting line.
    If line convergent to local minima:
      set p_i and p_j as not connected.
  use Dijkstra to calculate shortest path from the graph.
  If no path is found: increase m.

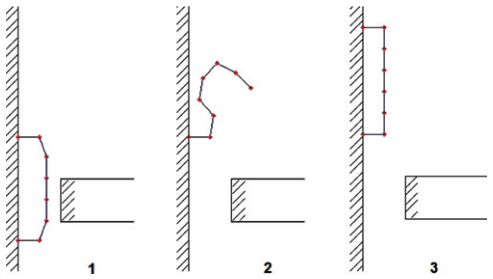


Fig. 6. (Colour online) A three-anchoring point solution via CPRM.

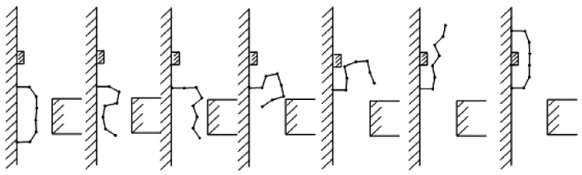


Fig. 7. A seven-anchoring point solution via CPRM.

Remark 4.6. Note that, though not implemented here, graph connectivity algorithms may accelerate calculations when selectively adding new nodes that should be checked for pairwise connectivity.

4.4. Parallelizing CPRM

Naturally, one may accelerate the calculation runtime by applying parallel computation using multicore CPU (modern computers commonly possess 2–8 cores). Plaku et al.[19] have shown that RoadMap motion planning algorithms may have efficient parallel versions, up to almost linear speedup. Further runtime improvement can be achieved using a GPGPU approach for motion planning (see refs. [20, 21]). One can apply the single instruction multiple data (SIMD) methodology over Algorithm 2. Recall that Algorithm 2 determines the connectivity between all pairs of anchoring points and therefore is suitable for parallel computing (requires limited size of data).

5. An Example: An Inchworm-Inspired Climbing Robot

5.1. Simulation results

A MATLAB program for controlling the robot was written (see simulation results in Figs. 6, 7, 13). Commands were

https://doi.org/10.1017/S0263574713000489 Published online by Cambridge University Press

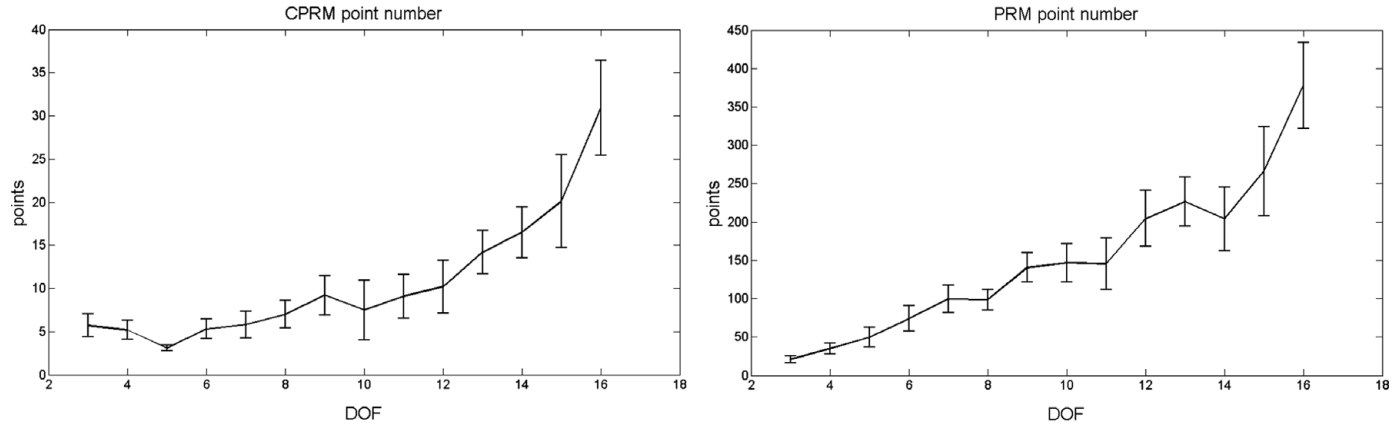


Fig. 8. Number of anchoring points.

sent to a 16 servo *pololu* controller. Our implementation for crawling on the hyper-surfaces was carried out by small steps with constant length. We compared the CPRM algorithm with the naive PRM algorithm. The program was given start and goal configurations and obstacle functions: wall obstacle, a rectangle obstacle, torque limitation, and self-intersection limitation. We determined start and goal configurations as

$$p_g = \underbrace{(\pi/2, 0, \ldots, 0, \pi/2)}_{n}, \quad p_s = \underbrace{(-\pi/2, 0, \ldots, 0, -\pi/2)}_{n}$$

for a sequence of dimensions $n = 3, 4, 5, ..., 16$. The naive algorithm selects $k = 1$ random anchoring points in $\mathcal{C}_{\text{free}}$. Mutual connectivity for all pairs of points is calculated to form a graph, each arc is given weight corresponding to the energy consumption needed for the given move. The shortest path is then calculated using the Dijkstra algorithm. If no path is found, an additional point is selected to form a weighted graph with $k + 1$ nodes. Notice that using this method is highly time-consuming. Still, since our goal is to compare anchoring point and calculation time for a successive experiment, calculation time is reset to zero when k is increased. The algorithm proceeds in adding points until a solution is found (for, say, $k = k_f$ points). Finally, we keep track of the elapsed time for the final set k_f of anchoring points. Similarly, we ran the CPRM algorithm using the same procedure, i.e. starting with $k = 1$ random points, adding points when needed up to k_f. Again, we make record of time elapsed for the successful k_f (see Fig. 8). Note that this is a stringent measure since the total difference in the elapsed time between both motion planning schemes is bigger than that depicted (may be proved inductively). Due to the additional crawling feature, the number of anchoring points needed for obtaining a path from start to goal is presumed to be smaller. Figure 8 depicts an average of 10 experiments for each dimension. Indeed, any PRM algorithm results in an exponential number of anchoring points. The CPRM demands an average of less than 10% of the PRM anchoring points. Figure 9 compares CPRM and PRM elapsed times. Needless to say, reducing the number of nodes is essential for real-time systems and, as seen here, CPRM presents a major advantage. Although one may suspect that the PRM algorithm establishes a pseudo-linear approximation of the CPRM algorithm, the length of the path in each algorithm is different, as can be seen in Fig. 10. This also points to the efficiency of the CPRM.

6. Additional Experiments

In order to get a clear estimate of our algorithm performance, we have conducted two additional experiments using the well-known *cage setup* and the *puma-bars setup*.

The cage setup is depicted in Fig. 11, where an "L" shape is confined in an octagon prism bird's cage. The task is to exit the cage. This setup was experimented by Yershova and Lavelle[22] (a tighter version known as the

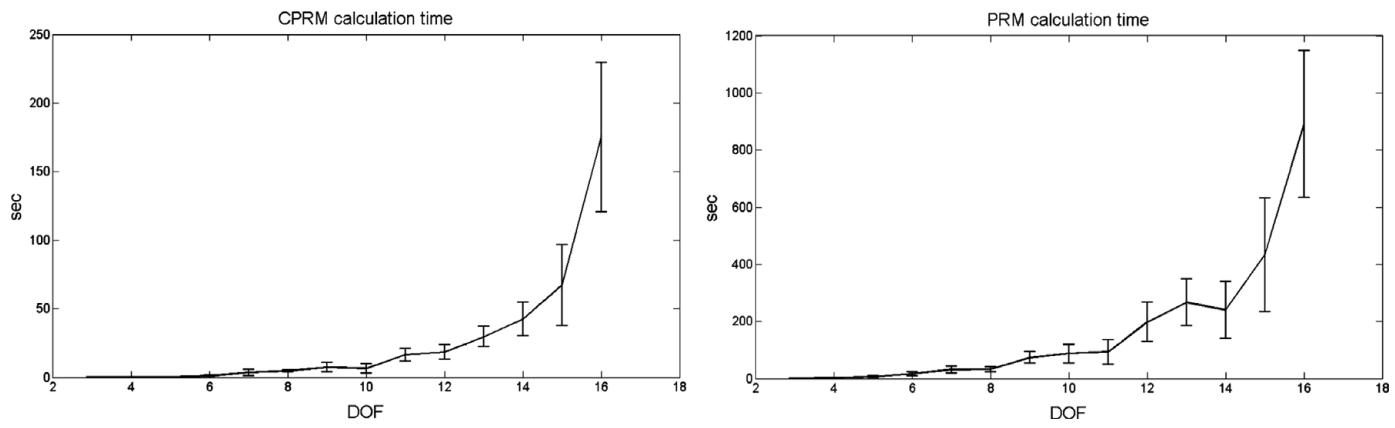


Fig. 9. Elapsed time.

https://doi.org/10.1017/S0263574713000489 Published online by Cambridge University Press

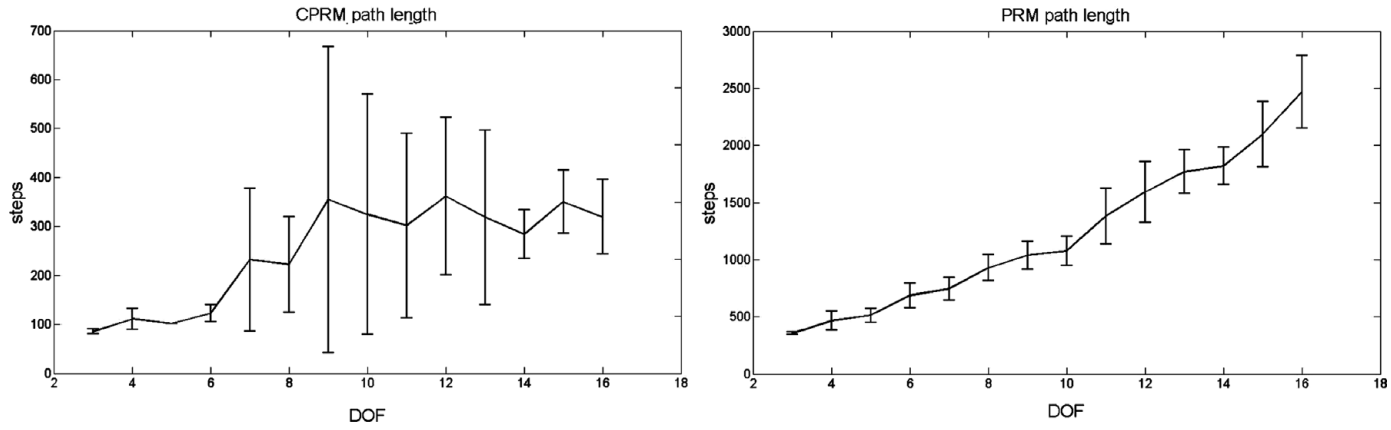


Fig. 10. Path length.

Table I. Simulation results of some motion planners for the cage setup.

	MPNN	PRM	Cover tree	*sb* (s)	CPRM
Computation time (s)	191	2,302	187	361	7.8
Number of nodes	37,634	37,186	35,814	37,922	3

Hedgehog problem was solved but not formally published by V. Vonasek). Yershova and Lavelle[23] reported a set of 37,634 nodes required for solution and a computation time of 191 s for the MPNN algorithm; 37,186 nodes and 2,302 s for a naive PRM; 35, 814 nodes and 187 s for the *Cover tree* algorithm, and 37,922 anchoring points with a 361 s calculation time applying *sb*(s) (see Table 1). The shortest calculation time given in ref. [22] is reported as 21 s.

We have performed a set of 100 tests using CPRM. Initial and target configurations were randomly selected (in and out of the cage). The "L" shape taken as unit length for both its horizontal and vertical segments, with a 0.3×0.3 rectangle profile, the bars were taken as 0.1 diameter and 2.2 for the overall (middle) diameter of the cage. Collisions were calculated as intersections between line segments (rather than points getting too close as in our climbing planar case).

Our results show that an average of only 3 anchoring points are needed while the averaged calculation time was 7.8 seconds. This shows that our algorithm performs well under tight environments. Furthermore, since the number of anchoring points has been reduced, the six-dimensional motion trajectory is very smooth as depicted in Fig. 11.

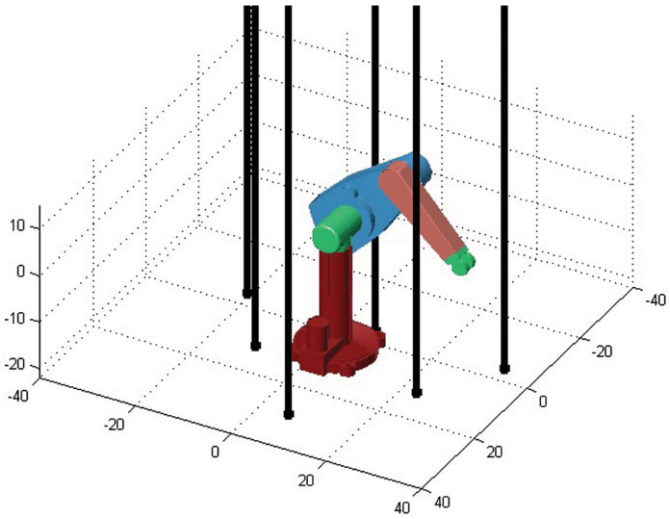


Fig. 12. (Colour online) The puma-bar setup. Vertical bars' diameter was set to 4 in., while the robot's forearm length is 17.05 in.

The *puma-bars setup* comprises *Puma560*—a 6R articulated robot arm together with six rods in its workspace. Kavraki et al.[19] used a sampling-based road map of trees (SRT) for solving the motion planning problem (considering random initial and goal configurations). In particular, they compared two single-query planners: the *Expansive Space Trees* and the *Rapidly-exploring Random Trees*. They further

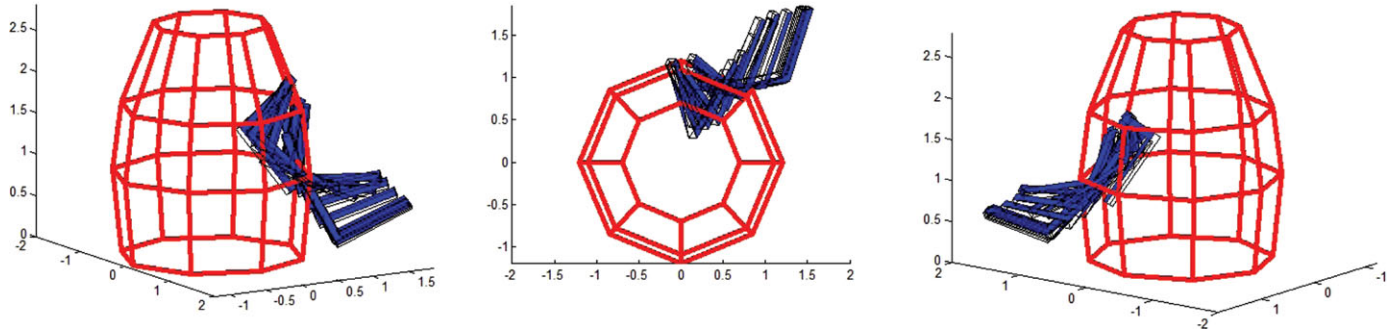


Fig. 11. (Colour online) A typical trajectory solution.

https://doi.org/10.1017/S0263574713000489 Published online by Cambridge University Press

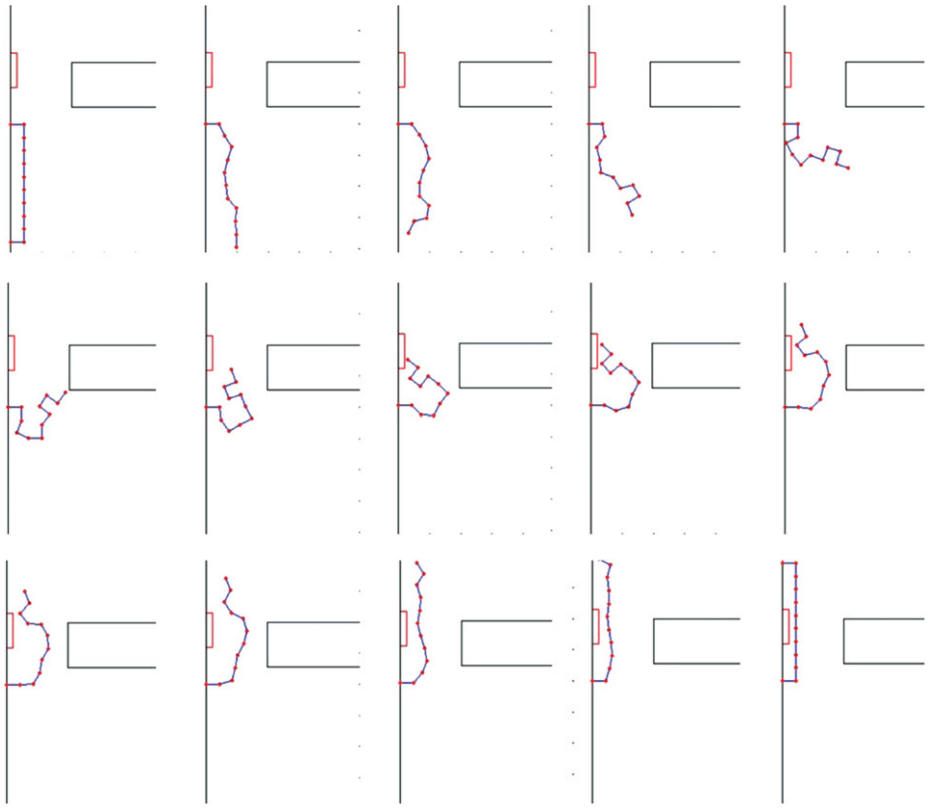


Fig. 13. (Colour online) The algorithm applied on the 10-DOF climbing robot.

Table II. Simulation results of some motion planners for the puma-bars setup.

	Serial SRT[RRT]	Serial SRT[EST]	Parallel SRT[RRT]	Parallel SRT[EST]	CPRM
Computation time (s)	8,097	10,207	327	414	40

compared serial computations with parallel ones using 22 processors. Results are presented in Table II.

We applied the CPRM algorithm on the puma-bar problem (Fig. 12) using serial computation. Collisions were calculated as intersections between line segments (as in the cage setup). A set of 100 tests were conducted. An average of 2.7 encoring points (with standard deviation $\sigma = 0.9$) were needed for solving the motion planning problem. This yields an average calculation time of 40 s (with $\sigma = 38$ s).

7. Conclusions

Motion planning via the configuration space is necessary for hyper-redundant robots requiring challenging maneuvers. Our results imply that combining the naive PRM algorithm with the potential field method significantly shortens calculation time. The CPRM algorithm is applicable for mechanisms whose configuration spaces are algebraically known, yet one may also apply CPRM for experimentally derived configuration spaces whose algebraic form is not known, and is too complex for real-time calculations. This may be done via configuration space compression and region-of-interest decompression. We plan to investigate such a generalized approach in the future.

Acknowledgements

We acknowledge the anonymous reviewers for their fruitful comments.

References

1. Z. Jianwen, S. Lining, D. Zhijiang and Z. Bo, "Self-Motion Analysis on a Redundant Robot with a Parallel/Series Hybrid Configuration," **In**: *Proceedings of the 9th International Conference on Control, Automation, Robotics and Vision (ICARCV)*, Singapore (Dec. 2006), pp. 1–6.
2. H. Brown, M. Schwerin, E. Shammas and H. Choset, "Design and Control of a Second-Generation Hyper-Redundant Mechanism," **In**: *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, San-Diego, California (2007) pp. 2603–2608.
3. S. Kornienko, O. Kornienko, A. Nagarathinam and P. Levi, "From Real Robot Swarm to Evolutionary Multi-Robot Organism," **In**: *Proceedings of the IEEE Congress on Evolutionary Computation (CEC)*, Singapore (2007) pp. 1483–1490.
4. H. Maruyama and K. Ito, "Semi-Autonomous Snake-Like Robot for Search and Rescue," **In**: *Proceedings of the 2010 IEEE International Workshop on Safety Security and Rescue Robotics (SSRR)*, Bremen, Germany (Jul. 2010) pp. 1–6.
5. T. Ota, A. Degani, D. Schwartzman, B. Zubiate, J. McGarvey, H. Choset and M. Zenati, "A highly articulated robotic surgical system for minimally invasive surgery," *Ann. Thoracic Surg.* **87**, 1253–1256 (May 2009).
6. H.-S. Yoon and B.-J. Yi, "A 4-dof Flexible Continuum Robot using a Spring Backbone," **In**: *Proceedings of the International Conference on Mechatronics and*

https://doi.org/10.1017/S0263574713000489 Published online by Cambridge University Press

Automation (ICMA), Changchun, Jilin, China (Aug. 2009) pp. 1249–1254.
7. K. Xu and N. Simaan, "Actuation Compensation for Flexible Surgical Snake-Like Robots with Redundant Remote Actuation," **In**: *Proceedings of the 2006 IEEE International Conference on Robotics and Automation (ICRA)*, Orlando, Florida (May 2006) pp. 4148–4154.
8. M. Akhtaruzzaman and A. Shafie, "Evolution of Humanoid Robot and Contribution of Various Countries in Advancing the Research and Development of the Platform," **In**: *Proceedings of the 2010 International Conference on Control Automation and Systems (ICCAS)*, Gyeonggi-do, Korea (Oct. 2010) pp. 1021–1028.
9. B. Siciliano and O. Khatib, *Springer Handbook of Robotics* (Springer-Verlag, New York, 2008).
10. J. Kuffner, K. Nishiwaki, S. Kagami, M. Inaba and H. Inoue, "Motion planning for humanoid robots," *Robot. Res.*, **21**(1), 365–374 (2005).
11. D. Yagnik, J. Ren and R. Liscano, "Motion Planning for Multi-Link Robots Using Artificial Potential Fields and Modified Simulated Annealing," **In**: *Proceedings of 2010 IEEE/ASME International Conference on Mechatronics and Embedded Systems and Applications (MESA)*, Vancouver, Canada (Jul. 2010) pp. 421–427.
12. A. Masoud, "A Discrete Harmonic Potential Field for Optimum Point-to-Point Routing on a Weighted Graph," **In**: *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, Beijing, Caina (Oct. 2006), pp. 1779–1784.
13. S. Khanmohammadi and A. Mahdizadeh, "Density Avoided Sampling: An Intelligent Sampling Technique for Rapidly-Exploring Random Trees," **In**: *Proceedings of the 8th International Conference on Hybrid Intelligent Systems, 2008 (HIS '08)*, Barcelona, Spain (Sep. 2008), pp. 672–677.
14. L. Kavraki, J. Latombe, R. Motwani and P. Raghavan, "Randomized Query Processing in Robot Path Planning," **In**: *Proceedings of the 27th Annual ACM Symposium on Theory of Computing*, Las Vegas, Nevada (1995) pp. 353–362.
15. O. Medina, A. Taitz, B. B. Moshe, and N. Shvalb, "C-space compression for robots motion planning," *Int. J. Adv. Robotic. Syst.* **10**(6), 1–7 (2013).
16. E. Rimon and D. Koditschek, "Exact robot navigation using artificial potential functions," *IEEE Trans. Robot. Autom.* **8**(5), 501–518 (1992).
17. X. Ni, M. Garland and J. Hart, "Fair morse functions for extracting the topological structure of a surface mesh," *ACM Trans. Graph.* **23**(3) 613–622 (2004).
18. J. Milnor, *Morse Theory* (Princeton University Press, NJ, 1963).
19. E. Plaku, K. Bekris, B. Chen, A. Ladd and L. Kavraki, "Sampling-based roadmap of trees for parallel motion planning," *IEEE Trans. Robot.* **21**(4), 597–608 (2005).
20. A. Bleiweiss, "GPU Accelerated Pathfinding," **In**: *Proceedings of the 23rd ACM SIGGRAPH/EUROGRAPHICS Symposium on Graphics Hardware*, Eurographics Association, Sarajevo, Bosnia (2008) pp. 65–74.
21. J. Kider, M. Henderson, M. Likhachev and A. Safonova, "High-Dimensional Planning on the GPU," **In**: *Proceedings of 2010 IEEE International Conference on Robotics and Automation (ICRA)*, Anchorage, Alaska (2010) pp. 2515–2522.
22. A. Atramentov and S. LaValle, "Efficient Nearest Neighbor Searching for Motion Planning," **In**: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, Vol. 1, IEEE, Washington, D.C. (2002) pp. 632–637.
23. A. Yershova and S. LaValle, "Improving motion-planning algorithms by efficient nearest-neighbor searching," *IEEE Trans. Robot.* **23**(1), 151–157 (2007).

https://doi.org/10.1017/S0263574713000489 Published online by Cambridge University Press