Smooth Feedback for Car-Like Vehicles in Polygonal Environments

Stephen R. Lindemann and Steven M. LaValle

Abstract— **We introduce a method for constructing provably safe smooth feedback laws for car-like robots in obstacle-cluttered polygonal environments. The robot is taken to be a point with motion that must satisfy bounded path curvature constraints. We construct a global feedback plan (or control policy) by partitioning the environment into convex cells, computing a discrete plan on the resulting cell complex, and generating local control laws on the state space that are safe, consistent with the high level plan, and satisfy smoothness conditions. The trajectories of the resulting global feedback plan are smooth and stabilize the position of the robot in the plane, neglecting the orientation. We also extend our basic results to the case of a disc robot moving among polygonal obstacles.**

I. Introduction

Navigation is a fundamental problem in mobile robotics. In contrast to free-flying robots, mobile robots must navigate to a goal location while satisfying nonholonomic constraints which restrict how they can move. For example, differential drive (or unicycle) robots can drive forward and rotate in place, but cannot slide sideways. Car-like robots, which are another important class of mobile robots, are even more restricted in their motion. They have a limited steering angle which prevents them from rotating in place, causing them instead to move along paths of bounded curvature. In environments that are free of obstacles, feedback control techniques provide a highly robust and effective solution to the global navigation problem. Feedback controllers define a vector field over the state space, the integral curves of which are guaranteed to asymptotically converge to the goal state. This guarantees that the goal will be reached, regardless of the initial state. Unfortunately, the presence of obstacles typically invalidates standard control techniques because it is difficult or impossible to guarantee *both* convergence to the goal state and obstacle avoidance. Motion planning algorithms offer an efficient solution to the difficulties posed by the presence of obstacles, but at the cost of producing only open loop trajectories rather than closed loop feedback controls. In this work, we describe an algorithm that computes global feedback controllers that provide *guarantees* that trajectories from any initial state will converge to the goal state and avoid obstacles.

Stabilization of nonholonomic systems in obstacle-free environments has been studied in depth [4], [6], [9]. Car-like robots are common and have received significant attention [10], [21], [22], [26], [27]. Important research goals include stabilization to a point or trajectory and path following. Obstacles are generally not considered. The standard way to solve problems with obstacles is to plan an open loop path using a motion planning algorithm, and then use the local controller to track the path or move from waypoint to waypoint. This approach can be reasonably successful in practice, assuming that the distance to the obstacles is large relative to the intra-waypoint distance. However, we again emphasize that this approach *cannot* generally guarantee both obstacle avoidance and global convergence. This is a result of decoupling planning and feedback; by integrating the two more closely together, we will be able to make stronger guarantees.

One of the primary motivations of the decoupled approach is that non-convex obstacle constraints make traditional feedback control ideas difficult to apply. One solution is to use state space sampling along with dynamic programming to achieve not only feedback, but approximately optimal trajectories [3], [17], [31]. This may be feasible for low-dimensional spaces, but both the time- and space-complexity is exponential in the dimension of the state space. Even for the four-dimensional state space we consider here, dynamic programming is quite slow relative to our algorithm.

Other approaches to feedback navigation in the presence of obstacles typically rely on scalar potential functions [15], [24]. For fully actuated systems, the gradient of the potential function is used as the feedback control. If the potential function is uniformly maximal along the obstacle boundary and has no local minima other than the goal state, then the potential function serves as a suitable Lyapunov function and convergence can be guaranteed. However, constructing such functions is quite difficult, and it is difficult to apply potential function or navigation function methods to systems with nonholonomic constraints (see [29], [30] for an approach for robots with unicycle dynamics).

Our approach is based on decomposing the two-dimensional environment into simple cells, constructing local controllers on the region of the state space corresponding to each cell, and combining them together. Several other approaches based on cell decompositions include [2], [8], [12], [25]. These generally consider the problem of control of an affine system on a arbitrary dimensional simplex or polytope. We have previously addressed a fully actuated robot [19], [20] and a point unicycle robot [18]. The present work is much more challenging because curvature bounds force solution trajectories to have velocity reversals, which are not necessary for a fully actuated or unicycle robot.

S. R. Lindemann is with the Department of Electrical Engineering, University of Illinois at Urbana-Champaign, Urbana, IL 61801 USA slindema@uiuc.edu

S. M. LaValle is Associate Professor in the Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, IL 61801 USA. lavalle@uiuc.edu

This work was funded in part by an NSF Graduate Research Fellowship and NSF Award 0535007

II. Problem Formulation

In this work, we consider the global feedback navigation problem for car-like robots. The input to our algorithm is a polygonal environment and a goal state, x_g. We construct a feedback controller (equivalently, a vector field) such that from any initial state, following the integral curves of the vector field causes the robot to asymptotically converge to the goal state. Moreover, our algorithm computes a *smooth* feedback controller, which guarantees that all system trajectories are C^∞ differentiable.

Car-like robots are characterized by a bounded curvature constraint. This can be expressed as a constraint on the angular velocity: $|\dot{\theta}| \leq \alpha v_f$, in which $\alpha > 0$ and v_f is the forward velocity. The state transition equation for the system is then:

$$\begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \\ \dot{v}_f \end{pmatrix} = \begin{pmatrix} v_f \cos\theta \\ v_f \sin\theta \\ v_\theta \\ u_f \end{pmatrix}, \tag{1}$$

in which we again have $|v_\theta| \leq \alpha v_f$. This is illustrated in Figure 1. The control variables for this model are v_θ and u_f. Car-like robots pose a much more challenging control problem than differential drive (unicycle) robots due to the bounded curvature constraint. A unicycle is capable of going around a sharp corner without any difficulty because of its ability to rotate in place. A car-like robot, however, may need to reverse directions a number of times to move through an area with small clearance. The classic example of this is the problem of parallel parking a vehicle in a tight space.

Note that for car-like robots, there is no smooth (or even continuous) vector field over the configuration variables (x, y, θ) alone that can reverse the velocity of the vehicle. Any such trajectory could at best converge to $v_f = \dot{\theta} = 0$, since $v_f = \dot{\theta} = 0$ is an equilibrium, and there is no way to go from $v_f > 0$ to $v_f < 0$ except through the equilibrium. Taking v_f as a state variable allows us to define smooth controls that reverse velocity. In the examples in Section IV, what may appear to be "cusps" in the configuration space are actually smooth curves in the expanded (x, y, θ, v_f) space. Although we include v_f in the state space, note that there are no bounds on the inputs. This is important for obstacle avoidance, because there is no way to guarantee that the obstacles will be avoided without having unbounded inputs as the robot approaches the obstacle boundary.

Another notable feature of this model is the absence of a steering angle. In standard car models, the angular velocity is typically $\dot{\theta} = v_f/L \tan\phi$, in which L is the length of the car (i.e., the distance between the front and rear wheels), and ϕ is the steering angle. In our model, however, the robot is a point, and $L = 0$. As a result, steering angle ceases to be meaningful. That said, an implicit steering angle can be computed from the v_θ determined by our feedback controller, given some $L > 0$.

Brockett's condition [5] implies that no static, smooth vector field (feedback control) can stabilize the complete state of the system; therefore, we only attempt to stabilize the position of the robot, neglecting the orientation. If desired, other controllers can be used to stabilize the full state of the system after the robot reaches a sufficiently small neighborhood of the goal state. This is still a significant result, because other approaches are either open loop (losing the robustness of feedback control) or cannot guarantee obstacle avoidance as well as global asymptotic convergence.

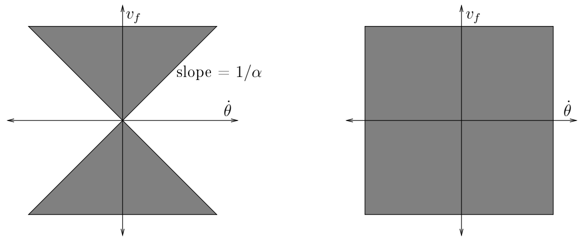

Fig. 1. Angular versus linear velocity, for a car-like robot (left) and a unicycle (right).

Our algorithm quickly and automatically computes *smooth feedback plans* for car-like robots in polygonal environments. Consider a system with state space X, input space U, and state transition equation $\dot{x} = f(x, u)$. For some goal state x_g, a smooth feedback plan is a vector field V on the free space (i.e., the state space minus the obstacles) that satisfies the following properties:

1) At every point x, $V(x)$ is admissible (i.e., there exists a $u \in U$ such that $V(x) = f(x, u)$).
2) All integral curves of V are smooth.
3) All integral curves of V avoid the obstacles and asymptotically converge to the goal state.

By *smooth* we mean infinitely differentiable, i.e., C^∞. Since the vector field V is admissible, it is by definition equivalent to a feedback controller defined as a mapping $\pi : X \rightarrow U$.

III. Feedback Plans for Car-Like Robots

In this section, we describe how to compute smooth feedback plans for car-like robots among obstacles. Initially, we will restrict our discussion to point robots; in the next section, we will discuss the extension to the case of a disc robot. As in our previous work, our strategy is as follows:

1) Decompose the environment into convex cells and compute a discrete plan over the cell connectivity graph.
2) Define local controllers (vector fields) corresponding to each cell and each face separating adjacent cells.
3) Construct a global controller by interpolating between vector fields defined in each cell.

By using a smooth interpolating function, we are able to guarantee smoothness in the interior of each cell as well as across cell boundaries.

A. Constructing a Smooth Feedback Plan

First, the environment (a polygonal subset of $\mathbb{R}^2$) must be decomposed into convex cells and a discrete plan computed. Computing convex decompositions is well-studied;

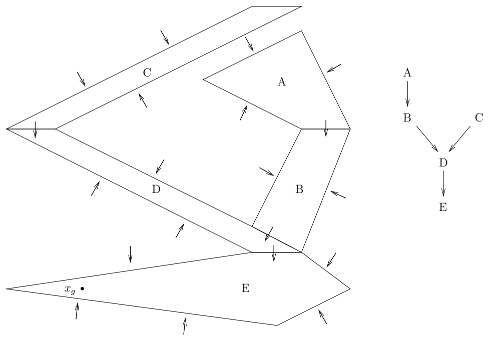

Fig. 2. An environment, corresponding face vector fields, and the graph showing how to reach the goal cell from any other cell.

algorithms by Keil [14] and Greene [11] compute convex decompositions with a minimum number of cells, and algorithms exist which compute more restricted types of decompositions such as vertical decomposition [13], [28] and triangulation. Keil's algorithm requires $O(nr^2 \log n)$ time, in which n is the number of vertices and r the number of reflex vertices. Triangulation can be done in linear time [7], and a practical implementation based on Seidel's algorithm that requires $O(n \log^* n)$ time is available [23]. Vertical decomposition (also known as trapezoidal decomposition) requires $O(n \log n)$ time. In addition to having favorable asymptotic performance, these algorithms are very practical. For example, the triangulation algorithm based of Seidel's algorithm can compute decompositions with hundreds of cells in a few milliseconds on a standard workstation.

Once the decomposition has been computed, consider the connectivity graph of the cells which is the dual of the decomposition. Let the cell containing the goal point, x_g, be denoted as C_g. Then, beginning with C_g, search the connectivity graph to obtain a chain of cells from any cell to C_g. Any graph search algorithm can be used, with or without any optimality criteria. For example, breadth-first search can be used, with a corresponding linear time bound. The resulting directed graph defines a "successor" for every cell except the goal cell; the successor of a cell is the next cell on the route to the goal. We call every cell with a successor an *intermediate* cell, in distinction with the goal cell, which has no successor. See Figure 2.

The remaining task is to construct local controllers that avoid obstacles, are consistent with the computed discrete plan, and satisfy the smoothness requirement. Since each node in the graph corresponds to a convex cell, consistency with the high level plan is equivalent to solving the *control to facet* problem: that is, all integral curves must exit from a particular facet in finite time while avoiding all other facets. We will construct such controllers by defining a vector field over the cell, as well as one corresponding to each face. In the interior of the cell, we will interpolate between these vector fields in such a way that all the requirements are met. In particular, the vector field at a point p in a particular cell is defined as

$$V(p) = b(p)V_i(p) + (1 - b(p))V_c(p), \tag{2}$$

in which b is the interpolating function, V_i is the vector field corresponding to some face f_i, and V_c is the vector field corresponding to the cell.

Interpolation is accomplished using the generalized Voronoi diagram (GVD) [1]. See Figure 3. The GVD partitions the cell into regions corresponding to each face, such that face f_i is the closest face to every point p in the region corresponding to f_i (denote this as the region of influence of f_i). Within the region of influence of f_i, we interpolate between the cell vector field (on the GVD faces) and the vector field corresponding to f_i (on the face itself). We smoothly interpolate using *bump functions*, which are defined as follows:

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

1) *The support of ρ is contained in U.*
2) *For every $x \in K$, $\rho(x) = 1$.*

We construct a bump function that transitions smoothly from 0 to 1 on the unit interval as follows. First, define

$$\lambda(s) = (1/s)e^{-1/s}. \tag{3}$$

The bump function is then defined as

$$b(s) = \begin{cases} 0 & s \le 0 \\ \frac{\lambda(s)}{\lambda(s)+\lambda(1-s)} & 0 < s < 1 \\ 1 & 1 \le s. \end{cases} \tag{4}$$

An illustration of this bump function is given in Figure 4. The bump function has the important property that all derivatives equal zero at the endpoints of the unit interval. The parameter we use for the bump function is the product of a number of analytic switches. The parameter function must be smooth over the interpolation region, and the bump function makes sure that the derivatives match across the cell and GVD boundaries. For any point p in the region of influence of face f_i, let

$$s(p) = 1 - \prod_{j \ne i} \frac{d(p, f_j) - d(p, f_i)}{d(p, f_j)}. \tag{5}$$

This function is smooth (except at the cell vertices), and has the desired property of being identically equal to zero on the cell face and one on the faces of the GVD, where points are equidistant from multiple faces. Using the shorthand, $b(p) = b(s(p))$, we obtain the interpolation function in Equation 2.

Now, we describe the cell and face vector fields in detail. The vector field corresponding to the cell, V_c, contains only a rotational component. The purpose of the rotational vector field is to orient the robot so it may freely cross the exit face into the next cell, without encountering another face first. Assume that we are in an intermediate cell (we will consider the case of the goal cell later), and that we are given a point on the exit face of the cell; such a point is trivial to compute. Define θ_e as the angular error of the robot to the

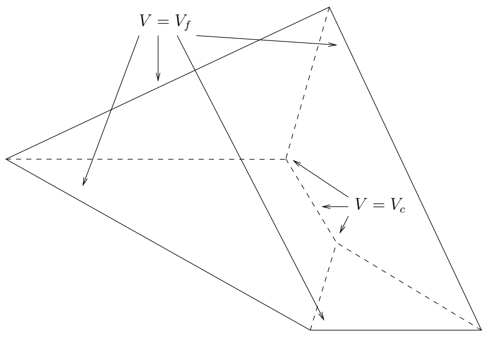

Fig. 3. An individual cell, partitioned using the generalized Voronoi diagram.

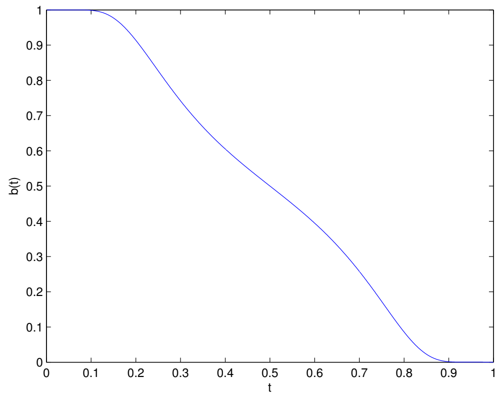

Fig. 4. The bump function given by Equation 3.

point, given the current orientation. The absolute angle to the point from the robot is $\theta_p = \text{atan2}(y_p - y, x_p - x)$. We then have $\theta_e = \min\{\theta - \theta_p, 2\pi - \theta + \theta_p\}$. Clearly, this has a maximum of π when the robot is pointing directly away from the point and a minimum of zero when the robot is oriented directly at the point. Let $\theta_e = \theta - \theta_p$. For any simple control law of the form $\dot{\theta} = -K\theta_e = -K(\theta - \theta_p)$ with K ¿ 0, it is clear that θ_e will remain less than π for all time and therefore $\theta_e = \theta - \theta_p$ always (a transition to $\theta_e = 2\pi - \theta + \theta_p$ is never made). The importance of this fact will be made clear. Define the cell vector field:

$$V_c = -\alpha |v_f| \text{sgn}(\theta_e) b(\theta_e/\epsilon) \frac{\partial}{\partial \theta}, \tag{6}$$

for some ϵ ¿ 0, with b the bump function above and v_f the velocity. The presence of αv_f in the product guarantees that the bounded curvature constraint is satisfied by this vector field. Also, θ_e never switches between terms in the min expression, which is important for preserving smoothness. Finally, the sign of θ_e never changes, as we argued above; this is also necessary for smoothness. The effect of the rotational vector field is to orient the robot toward the point p on the exit face. Even though the value of θ_e is not monotonically decreasing, it should be clear that it will eventually converge to zero. We will prove this later.

Now we consider the vector fields $\{V_i\}_1^n$ corresponding to the faces $\{f_i\}_1^n$. The face vector fields will have no rotational component, but will consider only the forward velocity. The purpose of the face vector fields is to prevent the robot from reaching any face other than the exit face (by reversing the velocity of the robot). We will construct each face vector field using two other vector fields; one to decelerate the robot and prevent it from hitting the face, and one to accelerate the robot in the opposite direction.

A couple of preliminary definitions are required. Assume that the point $q = (x, y, \theta, v_f)$ is in the region of influence of f_i (i.e., the Voronoi region corresponding to f_i). Then, define the *hitting time* t_h to be the time until the robot hits the f_i, while maintaining the current heading and forward velocity. If the integral curve through q does not hit f_i but leaves the region of influence of f_i, let $t_h = \infty$. Note that the integral curve containing q does not depend on the velocity v_f, since the rotational component defined by V_c is proportional to v_f; thus, whether or not the integral curve hits f_i can be determined without considering V_i. Define the saving vector field:

$$V_s(q) = -(v_f/t_h) \frac{\partial}{\partial v_f}. \tag{7}$$

Observe that if the robot travels in a straight line, this is sufficient to stop the robot before the edge is reached, assuming $|v_f| \leq 1$.

In addition to the saving vector field, we need a vector field over the entire cell that accelerates the robot away from the face. This will increase the forward velocity if the robot is moving away from the face (i.e., $t_h = \infty$) and slow the robot down if it is moving toward the face ($t_h < \infty$). Let $V_+ = \text{sgn}(v_f)(1 - b((|v_f|/\epsilon) - (1 - \epsilon)))$; this vector accelerates the robot to its maximum speed of one. Also define $V_- = -\text{sgn}(v_f)$, which decelerates the robot. Let θ_i be the absolute value of the angle between the robot's velocity and the inward pointing normal of face f_i; then define the acceleration vector field:

$$V_a(q) = b((\theta_i/\epsilon)V_+(q) - (1 - b((\theta_i + \epsilon)/\epsilon))V_-(q) \tag{8}$$

If the integral curve does not hit the face (with $t_h = \infty$ as a result), simply let $V_a(q) = V_+(q)$. Simply, this vector field decelerates if $\theta_i < 0$ and accelerates if $\theta_i > 0$, smoothly interpolating between the two in the interval $(-\epsilon, \epsilon)$.

The face vector field for face f_i is then defined as:

$$V_i = b(t_h - t_{safe})V_s(q) + (1 - b(t_h - t_{safe}))V_a(q), \tag{9}$$

for some $t_{safe} > 0$. The face vector field smoothly interpolates between the saving vector field (when collision is imminent) and the ordinary acceleration vector field (when collision is at least t_{safe} time from occurring). The only exception is when the face is the exit face, in which case we set $V_i = V_a$, since we want the integral curve to reach the exit face.

We have completely described the construction of the vector field over the intermediate cells; now we consider the case of the goal cell. In this case, the target point is the goal point x_g which is in the interior of the cell, rather than on the boundary as in the previous case. This would seem to

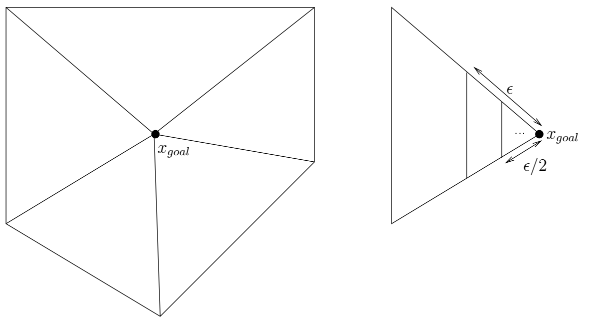

Fig. 5. A goal cell, subdivided. On the right, an infinite sequence of cells is created, each of which is guarantee that the robot is within $\epsilon/2^i$ for some i.

complicate things because it is no longer trivial to argue that rotating in the same direction will lead to orientation toward the target point. Since x_g is in the interior of the cell, it is possible for the integral curves to converge to a circular orbit about the goal point. The most straightforward way to remedy this to partition the goal cell into new cells such that the goal point is once again on the boundary. This is easily accomplished; partition the cell into triangles, one per face, the vertices of which are the vertices of that face together with the goal point. Then, for some $\epsilon > 0$, add a new edge with vertices on the two edges incident on x_g and ϵ distant from it. Thus, crossing that edge ensures that the robot is no more than ϵ away from the goal point. At this point, the vector field is constructed exactly as we have already described. This procedure is illustrated in Figure 5. Inside the new "corner" cell, the procedure can be repeated by trimming off the corner at distance $\epsilon/2$, and so on. This will guarantee global convergence to the goal state (in position, not orientation).

B. Theoretical Analysis

To establish that our algorithm correctly computes feedback plans, we need to prove several important results. First, we verify that the controller satisfies the bounded curvature constraints. Second, it is straightforward to show that the controller is safe, avoiding all obstacles. Then, we need to show that all integral curves of the global vector field V are smooth. Finally, we prove that the controller stabilizes the position of the robot. These results directly follow from the construction of the vector fields over the individual cells.

Theorem 2 *The vector field V satisfies the bounded curvature constraint.*

Proof: Our two control inputs, as determined by V, are $\dot{v}_f$ and $\dot{\theta}$. From the definition of the cell vector field V_a, we see that $\dot{\theta} \leq \alpha v_f$, which is precisely the bounded curvature constraint. ■

Theorem 3 *The controller defined by V avoids all obstacles.*

Proof: Each obstacle face is a face on one of the cells in the convex decomposition. Consider first the case of a trajectory approaching a point in the interior of a face f_i. From the construction of V, we see that $V = V_i$ on the face. Therefore, we need only consider the face vector field V_i. As the robot approaches the face, the hitting time t_h goes to zero and $V_i \to V_s$, the saving vector field. We have already mentioned (and it is trivial to verify) that the saving vector field is always strong enough to reverse the velocity before the face is reached.

The other important case is when the integral curve approaches an endpoint of the face, rather than a point in its interior. It is somewhat more difficult to show that safety is preserved in this case. The difficulty lies in the fact that V does not approach V_i as the robot nears the vertex, due to the (unavoidable) non-smoothness in the interpolation function. We do not give the details here, but it suffices to show that for any $0 < \beta < 1$, $V = \beta V_i$ is sufficient to keep the robot from reaching the vertex. Showing this result covers all cases except that in which the integral curve is tangent to the GVD face at the vertex. One can show that it is always possible to choose the saving vector field V_s large enough that obstacle collision is avoided in this case as well. ■

Theorem 4 *All integral curves of V are smooth.*

Proof: This is also simple to verify from the construction of V. The bump function b is smooth, as is the analytic switch we use to interpolate between the face and cell vector fields in each face's region of influence. So we simply need to consider the smoothness of the cell and face vector fields. Consider the face vector fields. If the velocity-independent integral curve does not lead to the face, then we have $t_h = \infty$ and the velocity along that integral curve smoothly increases to the maximum velocity, $|v_f| = 1$. Consider, then, the integral curves along which the system must reverse direction so that the face is not reached. In this case, the vector field is a smooth interpolation of the saving vector field V_s and the acceleration field V_a. The saving vector field is clearly smooth, since the hitting time and velocity are smooth. The only potential issue is when the the hitting time goes to infinity, which happens when $v_f = 0$ or θ becomes parallel to the face. At this point, however, we have $V_i = V_a$ and all derivatives of the bump function are identically zero; therefore, smoothness is preserved. Likewise, V_a is also smooth.

Now, consider the cell vector field, which controls rotation. The velocity v_f is smooth, as we have shown above. We have also discussed the fact that θ_e is smooth, since the system always rotates in the same direction. Therefore, the cell vector field V_a is smooth. Therefore, all component functions and vector fields are appropriately smooth, so all integral curves of V are smooth. ■

Theorem 5 *All integral curves of V converge to the goal state, x_g.*

Proof: To show this, we will prove that for any cell, all integral curves will reach the exit face of that cell in finite time. After a number of such faces are crossed, we can guarantee that the robot is at most $\epsilon, \epsilon/2, \epsilon/4, \ldots$, distant from the goal state, which establishes convergence.

First, we verify that when the robot changes direction to avoid hitting a face, it actually changes direction rather than simply converging to a point. The robot will not converge to a point under the acceleration vector field V_a, because the only place where $V_a = 0$ is when $\theta = \theta_i$. The only time when $v_f = 0$ is when the robot reverses direction, but it is not possible for θ to equal θ_i at this point, because then the robot would not be at risk of hitting the face (i.e., it would proceed to another face's region of influence instead). Therefore, the robot would not decelerate and change direction. The case where $v_f = 0$ and $\theta = \theta_i$ does correspond to an equilibrium point, but one which has no region of attraction (if the system starts at this point, perturb it; otherwise, no integral curve converges to the state).

Consider, then, the saving vector field V_s. The term v_f/t_h implies that the deceleration applied is independent of the magnitude of the velocity, since t_h depends linearly on v_f. Since the deceleration is velocity-independent, this will never lead to the robot converging to a point.

One potential problem is on the faces of the GVD, where $V = V_c$. There is no linear acceleration on the GVD face since V_c is purely rotational, and since V_c depends linearly on v_f, any point on the GVD such that $v_f = 0$ is an equilibrium point. However, it can be seen that these have no region of attraction, since any perturbation will lead away from the equilibrium region. This means that the robot will never converge to this region. If the robot is in this location due to an initial condition, a random perturbation will free it from the equilibrium, so there is no problem. Note that while we do not describe it here, it is possible to add a linear acceleration component to V_c which will eliminate this issue entirely.

Now, we need to show that with respect to its orientation, the robot will eventually be oriented toward the target point enough that it will be able to leave via the exit face of the cell. Recall that we have already discussed the fact that within a cell, the robot will always rotate in the same direction, due to the cell vector field. Define a *trajectory segment* to be an interval of an integral curve between two velocity reverals. Consider a sequence of trajectory segments t_1, t_2, t_3, t_4. Denote by $e_1, \ldots, e_4$ the linear extrapolation of the endpoints on the faces of the polygon. Assuming that we are rotating in the positive direction, we know that e_3 is located counter-clockwise of e_1 along the boundary of the polygon, and e_4 is likewise counter-clockwise of e_2. Taking then a sequence of every other endpoint, we can see that this must eventually reach the exit face. There can be no limit point at any other point along the boundary, because this would imply that the angular change over the trajectory segments goes to zero, which is impossible. Therefore, the robot will always rotate in such a way to be oriented toward the exit face and therefore exit the cell via that face. ■

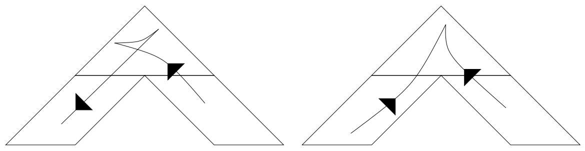
Fig. 6. On the left, the robot makes an extra reversal but always crosses edges moving forwards. On the right, the robot crosses moving backwards but saves a reversal.

IV. Extensions and Practical Results

Our feedback planning algorithm can be improved and extended in a number of ways. First, we will discuss alternative vector field choices that may lead to more desirable system trajectories. We will also describe how to modify the method for the case of a disc robot, as opposed to the point robot which we have considered so far.

In Section III, we defined the cell vector field in such a way that the robot always rotates in the same direction. We utilized this in the convergence proof above. Practically, this means that the robot will leave a cell with a velocity identical in sign to the velocity it entered with. This means that if the robot enters the cell moving forward, it will exit the cell moving forward. This is desirable behavior, if there is a practical difference between forwards and backwards for the robot. If there is not such a distinction, then it may be possible to improve path quality by allowing the robot to reverse the direction with which it crosses the edge. This can be seen in Figure 6. In order to do this, the robot may no longer always rotate in the same direction, but may reverse the direction of rotation when the robot is moving away from the target point. Following this strategy may lead to many fewer path reversals, especially when the robot must go around sharp corners. We have not proven convergence for this approach, but we conjecture that it will hold.

A second way to modify the vector field is to permit the robot to reverse direction even when a collision with the edge is not imminent. For example, if the robot is moving away from the exit face but oriented toward it, it is advantageous to for the robot to immediately reverse directions rather than waiting until an edge is approached. We have not investigated this approach at present.

It is straightforward to extend this method to the case of a disc robot. Computing the configuration space obstacles for a disc robot is well-known [16]; each obstacle is transformed to its Minkowski sum with the robot body. The obstacle boundary, then, no longer consists solely of line segments but line segments and circular arcs. Two ways to deal with this are readily apparent. First, the arcs can be conservatively approximated with line segments, and our method applied as we have already described. If this approach is followed, care must be taken to preserve the topology of the configuration space, so that completeness is preserved.

The second method is more complicated. First, replace each arc with a single line segment (the chord of the circle sharing the endpoints of the arc), and perform the cell decomposition as before. Reinsert the chords into the de-

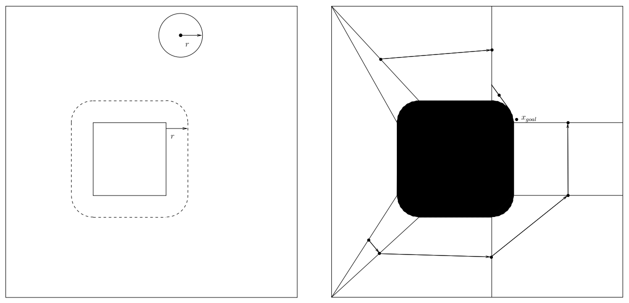

Fig. 7. A disc robot with a square obstacle. On the right, the grown obstacle and an example decomposition. Target points on the exit edges are drawn; the goal cell must be subdivided because it is not star-shaped with respect to the goal.

composition, yielding cells which are no longer polygonal or convex. Further decompose cells containing circular arcs so that every cell is star-shaped with respect to its target point. Then, the method we described above can be applied (with appropriately modified GVD, distance functions, etc.). The star-shaped requirement is important because the robot must be able to travel directly to the exit edge if it has converged sufficiently in rotation. See Figure 7 for an example.

Although our method defines a global feedback plan, it can also be used to generate open loop trajectories. In fact, dynamic programming can be used to reduce the number of path reversals in an individual open loop trajectory considerably, in the following way. Consider an initial state x_i. If there are n possible exit edges, then there are n possible choices of cell vector field; add an open node corresponding to each choice to a priority queue. For each of these, the integral curve through the initial state will reach some face, either crossing it or reversing velocity. For each possibility, insert a new node in a priority queue with cost equal to the number of reversals so far (in the first iteration, the node corresponding to crossing the edge would have lower cost with zero reversals, and the state remaining in the current cell would have high cost with one reversal). Each time a cell is reached, new nodes are added based on the number of possible exit edges. Using dynamic programming, continue to extend each trajectory segment, adding new elements to the queue every time a cell boundary is reached. Once the goal is reached, we can guarantee that the minimum number of reversals has been achieved for that open loop query, from the class of all possible controllers constructed according to our basic method.

Finally, we present several examples of paths computed using our method in Figures 8 and 9. These figures depict only two-dimensional projections of individual integral curves; however, our method determines a global feedback plan over the entire four-dimensional state space. In some cases, it may seem that there are unnecessarily many path reversals. To some extent, reversals are unavoidable because of the bounded curvature restriction; several properties of our algorithm increase the number of path reversals. In particular, many reversals can be induced by the requirement that once a robot enters a particular cell, it cannot leave that cell except via the exit face (small cells greatly magnify this problem).

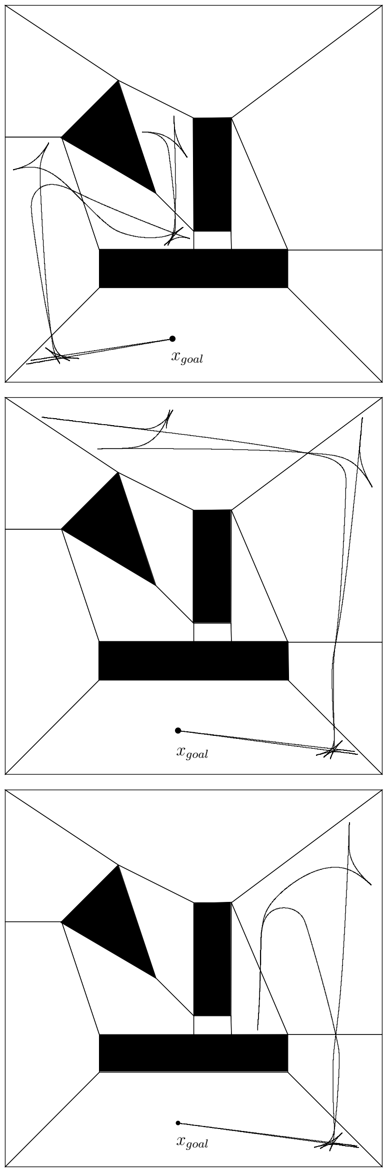

Fig. 8. Several paths through an obstacle cluttered environment, for two robots with different turning radii.

Other factors that contribute to reversals are the choice of the target point and the fact that edges are crossed with zero angular velocity. It is simple to modify the choice of target point; for example, it could be replaced with a target interval on the exit face, which would still guarantee convergence but would not "compress" the integral curves to the target point as is seen in the examples (in the examples, the target point is the midpoint of the edge). It may be possible to add a rotational component to the face vector fields, which would likely improve path quality; however, we have not investigated whether global convergence would hold under such a scheme.

V. Conclusion

In conclusion, we have described a method for constructing provably safe smooth feedback laws for kinematic car-like robots in obstacle-cluttered polygonal environments. We

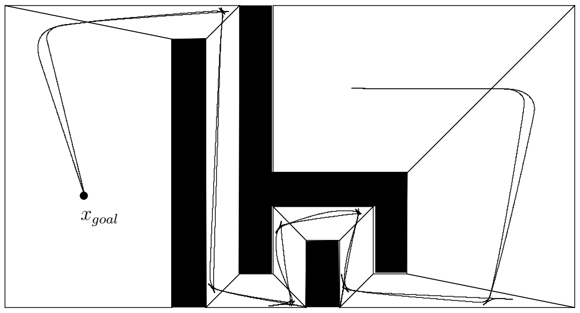

Fig. 9. Paths through a winding corridor, for robots with different turning radii.

have proven that the vector field constructed by our algorithm has the appropriate safety and smoothness properties and that it stabilizes the position of the robot in the plane. We neglect orientation because Brockett's condition implies that no smooth time-invariant feedback law can stabilize the entire state of the robot. We have also presented several ways to extend our method through different choices of component vector fields and have extended our method to the case of a disc robot.

Our primary goal in future work is to extend our approach to robots with polygonal bodies. Although some robots can be reasonably approximated with discs, many cannot be; in these cases, we would still like to compute global smooth feedback plans. This is a challenging problem, but one with significant practical impact.

REFERENCES

[1] F. Aurenhammer, "Voronoi diagrams – A survey of a fundamental geometric structure," *ACM Computing Surveys*, vol. 23, pp. 345–405, 1991.

[2] C. Belta, V. Isler, and G. J. Pappas, "Discrete abstractions for robot motion planning and control in polygonal environments," *IEEE Transactions on Robotics*, vol. 21, no. 5, pp. 864–874, Oct. 2005.

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

[4] A. Bloch, J. Baillieul, P. Crouch, and J. Marsden, *Nonholonomic Mechanics and Control*. New York, NY: Springer-Verlag, 2003.

[5] R. A. Brooks, "Solving the find-path problem by good representation of free space," *IEEE Trans. Syst., Man, Cybern.*, vol. 13, no. 3, pp. 190–197, 1983.

[6] F. Bullo and A. D. Lewis, *Geometric Control of Mechanical Systems*. Berlin: Springer-Verlag, 2004.

[7] B. Chazelle, "Triangulating a simple polygon in linear time," *Discrete and Computational Geometry*, vol. 6, no. 5, pp. 485–524, 1991.

[8] D. C. Conner, A. A. Rizzi, and H. Choset, "Composition of local potential functions for global robot control and navigation," in *Proceedings IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2003, pp. 3546–3551.

[9] J. Cortés, S. Martínez, J. P. Ostrowski, and H. Zhang, "Simple mechanical control systems with constraints and symmetry," *SIAM Journal on Control and Optimization*, vol. 41, no. 3, pp. 851–874, 2002.

[10] M. Egerstedt, X. Hu, and A. Stotsky, "Control of a car-like robot using a virtual vehicle approach," in *Proceedings IEEE Conference on Decision & Control*, 1998, pp. 1502–1507.

[11] D. H. Greene, "The decomposition of polygons into convex parts," in *Computational Geometry: volume 1 of Advances in Computing Research*, F. P. Preparata, Ed. London: JAI Press, 1983, pp. 235–259.

[12] L. C. G. J. M. Habets and J. H. van Schuppen, "Control to facet problems for affine systems on simplices and polytopes – with applications to control of hybrid systems," in *Proceedings IEEE Conference on Decision & Control, and the European Control Conference*, 2005.

[13] D. Halperin, "Arrangements," in *Handbook of Discrete and Computational Geometry, 2nd Ed.*, J. E. Goodman and J. O'Rourke, Eds. New York: Chapman and Hall/CRC Press, 2004, pp. 529–562.

[14] J. M. Keil, "Decomposing a polygon into simpler components," *SIAM Journal of Computing*, vol. 14, pp. 799–817, June 1985.

[15] O. Khatib, "Real-time obstacle avoidance for manipulators and mobile robots," *International Journal of Robotics Research*, vol. 5, no. 1, pp. 90–98, 1986.

[16] S. M. LaValle, *Planning Algorithms*. Cambridge, U.K.: Cambridge University Press, 2006, available at http://planning.cs.uiuc.edu/.

[17] S. M. LaValle and P. Konkimalla, "Algorithms for computing numerical optimal feedback motion strategies," *International Journal of Robotics Research*, vol. 20, no. 9, pp. 729–752, Sept. 2001.

[18] S. R. Lindemann, I. I. Hussein, and S. M. LaValle, "Real time feedback control for nonholonomic mobile robots with obstacles," in *Proceedings IEEE Conference on Decision & Control*, 2006.

[19] S. R. Lindemann and S. M. LaValle, "Smoothly blending vector fields for global robot navigation," in *Proceedings IEEE Conference Decision & Control*, 2005, pp. 3353–3559.

[20] ——, "Computing smooth feedback plans over cylindrical algebraic decompositions," in *Proceedings Robotics: Science and Systems*, 2006.

[21] A. D. Luca, G. Oriolo, and C. Samson, "Feedback control of a nonholonomic car-like robot," in *Robot Motion Planning and Control*, J.-P. Laumond, Ed. Berlin: Springer-Verlag, 1998, pp. 171–253.

[22] R. M. Murray and S. Sastry, "Nonholonomic motion planning: Steering using sinusoids," *IEEE Transactions on Automatic Control*, vol. 38, no. 5, pp. 700–716, 1993.

[23] A. Narkhede and D. Manocha, "Fast polygon triangulation based on seidel's algorithm," in *Graphics Gems V*, A. Paeth, Ed. New York: Academic, 1995, pp. 394–397.

[24] E. Rimon and D. E. Koditschek, "Exact robot navigation using artificial potential fields," *IEEE Transactions on Robotics & Automation*, vol. 8, no. 5, pp. 501–518, Oct. 1992.

[25] B. Roszak and M. E. Broucke, "Necessary and sufficient conditions for reachability on a simplex," in *Proceedings IEEE Conference on Decision & Control, and the European Control Conference*, 2005.

[26] C. Samson, "Time-varying feedback stabilization of car-like wheeled mobile robots," *International Journal of Robotics Research*, vol. 12, pp. 55–65, 1993.

[27] C. Samson and K. Ait-Abderrahim, "Feedback control of a nonholonomic wheeled cart in Cartesian space," in *Proceedings IEEE International Conference on Robotics & Automation*, 1991, pp. 1136–1141.

[28] R. Seidel, "A simple and fast incremental randomized algorithm for computing trapezoidal decompositions and for triangulating polygons," *Computational Geometry: Theory and Applications*, vol. 1, no. 1, pp. 51–64, 1991.

[29] H. G. Tanner, S. Loizou, and K. J. Kyriakopoulos, "Nonholonomic stabilization and collision avoidance for mobile robots," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2001, pp. 1220–1225.

[30] H. G. Tanner, S. G. Loizou, and K. J. Kyriakopoulos, "Nonholonomic navigation and control of cooperating mobile manipulators," *IEEE Transactions on Robotics and Automation*, vol. 19, no. 1, pp. 53–64, Feb. 2003.

[31] J. N. Tsitsiklis, "Efficient algorithms for globally optimal trajectories," *IEEE Transactions on Automatic Control*, vol. 40, no. 9, pp. 1528–1538, Sept. 1995.