Proceedings of the
44th IEEE Conference on Decision and Control, and
the European Control Conference 2005
Seville, Spain, December 12-15, 2005

Smoothly Blending Vector Fields for Global Robot Navigation

Stephen R. Lindemann and Steven M. LaValle
Department of Computer Science
University of Illinois
Urbana, IL 61801 USA
{*slindema, lavalle*}*@uiuc.edu*

***Abstract*— We introduce a new algorithm for constructing smooth vector fields for global robot navigation. Given a d-dimensional cell complex with each cell a convex polygon, our algorithm defines a number of local vector fields: one for each cell, and one for each face connecting two cells together. We smoothly blend these component vector fields together using bump functions; the precomputation of the component vector field and all queries can be done in linear time. The integral curves of the resulting globally-defined vector field are guaranteed to arrive at a neighborhood of the goal state in finite time. Except for a set of measure zero, the vector field is smooth. The resulting vector field can be used directly to control kinematic systems or can be used to develop dynamic control policies. We prove convergence for the integral curves of the vector fields produced by our algorithm and give examples illustrating the practical advantages of our technique.**

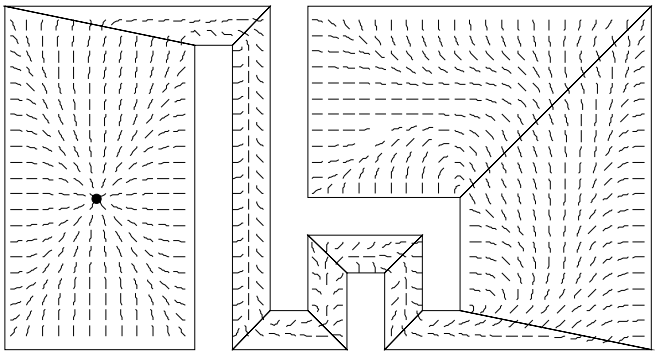

Fig. 1. A smooth vector field generated using our algorithm in a two-dimensional environment.

I. Introduction

Finding smooth vector fields for global robot navigation is a long-standing problem in mobile robotics. Traditional feedback control methods fail due to non-convex constraints induced by obstacles in the environment. In the motion planning community, however, most approaches compute open-loop plans while relegating important feedback concerns to secondary status. Some have tried to make feedback more central through the construction of potential fields that have no local minima other than the goal state. If such a potential field can be found, the gradient of the field can be used as the velocity command for the robot. We adopt a more direct approach. Instead of beginning with a potential field and taking the gradient, we directly construct a smooth vector field to use as the velocity command. We do this by smoothly combining locally-defined vector fields using bump functions. The result is a globally-defined smooth vector field the integral curves of which converge to a goal state. [1] An illustration of a vector field produced by our algorithm is given in Figure 1. Our vector fields can be used directly for kinematic systems, or they can be used to develop dynamic control policies. For example, a control policy

$$u = K(V(p) - \dot{p}) + \dot{V}(p)$$

can be used [18]. Under certain conditions, it can be shown that the system will converge to the integral curves of $V(p)$ [4], [18].

An important advantage of our algorithm is that it requires little preprocessing to compute our vector field and the value of the vector field at any point can found extremely quickly during operation. In particular, the vector field can be computed in $O(n)$ time in the complexity of the environment[2]. During operation, the value of the vector field at any point can be found in $O(n)$ time, in which n is the complexity of the cell in which it is located.

In the following section, we will review related work, focusing on the different ways this problem has been addressed within the robotics community. We will then describe our algorithm in detail and give proofs of the convergence of the integral curves of our vector field to the goal state. Finally, we examine several practical issues with the goal of illustrating how to use our technique to develop good paths for mobile robotics applications.

II. Related Work

The problem of finding a global motion plan in complex environments is difficult. Motion planning problems in robotics typically involve non-convex constraints resulting

[1] To be precise, the vector field is smooth except for a set of measure zero; however, all integral curves obtained by starting at an initial state and following the vector field are smooth. This qualification is assumed throughout the paper.

[2] The complexity is defined to be the total number of k-cells in the complex, summing k from 0 to d

0-7803-9568-9/05/$20.00 ©2005 IEEE

from obstacles in the environment. This presents a significant problem for traditional feedback control methods. One solution might be to use state-space sampling along with dynamic programming to achieve not only feedback, but approximately-optimal trajectories [1], [13], [21]. This may be feasible for low-dimensional spaces, but both the time- and space-complexity is exponential in the dimension of the state space, assuming that the sampling resolution remains fixed. The difficulty of feedback control for these problems motivates the development of open-loop motion planning algorithms, which can at least find feasible paths through obstacle-cluttered environments. Such algorithms have been extensively studied [10], [12]. Many motion planning algorithms have been developed for kinematic systems; several, such as RRTs [14] and PDST-EXPLORE [8] are specifically designed for systems with dynamics. Kinematic motion planning algorithms find paths which need post-processing (e.g., time-scaling [2], [20], steering [9], [16], or other transformations [11], [19]) to be transformed into trajectories for dynamical systems. In contrast, RRTs and similar planners find such trajectories directly. In either case, an open-loop trajectory for the system is found. This trajectory can then be tracked using feedback.

This approach has several disadvantages, however. First, paths generated by motion planning algorithms often appear to be of poor quality, having unnecessary turns and bends in them. This may result in them being difficult to follow for a dynamical system. Second, this approach does not produce a global feedback plan, but only a local feedback plan in a neighborhood of the nominal trajectory. It would be better to solve the feedback problem once for the entire space.

Another approach, made plausible through tremendous advances in computational power, is to use motion planning algorithms themselves as the feedback mechanism. In such a model, any time the system deviated from the prescribed trajectory, the trajectory would be re-planned (probably from scratch) based on the new state of the system. This approach is extremely problematic as well. First, it has a very high computational cost, and may not be suitable for real-time applications. Second, this approach is not even guaranteed to bring the system to the goal state, although in practice might expect it to.

These approaches, which add feedback almost as an afterthought to open-loop trajectories, have significant problems, as we have seen. Consequently, there have been some attempts within the robotics community to incorporate feedback more directly. For example, the sampling-based neighborhood graph (SNG) covers the free space with balls, each of which is equipped with a local navigation function which is guaranteed to convey the robot into a ball nearer to the goal state. Other approaches to feedback motion planning in the presence of obstacles are often based on potential fields. Khatib [6] developed a method which utilized a potential field over the operational space to guide a manipulator or mobile robot to the goal. His approach suffers from local minima, however, as do many potential field methods. A highly-influential potential field method is that of Rimon and Koditschek [17], who show how to develop *navigation functions* (potential functions with a unique minimum at the goal and meeting certain other criteria) using potential functions in a generalized sphere world. Waydo and Murray give a stream function method for navigation in two-dimensional environments [22].

Recent work by Conner *et al.* [4] bears similarity to the present work. They consider an cell-complex environment in d-dimensional Euclidean space. They then impose a potential field over each individual cell, taking as the field the pullback of a potential function on a disk, which has a closed-form solution. They require that the gradients of the potential fields be perpendicular to the cell boundaries, so that adjoining potential fields can be easily pieced together. Putting the individual "component control policies" together guarantees that the global control policy brings the robot to the goal. In addition to specifying a control policy for kinematic systems, they develop control policies for systems with dynamical constraints. Conner's work (and ours) can be seen in the context of the sequential composition of funnels approach [15], in which a collection of controllers is developed, each of which converges to a goal set which is either the actual goal state or in the domain of another controller. Following a sequence of these controllers will cause the system to arrive at the goal state. This idea was further developed in [3], [18]

III. Our algorithm

Consider a point robot whose environment is a d-dimensional cell complex; each cell is a (bounded) d-dimensional convex polytope. Typically, the complex might result from the convex decomposition of a general polygonal environment. Let the goal state be x_g, and let the cell containing x_g be C_g. Then, using the connectivity of the convex cells, construct a graph and use a graph search algorithm (such as Dijkstra's algorithm) to determine a path from each cell to C_g. Hence, for each cell other than C_g, we have a "successor" that represents the next cell on the path to the goal cell. We call every cell with a successor an *intermediate* cell, in distinction with the goal cell, which has no successor. See Figure 2 for an illustration. Within each cell, we will define a vector field that will take every point in that cell to the face between it and its successor (denote this face the *exit face* of the cell). In the case of the goal cell, the vector field will take every point to the goal point.

At this point, two primary issues must be addressed. First, how can the vector fields for the individual cells be designed so that the global vector field is smooth on the boundaries between cells? Throughout this paper, the term *smooth* is used to denote C^∞-continuous. Second, how can the cell vector fields be designed so that they guarantee that any trajectory beginning in the cell will not exit that cell except at the exit face? Conner *et al.* address these issues through the careful construction of their potential fields. We deal with these issues through the use of *face vector fields*. Each face (specifically, each $(d-1)$-dimensional boundary hyperplane) of each cell is assigned a vector field, and we require that our eventual global vector field agree with the face vector fields

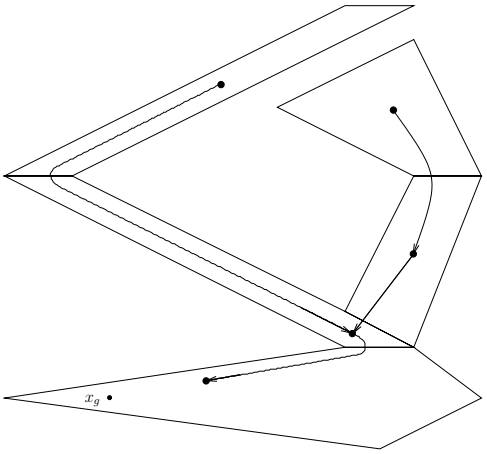

Fig. 2. Paths found using the graph representation of the cells in the environment.

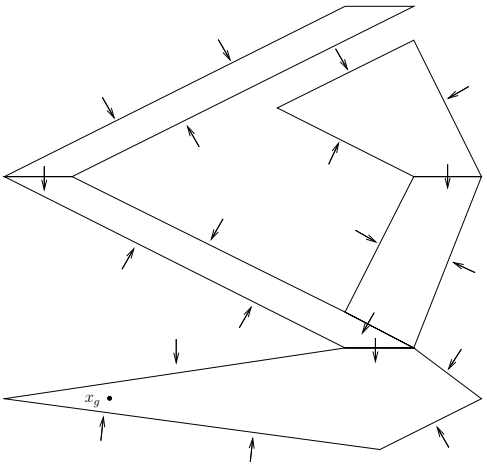

Fig. 3. Vector fields assigned to faces of cells in the environment.

on the faces. The face vector fields are constructed in such a way as to guarantee that the above issues are satisfactorily addressed; formal proofs are given in Section IV.

Figure 3 shows face vector fields for an example. Note that although the vector fields are shown to be perpendicular to the faces (similar to Conner *et al.*), this is not required.

Through the use of face vector fields, the problem of constructing a global smooth vector field is reduced to constructing a smooth vector field within each individual cell. In the case of the goal cell, the integral curves should all converge to the goal point; for all other cells, the integral curves should all go to the exit face of the cell (and hence continue to the successor cell). In order to construct this vector field, we need to smoothly transition between face vector fields (since the overall vector field is required to match the face vector fields at the boundary of the cell). We achieve this using the interior generalized Voronoi diagram (GVD) of the cell. Using the GVD, the region of influence of each face (and hence each face vector field) is defined to be the set of points inside the polygon which are closest to that particular face.

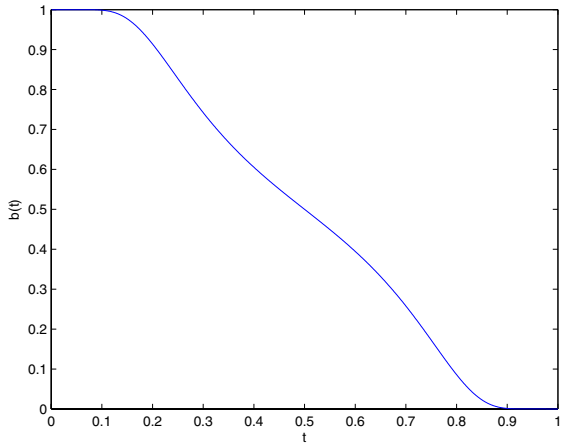

Fig. 4. A bump function. If we let $\lambda(t) = (1/t)e^{-1/t}$, then $b(t) = 1 - \lambda(t)/(\lambda(t) + \lambda(1-t))$.

Within each cell, the face vector field is blended with an *attractor vector field*, which is defined over the entire cell. The resulting vector field is equal to the face vector field on the boundary of the cell, and equal to the attractor vector field on the GVD. However, we still need a way to smoothly blend the face vector field and the attractor vector field. We accomplish this through the use of *bump functions*, which are defined as follows:

Definition 3.1: Let X be a smooth manifold, and let K be a closed set and U an open set, $K \subset U \subseteq X$. Then a bump function over U is a smooth, real-valued function $\rho : X \to [0, 1]$ such that:

1) ρ has support contained in U.
2) $\rho(x) = 1$ for every $x \in K$.

For our purposes, we will use a bump function on the real line which transitions smoothly from 1 to 0 on the unit interval. An illustration of such a bump function is given in Figure 4.

The parameter we use for our bump function is essentially the product of all fractional distances to the faces of the GVD. Formally, for any point p we have

$$t(p) = 1 - \prod_{i=1}^{n} \frac{d(p, f_i)}{d(p, f_i) + d(p, f_x)}, \tag{1}$$

in which $\{f_i\}_1^n$ are the faces of the GVD and $d(p, f_i)$ is the distance from p to face f_i. This function is smooth (except at the vertices of the cell), and has the desired property of being identically equal to one on the exit face and zero on all other boundaries.

Since the bump function smoothly blends the face and attractor vector fields together, we obtain a vector field which is smooth over the entire cell. With small modifications, the above approach can be used in the goal cell as well; piecing the cells together, we obtain a vector field which is smooth over the entire free space. At each point, the global vector field $V(p)$ is defined as $V(p) = \mathrm{norm}(b(p)V_f(p) + (1 - b(p))V_a(p))$, in which V_f is the face vector field for that

point, V_a the attractor vector field, b the bump function, and norm is the normalization function, so that V is a unit vector field. Formal proofs and conditions on the face and attractor vector fields are given in the following section.

IV. Theoretical Analysis

In this section, we formally show that the integral curves of the vector fields our algorithm produces are guaranteed to terminate at the goal state. In so doing, we give sufficient conditions to be satisfied by the face and attractor vector fields. In the following section, we give examples of how to automatically construct vector fields which satisfy these conditions and offer good practical performance. The most difficult result to show is that for every intermediate cell, every integral curve goes to the exit face of the cell. We will begin by showing that this is the case; after this is complete, we will briefly show that the vector field in the goal cell will bring every integral curve to the goal state.

We begin by formally defining attractor vector fields for intermediate cells. Recall that the GVD consists of subsets of hyperplanes which are equidistant from two faces of the cell. Essentially, we require that for any such subset, the vector field can only cross it in a single direction. This enables is to construct a directed graph corresponding to the possible transitions between Voronoi regions of the cell. We then require that the paths through this graph all terminate in the node corresponding to the Voronoi region of the exit face. Formally, we have the following:

Definition 4.1: Let C be a convex cell with exit face f_x, and consider the interior GVD of C. An attractor vector field V_a is a smooth unit vector field on C which satisfies the following:

1) V_a is always directed toward some point in f_x.
2) Let h be a hyperplane of the GVD with normal n. If $V_a \cdot n = 0$ anywhere on the intersection of the GVD with h, then $V_a \cdot n = 0$ everywhere on that intersection.
3) The resulting directed transition graph is acyclic and every path through the graph terminates at the node corresponding to the exit edge.

While this definition permits many different types of attractor vector fields, we use a particular formulation. Consider a convex cell C with exit face f_x, having normal n_x, and consider all neighboring faces of f_x. If the neighboring faces intersect at a single point p, then let V_a be a unit vector field which always points toward p or away from p (the appropriate choice will have positive dot product with n_x in C). If the faces do not intersect at a point (i.e., two or more faces are parallel), then let V_a be a constant vector field, the projection of n_x onto one of the parallel faces.

Observation 4.2: V_a is an attractor vector field.

Note that an attractor vector field is also attained if V_a is assigned to always point to some point p on the exit face (the centroid, for example). We will make use of this fact later. The definition for an face vector field is simple:

Definition 4.3: Let C be a convex cell with exit face f_x. A face vector field corresponding to a face f (with inward normal n) is a smooth unit vector field V_f such that for every $p \in f$, $V_f \cdot n < 0$ if $f = f_x$, or $V_f \cdot n > 0$ otherwise.

Now, given an additional condition on the face vector fields, we are able to show that the integral curves all reach the exit face. It is obvious that they do so without leaving the cell, because the face vector fields all point inward at the boundary (except at the exit face, where the vector field points outward). As before, consider a convex cell C with exit face f_x. Consider a non-exit face f, and denote the hyperplane of points equidistant to f and f_x by b_f. Denote the unit normal vector of this hyperplane to be n_{bf} and let it be oriented so that $n_{bf} \cdot n_x \geq 0$. Finally, consider a point p in the Voronoi region corresponding to f, and let $V_f(p)$ be the value of the face vector field at that point.

Theorem 4.4: If $f = f_x$, then let $V_f(p) \cdot n_x \geq 0$, otherwise let $V_f(p) \cdot n_{bf} \geq 0$. Then all integral curves of V reach the exit face.

Proof: First, we know that at any point p not in the Voronoi region of the exit face, the attractor vector field satisfies $V_a(p) \cdot n_{bf} > 0$. the total vector field at p, $V(p)$, is a linear sum of V_a and V_f, this implies that $V(p) \cdot n_{bf} > 0$. In other words, on any integral curve the distance to the hyperplane b_f is always decreasing. As a result, two things can happen: either the hyperplane is reached or some other hyperplane in the GVD is reached. If the hyperplane itself is ever reached, then the integral curve has reached Voronoi region of the exit face and will clearly continue to the exit face.

Assume that another hyperplane in the GVD is reached. On the GVD, the vector field is equal to the attractor field, and the hyperplane is crossed in one direction only. If the attractor field is such that the integral curve stays in the Voronoi region corresponding to f, then the same argument holds: the distance to b_f continues to decrease. Recall that we used the attractor vector field to build a directed, acyclic graph with all paths terminating at the exit cell. Hence, if the integral curve continues to another Voronoi cell, this corresponds to following an edge in the graph from one node to another. Eventually, we must reach a node in which the only outgoing edge is to the node corresponding to the exit region. Since the integral curve cannot cross any hyperplane but b_f, and the distance to b_f is always guaranteed to decrease, we obtain the result that b_f must eventually be reached. At this point, we have reached the Voronoi region of the exit face and the integral curve will continue to the exit face. ■

We have shown that for every intermediate cell, every integral curve of the generated vector field will reach the exit face of that cell, and hence continue to the next cell. Consequently, we have shown that all integral curves will reach the goal cell. However, it remains to be shown that all integral curves in the goal cell will reach the goal point.

For intermediate cells, we divided each cell into regions of influence for each edge vector field using the GVD. In the goal cell, we divide the cell somewhat differently. The region of influence for each face is defined to the convex hull of the vertices of that face together with the goal point. Doing

this for every face clearly results in a subdivision of the cell. As before, we blend the face vector field with the attractor vector field over each region. We again use the product of fractional distances as the parameter in the bump function.

We also simplify our requirements on the attractor vector field. We now require only that $V_a(p) \cdot (x_g - p) > 0$ for any point p. Practically, we use an attractor vector field which is always oriented toward x_g to satisfy this condition. For a convex cell C containing a goal point x_g, consider a point p in the region of influence of some face f. Let $V_f(p)$ be the value of the face vector field at that point.

Theorem 4.5: All integral curves in C reach the goal point if for all such p and f, $V_f(p) \cdot (x_g - p) \geq 0$.

Proof: For any point p, the distance to the goal point is always decreasing for any V resulting from a linear combination of V_a and V_f, since the distance is decreasing for V_a and non-increasing for V_f. Consequently, every integral curve must terminate at x_g. ■

Given the previous theorems, the following theorem holds true:

Theorem 4.6: The integral curves of the vector field V, defined over the entire free space, all terminate at the goal state x_g.

Proof: From Theorem 4.4, any integral curves in an intermediate cell proceeds to the exit edge as defined by the shortest-path through the graph corresponding to the cell complex. All integral curves consequently proceed to the goal cell. Then, Theorem 4.5 implies that the integral curves terminate at the goal state. ■

Thus far, we have claimed that the global vector field resulting from the blending of individual face and attractor vector fields is smooth, excepting only a set of measure zero. We now elaborate on this qualification. First, it is obvious that if the cell complex is not simply connected, then the resulting vector field cannot be globally smooth. There will be some faces for which the integral curves move in opposite directions on the opposite sides of the face. An example of an environment which is not simply connected can be seen in Figure 5. Second, it is clearly not smooth at the goal point because all integral curves of the unit vector field arrive there. Practically, it is simple to apply a deceleration policy as the robot approaches the goal state, which still allows you to arrive at a neighborhood of the goal state in finite time. Finally, the vector field is not guaranteed to be smooth on the $(d-2)$-dimensional intersections of neighboring cells. This is the case because the face vector fields must have different values. If these are cut out of the free space (which can be done without consequence), smoothness is preserved. Note that in the two-dimensional case, this corresponds to eliminating any Steiner points used in the convex decomposition of the free space. While this is the case, simple computations show that the vector field and all its derivatives match on the $(d-1)$-dimensional faces connecting adjacent cells, so our claim of smoothness on them is correct.

Finally, we claimed in the introduction that our method is extremely fast to compute. There are three primary computation costs. First, there is the cost to compute the component vector fields given an environment and a goal state. Second, there is the problem of "initializing" the vector field given a new initial state. Finally, there is the problem of computing the new vector field values when following an existing trajectory. All of these can be done quickly. First, consider the precomputation phase. If breadth-first search is done on the graph corresponding to the cell complex, the successor of each cell can be found in $O(n)$ time, in which n is the complexity of the cell complex[3]. The face vector fields can be assigned in linear time if perpendicular face vector fields are used. The attractor vector fields likewise require only linear time, since they can be assigned to point to the centroids of the exit faces.

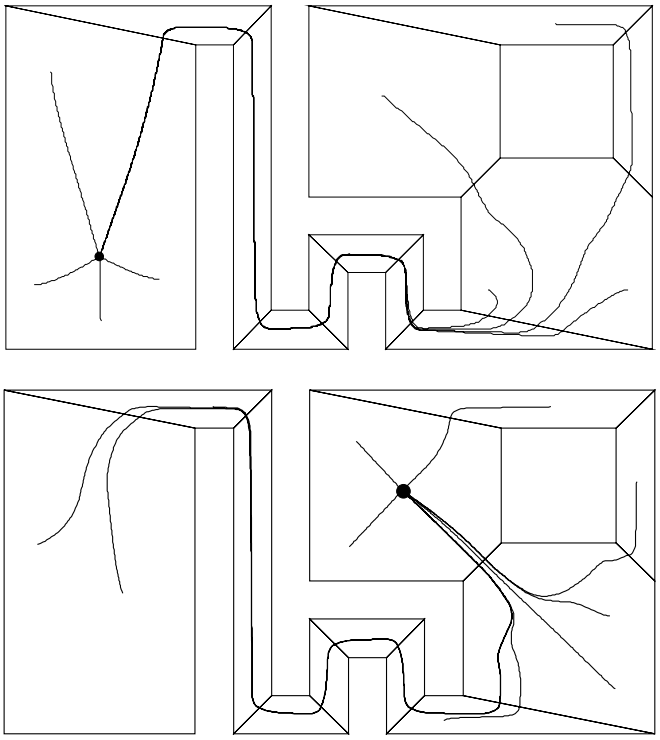

Fig. 5. An environment which is multiply connected, with two different goal states shown.

Second, consider the initialization. A point location query must be answered to determine in which cell the point lies. This can clearly be answered in linear time, and may be answered in logarithmic time if some preprocessing is done. In two dimensions, the optimal preprocessing bound is $O(n)$ time, but practical algorithms typically require $O(n \log n)$. Also, only linear space is required in two dimensions. A good algorithm for this purpose is Kirkpatrick's triangulation refinement method [7]. In higher dimensions, the results are not as good: logarithmic query time (more precisely, $O(d \log n)$, in which d is the dimension) can be attained, but only at the cost of exponential space: $O(n^{2^d})$ [5].

After the initialization is complete, no more point location

[3]Note that both the cells and the faces connecting cells are included in n, justifying the $O(n)$ bound.

queries must be answered in the course of computing a trajectory. Consider two successive query points: they must either lie in the same cell, or the second one lies in the cell which is the successor to the first one. This is guaranteed as long as we assume that the vector field is queried at a high enough rate. This is a very weak assumption, since if this does not hold, having a smooth vector field is without benefit. The most reasonable assumption is that the vector field is queried nearly continuously, which will result in the condition holding true. Once the cell of a query point is known, it requires linear time (in the complexity of the cell along with one neighboring cell). to compute the vector field value. Finding the closest edge requires only linear time in the complexity of the current cell. If the radius of the ball in the Voronoi cell must be computed, this could require examining all the faces in the current cell and in one neighboring cell. Hence, the vector field value can be computed in linear time.

V. Practical Considerations

In the previous section, we stated fairly general conditions on the face and attractor vector fields under which convergence is guaranteed. The purpose of doing this is to permit a great deal of design flexibility, rendering our algorithm highly practical. In this section, we will outline our approach for designing face vector fields. We also give several concrete examples to illustrate the impact of the choice of convex subdivision, face, and attractor vector fields on the quality of the resulting paths.

Consider an face vector field for some face other than the exit face of a cell. As we have seen, such a vector field must be directed inward at the face itself and must always point toward to hyperplane bisecting the face and the exit face. Moreover, in the case when the face is also the exit face of another cell, this places additional restrictions on the field, since in that cell the vector field must always point toward the exit face. While these requirements are satisfied by a variety of different vector fields, we prefer constant vector fields for the sake of simplicity. In general, there still remains a great deal of design freedom under the constant vector field restriction.

First, consider the case where we have a constant face vector field with only the restriction that it must be inward-pointing at the face itself. If we have this much freedom, where might we want the vector field to point? Four possibilities are obvious. First, we might want the vector field to point toward the exit face as much as possible, to promote short paths. Second, we might want it to point as far away from the exit face as possible, to avoid the sharp turns that the first approach might induce. Third, it might be preferable to make each face vector field perpendicular to its face; this is the simplest approach. Finally, we might wish to compute the centroid of the cell or of the exit face and direct the vector field toward it (say, from the centroid of the face). In different situations, each of these approaches could offer advantages; we have not investigated this in depth at present.

So, we have made general comments about what a constant face vector field might look like, in the absence of constraints. It is trivial to apply constraints to the desired vector field, since each is a simple hyperplane constraint (i.e., each constraint requires that the vector field have positive dot product with the normal of some hyperplane). As in the previous section, let f be a face under consideration, and denote the "bisector" hyperplane between f and f_x by b_f. Denote the unit normal vector of this hyperplane to be n_{bf}. Recall that n_{bf} satisfies the following inequality with the outward-pointing normal of the exit face: $n_{bf} \cdot n_x \geq 0$. Then, we require that the face vector field have positive dot product with n_{bf}. If the desired vector field does not satisfy this, simply project the vector field onto b_f, also adding an arbitrarily-small fraction of n_{bf} to attain a positive dot product.

If the face vector field is also the exit face vector field of a previous cell, we have more constraints to apply. Their application, however, is identical to that just described. Now, consider an exit face vector field, and take any adjoining face f. Then we again have a bisecting hyperplane and its normal. The exit face vector field must satisfy a positive dot product constraint with each bisector normal n_{bf}. In doing so, the vector field is guaranteed to cross the exit face as required. Finally, it is worth noting that a perpendicular vector field always satisfies all the constraints we have outlined. If a different vector field is desired, however, each constraint must be examined and satisfied in order for convergence to be guaranteed. In our experience, we have found that a good paths are typically attained if the exit face vector fields are set to point to the centroid of the next exit face and all all other face vector fields set to be perpendicular. Assuming a reasonable choice of face vector fields, the attractor vector fields seem to exert a stronger influence on path quality.

In addition to the choice of attractor field discussed above, a variety of other choices are possible. One possibility is to place the "attractor point" on the exit face, and have the vector field always oriented toward that point. Depending on where this point is placed on the exit face, it tends to strongly influence the vector field to leave the cell near the attractor point. Sometimes, it may also result in sharp turns, which can be undesirable.

Finally, the choice of convex decomposition can greatly affect the quality of the resulting paths. This is particularly important when the face vector fields are chosen to be perpendicular to the edges. See Figure 6 for an illustration of this point.

VI. Conclusion

In conclusion, we have introduced a new algorithm for addressing the problem of global robot navigation. In contrast with previous work, which uses the gradient of a suitable potential field as a velocity command, we directly build a smooth vector field suitable for robot navigation. Using bump functions, we combine face and attractor fields so that the integral curves of the resulting vector field are guaranteed to terminate at the goal state. In addition to giving

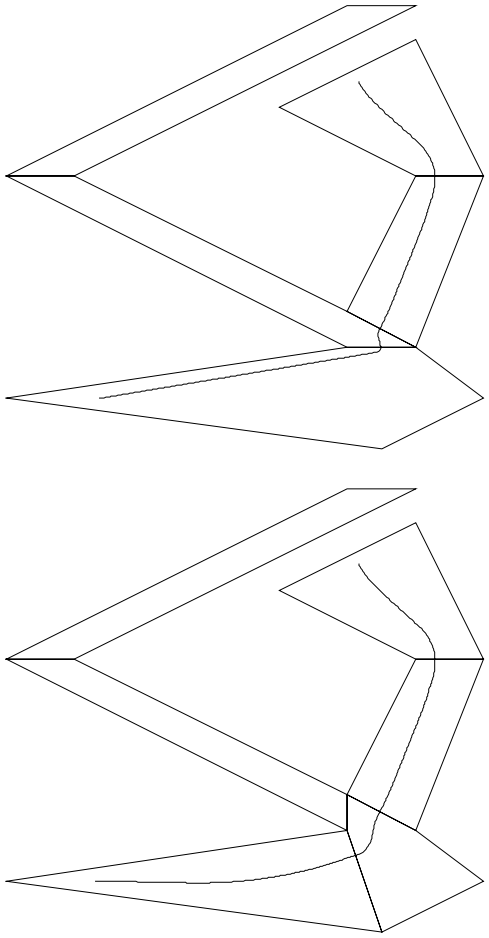

Fig. 6. The influence of different convex decompositions. On top, a path with sharp turns arising from the choice of decomposition and perpendicular face vector fields; on bottom, the artifacts eliminated through a better decomposition.

theoretical proofs of the convergence of the vector fields generated with our algorithm, we show illustrations of how to make design choices which result in high-quality paths for practical applications. This vector field can be used directly to control kinematic systems, or can be used to develop dynamic control policies as in [4], [18].

In the future, we plan to continue to improve our methods of choosing component vector fields and blending strategies. In addition to this, we intend to extend these ideas to the case of a polygonal robot translating and rotating in the plane. We would also like to integrate the ideas in this paper with sampling-based motion planning techniques in order to develop global or local feedback motion plans for arbitrary robotic configuration spaces.

ACKNOWLEDGMENTS

This work was funded in part by NSF Awards 9875304, 0118146, and 0208891.

REFERENCES

[1] D. Bertsekas. *Dynamic Programming and Optimal Control: Volume I*. Athena Scientific, Belmont, MA, USA, 2000.

[2] J. E. Bobrow, S. Dubowsky, and J. S. Gibson. Time-optimal control of robotic manipulators along specified paths. *Int. J. Robot. Res.*, 4(3):3–17, 1985.

[3] R. R. Burridge, A. A. Rizzi, and D. E. Koditschek. Sequential composition of dynamically dexterous robot behaviors. *Int. J. Robot. Res.*, 18(6):534–555, 1999.

[4] D. C. Conner, A. A. Rizzi, and H. Choset. Composition of local potential functions for global robot control and navigation. In *IEEE/RSJ Int. Conf. on Intelligent Robots & Systems*, pages 3546–3551, 2003.

[5] J. E. Goodman and J. O'Rourke. *Handbook of Discrete and Computational Geometry, Second Edition*. CRC Press, Boca Raton, FL, 2004.

[6] O. Khatib. Real-time obstacle avoidance for manipulators and mobile robots. *Int. J. Robot. Res.*, 5(1):90–98, 1986.

[7] D. G. Kirkpatrick. Optimal search in planar subdivisions. *SIAM J. Computing*, 12:28–35, 1983.

[8] A. M. Ladd and L. E. Kavraki. Fast exploration for robots with dynamics. In *Proc. Workshop on Algorithmic Foundation of Robotics*, 2004.

[9] F. Lamiraux and J.-P. Laumond. Flatness and small-time controllability of multibody mobile robots: Applications to motion planning. *IEEE Transactions on Automatic Control*, 45(10):1878–1881, April 2000.

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

[11] J. P. Laumond, S. Sekhavat, and F. Lamiraux. Guidelines in nonholonomic motion planning for mobile robots. In J.-P. Laumond, editor, *Robot Motion Plannning and Control*, pages 1–53. Springer-Verlag, Berlin, 1998.

[12] S. M. LaValle. *Planning Algorithms*. [Online], 2005. Available at http://msl.cs.uiuc.edu/planning/.

[13] S. M. LaValle and P. Konkimalla. Algorithms for computing numerical optimal feedback motion strategies. *International Journal of Robotics Research*, 20(9):729–752, September 2001.

[14] S. M. LaValle and J. J. Kuffner. Rapidly-exploring random trees: Progress and prospects. In B. R. Donald, K. M. Lynch, and D. Rus, editors, *Algorithmic and Computational Robotics: New Directions*, pages 293–308. A K Peters, Wellesley, MA, 2001.

[15] T. Lozano-Pérez, M. T. Mason, and R. H. Taylor. Automatic systhesis of fine-motion strategies for robots. *Int. J. Robot. Res.*, 3(1):3–24, 1984.

[16] R. M. Murray and S. Sastry. Nonholonomic motion planning: Steering using sinusoids. *Trans. Automatic Control*, 38(5):700–716, 1993.

[17] E. Rimon and D. E. Koditschek. Exact robot navigation using artificial potential fields. *IEEE Trans. Robot. & Autom.*, 8(5):501–518, October 1992.

[18] A. A. Rizzi. Hybrid control as a method for robot motion programming. In *IEEE Int. Conf. Robot. & Autom.*, pages 832–837, 1998.

[19] S. Sekhavat, P. Svestka, J.-P. Laumond, and M. H. Overmars. Multilevel path planning for nonholonomic robots using semiholonomic subsystems. *Int. J. Robot. Res.*, 17:840–857, 1998.

[20] K. G. Shin and N. D. McKay. Minimum-time control of robot manipulators with geometric path constraints. *IEEE Trans. Autom. Control*, 30(6):531–541, 1985.

[21] J. N. Tsitsiklis. Efficient algorithms for globally optimal trajectories. *IEEE Trans. Autom. Control*, 40(9):1528–1538, September 1995.

[22] S. Waydo and R. M Murray. Vehicle motion planning using stream functions. In *IEEE Int. Conf. Robot. & Autom.*, pages 2484–2491, 2003.