Randomized path planning on vector fields

Inyoung Ko, Beobkyoon Kim and Frank Chongwoo Park

The International Journal of Robotics Research 2014 33: 1664 originally published online 9 October 2014
DOI: 10.1177/0278364914545812

Article

The International Journal of Robotics Research
2014, Vol. 33(13) 1664–1682

DOI: 10.1177/0278364914545812
ijr.sagepub.com

Randomized path planning on vector fields

Inyoung Ko, Beobkyoon Kim and Frank Chongwoo Park

Abstract

Given a vector field defined on a robot's configuration space, in which the vector field represents the system drift, e.g. a wind velocity field, water current flow, or gradient field for some potential function, we present a randomized path planning algorithm for reaching a desired goal configuration. Taking the premise that moving against the vector field requires greater control effort, and that minimizing the control effort is both physically meaningful and desirable, we propose an integral functional for control effort, called the ***upstream criterion****, that measures the extent to which a path goes against the given vector field. The integrand of the upstream criterion is then used to construct a rapidly exploring random tree (RRT) in the configuration space, in a way such that random nodes are generated with an a priori specified bias that favors directions indicated by the vector field. The resulting planning algorithm produces better quality paths while preserving many of the desirable features of RRT-based planning, e.g. the Voronoi bias property, computational efficiency, algorithmic simplicity, and straightforward extension to constrained and nonholonomic problems. Extensive numerical experiments demonstrate the advantages of our algorithm vis-à-vis existing optimality criterion-based planning algorithms.*

1. Introduction

The problem of path planning on vector fields is one that arises in a wide range of robotics applications, but has yet to receive the same attention as the classical configuration space path planning problem. In field robotics, for example, a wheeled rover may be asked to navigate rocky, hilly terrain in the presence of strong, shifting winds, or a swimming robot may need to traverse an obstacle-cluttered underwater environment in the presence of strong, shifting water currents. In such cases the shortest path may not necessarily be the best path, as moving directly against the wind or current field for extended periods may quickly deplete the available power. Other examples include robots that must steer through some medium (e.g. a needle robot through skin, or a digging robot through the earth) that more easily allows motion in certain directions and resists motion in other directions. Planning on any configuration space in which a potential function value can be associated with each point can also be cast as a vector field planning problem, by taking the (negative) gradient field of this potential function as the vector field.

The class of planning problems described above can most naturally be framed as a planning problem on a vector field: given a robot's configuration space, a tangent vector indicating the preferred direction of movement, e.g. the direction of the wind or current, or the preferred direction of cutting or slicing through a fibrous medium, is assigned to each point of the configuration space. The resulting collection of vectors attached to the configuration space then constitutes a vector field. Naturally the most efficient thing to do is to always move along the direction of the vector field. Doing so, however, may make it impossible to reach the goal configuration.

Despite the ubiquity of the problem, the literature on path planning on vector fields is mostly confined to low-dimensional, application-specific contexts. Existing algorithms for optimal path planning on vector fields are primarily grid-based, and often make use of level set (Lolla et al., 2012) or fast marching methods (Sethian, 1999; Petres et al., 2007). These deterministic methods have proven to be quite effective in low-dimensional spaces, but in higher-dimensional spaces their efficiency degrades significantly. Among wavefront methods, the spatiotemporal wavefront algorithm of Thompson et al. (2010) and the sliding wavefront expansion algorithm of Soulignac (2011) both seek the minimum time path to some spatial location in a

Robotics Laboratory, College of Engineering, Seoul National University, Daehak-dong, Gwanak-gu, Seoul, Republic of Korea

Corresponding author:
Inyoung Ko, Robotics Laboratory, College of Engineering, Seoul National University, Daehak-dong, Gwanak-gu, Seoul, Republic of Korea.
Email: iyko2001@gmail.com

given current field, but like the previous methods are deterministic, and less effective in high-dimensional settings.

In the randomized path planning literature, the work of Jaillet et al. (2010) comes closest to addressing the general vector field planning problem addressed in this paper. Here a potential function is assumed pre-defined on the configuration space; in this case the preferred direction of motion is identified with the negative gradient of the potential function, and the resulting vector field is by construction always conservative. Given a path on the configuration space, a mechanical work-like cost is then associated with the path, in which only those segments of the path along which the configuration space potential value is increasing contribute to the net mechanical work. This criterion is then used in a randomized motion planning setting, in which a rapidly exploring random tree (RRT) is constructed in a way such that random nodes that lower the overall mechanical work are generated.

More generally, the rapidly exploring random tree (RRT) (LaValle, 1998; LaValle and Kuffner, 2001) has become an essential component of randomized motion planning algorithms, not only because of its simplicity and efficiency, but also because of the ease with which it can be modified to address a wide range of planning objectives and constraints. The efficiency of the basic RRT algorithm is derived in part by settling for any feasible path that is obtained, regardless of the final quality of the path. To obtain better quality paths, in principle one could, e.g., use a path obtained from a basic RRT planner as an initial feasible trajectory to some optimization procedure. Such an approach not only forfeits the inherent efficiency of sampling-based planning, but there is no guarantee that the optimization procedure—which is likely highly nonlinear and complicated—will converge to a better path.

Recent work in the literature has instead attempted to add elements of optimization to the random sampling stage in the basic RRT algorithm, while still preserving the efficiency and random exploratory characteristic of RRTs. Existing methods can be loosely classified into two basic approaches: (a) biased sampling and (b) tree refinement. Biased sampling approaches generate new random nodes that are biased toward a preferred direction (usually the direction that lowers the path cost up to the current node). In Urmson and Simmons (2003), the integral path cost from the initial to the newly sampled node is heuristically estimated, and a node acceptance probability is assigned based on this estimated cost. Other notable features include the replacement of the *NearestNeighbor* function by *k*-nearest neighbors to improve performance in narrow passageways. In Ferguson and Stentz (2006), adjustable distance and cost bias parameters, and a range of node extension methods, are introduced into the basic framework described in Urmson and Simmons (2003); this leads to an algorithm that generates a series of RRTs, in which each tree reuses information from previous trees to iteratively improve the path quality.

The algorithm of Jaillet et al. (2010) based on the notion of mechanical work is another example of a biased sampling method. Here the probability of node creation decreases exponentially with increasing mechanical work, and the statistical test for acceptance (or transition test, as referred to in the paper; the algorithm is subsequently called the transition RRT, or T-RRT, algorithm) is based on a Metropolis criterion similar to those used in simulated annealing algorithms. To improve performance in narrow, low-potential value regions, the GradienT-RRT of Berenson et al. (2011) enhances the basic algorithm of Jaillet et al. (2010) by moving a fixed amount along the gradient direction and re-applying the transition test. Both Jaillet et al. (2010) and Berenson et al. (2011) demonstrate improved performance over earlier methods that heuristically estimate the cost. In practice, however, the transition test can reject potential nodes that lie on or near the optimal path, limiting the exploration efficiency. More fundamentally, qualitative features of paths that minimize mechanical work, while claimed to be natural and intuitive, are not always so, as we elaborate on later.

Tree refinement methods modify the branches within some fixed radius of a given node, so as to lower the overall cost associated with the final path. Representative of these methods is the RRT* algorithm of Karaman and Frazzoli (2011), which dynamically adjusts the tree by changing the parent nodes of select branches. RRT* is the first algorithm in which the generated path provably converges to the optimal one as the number of samples increases to infinity. In Alterovitz et al. (2011), optimal paths are found by converting into roadmaps those subtrees most likely to contain the final path. While both Karaman and Frazzoli (2011) and Alterovitz et al. (2011) have the appealing property of converging to the optimal path as the number of samples increases, in practice the optimization proceeds at a very slow rate, particularly in the early stages when there are only a few branches in the configuration space. As Karaman and Frazzoli (2011) also reports, in certain scenarios the running time of RRT* can be up to 30 times longer than that of basic RRT. Tree refinement approaches hold much promise and are useful when finding nearly optimal paths takes precedence over real-time planning, but for planning problems where computation times are important, biased sampling methods generally tend to be more efficient.

While it is possible to relate some of the above approaches to our vector field planning problem, by, e.g., taking the negative gradient direction of the potential function to be the vector field, only conservative vector fields can be handled in such cases. Moreover, most of the algorithms contain elements that are tightly bound to the RRT framework. What is needed before developing an alternative sampling-based vector field planning algorithm, we believe, is a more thorough and rigorous analysis of possible performance criteria for planning on vector fields.

As the first contribution of this paper, we formalize the vector field planning problem in an optimal control setting, and propose a mathematical criterion that measures the extent to which a certain direction deviates from the

preferred direction as specified by the vector field. This path integral functional, which we call the **upstream criterion**, is smoothly dependent on the path (provided the given vector field is smooth), and is a direct consequence of the Cauchy–Schwarz Inequality. Minimum upstream paths possess a number of desirable mathematical and physical properties:

- Minimum upstream paths can be identified with minimum control effort paths in the following sense (here we take the configuration space to be $\mathbb{R}^n$ with local coordinates $q \in \mathbb{R}^n$): Given a vector field $f(q)$, minimum upstream paths minimize the functional $\int \|u(t)\|^2 dt$, where $u(t)$ is the control term in the state equation $\dot{q} = f(q) + u$. The vector field $f(q)$ here is appropriately regarded as a drift term; the control u then reflects the effort expended to steer the system in directions other than the drift.
- For conservative vector fields, unique paths are produced, and the paths are also reversible, i.e. the path between two configurations is unique without regard to which is designated the start configuration.

We next propose a sampling-based planning algorithm based on the upstream criterion, in which random sampling is biased to favor those nodes that minimize the upstream criterion. Whereas existing optimality criterion-based planning algorithms have the k-nearest neighbor searching or an explicit test in which randomly sampled nodes are rejected or accepted, in our method the randomly sampled node is adjusted toward the vector field direction. As a result the algorithm is more efficient; computational times are of the same order as the basic RRT algorithm, while producing trajectories that are closer to being optimal.

The remainder of the paper is organized as follows. In Section 2 we define the upstream criterion and establish its mathematical and physical properties. Section 3 describes our sampling-based planning algorithm based on the upstream criterion. Section 4 presents results of numerical experiments with our algorithm, including comparisons with alternative planning algorithms on vector fields. Section 5 presents a smoothing procedure for paths produced by our algorithm. Conclusions and possible extensions are discussed in Section 6.

2. Upstream criterion

To keep things simple we introduce concepts and notation in a Euclidean space setting; remarks on the generalization to curved configuration spaces are given later in the paper. Let $\mathbb{R}^n$ denote n-dimensional Euclidean space equipped with the standard Euclidean inner product $\langle \cdot, \cdot \rangle$. Let $\mathcal{Q} \subseteq \mathbb{R}^n$ denote the n-dimensional configuration space manifold, $\mathcal{Q}_{\text{obs}} \subseteq \mathcal{Q}$ the region occupied by obstacles, and $\mathcal{Q}_{\text{free}} \subseteq \mathcal{Q}$ the free space. The tangent space of $\mathcal{Q}$ is denoted $T\mathcal{Q}$. Given initial and final configurations q_{init} and q_{final}, let $q : [0, L] \to \mathcal{Q}_{\text{free}}$ be an arc-length parametrized continuous curve connecting q_{init} and q_{final} (i.e. L is the arclength, $q(0) = q_{\text{init}}$ and $q(L) = q_{\text{final}}$, and the unit-speed condition $\|dq/ds\| = 1$ holds at all points on the arclength-parametrized curve $q(s)$). We further assume a piecewise continuous vector field $f : \mathcal{Q} \to T\mathcal{Q}$ is given that describes, at each point in the configuration space, an optimal or desired direction of motion. Recall that if f is conservative, i.e. there exists some continuous and piecewise $\mathcal{C}^1$ function $V : \mathcal{Q} \to \mathbb{R}$ such that $f(q) = -\nabla V(q)$, then any line integral of f is independent of path, i.e.

$$\int_0^L \langle f(q(s)), q'(s) \rangle \, ds = V(q(L)) - V(q(0))$$

We now recall the Cauchy–Schwarz Inequality: for any $u, v \in \mathbb{R}^n$,

$$|\langle u, v \rangle| \leq \|u\| \cdot \|v\|$$

with equality holding if and only if u and v are linearly dependent. As is well known, the Cauchy–Schwarz Inequality follows directly by minimizing $\langle u,v \rangle^2$ subject to the constraints $\|u\| = \|v\| = \text{constant}$. The upstream criterion is a natural measure of the extent to which a path $q(s)$ goes against the vector field $f(q(s))$; that is, identifying the vector field $f(q(s))$ with u and $q'(s) = \frac{dq}{ds}$ with v, the upstream functional is defined as follows:

Definition 2.1. *Given a unit-speed path $q : [0, L] \to \mathcal{Q}_{\text{free}}$ and a piecewise continuous vector field $f : \mathcal{Q} \to T\mathcal{Q}$, the* **upstream criterion** *$\mathcal{U}(q)$ is defined as follows:*

$$\mathcal{U}(q) = \int_0^L \left(\|f(q(s))\| - \langle f(q(s)), q'(s) \rangle \right) ds \qquad (1)$$

where $\|\cdot\|$ is the norm induced from $\langle \cdot, \cdot \rangle$.

The first term of the integrand is a result of the unit speed assumption $\|q'(s)\| = 1$.

We now show that paths that minimize the upstream functional can be regarded as minimum control effort paths in the following sense.

Proposition 2.2. *Given the system $\dot{x} = f(x) + u$, where $x(t) \in \mathbb{R}^n$ and $f : \mathbb{R}^n \to \mathbb{R}^n$ is given, suppose the trajectory $x^*(t)$ minimizes*

$$\mathcal{J} = \frac{1}{2} \int_0^{t_f} \|u(t)\|^2 dt \qquad (2)$$

for $x(0)$, $x(t_f)$ given and t_f free. Let the curve $q^*(s) \in \mathbb{R}^n$ be obtained by reparametrizing $x^*(t)$ with respect to arclength. Then $q^*(s)$ minimizes the upstream functional of equation (1).

Proof. The Hamiltonian H is

$$H = \frac{1}{2} \|u\|^2 + \lambda^{\mathrm{T}} (u + f(x))$$

The optimality condition $\frac{\partial H}{\partial u} = 0$ implies $\lambda = -u$, so

$$H = -\frac{1}{2}\|u\|^2 - u^{\mathrm{T}} f(x) = 0 \tag{3}$$

Since the final time t_f is free and t does not appear explicitly in H, it follows that $H = 0$. Substituting $u = \dot{x} - f(x)$ into (3),

$$H = -\frac{1}{2}\|\dot{x}\|^2 + \frac{1}{2}\|f(x)\|^2 = 0$$

from which it follows that $\|\dot{x}\| = \|f(x)\|$. Substituting $u = \dot{x} - f(x)$ into the objective function (2),

$$\mathcal{J} = \int_0^{t_f} \left(\frac{1}{2}(\|\dot{x}\|^2 + \|f(x)\|^2) - \dot{x}^{\mathrm{T}} f(x)\right) dt \tag{4}$$

From $\|\dot{x}\| = \|f(x)\|$, equation (4) can be written in the equivalent form

$$\mathcal{J} = \int_0^{t_f} \left(\|f(x)\|\|\dot{x}(t)\| - \dot{x}(t)^{\mathrm{T}} f(x)\right) dt \tag{5}$$

Now recall that the arclength $s(t)$ is given by

$$s(t) = \int_0^t \|\dot{x}\| dt$$

and its inverse $t(s)$ is well-defined. Define $q(s) = x(t(s))$; then $q'(s) = \frac{dx}{dt}\frac{dt}{ds}$, or $\dot{x} = q'(s)\dot{s}$. Under the change of coordinates $t \mapsto s(t)$, the integral (5) becomes the minimum upstream functional of equation (1). Therefore, if $x^*(t)$ minimizes the system (2), then $q^*(s) = x^*(t(s))$ minimizes the corresponding upstream functional. □

The remaining propositions assume the vector field $f(q)$ is conservative, i.e. that there exists some continuous and piecewise $\mathcal{C}^1$ function $V : \mathbb{R}^n \to \mathbb{R}$ such that $f(q) = -\nabla V(q)$. The next proposition shows that over conservative vector fields, minimum upstream paths follow the "path of least resistance":

Proposition 2.3. *Let $f(q)$ be a conservative vector field, and $q^*(s)$ be a unit-speed curve of length L that minimizes the upstream functional. Then for any arbitrary unit-speed curve $q(s)$ with the same endpoints as $q^*(s)$, the following holds*:

$$\int_0^L \|f(q^*(s))\| \, ds \leq \int_0^L \|f(q(s))\| \, ds$$

Proof. Recalling that the upstream functional is

$$\mathcal{U}(q) = \int_0^L \|f(q(s))\| ds - \int_0^L \langle f(q(s)), q'(s)\rangle \, ds$$

The second term corresponds to the negative work $W(q)$, which results from the assumption that $f(q)$ is conservative is path-independent. □

The next proposition shows that under a constant norm vector field, minimum upstream paths correspond to minimum length paths.

Proposition 2.4. *If the conservative vector field $f(q)$ is of constant norm everywhere, then the minimum upstream path is a minimum length path*.

Proof. Since $\|f(q)\|$ is constant everywhere,

$$\mathcal{U}(q) = \|f\| \int_0^L ds - W(q)$$

from which the proposition follows.

The next proposition shows that the reverse path (i.e. the path is traversed in reverse, from q_{final} to q_{init}) of a minimum upstream path is also a minimum upstream path.

Proposition 2.5. *Let $q_1(s)$ and $q_2(s)$ be two unit-speed curves of lengths $L_1$and L_2, respectively, both connecting the same pair of endpoints (q_{init}, q_{final}). Let $r_1(s) = q_1(L_1 - s)$ and $r_2(s) = q_2(L_2 - s)$. Let $f(q)$ be a conservative vector field, and let $\mathcal{U}(q_i)$ be the upstream functional evaluated for path q_i, $i = 1, 2$. If $\mathcal{U}(q_1) > \mathcal{U}(q_2)$, then $\mathcal{U}(r_1) > \mathcal{U}(r_2)$.*

Proof. Using the notation $I(q_i) = \int_0^{L_i} \|f(q_i(s))\| \, ds$, we have

$$\mathcal{U}(r_i) = I(r_i) + W(r_i)$$

where $W(r_i)$ denotes the work associated with the reverse path r_i. Clearly $I(r_i) = I(q_i)$. On the other hand $W(r_i) = V(q_{\text{final}}) - V(q_{\text{init}}) = -W(q_i)$, from which it follows that $\mathcal{U}(r_i) = I(q_i) - W(q_i) = \mathcal{U}(q_i) - 2W(q_i)$. Since $W(q_i)$ is constant over a conservative vector field, the proposition now follows. □

We now provide several examples that highlight the differences between minimum upstream paths and those that minimize the usual physics notion of work, and also the mechanical work criterion of Jaillet et al. (2010). In the examples below, the configuration space is the two-dimensional x–y plane, and a vector field $f(x, y)$ is defined for each configuration point (x, y). For conservative vector fields, $f(x, y)$ is expressed as the negative gradient of the potential function $V(x, y)$.

Recall that for conservative vector fields, the usual notion of work is unable to distinguish between paths that have the same endpoints—the work is independent of paths and given by $V(q_{\text{init}}) - V(q_{\text{final}})$. The mechanical work criterion of Jaillet et al. (2010) overcomes some of the deficiencies associated with the standard physics notion of work. Given a continuous and piecewise $\mathcal{C}^1$ potential function $V : \mathbb{R}^n \to \mathbb{R}$, set $f(q) = -\nabla V(q)$, and define the mechanical work $W_m(q)$ of a path q as

$$W_m(q) = \int_0^L \sigma(q(s)) \, ds + \epsilon \int_0^L ds \tag{6}$$

where ϵ is some arbitrarily chosen constant, and

$$\sigma(q(s)) = \begin{cases} \langle -f(q(s)), q'(s) \rangle & \text{if } \langle -f(q(s)), q'(s) \rangle > 0 \\ 0 & \text{otherwise} \end{cases} \tag{7}$$

While Jaillet et al. (2010) does not explicitly do so, it is straightforward to generalize the mechanical work criterion to nonconservative vector fields, simply by replacing $-\nabla V(q)$ by the given nonconservative vector field. Note that the mechanical work criterion does not depend smoothly on the path.

In the first example, we consider the problem of swimming across a river, in which currents in the middle of the river are the strongest. The vector field is then given by this nonconservative and piecewise continuous current flow, and we ask what is the most efficient way to swim across the river to some designated point on the opposite bank. Intuitively, one would likely try to traverse more directly across the width of the river in regions where the current is relatively weaker. From Figure 1, this is precisely what the minimum upstream path does. In contrast the minimum mechanical work path is always a straight line regardless of the relative strengths of the current.

In the second example, we consider the gradient field of the piecewise-planar potential function of Figure 2(a); the robot moves from a high potential value configuration to a low potential value configuration. Using the river analogy, this time the current strength varies not across the width, but as one goes downstream, as, e.g., in a whitewater rapid. The black line segment depicts the path that minimizes mechanical work, while the red path, consisting of three connected line segments, represents the minimum upstream path.

In the third example, the potential function is given by the arbitrary surface of Figure 3, with the endpoints separated by a series of hills and valleys. It can be seen that the minimum mechanical work path (indicated in Figures 3(a) and 3(c)) goes against the vector field much more so than the minimum upstream path (indicated in Figures 3(b) and 3(d)).

In the final example we consider the nonconservative rotational vector field of Figure 4. The minimum work and mechanical work paths are indicated in blue and black, respectively, while the minimum upstream path is indicated in red. Here as in the previous examples, the minimum upstream path tends to be more aligned with the vector field than the minimum mechanical work path.

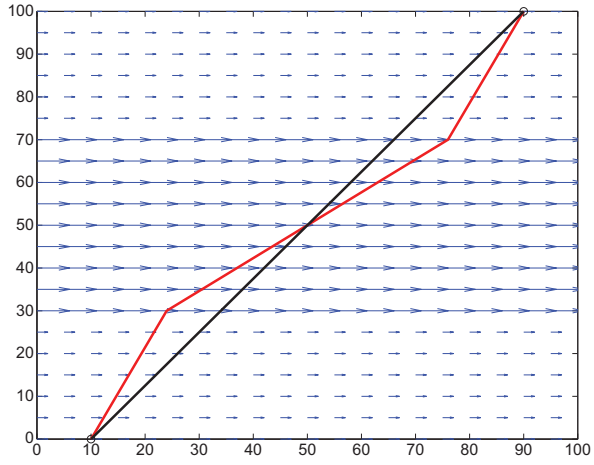


Fig. 1. River crossing problem: the minimum work path is not unique, while the minimum mechanical work path (black) is a straight line. The minimum upstream path (red) is piecewise linear.

3. Algorithm for randomized planning on vector fields

We now present a randomized path planning algorithm based on the upstream criterion. The basic structure of the algorithm is similar to the *ExtExt* version of the standard RRT algorithm presented in LaValle and Kuffner (2001), but each randomly sampled node is adjusted toward the vector field direction. Existing variants of the RRT algorithm that attempt to find better quality paths often include a node acceptance–rejection test or a nearest-k-nodes search step, which often degrade computational efficiency. Our proposed algorithm, which we call the Vector Field Rapidly Exploring Random Tree (VF-RRT), improves computational efficiency primarily by eliminating these steps. We first describe the structure and biased sampling method of our algorithm on a unit magnitude vector field (i.e. the vectors are all of unit magnitude), then extend the algorithm to more general vector fields. In what follows we assume some familiarity with the most basic version of the RRT algorithm.

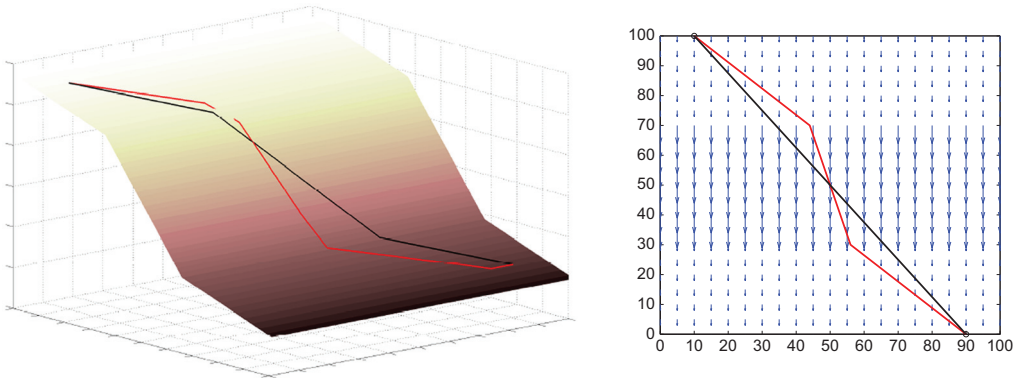
3.1. VF-RRT algorithm for unit magnitude vector fields

3.1.1. Algorithm structure. We assume a unit magnitude vector field $f(q)$ (i.e. $\|f(q)\| = 1$ for all q) defined on some configuration space, with the initial configuration q_{init} taken to be the root node of the tree, and the final configuration q_{final} given. Referring to Figure 5, given a random node q_{rand}, the nearest tree node from this random node is denoted q_{near}; the unit vector directed from q_{near} toward q_{rand} is then denoted $\hat{v}_{\text{rand}}$. We further define three additional unit vectors:

1. $\hat{v}_{\text{field}} = f(q_{\text{near}})$ is the vector field at q_{near}.
2. $\hat{v}_{\text{new}}$ is the direction in which the tree extends.
3. $\hat{v}_{\text{input}}$ is a unit vector lying on the plane spanned by $\hat{v}_{\text{rand}}$ and $\hat{v}_{\text{field}}$ such that for some appropriately chosen scalar $\omega \in \mathbb{R}$, the following equality holds:

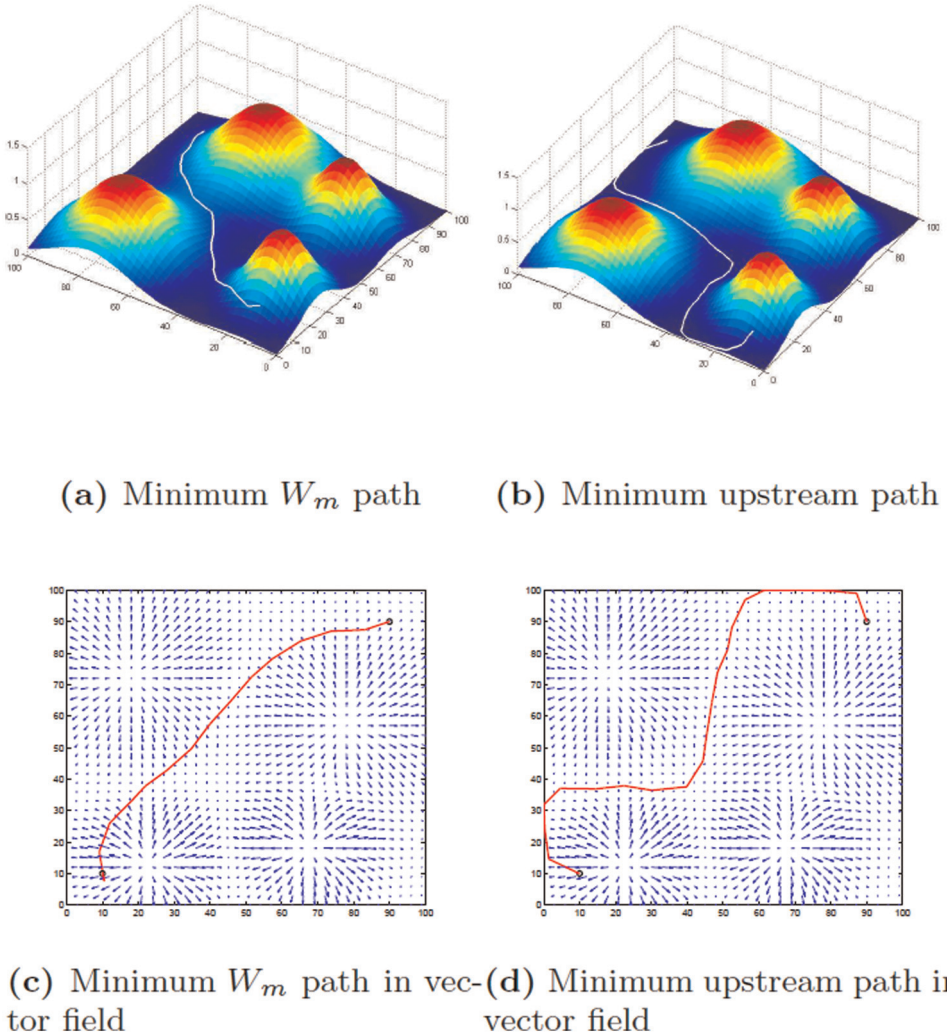
$$\hat{v}_{\text{new}} = \hat{v}_{\text{field}} + \omega \hat{v}_{\text{input}} \tag{8}$$

Now, given the time-parametrized system $\dot{x} = f(x) + u$ as in Proposition 2.2, let $s(t)$ denote the arclength s as a function of t. The arclength parametrized path $q(s) = x(t(s))$ is then governed by



(a) Surface of potential function (b) x–y plane

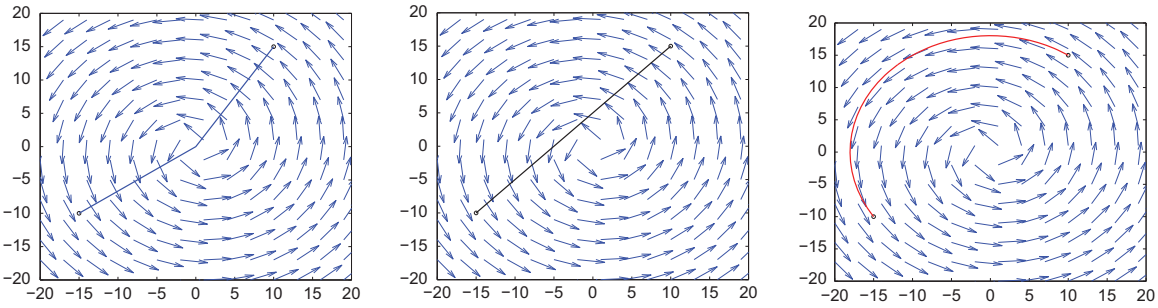
Fig. 2. The minimum mechanical work path (black) projects to a straight line in configuration space (the $x-y$ plane). The minimum upstream path (red) is piecewise linear.



(a) Minimum W_m path (b) Minimum upstream path

(c) Minimum W_m path in vector field (d) Minimum upstream path in vector field

Fig. 3. The potential value is strictly monotone decreasing along the minimum mechanical work path, while the minimum upstream path traverses lower potential value regions.



(a) Minimum work path (b) Minimum W_m path (c) Minimum upstream path

Fig. 4. Nonconservative (rotational) vector field: the minimum work path (blue) is piecewise linear, the minimum mechanical path (black) is a straight line, while the minimum upstream path (red) is aligned with the vector field.

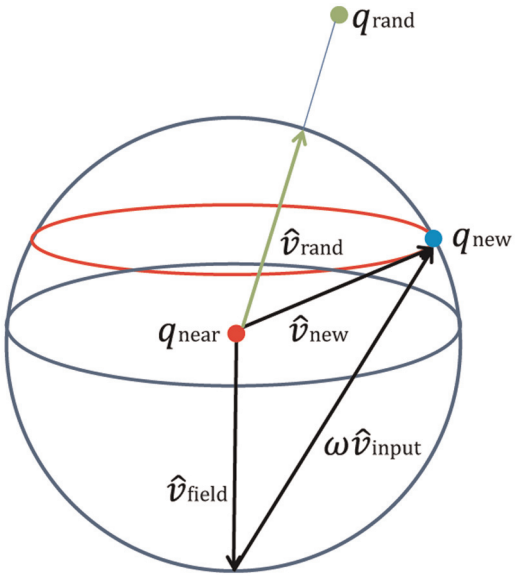


Fig. 5. The new node is given by the sum of $\omega\hat{v}_{\text{input}}$ and $\hat{v}_{\text{field}}$.

$$q'(s) = f(q) + u \tag{9}$$

with $\|q'(s)\| = 1$. Comparing (8) with (9), we can identify $\hat{v}_{\text{new}}$ and $\hat{v}_{\text{field}}$ with $q'(s)$ and $f(q)$, respectively, and $\omega\hat{v}_{\text{input}}$ with the control u.

With respect to (8), we now consider a path of length L, formed by the sequence of unit vectors

$$\hat{v}^i_{\text{new}} = \hat{v}^i_{\text{field}} + \omega_i\hat{v}^i_{\text{input}}, \quad i = 1, \ldots, L \tag{10}$$

The objective function (2) of this path can then be expressed as

$$\mathcal{J} = \frac{1}{2}\sum_{i=1}^{L}\omega_i^2 \tag{11}$$

The problem of finding the optimal path as described in Proposition 2.2 can then be formulated in the following way: given the vector field $\hat{v}_{\text{field}} = f(q)$, find the sequence $\{\omega_i\}$ and $\{\hat{v}^i_{\text{input}}\}$ so as to minimize $\mathcal{J}$ while satisfying (8) and the constraints $\|\hat{v}^i_{\text{new}}\| = \|\hat{v}^i_{\text{input}}\| = 1$, $i = 1, \ldots, L$. From the unit magnitude constraints it follows that each ω_i must lie in the interval $\omega_i \in [0, 2]$. Moreover, once ω_i has been determined, we choose the unit vectors $\hat{v}^i_{\text{input}}$ and $\hat{v}^i_{\text{new}}$ to lie in the span of $\hat{v}^i_{\text{rand}}$ and $\hat{v}^i_{\text{field}}$ (see Figure 5). If we further restrict the $\hat{v}^i_{\text{rand}}$ component to be positive, then $\hat{v}^i_{\text{new}}$ and $\hat{v}^i_{\text{rand}}$ are obtained uniquely. That is,

$$\hat{v}^i_{\text{new}} = \alpha\hat{v}^i_{\text{field}} + \beta\hat{v}^i_{\text{rand}} \tag{12}$$

with $\|\hat{v}^i_{\text{new}}\|^2 = \alpha^2 + \beta^2 + 2\alpha\beta\langle\hat{v}^i_{\text{field}}, \hat{v}^i_{\text{rand}}\rangle - 1 = 0$ and $\beta \geq 0$. From equations (10) and (12), both $\hat{v}^i_{\text{new}}$ and $\hat{v}^i_{\text{input}}$ are then determined uniquely from sampled ω_i.

Given q_{init} and q_{final}, the objective is to find a sequence $\{\omega_i, \hat{v}^i_{\text{input}}\}$, $i = 1, \ldots, L$, that connects q_{init} and q_{final} while minimizing (11). Smaller values of ω_i clearly lead to a smaller total path objective function $\mathcal{J}$, and we seek a probabilistic sampling-based method that retains many of the desirable features of the RRT algorithm (e.g. computational efficiency, Voronoi bias) while producing a path that attempts to reduce $\mathcal{J}$.

The complete algorithm is summarized in Algorithms 1 through 4. Algorithm 1 presents the main steps of the algorithm: a random node q_{rand} is sampled, and the nearest tree

Algorithm 1. Vector Field RRT Algorithm.

```
1:  Tree
2:  Tree.AddVertex(q_init)
3:  while i < I_max do
4:      i ← i + 1
5:      q_rand ← RandomSampling ()
6:      q_near ← FindNearestNodeOnTree (Tree, q_rand)
7:      v̂_new ← GetNewDirection(q_near, q_rand, field, Tree)
8:      q_new ← Extend(Tree, q_near, v̂_new)
9:      if Connect (q_new, q_final) then
10:         return Path(Tree)
11:     end if
12:   end while
13:   return NULL
```

Algorithm 2. GetNewDirection (q_1, q_2, field, Tree).

1: $\hat{v}_{\text{rand}} \leftarrow \frac{q_2 - q_1}{\|q_2 - q_1\|}$
2: $\hat{v}_{\text{field}} \leftarrow \text{GetVectorField}(q_1, \text{field})$
3: $\omega \leftarrow \text{BiasedSampling}(\hat{v}_{\text{rand}}, \hat{v}_{\text{field}}, \text{Tree})$
4: $\alpha, \beta \leftarrow \text{Get}\alpha\beta(\omega, \hat{v}_{\text{rand}}, \hat{v}_{\text{field}}, \text{Tree})$
5: $\hat{v} = \alpha\hat{v}_{\text{field}} + \beta\hat{v}_{\text{rand}}$
6: **return** $\hat{v}$

Algorithm 3. BiasedSampling($\hat{v}_1$, $\hat{v}_2$, Tree).

1: $\sigma \leftarrow \text{GetVariate}(\hat{v}_1, \hat{v}_2)$
2: **At each** n-step
3: $\lambda \leftarrow$ UpdateGain (Tree)
4: $\phi \leftarrow$ GetNormalizingFactor (λ)
5: $\omega \leftarrow$ GetWeight (σ, λ, ϕ)
6: **return** ω

Algorithm 4. Extend (Tree, q_1, $\hat{v}$).

1: $q \leftarrow q_1 + \delta\hat{v}$
2: **if** CollisionOccur(q) **then**
3: Tree.nq_{ineff} = Tree.nq_{ineff} + 1
4: **return** NULL
5: **end if**
6: UpdateExplorationEfficiency(Tree, q)
7: Tree.AddVertex(q), Tree.AddEdge (q_1, q)
8: **return** q

node q_{near} from q_{rand} is found, and $\hat{v}^i_{\text{new}}$ determined. The method for determining $\hat{v}^i_{\text{new}}$ is shown in Algorithm 2: $\hat{v}^i_{\text{new}}$ is chosen by sampling ω and calculating (10) and (12). The method for sampling ω_i is presented in Algorithm 3. After determining $\hat{v}_{\text{new}}$ from ω_i, q^i_{new} is obtained using the standard extension procedure from q_{near} as described in Algorithm 4, where δ is taken to be the stepsize parameter in the standard RRT algorithm.

3.1.2. Determination of ω. We propose a method for determining ω based on the following key requirements and ideas:

1. Each $\omega_i \in [0, 2]$ is sampled from some suitably chosen decreasing probability density function.
2. The probability of any given path being generated is inversely proportional to the upstream cost associated with the path. That is, paths with a lower upstream cost should have a higher probability of being generated than those with a higher upstream cost.
3. $\hat{v}^i_{\text{new}}$ should lie in the positive span of $\hat{v}^i_{\text{rand}}$ and $\hat{v}^i_{\text{field}}$. More specifically, $\hat{v}^i_{\text{new}}$ should satisfy $\left\|\hat{v}^i_{\text{rand}} - \hat{v}^i_{\text{field}}\right\| \geq \left\|\hat{v}^i_{\text{new}} - \hat{v}^i_{\text{field}}\right\|$.

With respect to requirements (1) and (2), we make the assumption that the probability of any path being generated decreases exponentially with the associated upstream cost of the path. That is, given a path $\bar{q}(s)$ with associated upstream cost $\mathcal{U}(\bar{q})$, the probability of the path $\bar{q}(s)$ being generated is of the form

$$\text{Prob}(\bar{q}) = \Phi(\lambda) e^{-\lambda \mathcal{U}(\bar{q})}$$

where λ is a positive scalar reflecting the rate of descent, and $\Phi(\lambda)$ is some suitable normalizing factor. The following proposition specifies the exact form of the probability density function for ω_i that meets our earlier requirements.

Proposition 3.1. *Let $z = \omega^2/2$, and define an exponentially decreasing probability density function $\rho(z)$ by*

$$\rho(z) = \phi(\lambda) e^{-\lambda z} \tag{13}$$

where $\phi(\lambda) = \lambda/(1 - e^{-2\lambda})$ is a normalizing factor such that $\int_0^2 \rho(z)dz = 1$. Suppose ω_i, $i = 1, \ldots, L$, are generated by drawing L independent samples of $z_i = \omega_i^2/2$ from $\rho(z)$, and $\hat{v}^i_{\text{field}}$ is constant during the one-step extension. Then the resulting path corresponding to the sequence of samples $\{\omega_1, \ldots, \omega_L\}$ is generated with probability

$$\text{Prob}(\bar{q}) = \Phi(\lambda) e^{-\mathcal{U}(\bar{q})} \tag{14}$$

Proof. The probability of any path $\bar{q}$ of length L being generated becomes

$$\begin{aligned}
\text{Prob}(\bar{q}) &= \prod_{i=1}^{L} \rho(z_i) \\
&= \phi(\lambda)^L e^{-\lambda \sum_{i=1}^{L} z_i} \\
&= \phi(\lambda)^L e^{-\lambda \sum_{i=1}^{L} \frac{\omega_i^2}{2}} \\
&= \phi(\lambda)^L e^{-\frac{\lambda}{2} \sum_{i=1}^{L} \left\|\hat{v}^i_{\text{new}} - \hat{v}^i_{\text{field}}\right\|^2} \\
&= \phi(\lambda)^L e^{-\lambda \sum_{i=1}^{L} (1 - \langle \hat{v}^i_{\text{field}}, \hat{v}^i_{\text{new}} \rangle)} \\
&= \phi(\lambda)^L e^{-\lambda \mathcal{U}(\bar{q})}
\end{aligned} \tag{15}$$

Setting $\phi(\lambda)^L = \Phi(\lambda)$, we obtain the probability (14). □

The only remaining requirement is (3). This can be satisfied by limiting

$$z_i \leq \frac{\left\|\hat{v}^i_{\text{rand}} - \hat{v}^i_{\text{field}}\right\|^2}{2} = z^i_{\max}$$

and ensuring that the above is satisfied when generating exponential random variates. For this purpose it is more convenient to use the cumulative distribution function $F(z)$ for the density (13):

$$F(z) = \frac{\phi(\lambda)(1 - e^{-\lambda z})}{\lambda} = \sigma$$

A closed form expression for the inverse of $F(z)$ exists:

$$z = -\frac{\log\{1-\sigma\lambda\phi(\lambda)^{-1}\}}{\lambda}$$

Exponential variates are typically obtained by uniformly sampling $\sigma \in [0, 1)$. If σ_i is too large, however, requirement (3) may be violated. To ensure that this is not the case, rather than uniform sampling σ_i in this interval, we set

$$\sigma_i = \frac{z^i_{\max}}{2}$$

which then guarantees that $\left\|\hat{v}^i_{\text{rand}} - \hat{v}^i_{\text{field}}\right\| \geq \left\|\hat{v}^i_{\text{new}} - \hat{v}^i_{\text{field}}\right\|$ is satisfied.

3.1.3. Determination of λ. The choice of the parameter $\lambda \in (0, \infty)$ in the exponential probability density $\rho(z)$ of equation (13) determines the rate of descent of the density function. For larger values of λ, the density decreases more sharply, resulting in trees that tend to more closely follow the drift vector field. One way to ensure that trees do not bunch into overly narrow regions (and compromising the Voronoi bias property of RRTs) is to adaptively adjust the value of λ according to the current status of the tree. Toward this end, we introduce the notion of a tree's exploration inefficiency. A candidate node q is classified as being efficient or inefficient based on its distance from the current tree; that is, if q is a collision free node and

$$\text{Dist}(\text{Tree}, q) \geq \delta$$

for some prescribed distance δ (for our purposes δ can be taken to be the standard RRT stepsize parameter), where Dist (Tree, q) denotes the minimum distance between q and the tree, then the node is classified as being efficient; all other remaining candidate nodes are classified as inefficient (Figure 6; Algorithm 5). A tree's exploration inefficiency E_{ineff} is then defined as follows:

$$E_{\text{ineff}} = \frac{\text{Number of inefficient nodes}}{\text{Number of total candidate nodes}}$$

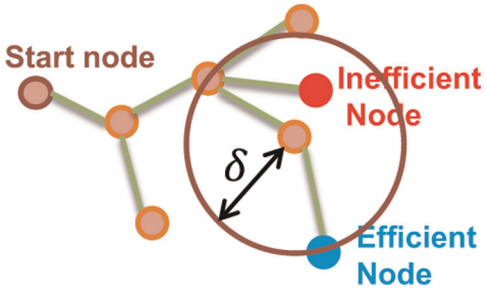


Fig. 6. Efficient and inefficient nodes. If a new node is within some prescribed distance δ of the tree, it is classified as inefficient; otherwise it is an efficient node.

Algorithm 5. UpdateExplorationEfficiency (Tree, q).

1: $q_{\text{near}} \leftarrow$ FindNearestNodeOnTree(Tree, q)
2: **if** Distance(q_{near}, q) $< \delta$ **then**
3: Tree.nq_{ineff} = Tree.nq_{ineff} + 1
4: **else**
5: Tree.nq_{eff} = Tree.nq_{eff} + 1
6: **end if**
7: Tree.E_{ineff} = $\frac{\text{Tree.n}q_{\text{ineff}}}{\text{Tree.n}q_{\text{eff}}}$ + Tree.nq_{ineff}

Algorithm 6. UpdateGain (Tree).

1: $\lambda \leftarrow \lambda$ (1 − Tree.E_{ineff} + E_s)
2: Tree.nq_{eff}, Tree.nq_{ineff}, Tree.E_{ineff} = 0
3: **return** λ

The value for λ can now be adjusted according to the changing value of E_{ineff} as the tree progresses. Given a user-specified reference exploration inefficiency factor $E_s \in (0, 1)$, λ is then updated according to Algorithm 6: λ is increased when $E_{\text{ineff}} < E_s$, and decreased when $E_{\text{ineff}} > E_s$.

3.2. VF-RRT Algorithm on General Vector Fields

Thus far our development of the VF-RRT algorithm has been restricted to unit magnitude vector fields. Our algorithm can be generalized to the more general case of non-unit magnitude vector fields in a number of ways. One straightforward way is to simply scale the parameter λ, to $\lambda' = \lambda\|f(q)\|/m$, where m denotes the average value of $\|f(q)\|$ over the configuration space region of interest. For our purposes an approximate estimate of m is sufficient, which can be obtained by, e.g., Monte Carlo sampling the magnitude of $f(q)$ over the region of interest. The vector field $f(q)$ is transformed to a unit vector field by standard normalization, i.e. $\hat{v}_{\text{field}} = f(q)/\|f(q)\|$.

The basis for this particular generalization of the VF-RRT algorithm is that path probabilities still remain exponential with respect to the local upstream criterion. To illustrate, without loss of generality set $m = 1$ and λ to some constant value. Equation (15) then becomes

$$\begin{aligned}
\text{Prob}(\bar{q}) &= \prod_{i=1}^{L} \phi(\lambda'_i) e^{-\lambda\|f(q_i)\|(1-\langle\hat{v}^i_{\text{field}}, \hat{v}^i_{\text{new}}\rangle)} \\
&= \left[\prod_{i=1}^{L} \phi(\lambda'_i)\right] e^{-\lambda\sum_{i=1}^{L}\|f(q_i)\|(1-\langle\hat{v}^i_{\text{field}}, \hat{v}^i_{\text{new}}\rangle)} \\
&= \left[\prod_{i=1}^{L} \phi(\lambda'_i)\right] e^{-\lambda\mathcal{U}(\bar{q})} \\
&= \Phi'(\lambda') e^{-\lambda\mathcal{U}(\bar{q})}
\end{aligned}$$

The lower upstream L-length path does not guarantee the higher probability of being generated any more.

Since the value of $\Phi'(\lambda')$ is path-dependent, a path with a lower upstream value $\mathcal{U}(q)$ does not necessarily imply a

Algorithm 7. GetNewDirection (q_1, q_2, field, Tree) (general case).

1: $\hat{v}_{\text{rand}} \leftarrow \frac{q_2 - q_1}{\|q_2 - q_1\|}$
2: $f \leftarrow$ GetVectorField (q_1, field)
3: $\hat{v}_{\text{field}} \leftarrow \frac{f}{\|f\|}$
4: $\omega \leftarrow$ BiasedSampling($\hat{v}_{\text{rand}}, \hat{v}_{\text{field}}, f$, Tree)
5: $\alpha, \beta \leftarrow$ Get$\alpha\beta$($\omega, \hat{v}_{\text{rand}}, \hat{v}_{\text{field}}$, Tree)
6: $\hat{v} = \alpha\hat{v}_{\text{field}} + \beta\hat{v}_{\text{rand}}$
7: **return** $\hat{v}$

Algorithm 8. BiasedSampling ($\hat{v}_1, \hat{v}_2, f$, Tree) (general case).

1: $\sigma \leftarrow$ GetVariate($\hat{v}_1, \hat{v}_2$)
2: **At each** n-step
3: $\lambda \leftarrow$ UpdateGain(Tree)
4: $\lambda' \leftarrow$ GainNormalizing(λ, f)
5: $\phi \leftarrow$ GetNormalizingFactor(λ')
6: $\omega \leftarrow$ GetWeight (σ, λ', ϕ)
7: **return** ω

higher probability of being generated. Over each path segment, however, higher upstream edges still have a lower extension probability. Moreover, for larger $\|f(q_i)\|$, the $e^{-\lambda \mathcal{U}(\bar{q})}$ term dominates since $\phi(\lambda'_i)$ increases approximately linearly, whereas $e^{-\lambda \mathcal{U}(\bar{q})}$ decreases exponentially. Therefore, as before, the VF-RRT algorithm still tends to generate low upstream value paths with a higher probability. The modified versions of the functions GetNewDirection and GetWeight for the general vector field case are respectively described in Algorithms 7 and 8.

3.3. Probabilistic completeness

The VF-RRT algorithm inherits the probabilistic completeness property more or less directly from the standard RRT algorithm. In this section we verify the relevant fundamental lemma as given in LaValle and Kuffner (2001) for the VF-RRT algorithm.

Proposition 3.2. *Suppose $\mathcal{Q}_{\text{free}} \subset \mathcal{Q}$ is convex and open. For any configuration $q(s) \subset \mathcal{Q}_{\text{free}}$ and any positive constant ϵ, the probability* Prob (Dist (Tree, $q(s)$)) $< \epsilon$) *approaches unity as the number of iterations goes to infinity.*

Proof. Let q_i be the tree node that is nearest to $q(s)$, and $R(q_i)$ be the region from which the random node is drawn, such that the extended node from q_i is closer to $q(s)$ than q_i. Let $B(R(q_i))$. denote the volume of $R(q_i)$. If the tree can extend toward all directions from q_i, then $B(R(q_i)). > 0$ for any $q(s)$. Recall also that in the VF-RRT algorithm, $\omega \in [0, 2\|f\|]$. q_i can therefore be extended in all directions. Moreover, $B(R(q_i)) > 0$ always regardless of the value of $q(s)$ and $f(q_i)$, implying that the probability that a random point is sampled in $R(q_i)$ is strictly positive. Therefore, as the VF-RRT algorithm iterates infinitely, the probability

$$\text{Prob}(\text{Dist}(\text{Tree}, q(s)) < \epsilon) \rightarrow 1$$

as claimed. □

4. Case studies and discussion

In this section, we present results of numerical experiments with the VF-RRT algorithm. We begin with some simple examples that examine the shape of VF-RRT trees on a nonconservative vector field. We then apply the VF-RRT algorithm to conservative vector fields. We also compare the performance of our VF-RRT algorithm with the basic RRT and T-RRT algorithm (Tables 1–6) (we remind the reader that the VF-RRT minimizes the upstream criterion of (1), while the T-RRT algorithm attempts to minimize the mechanical work of (6)). Both the greedy and tempered versions of the T-RRT algorithm are tested. RRT algorithms for the most part are implemented bi-directionally (that is, the trees are grown simultaneously from the start and goal nodes), with the exception of the comparison with the RRT* algorithm in Section 4.5. In the latter example, the uni-directional versions are compared, with the upstream criterion applied to RRT* as the cost.

For each case study, the relevant mean statistics are obtained for 50 trials. To compare the algorithms in a fair manner, all case studies use the MPNN algorithm of Yershova and LaValle (2007) for finding the nearest neighbor node. Each algorithm is implemented in C++, and executed on a PC with a 3.40GHz Intel(R) core(TM) i7 quad core processor and 8.0GB RAM.

4.1. Nonconservative wind velocity field

The first example involves path planning on a nonconservative two-dimensional wind velocity field. For the tree emanating from the goal node q_{final} in the bi-directional implementation, we set $\hat{v}_{\text{field}}$ to be the reverse of the wind vector direction. As shown in Figure 7, the branches are extended following the direction of the wind. Both the T-RRT algorithm and VF-RRT algorithm find better quality paths than the basic RRT algorithm. The T-RRT algorithm reduces the upstream criterion by only 35.8%, with the tempered version of the T-RRT algorithm showing poorer performance than the greedy version. For the VF-RRT algorithm, path quality improves with increasing E_s, but so does the overall computation time. Comparing the quality of the paths with similar computation times, the upstream criterion for paths produced by VF-RRT tend to be much smaller than those produced by T-RRT.

We also perform further experiments on the statistical test for node acceptance, by executing the T-RRT algorithm with the upstream criterion, i.e. the integrand of the upstream functional $\|f(q(s))\| - \langle f(q(s)), q'(s)\rangle$ is applied to the transition test instead of (7). Comparing the results of

Table 1. Wind velocity field: bi-directional versions.

	RRT	T-RRT		VF-RRT				
		Greedy	Tempered	$E_s = 0.05$	$E_s = 0.25$	$E_s = 0.45$	$E_s = 0.65$	$E_s = 0.85$
Upstream cost ($\mathcal{U}$)	352.8	226.6	229.0	196.5	166.8	145.5	109.0	71.3
Std dev. of $\mathcal{U}$	91.5	28.0	29.9	86.9	70.2	70.9	55.2	11.8
No. iterations	230.7	789.3	1659.8	288.3	596.6	1368.1	3988.7	23,709.7
Path length	189.9	199.1	201.9	187.8	192.4	207.8	223.2	228.1
Time (s)	0.025	0.087	0.182	0.026	0.053	0.196	0.681	4.047

Table 2. T-RRT with upstream criterion: bi-directional version.

	Upstream cost ($\mathcal{U}$)	Std dev. of $\mathcal{U}$	No. iterations	Path length	Time (s)
T-RRT (greedy)	171.0	42.1	21582	222.5	3.63

Table 3. Complex potential surface: bi-directional versions.

	RRT	T-RRT		VF-RRT				
		Greedy	Tempered	$E_s = 0.05$	$E_s = 0.15$	$E_s = 0.25$	$E_s = 0.35$	$E_s = 0.45$
$\int V\,dq$	449.0	364.5	407.6	378.3	358.3	356.8	355.0	374.8
$\mathcal{S}(\int V dq)$	32.96	26.0	74.3	38.9	34.9	27.7	19.1	26.6
Upstream cost ($\mathcal{U}$)	0.900	0.663	0.936	0.825	0.683	0.660	0.666	0.743
Std dev. of $\mathcal{U}$	0.062	0.075	0.365	0.116	0.075	0.054	0.066	0.097
No. iterations	552.2	12,917.0	53,0,368.1	800.4	1043.6	1258.8	1642.6	2178.1
Path length	412.9	497.6	673.1	473.2	513.5	552.3	578.4	627.5
Time (s)	0.086	2.378	78.319	0.364	0.474	0.572	0.704	0.949

Table 2 with Table 1, it can be seen that the obtained paths are of better quality than those obtained using T-RRT with mechanical work, and similar to those obtained with VF-RRT for $E_s = 0.25$. However, it is worth noting that the computational time for T-RRT with the upstream criterion is longer than for the other two methods. For values of E_s greater than 0.25, VF-RRT produces better paths in a shorter amount of time than T-RRT with the upstream criterion. The tempered version of T-RRT fails to find a path even after a million iterations. Unlike mechanical work, the integrand of the upstream functional is always positive. Thus, if T-RRT is run with the upstream criterion, potential nodes are likely to be rejected far more frequently than T-RRT run with mechanical work. These results provide fairly strong evidence that the statistical test for node acceptance is less efficient than for VF-RRT.

4.2. Conservative vector fields

Conservative vector fields are constructed by first defining potential functions $V(q(s))$. on the configuration space; the vector fields are then given by the negative gradient field $f(q(s)) = -\nabla V(q(s))$.

4.2.1. Complex potential surface. We first construct an arbitrary complex two-dimensional potential surface as shown in Figure 8, where the surface height z represents the potential value associated with the corresponding configuration space point (x, y).

The results are presented in Table 3. Here the path integral of the potential function and upstream cost as well as their respective standard deviations, number of iterations, length of the final path, and total computation time are compared for the basic RRT, T-RRT, and VF-RRT algorithms. For this example, the final paths produced by the VF-RRT algorithm and T-RRT are very similar to each other, as are the performance statistics obtained for the VF-RRT and the greedy version of the T-RRT algorithm. Note however that the computation times for VF-RRT are significantly faster than those for T-RRT. The primary reason is that because of the hilly nature of the potential surface, T-RRT rejects an inordinately high number of nodes, considerably increasing the number of iterations and thus the overall computation time.

4.2.2. Spatial robot arm. We now consider a six degree of freedom spatial open chain robot arm, subject to two different sets of vector fields:

Table 4. Vector fields for 6-dof spatial robot arm: bi-directional versions.

	RRT	(a) Unit magnitude attractor vector field.						
		T-RRT		VF-RRT				
		Greedy	Tempered	$E_s = 0.05$	$E_s = 0.25$	$E_s = 0.45$	$E_s = 0.65$	$E_s = 0.85$
Upstream cost ($\mathcal{U}$)	520.2	289.0	208.9	410.0	209.5	114.7	67.5	47.7
Std dev. of $\mathcal{U}$	181.0	169.7	43.3	205.2	104.4	60.5	23.4	9.5
No. iterations	3927.0	53,355.7	143,884.8	3147.6	3070.5	2342.4	729.4	800.1
Path length	255.2	191.8	164.9	220.4	156.0	116.7	97.0	88.4
Time (s)	0.609	5.86	16.69	0.564	0.524	0.541	0.065	0.048
	RRT	(b) Negative gradient vector field for static torque minimization.						
		T-RRT		VF-RRT				
		Greedy	Tempered	$E_s = 0.05$	$E_s = 0.25$	$E_s = 0.45$	$E_s = 0.65$	$E_s = 0.85$
$\int V\, dq$	3296.77	970.4	882.9	1766.0	1400.0	790.6	409.8	346.8
Std dev. of $\int V\, dq$	1104.1	144.2	142.5	767.2	339.5	303.7	55.9	49.0
Upstream cost ($\mathcal{U}$)	7943.7	2451.4	1879.7	4755.1	2958.7	1529.3	731.1	572.0
Std dev. of $\mathcal{U}$	1969.5	603.7	247.0	1833.1	1095.3	592.1	144.3	97.9
No. iterations	3927.0	20,862.3	80,991.7	1338.0	1804.5	2457.1	4829.8	5593.0
Path length	255.2	281.8	213.5	254.0	257.4	246.0	166.1	231.8
Time (s)	0.609	3.84	11.96	0.608	0.773	1.07	2.58	3.31

- **Unit magnitude vector field directed toward the goal node (attractor)**: given a goal node q_{final}, the potential function is defined as

$$V(q(s)) = \|q(s) - q_{\text{final}}\| \tag{16}$$

  and the vector field is taken to be the negative gradient field. For the tree emanating from the goal node toward the start node q_{init}, we take q_{init} to be the goal node and take gradients accordingly. Some obstacles are placed between q_{init} and q_{final} to eliminate the trivial straight-line path as a solution. Since the gradients of (16) are unit vectors, from Proposition 2.4, we can expect the resulting path to be of minimal length.
- **Minimum static joint torque magnitude:** the potential function is taken to be the squared magnitude of the static joint torque vector $\tau(q(s))$:

$$V(q(s)) = \|\tau(q(s))\|^2,$$

  subject to the static equations $\tau = J^{\mathrm{T}}(q)F$, where $J(q)$ is the Jacobian of the forward kinematics, and F is a six-dimensional spatial force exerted at the arm tip corresponding to gravity.

The experimental results are shown in Figure 9 and Table 4. For the attractor vector field, VF-RRT produces shorter paths than both the standard RRT or T-RRT. Notably, VF-RRT takes even less computation time than standard RRT for this case. Also, as E_s is increased, the quality of the path improves as expected, and the computation time also commensurately decreases; at one level this is not surprising given that following the vector field will naturally lead the robot to the goal node.

For the minimum static torque vector field, the paths produced by VF-RRT not only have a lower upstream cost, but also a lower potential function path integral value (for paths where E_s is set to 0.45 or greater). Qualitatively, the path produced by VF-RRT agrees with our intuition; the arm first folds into a more compact configuration before moving, in order to reduce the overall rotational inertia and thus the overall static joint torque at each configuration. Even with E_s set to 0.85 (out of a possible maximum value of 1), computation times for VF-RRT are still lower than those for T-RRT.

4.2.3. Continuum robot. To evaluate the performance of our VF-RRT algorithm for high-dimensional problems, we now consider a continuum robot, modeled as a 14 degree-of-freedom open chain. The vector field is taken to be the negative gradient field of the potential function given in (16), with obstacles placed between q_{init} and q_{final} to eliminate trivial straight line paths as solutions (Figure 10). The T-RRT algorithm is excluded from Table 5, since it failed to produce paths even after several million iterations; for high-dimensional problems T-RRT generates an excessively high number of nodes that in our case exceeded our computer's memory capacity. Whereas the basic RRT algorithm takes between 15 and 20 min to generate a feasible path, a suitably tuned version of the VF-RRT algorithm can produce a path in 0.1–0.5 s. For this problem we observe that the computation time decreases dramatically as E_s is increased from 0.05 to 0.65, but then increases slightly as E_s approaches 0.85.

Table 5. High-dimensional open chain: bi-directional versions.

	RRT	VF-RRT				
		$E_s = 0.05$	$E_s = 0.25$	$E_s = 0.45$	$E_s = 0.65$	$E_s = 0.85$
Upstream cost ($\mathcal{U}$)	1114.2	630.4	137.9	76.1	51.1	39.9
Std dev. of $\mathcal{U}$	292.3	450.2	116.1	13.0	12.9	10.0
No. iterations	195,698.6	130,664.7	36,464.7	11,895.6	1032.1	2109.8
Path length	459.8	439.5	190.9	102.5	89.1	83.2
Time (s)	1019.8	630.4	137.901	28.592	0.134	0.488

Table 6. Path planning for a constrained 6-dof spatial arm: bi-directional versions.

	RRT	VF-RRT				
		$E_s = 0.05$	$E_s = 0.25$	$E_s = 0.45$	$E_s = 0.65$	$E_s = 0.85$
Upstream cost ($\mathcal{U}$)	273.6	218.9	107.7	67.9	47.0	38.7
Std dev. of $\mathcal{U}$	75.8	76.3	58.4	33.0	8.1	6.9
No. iterations	3246.3	1901.4	967.2	596.7	519.6	426.9
Path length	197.9	163.4	115.7	93.4	83.0	76.8
Time (s)	0.650	0.358	0.188	0.106	0.088	0.071

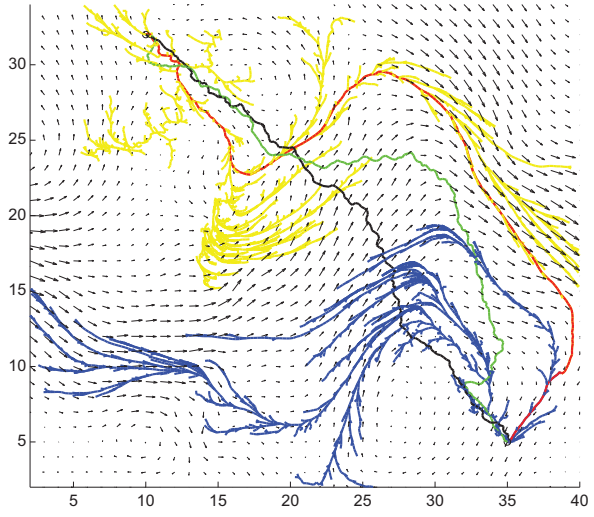


Fig. 7. Bi-directional VF-RRT trees in a two-dimensional wind velocity field. Yellow: A typical tree emanating from the start node. Blue: A typical tree emanating from the goal node. Final paths produced by RRT (black), T-RRT (green) and VF-RRT (red).

4.3. Planning on constraint manifolds

For robots with closed loops, and also for tasks that involve holonomic kinematic constraints, e.g. grasping and moving a tray with two arms, the feasible configuration space becomes a manifold (in the mechanisms and robotics literature such manifolds are typically referred to as constraint manifolds). For planning problems on constraint manifolds, the VF-RRT algorithm requires some modification, since any valid nodes generated must lie on the manifold. This requirement is in fact valid for the standard RRT algorithm and other variants, and several methods have been proposed for generalizing the standard vector space RRT algorithm to constraint manifolds (Berenson et al., 2009; Stilman, 2010; Suh et al., 2011).

The most straightforward way to extend our VF-RRT algorithm to constraint manifolds is to adopt the projection method of Berenson et al. (2009), in which q_{new} is first obtained from $\hat{v}_{\text{new}}$ of (12) without regard to any constraints, and then projected to the constraint manifold; this is done at each iteration. Figure 11 shows the results for a 6-dof arm whose tip is constrained to always maintain contact with a vertical wall. We take the vector field to be the negative gradient field of the potential function given in (16), with both q_{init} and q_{final} assumed to be points on the wall. As shown in Table 6, both the path length and path planning computation times are considerably shorter for the VF-RRT algorithm than for the standard RRT algorithm. We also remark that other constraint manifold RRT planning algorithms, e.g. in Stilman (2010) and Suh et al. (2011), can also be integrated in a similar manner with our VF-RRT algorithm.

4.4. Path planning for nonholonomic systems

The VF-RRT algorithm can also be applied to nonholonomic path planning problems. Here we consider nonintegrable Pfaffian constraints of the form

$$\omega_i(q)\dot{q} = 0, i = 1, \ldots, k$$

where $\omega_i(q)$ are n-dimensional row vectors. For the purposes of this paper we consider the associated drift-free nonholonomic control systems of the form

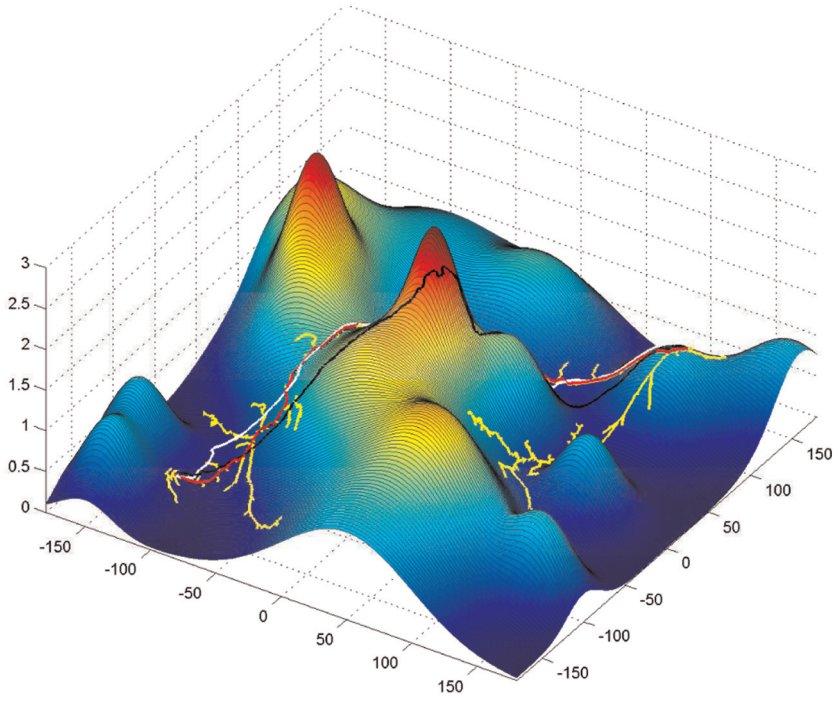
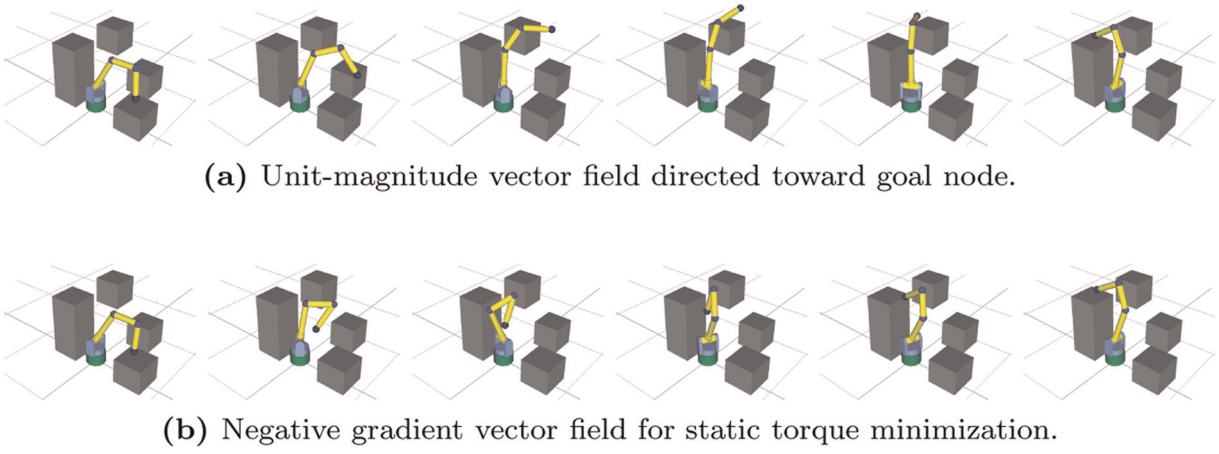


Fig. 8. Bi-directional VF-RRT trees for a complex potential surface. Yellow: A typical tree produced by VF-RRT. Final paths produced by RRT (black), T-RRT (white) and VF-RRT (red).



(a) Unit-magnitude vector field directed toward goal node.

(b) Negative gradient vector field for static torque minimization.

Fig. 9. Spatial robot arm: Typical paths generated by bi-directional VF-RRT for various vector fields.

$$\dot{q} = g_1(q)u_1 + \cdots + g_m(q)u_m \tag{17}$$

where $g_i(q)$, $i = 1, \ldots, m$ are n-dimensional vectors and u_i a scalar input (see e.g. Sastry (1999) and De Wit et al. (1996) for a further description of the properties of such systems). We propose a straightforward way to determine a set of inputs that lead to a direction closer to the vector field than a random direction.

1. q_{new} is first obtained by the standard procedure described in our earlier VF-RRT algorithm. That is, sample a random node in the state space, and determine $\hat{v}_{\text{new}}$ from the calculated ω, α, and β.
2. The inputs required to reach q_{new} can then be determined by a local planner. This local planner solves the optimal control problem for the objective function $\int \|u(t)\|^2 \, dt$ subject to the state equation (17). The initial and final conditions are given by $q(t_0) = q_{\text{near}}$ and $q(t_f) = q_{\text{new}}$, with t_f free. Computational and numerical methods for solving this optimal problem are presented in Sastry (1999) and Kirk (2012). Note that if the tree emanates from the goal node, the previous algorithm is applied with $q(t_0)$ and $q(t_f)$ switched.

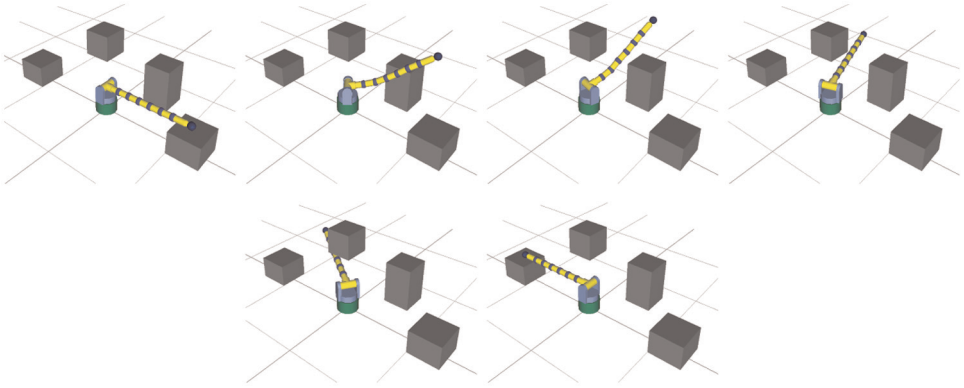


Fig. 10. High-dimensional open chain: Typical path generated by bi-directional VF-RRT for unit-magnitude vector field directed toward goal node.

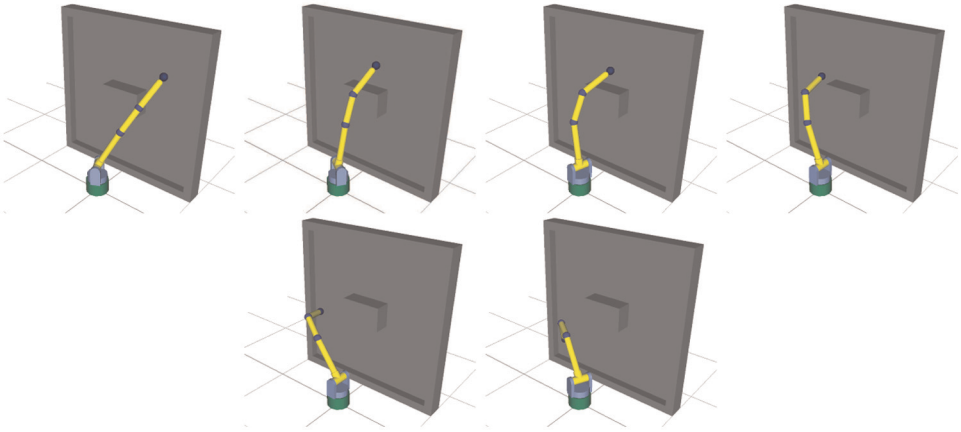


Fig. 11. A 6-dof spatial arm constrained to maintain tip contact with a vertical wall while moving from an initial to goal configuration: Typical path generated by bi-directional VF-RRT for unit-magnitude vector field directed toward goal node.

We now apply VF-RRT to a car-like nonholonomic system with state equations given in the following standard kinematic form:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \\ \dot{\phi} \end{bmatrix} = \begin{bmatrix} \cos\theta \\ \sin\theta \\ \tan\phi/\ell \\ 0 \end{bmatrix} u_1 + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} u_2$$

where $|u_1| \leq u_{\max}$ and $|\phi| \leq \phi_{\max}$ are the driving and steering velocity, respectively, and ℓ is the body length of a car. The path planning experimental results are shown in Figure 12 and Table 7. Compared to the paths produced by a classical RRT algorithm, a suitably tuned VF-RRT planner (that is, with appropriate choice of E_s) reduces the potential function path integral and the upstream criterion by up to 17.3% and 22.8%, respectively. The qualitative features of the path produced by VF-RRT are similar to the paths obtained for the holonomic case of Section 4.2.1.

*4.5. Comparing VF-RRT with RRT**

As noted in the introduction, the RRT* planning algorithm of (Karaman and Frazzoli, 2011) has the desirable feature of eventually converging to the optimal path, but in general the computation times can become quite long. Since VF-RRT attempts to generate good quality paths quickly rather than seeking the optimal path, a direct comparison between the two algorithms must take into consideration the different motivating philosophies. Note also that for certain vector fields such as the unit magnitude attractor field of (16), the VF-RRT algorithm is even faster than the basic RRT algorithm, whereas the RRT* algorithm cannot find a path within a given allotted time. To compare the performance of VF-RRT with RRT*, we therefore measure the quality of the intermediate paths produced by RRT* at intervals of 50,000 iterations. The upstream criterion is applied to the RRT* as a cost, and for a fair comparison we run uni-directional versions of VF-RRT and RRT*, since RRT* is implemented uni-directionally. We consider again the wind velocity field of Section 4.1, since for high-dimensional problems RRT* will take significantly longer to converge. After 350,000 iterations, or approximately 100 min with our computer hardware specifications, RRT* returns a path whose upstream cost is in the range of 140–150, while VF-RRT generates paths with a considerably lower upstream cost in less than 0.5 s.

Table 7. Path planning for a nonholonomic car-like system: bi-directional versions.

	RRT	VF-RRT				
		$E_s = 0.05$	$E_s = 0.15$	$E_s = 0.25$	$E_s = 0.35$	$E_s = 0.45$
$\int V\, dq$	421.3	397.6	359.5	348.5	351.0	377.5
Std dev. of $\int V\, dq$	54.9	34.2	31.7	29.3	29.6	30.6
Upstream cost ($\mathcal{U}$)	0.854	0.800	0.792	0.676	0.659	0.679
Std dev. of $\mathcal{U}$	0.130	0.086	0.101	0.115	0.110	0.076
No. iterations	1041.9	850.3	1421.4	1619.8	2395	3673.5
Path length	371.8	359.7	369.7	416.3	454.4	521.5
Time (s)	0.314	0.411	0.729	0.755	1.148	2.903

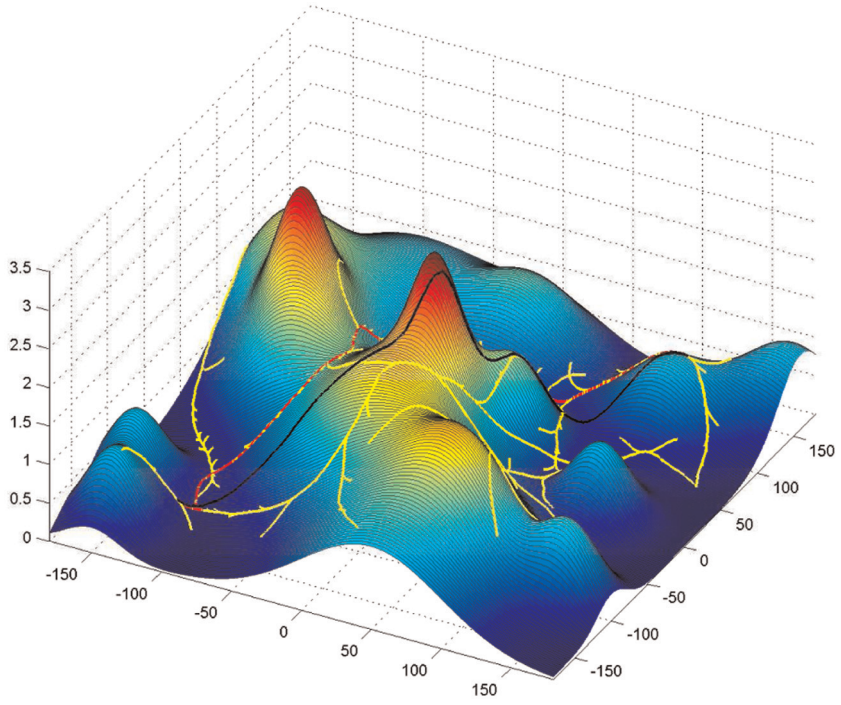


Fig. 12. Bi-directional VF-RRT trees for a car-like nonholonomic system on a complex potential surface. Yellow: A typical tree produced by VF-RRT. Final paths produced by RRT (black) and VF-RRT (red).

*4.6. VF-RRT**

Although the intent and objectives behind VF-RRT and RRT* are different, in cases where computational times are less critical, optimal paths may sometimes be desired. In such cases it is useful to combine VF-RRT with RRT*. More specifically, the sampling strategy of VF-RRT can also be applied to RRT*, by creating q_{new} as done in VF-RRT rather than the conventional RRT method adopted in RRT*.

We perform experiments with VF-RRT* for different values of E_s. For $E_s = 0.85$, the algorithm fails to find a path when the number of iterations is less than 5,000. Comparing Table 8 with Table 9, it can be seen that paths using VF-RRT* converge to the optimal path more quickly than RRT*. It is also possible to get a lower upstream cost path than using VF-RRT with $E_s = 0.85$, although it takes longer to obtain than VF-RRT.

5. Path smoothing

Like most randomized path planning algorithms, the paths generated by the VF-RRT algorithm sometimes contain inefficient intervals that lead to zig-zag paths, particularly when the path must go directly upstream against the vector field. Path smoothing is therefore an essential post-processing step of most randomized path planning algorithms, with methods like the shortcutting technique (Geraerts and Overmars, 2007) widely applied. While shortcutting is particularly useful for, e.g., conservative and constant norm vector fields in which minimum upstream paths are also minimal length (straight lines), for general vector fields where the straight line is not necessarily the best path, better smoothing methods can be developed. We now describe one such smoothing procedure.

Table 8. Comparison VF-RRT and RRT*: Uni-directional versions in a wind velocity field.

	(a) VF-RRT.				
E_S	0.05	0.25	0.45	0.65	0.85
Upstream cost ($\mathcal{U}$)	187.9	115.6	92.8	61.5	59.9
Std dev. of $\mathcal{U}$	49.0	37.1	34.3	9.6	6.7
No. iterations	2001.1	3160.8	4925.8	6091.2	29,093.6
Path length	255.2	156.0	116.7	97.0	88.4
Time (s)	0.163	0.260	0.435	0.683	2.832

	(b) RRT*.						
No. Iterations	50,000	100,000	150,000	200,000	250,000	300,000	350,000
Upstream cost ($\mathcal{U}$)	325.6	321.1	224.5	189.1	179.2	172.8	147.9
Std dev. of $\mathcal{U}$	84.7	57.3	40.9	26.4	17.6	13.2	9.3
Path length	190.7	191.5	198.3	200.1	201.0	201.8	207.5
Time (s)	173.88	683.80	1457.25	2554.71	3449.16	4709.43	6698.32

Table 9. Upstream Values of VF-RRT*: Uni-directional versions in a wind velocity field.

No. Iterations	5,000	10,000	20,000	50,000	100,000	150,000
$E_s = 0.05$	177.6	175.3	169.3	165.9	148.5	128.7
$E_s = 0.25$	115.3	114.8	110.2	105.0	95.4	77.9
$E_s = 0.45$	93.1	91.6	78.5	73.0	66.5	61.1
$E_s = 0.65$	63.6	57.8	56.8	55.4	53.6	50.1
$E_s = 0.85$	–	59.2	58.2	52.6	48.7	46.5

Given a final path of length L as defined by the ordered sequence of nodes $\{q_1, q_2, \ldots, q_i, \ldots, q_L\}$, a closed ball $B(q_i)$ of some sufficiently small radius ϵ and centered at node q_i is created. If a node q_{i+j} $(j > 0)$ on the final path is contained in $B(q_i)$, nodes q_{i+j} and q_i are then connected by a straight line $SL_{i,j}$, and the upstream of $SL_{i,j}$ is calculated and compared with the original interval. If the upstream cost of $SL_{i,j}$ is smaller than that of the original interval, the interval between q_i and q_{i+j} is substituted by $SL_{i,j}$. The temporary path is then constituted by the nodes $\{q_1, q_2, \ldots, q_i, SL_{i,j}, q_j, \ldots, q_L\}$. If nodes q_{i+j}, q_{i+k} $(k > j)$ are contained in $B(q_i)$, then all intervals $\{SL_{i,j}, q_{j+1}, \ldots, q_k\}$, $SL_{i,k}$, and the original interval must be compared to each other. For fixed ϵ this process is repeated for q_1 to q_{L-2}; ϵ is then gradually increased, and the above process is repeated (Algorithm 9).

We now compare the results of our smoothing method with the shortcut method. Both the shortcut method and our smoothing method are applied to the path generated by VF-RRT (Figure 13). Since the shortcut method involves taking random samples from the path, we take the average over 20 trials (note that our smoothing procedure is deterministic). In our method, we first set $\epsilon = \delta$, where δ is the prescribed stepsize, and increase it linearly by adding increments of δ.

The upstream cost of the original path is approximately 0.74. After sufficient smoothing, as can be seen in Figure 14, the upstream cost for our method is lower than that of the shortcut method. The upstream cost from the shortcut method is approximately 0.40, while our method produces a final path with upstream cost 0.38. More importantly, our smoothing procedure maintains the shape of the original path, while the shortcut method produces a nearly piecewise-linear path. In our method, to get the optimal path that minimizes the upstream cost, ϵ is increased up to the maximum distance between two points on the given

Algorithm 9. Path smoothing algorithm.

```
Path
radius ← ϵ
while ϵ < ϵ_max do
  while i < Length(Path) − 2 do
    index ← GetNodeIndexinBall(Path, i, ϵ)
    Interval ← GetIntervalPath(Path, i, index)
    SL ← GetStraightPath(Path, i, index)
    cost_original ← GetPathCost(Interval)
    cost_SL ← GetPathCost(SL)
    if cost_original > cost_SL then
      Path ← PathUpdate(Path, SL)
    end if
    i ← i+ 1
  end while
  Enlarge(ϵ)
end while
```

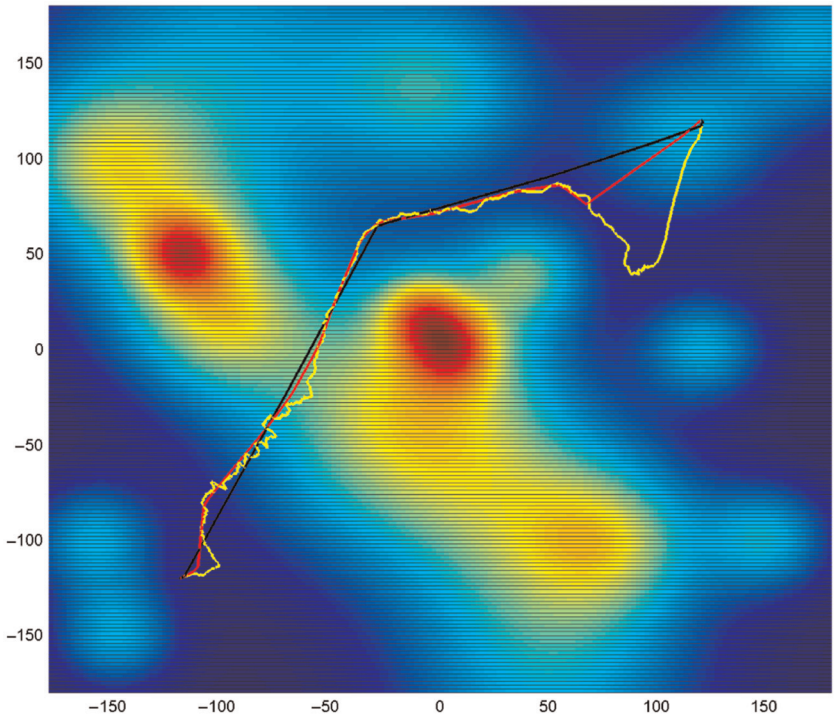


Fig. 13. Path generated by VF-RRT before smoothing (yellow), and after inefficient intervals have been smoothed by shortcut method (black) and our method (red).

path. According to Figure 14, 30 times the value of δ seems sufficient for this example.

6. Conclusions

This paper has proposed a randomized planning algorithm on general vector fields. The vector field can represent, e.g., nonconservative wind velocity fields, water current flow, or a gradient field with respect to some configuration space potential function such as a gravitational force field. Based on the premise that moving against the vector field requires greater control effort, and that minimizing the control effort is physically meaningful and desirable, we have proposed an integral functional for control effort, called the **upstream criterion**, that measures the extent to which a path goes against the given vector field. We have argued that the upstream criterion is more intuitive and physically natural than existing criteria for measuring path quality on vector fields, and established a number of fundamental properties for paths that minimize the upstream criterion.

A local version of the upstream criterion is then used to construct a rapidly exploring random tree (RRT) in the configuration space, in such a way that random nodes are generated with a bias that favors directions indicated by the vector field. The resulting algorithm, which we call the VF-RRT algorithm, has been tested over a wide range of numerical experiments, and its advantages over existing randomized vector field planning algorithms have been demonstrated both in terms of path quality and planning computation times. A user-specified scalar parameter E_s with values between 0 and 1 allows for a smooth transition from the classic RRT algorithm ($E_s = 0$) to a path planner that attempts to strictly follow the vector field ($E_s = 1$). Our experimental studies indicate that the choice of parameter can influence the computation times, with the following general trends observed: (a) for complex environments in which the vector fields are highly nonlinear and blocked by obstacles, larger values of E_s tend to increase computation

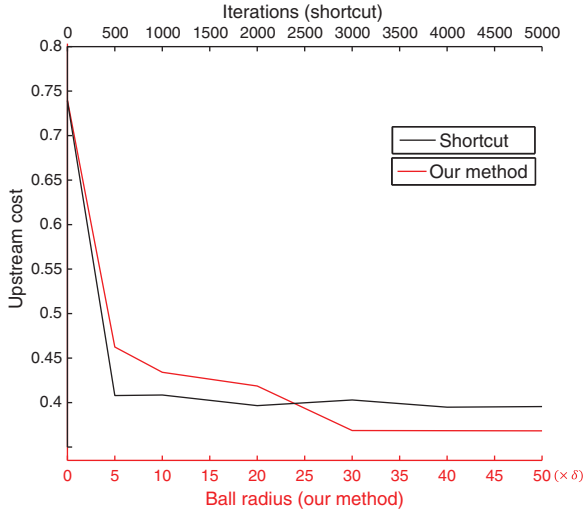


Fig. 14. Upstream cost comparison of shortcut method (black) with our method (red): upstream cost from our smoothing method becomes lower than that of the shortcut method.

times; (b) for simple environments, computation times generally decrease with increasing E_s; (c) for simple environments, the upstream cost decreases with increasing E_s; (d) for complex environments, the upstream cost is often a convex function of E_s, with a minimum achieved for some value of E_s achieved between 0 and 1. Of course, the above are general observations based on our experimental results and exceptions will undoubtedly exist.

Some natural extensions of the VF-RRT algorithm include alternative formulations for planning on constraint manifolds, including cases in which the configuration space has the structure of a Lie group (e.g. in the case of a multiple rigid bodies). These are currently topics of ongoing investigation.

Funding

This work was supported by the Biomimetic Robotics Research Center (UD130070ID), the SNU BK21+ Program in Mechanical and Aerospace Engineering (F14SN02D1310), SNU-IAMD and the Center for Advanced Intelligent Manipulation.

References

Alterovitz R, Patil S and Derbakova A (2011) Rapidly-exploring roadmaps: Weighing exploration vs. refinement in optimal motion planning. In: *Proceedings of the IEEE international conference on robotics and automation (ICRA)*, Shanghai, 9–13 May 2011, pp. 3706–3712.

Berenson D, Siméon T and Srinivasa SS (2011) Addressing cost–space chasms in manipulation planning. In: *Proceedings of the IEEE international conference on robotics and automation (ICRA)*, Shanghai, 9–13 May 2011, pp. 4561–4568.

Berenson D, Srinivasa SS, Ferguson D, et al. (2009) Manipulation planning on constraint manifolds. In: *Proceedings of the IEEE international conference on robotics and automation (ICRA)*, Kobe, Japan, 12–17 May 2009, pp. 625–632.

De Wit CC, Bastin G and Siciliano B (1996) *Theory of Robot Control*. New York: Springer-Verlag.

Ferguson D and Stentz A (2006) Anytime RRTs. In: *Proceedings of the IEEE/RSJ international conference on intelligent robots and systems (IROS)*, Beijing, 9–15 October 2006, pp. 5369–5375.

Geraerts R and Overmars MH (2007) Creating high-quality paths for motion planning. *The International Journal of Robotics Research* 26(8): 845–863.

Jaillet L, Cortés J and Siméon T (2010) Sampling-based path planning on configuration-space costmaps. *IEEE Transactions on Robotics* 26(4): 635–646.

Karaman S and Frazzoli E (2011) Sampling-based algorithms for optimal motion planning. *The International Journal of Robotics Research* 30(7): 846–894.

Kirk DE (2012) *Optimal Control Theory: An Introduction*. Mineola, NY: Courier Dover Publications.

LaValle SM (1998) Rapidly-exploring random trees: A new tool for path planning. TR 98-11, Department of Computer Science, Iowa State University.

LaValle SM and Kuffner JJ (2001) Randomized kinodynamic planning. *The International Journal of Robotics Research* 20(5): 378–400.

Lolla T, Ueckermann M, Yigit K, et al. (2012) Path planning in time dependent flow fields using level set methods. In: *Proceedings of the IEEE international conference on robotics and automation (ICRA)*, St Paul, MN, 14–18 May 2012, pp. 166–173.

Petres C, Pailhas Y, Patron P, et al. (2007) Path planning for autonomous underwater vehicles. *IEEE Transactions on Robotics* 23(2): 331–341.

Sastry S (1999) *Nonlinear Systems: Analysis, Stability, and Control*, Vol. 10. New York: Springer.

Sethian JA (1999) *Level Set Methods and Fast Marching Methods: Evolving Interfaces in Computational Geometry, Fluid Mechanics, Computer Vision, and Materials Science*, Vol. 3. Cambridge; New York: Cambridge University Press.

Soulignac M (2011) Feasible and optimal path planning in strong current fields. *IEEE Transactions on Robotics* 27(1): 89–98.

Stilman M (2010) Global manipulation planning in robot joint space with task constraints. *IEEE Transactions on Robotics* 26(3): 576–584.

Suh C, Um TT, Kim B, et al. (2011) Tangent space RRT: A randomized planning algorithm on constraint manifolds. In: *Proceedings of the IEEE international conference on robotics and automation (ICRA)*, Shanghai, 9–13 May 2011, pp. 4968–4973.

Thompson DR, Chien S, Chao Y, et al. (2010) Spatiotemporal path planning in strong, dynamic, uncertain currents. In: *Proceedings of the IEEE international conference on robotics and automation (ICRA)*, Anchorage, AK, 3–8 May 2010, pp. 4778–4783.

Urmson C and Simmons R (2003) Approaches for heuristically biasing RRT growth. In: *IEEE/RSJ international conference on intelligent robots and systems (IROS)*, Las Vegas, NV, 27 October–1 November 2003, Vol. 2, pp. 1178–1183.

Yershova A and LaValle SM (2007) Improving motion-planning algorithms by efficient nearest-neighbor searching. *IEEE Transactions on Robotics* 23(1): 151–157.