Integrated Planning and Control for Convex-bodied Nonholonomic systems using Local Feedback Control Policies

David C. Conner, Alfred A. Rizzi, and Howie Choset

CMU-RI-TR-06-34

August 2006

Robotics Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

Abstract

We present a method for defining a hybrid control system capable of simultaneously addressing the global navigation and control problem for a convex-bodied wheeled mobile robot navigating amongst obstacles. The method uses parameterized continuous local feedback control policies that ensure safe operation over local regions of the free configuration space; each local policy is designed to respect nonholonomic constraints, bounds on velocities (inputs), and obstacles in the environment. The hybrid control system makes use of a collection of these local control policies in concert with discrete planning tools in order to plan, and replan in the face of changing conditions, while preserving the safety and convergence guarantees of the underlying control policies. In this paper, we provide details of the system models and policy definitions used to validate the approach in simulation and experiment with a convex-bodied wheeled mobile robot. This work is one of the first that combines formal planning with continuous feedback control guarantees for systems subject to nonholonomic constraints, input bounds, and non-trivial body shape.

Contents

1 Introduction

The problem of simultaneously planning and controlling the motion of a convex-bodied wheeled mobile robot in a cluttered planar environment is challenging because of the relationships among control, nonholonomic constraints, and obstacle avoidance. The objective is to move the robot through its environment such that it arrives at a designated goal set without coming into contact with any obstacle, while respecting the nonholonomic constraints and velocity (input) bounds inherent in the system.

Conventional approaches addressing this problem typically decouple navigation and control [12, 31, 33]. First, a planner finds a path, and then a feedback control strategy attempts to follow that path. Often the methods assume a point or circular robot body; other non-trivial robot body shapes complicate the problem of finding a safe path, as the path must be planned in the free configuration space of the robot. This leaves the challenging problem of designing a control law that converges to the one-dimensional path, while remaining safe with respect to obstacles.

We address the coupled navigation and control problem by generating a vector field along which the robot can "flow." Unfortunately, determining a global vector field that satisfies all of these objectives for constrained systems can be quite difficult. Our approach, inspired by the idea of *sequential composition* [11], uses a collection of local feedback control policies coupled with a switching strategy to generate the vector field. Composing these relatively simple policies induces a piecewise continuous vector field over the union of the policy domains. A *policy* is specified by a configuration dependent vector field defined over a local region of the robot's free configuration space, termed a *cell*. From this vector field and knowledge of the current robot state, the control inputs are determined such that the closed-loop dynamics flows to the specified policy goal set within the cell. Figure 1 shows four paths induced by invoking the local policies for four different initial conditions in the robot experiments, as will be discussed in Section 6.

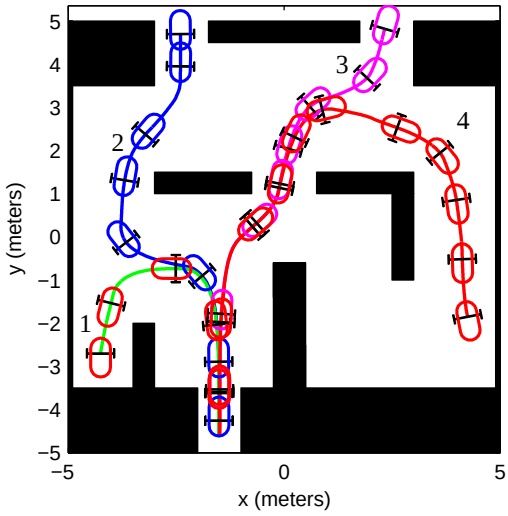


Figure 1: Experimental results of four robot runs using the proposed hybrid control framework. An explicit desired path is never calculated, the path is induced by sequentially chaining local feedback control policies together; the policy ordering is determined by a discrete planner. The induced path preserves the local guarantees of the local polices, and is thus free of collision while respecting the system constraints.

A discrete transition relation represented by a graph is induced by the transition between the domain of one policy to the domain of a second policy containing the policy goal set of the first. On-line planning, and re-planning under changing conditions, becomes more tractable on the graph, allowing us to bring numerous discrete planning tools to bear on this essentially continuous problem. By sequencing the local policies according to the ordering determined by a discrete planner, the closed loop dynamics induce the discrete transitions desired by the discrete plan. The overall hybrid (switched) control policy responds to system perturbations without the need for re-planning. In the face of changing environmental conditions, the discrete graph allows for fast online re-planning, while continuing to respect the system constraints.

By coupling planning and control in this way, the hybrid control system plans in the discrete space of control policies; thus, the approach represents a departure from conventional techniques. A "thin" path or trajectory is never explicitly planned; instead, the trajectory induced by the closed-loop dynamics flows along the vector field defined by the active policy, which is chosen according to an ordering determined by a discrete planner. A plan over the discrete graph associated with the collection of policies corresponds to a "thick" set of configurations within the domains of the associated local policies.

This approach offers guaranteed convergence over the union of the local policy domains. For the method to be complete, the entire free configuration space must be covered by policy domains (cells). This is a difficult problem due to the multiple constraints on the robot dynamics. Herein, the focus is on deploying a "rich enough" collection of policies that enables the navigation problem to be addressed over the bulk of the free configuration space.

This paper describes the basic requirements that any local policy must satisfy to be deployed in this framework; each local policy must be provably safe with respect to obstacles and guarantee convergence to a specified policy goal set, while obeying the system constraints within its local domain. We develop a set of generic policies that meet these requirements, and instantiate these generic policies in the robot's free configuration space. The underlying equations are detailed, and examples are provided. Finally, a concrete demonstration of controlling a real mobile robot using a collection of local policies is described.

2 Related Work

The approach advocated in this paper maintains the coupling between planning and control, unlike conventional approaches that decouple the problem into a path planning problem and a path-following control problem. This section presents a brief overview of conventional approaches, and concludes with a discussion of the sequential composition approach that motivates this work.

2.1 Motion Planning and Control with Nonholonomic Constraints

The control of systems with nonholonomic differential constraints is more complex than control of purely holonomic[1] systems. This section contains an overview of techniques used to plan paths and trajectories for nonholonomic systems, and address four basic control problems for nonholonomic systems – stabilization, path-following, trajectory-tracking, and point-to-point steering [18, 30, 43]. The discussions give insight into the difficulties associated with nonholonomic motion planning and control. For a more thorough discussion and reference list, refer to [30, 32, 33].

To recognize the complexity introduced by nonholonomic constraints, consider the "simple" problem of stabilizing a system about a given equilibrium point. Brockett's theorem provides necessary conditions for the existence of a smooth, time-invariant feedback control law that stabilizes the system about the given point [10]. It is well known that many nonholonomic systems, although small-time locally controllable, fail Brockett's test [30, 43]. This precludes the use of conventional feedback control based on potential functions. Several classes of stabilizing feedback control policies have been developed: discontinuous time invariant, time varying, and hybrid control policies [30]. Given the complexity of this control task, it is no surprise that the coupled navigation and control task is more complex than for holonomic systems.

Many approaches addressing the coupled navigation and control problem for nonholonomic systems decompose the problem into a motion planning problem, followed by a control problem. A common planning approach is to pretend the system is holonomic and hope for the best. In the holonomic case, the path or trajectory can be planned using any of a number of methods [12, 35]. Grid-based search algorithms are popular in path planning, as are roadmap methods such as Voronoi diagrams. Potential function methods, while not necessarily complete, are also widely used in path/trajectory planning.

Given a path through the environment, a path-following algorithm is then used to converge to the desired path. There are two basic formulations to path-following. In the first, a designated point on the robot traces a given path in the workspace, without concern for orientation [15]. This may fail in cluttered environments as the designated point may exactly follow a safe path, yet still allow the system to collide with an obstacle at another point on the robot body. The second formulation attempts to track position and orientation of a path in the free configuration space [17].

If the environment is not overly cluttered, so that deviations are not catastrophic, path following may work well. In fact, if a system is small-time locally controllable, any continuous path can be approximated arbitrarily well [34]. Unfortunately, the methods used to control the system along arbitrary paths often lead to highly oscillatory motions that require great control effort. If the path does not respect the system constraints, the resulting motions may require an inordinate number of reversals to follow the desired path. One approach to address this problem is to plan a path without regard to nonholonomic constraints, and perturb the path to respect nonholonomic constraints [51].

Techniques that consider the nonholonomic constraints during planning typically use only a discrete set of feasible motions. The shortest feasible path (SFP) metric is used to plan paths that approximate a holonomic path using a finite number of motions[42]. The motions are selected by choosing the shortest feasible path to a point on the holonomic path intersecting the largest ball defined by the SFP metric [56]. Dynamic programming methods determine an optimal path for a discrete set of controls (e.g. hard left, soft left,

[1] Here we follow the literature, and refer to systems lacking nonholonomic constraints as "holonomic".

straight, soft right, hard right) [3, 21, 36]. Probabilistic roadmaps (PRM) and rapidly-exploring random trees (RRT) are other discrete approaches to determining feasible paths for nonholonomically constrained systems [37, 54]. With the exception of dynamic programming, these discrete methods are not feedback based, and require replanning if the system deviates from the desired path.

Even if the planned path respects the nonholonomic constraints, the path must be followed by the robot. The presence of nonholonomic constraints renders path-following a non-linear controls problem. Control laws based on a Lyapunov analysis [17] or feedback linearization [18, 50] are common. Most path-following algorithms assume continuous motion, with a non-stationary path defined for all time. This (temporarily) avoids Brockett's problem with stabilization to a point [15, 17]. The path-following control policy asymptotically brings the error between the desired path and the actual path to zero. Typically, the control policy is constructed for a specific vehicle and class of paths [2, 17, 15, 55].

The path-following control laws are unaware of environmental obstacles; therefore, for a nonzero initial error or perturbation during motion, the system may collide with an obstacle. Path-following may be coupled with local obstacle avoidance, but this may invalidate the convergence guarantees. Thus, if the errors are large enough, the paths must be replanned, starting from the current location.

In addition to path-planning, trajectories that specify when the system arrives at points along the path may be planned [17, 50]. Trajectory-tracking problems can be problematic if the system is subject to a constraint that delays the tracking [17]. In this case, the accumulated error may make the system unstable or require unreasonably high inputs. Another possible problem is that trajectory-tracking controls may switch directions for certain initial conditions [50]. Unless the time matching along the trajectory is crucial, path-following is often a better formulation [17, 50].

In related work, several methods – including sinusoidal inputs, piecewise constant inputs, optimal control, and differentially flat inputs – solve the point-to-point steering problem between two positions using open loop controls [34, 43]. These methods are sometimes incorporated into the discrete planning systems. Each of these methods is an open loop control method that does not respect obstacles. Collision detection must be performed after the path is found to determine feasibility in a cluttered environment [27]. These point-to-point steering methods are strictly open loop, and not suitable for feedback control. The inevitable errors necessitate repeated applications to induce convergence to the goal point.

There have been a few attempts to address the coupled navigation and control problem for nonholonomic systems. A method based on potential fields uses resistive networks to approximate the nonholonomic constraints [14]. This approach requires a discretization of the configuration space, and is therefore subject to numerical difficulties when calculating derivatives necessary for feedback control. Other approaches define integral sub-manifolds in configuration space that contain the goal [25, 29, 41]. On the sub-manifold, the nonholonomic constraints are integrable, and the control naturally respects the constraints. The approaches drive the system to the sub-manifold, and then along the sub-manifold to the goal. While these methods are suitable for feedback control implementations, determination of a suitable manifold is sometimes difficult. For systems where the stable manifold is not given in closed form, an iterative process is used to approximate the manifold. The approaches also involve designing several functions that require insight into the specific problem and system constraints.

2.2 Sequential Composition

The research presented in this paper uses the technique of *sequential composition*, which enables the construction of switched control policies with guaranteed behavior and provable convergence properties [11].

The original form of sequential composition uses state regulating feedback control policies, whose domains of attraction partition the state space into cells [11]. The control policy can be thought of as a funnel, as shown in Figure 2, where the vertical represents the value of a Lyapunov function, and the domain is the

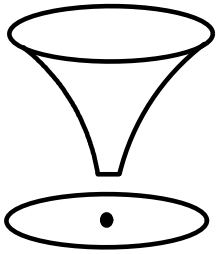


Figure 2: An idealized Lyapunov function and its associated domain of attraction represented by its shadow on the state space. The goal point is shown as a large dot within the domain.

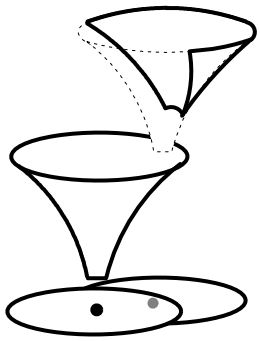


Figure 3: Composition of control policies leads to an enlarged domain of attraction. Note that the active policy switches to the highest priority policy as soon as the state enters its domain.

"shadow" cast by the funnel on the state space below [11]. The *domain* is the *safe domain of attraction*, which is the largest positive invariant region that does not intersect an obstacle

The basic idea behind sequential composition is to compose multiple control policies in a way that enlarges the overall domain of attraction, while preserving the underlying convergence guarantees. The approach specifies the goal of one policy to lie within the domain of another policy. By properly prioritizing the policies, and switching to a higher priority policy once the state enters the domain of the higher priority policy, it is possible to construct a switching control policy with a larger domain of attraction than any single policy [11]. The domain of attraction of the switched policy is equal to the union of the domains of the component policies, as shown in Figure 3. By composing policies with limited domains, constraints in the state space can be respected while building an almost globally convergent control policy. In this way, the free state space may be covered by incrementally covering an additional region of the free state space.

The policy composition is based on a formal notion of *prepares* [11]. For a given control policy, Φ_1, denote the domain of the policy $\mathscr{D}(\Phi_1)$ and the goal $\mathscr{G}(\Phi_1)$. Given two control policies, Φ_2 is said to *prepare* Φ_1, denoted $\Phi_2 \succeq \Phi_1$, if $\mathscr{G}(\Phi_2) \subset \mathscr{D}(\Phi_1)$. Let a finite collection of instantiated control policies, $\Lambda = \{\Phi_1, \ldots, \Phi_M\}$, defined over the free state space of a given system be given, and assume at least one policy stabilizes the overall goal. The prepares relationship between any two policies in the collection induces a directed graph, Γ_Λ, over the collection of instantiated control policies. In general the policy graph Γ_Λ is cyclic; however, an acyclic graph, Γ'_Λ, may be generated over the collection of policies by searching the original graph breadth first, beginning at the node corresponding to the policy that stabilizes the overall goal, and adding only those links and nodes that connect back to previously visited nodes. The acyclic graph Γ'_Λ is a partial order over the collection of control policies. By construction, the partial order graph is a connected graph containing a node corresponding to the policy that stabilizes the goal. Given the collection of policies Λ, the switching strategy defined by the associated partial order induces an overall switching control policy, denoted Φ. The union of the domains of the instantiated policies included in the partial order gives the domain of the overall policy; that is

$$\mathscr{D}(\Phi) = \bigcup_{\Phi_j \in \Gamma'_\Lambda} \mathscr{D}(\Phi_j) \ . \tag{1}$$

The overall control policy induced by sequential composition is fundamentally a hybrid control policy [6, 8, 24]. The composition of these local policies in a hybrid systems framework enables analysis on the discrete representation of the transitions between policy domains [8, 11]. We may analyze whether a goal node is reachable using the partial order graph Γ_Λ, which functions as a finite state automata [24]. Reachability of the goal state and decidability of the navigation problem for a particular collection of policies can be determined by the discrete transitions on Γ_Λ, without analyzing the underlying continuous system [24]. This analysis may be done prior to, or during, construction of the partial order.

The stability of the underlying control policies guarantees the stability of the overall switched policy because the partial order results in monotonic switching [11]. This obviates the need for complex hybrid stability analysis of the form given in [5, 7, 16, 38]. Disturbances are robustly handled provided their magnitude and rate of occurrence is small compared to the convergence of the individual policies [11].

The system in [11] uses multiple policies to demonstrate the inherent robustness of the technique. The deployment of the generic policies – including specification of the setpoints, control policy gains, and policy ordering – is accomplished manually. The deployment uses conservative approximations of the policy domains obtained through experimentation and simulation.

Sequential composition-like techniques have been applied to mobile robots. Many examples are for systems without nonholonomic constraints, where the policies are defined over simple cells – polytopes and balls – in configuration space [47, 48, 57]. Other approaches have used optimal control techniques on discrete state space representations to encode local policies [9, 22].

Sequential composition has also been used to control wheeled mobile robots using visual servoing [28, 44]. In these cases, the local control policies were designed based on careful analysis of the system, its constraints, and the problem at hand. These later approaches have the key feature that many of the individual policies are not designed to converge to a single point. In the later example, the "goal" of the highest priority policy is to drive through a doorway [44]. It is assumed that another control policy becomes active after the vehicle passes through the doorway. This idea of *flow-through* policies inspired work where generic policies are defined over polytopes that partition the environment into local cells [13]. This work was followed by similar approaches that used different methods of defining the control vector fields over the polytopes [4, 40].

These methods were originally developed for idealized holonomic systems, but may be applied to point nonholonomic systems using feedback linearization [4]. However, these approaches do not apply to systems with non-trivial body shapes, and cannot guarantee that the linearized system does not "cut a corner" between polytopes and collide with an obstacle.

A key feature of these approaches is that the policies are generic, and are therefore amenable to automated deployment [4, 13, 40]. These approaches also formed the base for later work in applying discrete planning techniques to specify logical constraints on the ordering of the local policies [19, 20]. Using the policy graph, Γ_Λ, a model checking system generates an open loop sequence of policies whose closed-loop behaviors satisfy the tasks specified in temporal logic.

In addition to model checking, there has been substantial work in the problem of planning and replanning on discrete graphs. The D* algorithm, originally developed for grid based path planning, facilitates fast replanning based on changes in the graph cost structure [53]. A similar, but algorithmically different version, called D*-lite has been applied to graph structures; including those, such as Markov Decision Processes (MDP), that support non-deterministic outcomes [39].

3 General Framework

This section presents the framework used to address the navigation problem described in this paper. First, a brief overview of the robot modeling framework is given. Next, the generic requirements for the the local policies are presented. This section concludes with an overview of planning based on the discrete transition graph induced by the policies.

3.1 System Models

The robot system consists of a single body that navigates through a planar environment that is cluttered with obstacles. The planar environment, or *workspace*, is a bounded subset $\mathcal{W} \subset \mathbb{R}^2$. Let $g = \{x, y, \theta\} \in \mathcal{G} \cong SE(2)$ denote the position and orientation, relative to the world frame, of a reference frame attached to the robot body. As the robot moves through its workspace, g evolves on the manifold $SE(2)$.

To address the navigation problem, the robot body must move along a path that reaches the overall goal, while avoiding obstacles along the way. Let $R(g) \subset \mathcal{W}$ denote the two-dimensional set of workspace points occupied by the body at position and orientation g. The obstacles in the workspace are represented as the union over a finite set of convex regions $\{O_k\} \subset \mathcal{W}$. Thus, for all body positions and orientations, g, along a collision free path,

$$R(g) \bigcap \bigcup_k O_k = \emptyset .$$

The robotic systems considered in this paper are driven by wheels in contact with the planar environment; the wheel-to-ground contact generates nonholonomic velocity constraints. The rotation of these wheels may be represented by internal *shape* variables [1]. Thus, the robot configuration is fully specified as $q = \{g, r\} \in \mathcal{Q} = \mathcal{G} \times \mathcal{R}$, where $r \in \mathcal{R}$ denotes the shape variables. The shape variable velocities are controlled via control inputs $u \in \mathcal{U} \subset \mathbb{R}^2$. For kinematic systems, $\dot{r} = u$; for second-order dynamical systems $\ddot{r} = u$. This paper is restricted to simple robot models where the nonholonomic constraints determine a linear mapping, $A(q)$, between velocities in the shape space and the velocity, $\dot{g} = A(q)\,\dot{r}$, of the body fixed frame.

The robot is induced to move along a path by application of the control inputs $u \in \mathcal{U}$; the closed-loop dynamics determine the velocity $\dot{g}$ via the mapping $A(q)$. The velocity $\dot{g}$ determines the robot path. The control problem is to specify control inputs $u \in \mathcal{U}$ such that the system moves along a collision free path and reaches the overall goal. To be a valid control, the inputs must be chosen from the bounded input space $\mathcal{U}$. As bounded inputs and the mapping $A(q)$ constrain $\dot{g}$, the navigation problem is tightly coupled to the control problem.

To address the navigation and control problem in a coupled manner, we define local control policies over local regions of $\mathcal{G}$ that are termed *cells*. Let Φ_i denote the i^{th} feedback control policy in a collection of policies and let $\Xi_i \subset \mathcal{G}$ denote the local cell. Define the cell Ξ_i in a subset of $\mathbb{R}^3$ that locally represents the $SE(2)$ configuration space; the cells are restricted to compact, full dimensional subsets of $\mathbb{R}^3$ without punctures or voids. It is assumed that the boundary of the cell, $\partial\Xi_i$, has a well defined unit normal, $n(g)$, that exists almost everywhere. The *policy goal set*, denoted $\mathcal{G}(\Xi_i)$, is a designated region in the local cell; that is $\mathcal{G}(\Xi_i) \subset \Xi_i$.

In this framework, the coupled navigation and control problem of moving the robot body through the space of free positions and orientations of the body fixed frame is converted to a problem of specifying safe feedback control policies over the local cells, and then composing these local policies to address the larger navigation problem. Over each local cell, define a feedback control policy $\Phi_i : \mathcal{TQ} \rightarrow \mathcal{U}$, where $\mathcal{TQ}$ denotes the tangent bundle over the robot's configuration space. In other words, the local policy maps the current robot state, whose configuration falls within the local cell, to an input in the bounded set of inputs $\mathcal{U}$ associated with the policy. The policy must be designed such the flow of $\dot{g}$ induced by $A(q) \circ \Phi_i$ enters

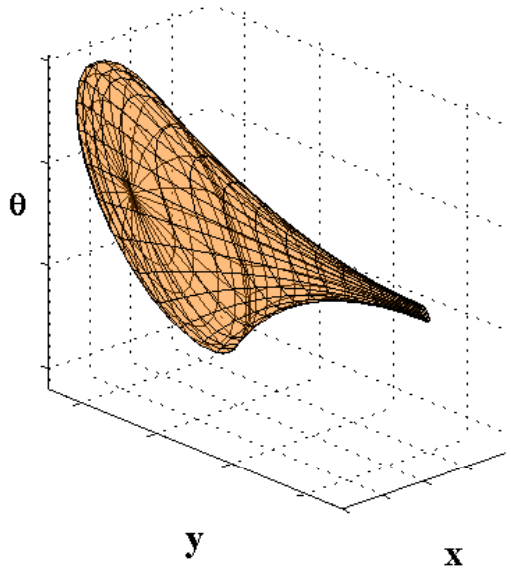


Figure 4: A cell defines a local region in the space of robot body positions and orientations. In this example, the cell is funnel shaped and the designated goal set is the small opening at the point of the funnel.

$\mathscr{G}(\Xi_i)$. Thus, instead of a specific path, the closed-loop dynamics of the local policy Φ_i induce a vector field flow over the cell Ξ_i that enters $\mathscr{G}(\Xi_i)$.

3.2 Generic Policy Requirements

For a local policy to be useful in the sequential composition framework, it must have several properties. First, to be practical, the local policies must have efficient tests to see if a specific local policy can be safely used; that is the system must be able to determine if the current robot state is within the domain of a given policy. Second, to induce safe motion, the cell must be contained in the space of free body positions and orientations so that collision with any obstacle is not possible while the body frame is within this cell. Third, under the influence of the policy, the induced trajectories must reach the policy goal set defined within the local cell without departing the associated cell from any initial position and orientation within the cell. Finally, the system must reach the designated goal set in finite time for any initial condition within the cell.

The requirements discussed below are generic with respect to the given robot model and any specific cell definition. The focus in the remainder of this section is on the generic policy requirements that enable the proposed hybrid control framework to work with a variety of system models.

3.2.1 Simple Inclusion Tests

A given policy is safe to activate if the robot state is within the policy domain, $\mathscr{D}(\Phi_i)$. The domain, which depends upon the policy definition, generally specifies both configurations and velocities. As the policies are defined over $\Xi_i \subset \mathcal{G}$, the first test of domain inclusion is that $g \in \Xi_i$. For kinematic systems $\mathscr{D}(\Phi_i) = \Xi_i$, because the state is simply the configuration. Therefore, only the cell inclusion test is needed to check to see if a policy is safe to activate for the simple models considered in this paper.

As the policies will need to be executed in real time, the cell inclusion test must be simple and relatively fast. There are several approaches to developing efficient inclusion tests for a given cell. Which approach is considered the best largely depends on the specific technique used to define the given cell. As such, discussion of specific tests will be deferred until the cells are defined. Section 4 will present a specific inclusion test for the cells developed therein.

3.2.2 Contained in Free Space

The second condition is that the cell must be contained in the space of free positions and orientations, $\mathcal{FS}_\mathcal{G}$, of the robot body; that is $\Xi_i \subset \mathcal{FS}_\mathcal{G} \subset \mathcal{G}$. In other words, any position and orientation within a valid cell is free of collision with an obstacle. Let $R_{\Xi_i} \subset \mathcal{W}$ be the composite set of points occupied by the robot over the entire cell; formally

$$R_{\Xi_i} = \bigcup_{g \in \Xi_i} R(g) \ .$$

Thus, R_{Ξ_i}, represents the swept volume of all the points occupied by the robot over all positions and orientations in the cell. The cell Ξ_i is contained in the free configuration space if

$$R_{\Xi_i} \bigcap \bigcup_k O_k = \emptyset \ .$$

Figure 5 shows the representation of R_{Ξ_i} for a given cell.

Testing that the cell is contained in $\mathcal{FS}_\mathcal{G}$ would seem to require constructing $\mathcal{FS}_\mathcal{G}$, and then performing tests on this complicated space. The conventional approach is to map the obstacles to configuration space and expand them based on the Minkowski difference between the obstacle boundaries and the robot body at various orientations [35]. Although there exist algorithms to construct these representations for obstacles and robots defined as semi-algebraic sets in $SE(2)$, the resulting representation of $\mathcal{FS}_\mathcal{G}$ is quite complex. This complexity can be avoided by testing the cells based on workspace measurements.

Appendix A presents a method for calculating R_{Ξ_i} given a representation of the body and the cell. By determining the boundary, ∂R_{Ξ_i}, the intersection of R_{Ξ_i} with any obstacle can be tested based on workspace

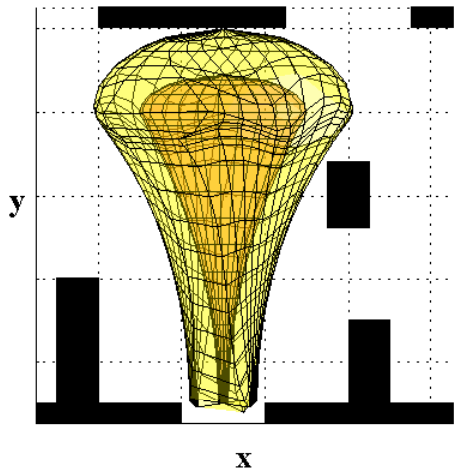


Figure 5: R_{Ξ_i} - Extent of robot body in workspace. The projection of the cell boundary is shown as the darker inner surface. If the minimum signed distance from the boundary ∂R_{Ξ_i} to the closest obstacle is positive, the cell is contained in the $\mathcal{FS}_\mathcal{G}$.

measurements. If no obstacle intersects the boundary of R_{Ξ_i} and no obstacle is completely contained in the interior of R_{Ξ_i}, then $\Xi_i \subset \mathcal{FS}_\mathcal{G}$. Given a representation of ∂R_{Ξ_i} and representations of the obstacles, Appendix A presents an automated procedure for verifying that the cell meets the necessary condition that it is contained in $\mathcal{FS}_\mathcal{G}$.

3.2.3 Conditionally Positive Invariant

The third necessary condition is a generalization of *positive invariance*, termed *conditional* positive invariance [29]. Earlier versions of sequential composition were defined for policies with positive invariant domains that converged to a single goal point. In contrast, we allow policies that cause the system to *flow-through* a designated goal *set* located on the cell boundary, and not come to rest within the goal set. For a conditionally positive invariant domain, the configuration remains in the domain of the policy until it enters the designated goal set. Coupled with the fact that the cell is contained in free configuration space, conditional positive invariance guarantees the policy is safe with respect to collision.

Formally, the domain $\mathscr{D}(\Phi_i)$ is *conditionally-positive invariant* under the influence of policy Φ_i with goal set $\mathscr{G}(\Phi_i)$, if $q(0) \in \mathscr{D}(\Phi_i)$, $q(T) \in \mathscr{G}(\Phi_i)$, and $q(t) \in \mathscr{D}(\Phi_i)$ for all $t \in [0, T]$; that is the configuration remains in the domain of the policy until it enters the designated goal set [29]. Thus, under the influence of policy Φ_i, the cell Ξ_i must be conditionally-positive invariant.

To maintain conditional positive invariance, the system must be able to generate a velocity that, while satisfying the system constraints, keeps the system configuration within the cell. Except, of course, for boundary points contained in $\mathscr{G}(\Xi_i)$; for flow-through policies the aim is to drive the system configuration through the goal set, $\mathscr{G}(\Xi_i)$. Thus, on the portion of the cell boundary that is not part of $\mathscr{G}(\Xi_i)$, the robot configuration velocity is restricted to the negative half-space defined by the outward pointing unit normal at the boundary point. Formally, for all $g \in \partial\Xi_i \backslash \mathscr{G}(\Xi_i)$, there must exist a $\dot{g}$ such that

$$n(g) \cdot \dot{g} < 0\,. \tag{2}$$

For flow-through policies the analogous necessary condition $n(q) \cdot \dot{g} > 0$ must hold for all points g in the interior of $\mathscr{G}(\Xi_i)$. In general, $n(q) \cdot \dot{g} > 0$ would only need to hold for an open set of points in the goal set interior; to simplify control policy design, we require it to hold over the entire interior of $\mathscr{G}(\Xi_i)$.

Using the general model, and substituting $\dot{g} = A(q)\,\dot{r}$, condition (2) can be rewritten in terms of shape velocities as $n(g) \cdot A(q)\,\dot{r} < 0$. Let $\omega(q) = n(g)^T A(q)$; $\omega(q)$ defines a half-space constraint on the shape velocities, $\dot{r}$.

Given a specific system model $A(q)$ and a bounded input set $\mathcal{U}$, these constraints limit the size and shape of the associated cell in the free configuration space. The constraint (2) specifies a necessary condition that must be satisfied on the boundary before a conditionally positive invariant policy can be defined; it does not guarantee the existence of such a policy.

3.2.4 Finite Time Convergence

The final necessary condition is that a valid control policy must bring any initial configuration within the cell to the specified goal set in finite time. This is implicit in the definition of conditional positive invariance if we require that for any initial condition $g(0) \in \Xi_i$, there exists some finite T such that $g(T) \in \mathscr{G}(\Xi_i)$. Addressing this requirement requires designing a provably correct feedback control policy over the domain.

Proving finite time convergence is necessary to guarantee that a sequence of policy transitions will eventually occur; that is, the system will not become stuck in one cell. If the system converges to the active policy goal set in finite time, and if that goal set is contained in a higher priority policy domain, then the higher priority policy is guaranteed to become enabled.

In general, proving finite time convergence for constrained systems can be difficult and must be considered for each policy design in conjunction with its cell and goal set specification. Section 4 addresses this issue for the specific policies used in this paper.

In summary, policies that respect the system constraints, have simple inclusion tests, are completely contained in the free configuration space, are conditionally positive invariant, and whose vector field flow converges to a well defined goal set in finite time may be deployed in the proposed hybrid control framework. Together these four requirements mean that the validity of a given policy can only be evaluated knowing the mapping $A(q)$, the associated cell Ξ_i, input space $\mathcal{U}_i$, and the designated goal set. In Section 4, we describe a family of policies that admit verification of these necessary conditions, and enable deployment of a collection of valid policies.

3.3 Discrete Planning in Space of Policies

Given finite time convergence and conditional positive invariance, the closed loop behavior of the control policies induces a discrete transition relation between the cell and its associated goal set. If the goal set of one policy is contained in the domain of another, the discrete transition relation is defined between policies. Recall from Section 2, that these discrete transitions between policies in a collection, Λ, may be represented by a directed, generally cyclic, graph structure, Γ_Λ. This allows the continuous dynamics to be represented by the more abstract discrete transitions on the graph. Using appropriate discrete planning tools, several high-level planning problems may be addressed using the underlying feedback control policies that induce the discrete transitions.

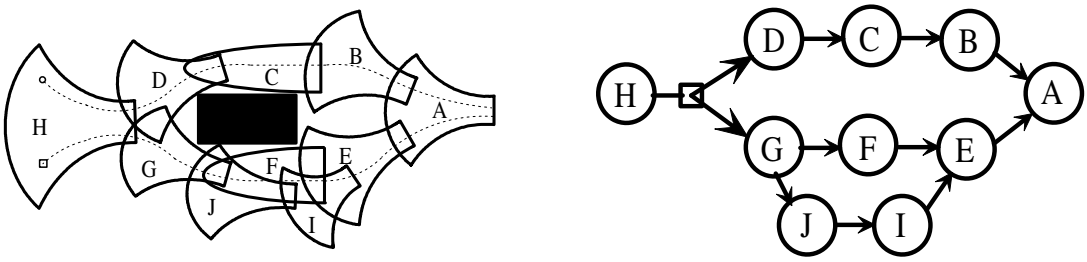
This subsection describes an extension to the conventional definition of *prepares*, and the impact on the resulting graph structure. The subsection concludes with a discussion of the pros and cons of some techniques for planning in the space of policies associated with the graph.

3.3.1 Extensions to Prepares Definition

Often the size of a valid cell that satisfies the policy requirements is tied to the size of the specified goal set. To enable larger cells, it is useful to consider policies where several policy domains cover the designated goal set, but no single policy domain covers the designated goal set. The conventional definition of *prepares*, $\Phi_j \succeq \Phi_i$ if $\mathscr{G}(\Phi_j) \subset \mathscr{D}(\Phi_i)$, describes a relation between two policies [11]. We extend this definition to a relation between a policy and a set of policies. A selected policy, Φ_i, *prepares* a set of policies if the goal set of the selected policy, $\mathscr{G}(\Phi_i)$, is contained in the union of the domains of the policies in the set, that is $\Phi_i \succeq \{\Phi_j\}$ if $\mathscr{G}(\Phi_i) \subset \bigcup_j \mathscr{D}(\Phi_j)$. Consider Figure 6-a, where $\Phi_H \succeq \{\Phi_D, \Phi_G\}$.

Under the influence of a given control policy, the flow along the induced vector field over a given cell is mathematically determinate; however, from the perspective of the discrete transition relation defined by the prepares relationship, the single policy could result in a transition to any of the policies in the union. In other words, while it is possible to determine the subset of the given control policy domain that enters a particular policy domain in the union, the resulting calculations are complicated. Restricting the domain representation to simple cells introduces indeterminacy into the graph structure Γ_Λ; this indeterminacy can be represented as an *action* with multiple *outcomes*, as shown in Figure 6-b. From the point-of-view of the discrete transitions encoded by Γ_Λ, this indeterminacy results in the outcome being imposed as an external choice that a discrete planner has no control over. Consider the example paths shown as dotted lines in Figure 6-a. The paths start from two different initial conditions, denoted $\circ$ and $\boxdot$, and exit the goal set of Φ_H in different locations, thereby entering different policy domains, Φ_D and Φ_G.

Using this extended definition of prepares allows flexibility in instantiating policies that might otherwise not be deployable using the conventional definition of prepares. This added flexibility comes at the cost of added complexity at the planning stage. If the extended definition of prepares is used to deploy policies,



a) Prepares relationship between a collection of policies.

b) Graph representation of the induced discrete abstraction.

Figure 6: This figure shows the relation between a set of policies and the discrete abstraction. The goal sets of the policies are the right edges of the bounded regions; either the small end of the funnel shapes or the minor axis of the half-ellipses. In this example, Φ_H prepares $\{\Phi_D, \Phi_G\}$. Note, that different initial conditions (denoted $\circ$ and $\boxdot$) in Φ_H lead to different paths through the policy graph.

then the discrete planner must take this indeterminacy into account. Therefore, the design choice to deploy policies using this extended definition of prepares must be evaluated given the particular planning problem at hand, the constraints in the specific robot model, the environmental complexity, and the available planning tools.

3.3.2 Planning in Policy Space

The discrete policy graph, Γ_Λ, represents the continuous dynamics of the problem as a finite set of transitions between nodes corresponding to policies. Thus, a fundamentally continuous navigation problem is reduced to a discrete graph search over Γ_Λ. If the current configuration is contained in the domain of a policy corresponding to a node in Γ_Λ, and a path through Γ_Λ to the overall goal node exists, then the given navigation problem is solvable with the collection of currently instantiated policies. Proving that such a graph traversal exists, and ordering the policies to generate a valid traversal, is the specialty of discrete planners.

To facilitate planning, each edge in Γ_Λ may be assigned a cost. Any non-negative cost is permitted, but the choice of cost impacts the resultant ordering, and hence the overall behavior of the system. In the experiments presented in this paper, a heuristic cost is assigned based on the relative complexity of each policy and a measure of distance between policy goal sets. By executing the feedback control policies according to an ordering determined by a discrete planner, the trajectories induced by the local policies solve the specified navigation problem. Given policies that meet the above requirements, the proposed technique is amenable to any number of discrete planning tools.

The most basic discrete planning approach is to order the policies in the graph according to the assigned costs [11]. First, a node in the graph is identified as the overall goal node; that is the node corresponding to the policy that flows to the overall goal set. From this goal node, basic graph search techniques may be used to order the graph and build the partial order Γ'_Λ; that is, convert the generally cyclic graph Γ_Λ into an acyclic tree Γ'_Λ. Dijkstra's Algorithm, A*, or D*, are natural candidates for this planning [12].

Given the ordered graph, Γ'_Λ, there are two basic approaches to executing the policies. The simplest is to convert Γ'_Λ into a total ordering of the policies [11]. Given the current state of the system, the hybrid control strategy searches for the highest priority policy whose domain contains the current state and executes that policy. This is simple to implement, and allows the hybrid control system to opportunistically switch to a higher priority policy. The drawback to this approach is that significant search time relative to real time operation may be required for large graphs; this approach makes having efficient inclusion tests imperative.

Another approach is to use use the local structure of the tree Γ'_Λ to guide the search for an active policy. This introduces an internal state where the system must keep track of the previously active policy, but allows the system to search a limited number of successors of the previously active policy that are expected to become active. If no successor is valid, the previously active policy is checked. If neither a policy corresponding to a successor in the tree nor the previously active policy is valid due to some system perturbation, then the hybrid control system can resort to searching the tree as a total order based on the node cost used during construction of the tree. This approach will typically require fewer policy domain inclusion tests, but limits the chance for opportunistic policy switching.

While the choice of switching strategy impacts the induced system trajectory, the convergence guarantees of the hybrid system are maintained so long as the graph ordering is observed. This inherent flexibility in the hybrid system approach may be exploited by a high-level planner by selecting a switching strategy for choosing between valid policies before the system execution, or possibly by using higher-level information to change the switching strategy on-the-fly. This later possibility is beyond the scope of this current work, but is allowed by the hybrid control framework.

Once the initial ordering is generated, the discrete representation allows for fast re-planning in the event of changing conditions. The D* algorithm and its variants are expressly designed to allow for existing plans to be modified if the cost structure is updated [39, 52]. For example, consider an ordering that uses a policy that navigates a particular doorway. If the doorway is unexpectedly closed, the robot may use this new information to update the graph cost structure – thereby invalidating the policy that passes through the doorway – and reorder the remaining policies. By operating on the discrete policy-based graph, which will generally have fewer nodes than a conventional grid-based path planning graph, the policy-based re-planning step is faster than conventional grid-based re-planning, while preserving the guarantees of feedback policies.

One of the key benefits to planning using the ordered graph Γ'_Λ is the inherent robustness. If an external disturbance invalidates a currently active policy, the system can quickly search the ordering for another valid policy. Consider the case shown in Figure 7, which is a close up of the layout in Figure 6. For the path shown, the system begins executing Φ_G then Φ_F; during execution of Φ_F an external disturbance moves the state outside $\mathscr{D}(\Phi_F)$. By searching the ordered graph, Φ_J may be activated without needing to stop and replan. This allows the system to plan once, and execute many times; in contrast to conventional approaches, that must completely re-plan the desired paths in response to significant perturbations.

The biggest drawback to the ordering approach is that only one navigation task at a time can be addressed. This works well for simple navigation tasks, but does not allow for higher level task specifications that require multiple goals. Recent work has addressed this issue by combining the policy graph Γ_Λ with temporal logic specifications [19, 20]. The approach uses model checking techniques to generate an open loop sequence of

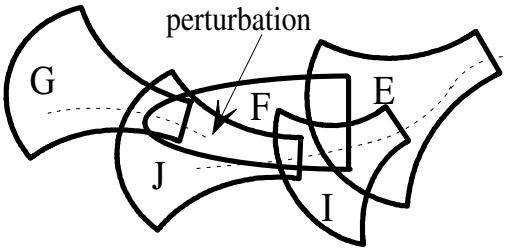


Figure 7: An ordering of policies provides robustness to external disturbances. When an external disturbance moves the state outside $\mathscr{D}(\Phi_F)$, Φ_J is activated without needing to stop and replan.

policies that satisfy the temporal logic specification. This approach works well for specifying multiple goals, but loses the robustness inherent in the partial order approach.

4 Local Policy Definitions

In defining the local policies, there are several competing goals. First, we desire policies with simple representations that are easy to parameterize. Additionally, we desire policies that have simple tests for inclusion so that the system can determine when a particular policy may be safely used. On the other hand, for a given policy goal set, we want the policy domain to capture as much of the free configuration space as possible given the system constraints. That is, we want the policies to be *expressive*. This paper considers how far a set of relatively simple policy parameterizations can be pushed.

To simplify the control policy design problem, the remainder of this paper is limited a particularly simple class of robots. First, the mapping $A(q)$ is only a function of the orientation of the robot body; that is $A(q) = A(g) = A(\theta)$. Second, the robot dynamics are kinematic; that is, the robot directly controls the shape variable velocities $\dot{r} = u$ for inputs $u \in \mathcal{U}$. These two restrictions allow the shape variables to be ignored, and thus $q = g$. Differential-drive robots, which are widely used, are in this class of robots provided their inputs are considered kinematic. This paper will focus on a specific robot model that meets these restrictions – the *kinematic unicycle* [12, 35]. The kinematic unicycle uses inputs that control of the forward speed, v, and rate of body rotation, ω. Given, $u = \begin{bmatrix} v & \omega \end{bmatrix}^T \in \mathcal{U} \subset \mathbb{R}^2$, the body position and orientation velocity is $\dot{g} = A(g)\, u$, where

$$A(g) = \begin{bmatrix} \cos\theta & 0 \\ \sin\theta & 0 \\ 0 & 1 \end{bmatrix}. \tag{3}$$

The mapping $A(g)$, which is defined by the control vector fields that annihilate the nonholonomic constraints, is fundamental to the policy design techniques developed in this section.

This section develops a class of generic policies based on a parameterized representation of the cell boundary. First, the basic cell definitions are presented. Given the cell definition, the four requirements from Section 3.2 are verified. As part of this verification, we use the mapping $A(g)$ to generate a vector field that satisfies the policy requirements and maps a configuration in the cell to a specific control input, u. The resulting closed-loop dynamics naturally satisfy the nonholonomic constraints inherent in $A(g)$.

The cells, Ξ_i, represent a set of configurations in the configuration space $\mathcal{G}$. The cells are defined in an $\mathbb{R}^3$ representation of $\mathcal{G}$ relative to a world coordinate frame with $\{x, y, \theta\}$-coordinate axes . We choose to define generic cells relative to a local coordinate frame attached to the center of the cell's designated goal set; let g_{goal} denote the goal set center in the world frame and $\{x', y', \theta'\}$ denote the coordinate axes of the local reference frame.

Given a generic cell specified in the local coordinate frame, the cell is instantiated in the configuration space by specifying the location of g_{goal} and the orientation of the local frame relative to the world frame. We restrict the policy goal set to lie within a plane orthogonal to the world frame x-y plane; thus the local θ'-axis is parallel to the world θ-axis. We restrict the positive x'-axis, which corresponds to the goal set normal, to be aligned with the robot direction of travel at the configuration corresponding to g_{goal}. Thus, the local x'-axis is normal to the cell goal set, and outward pointing with respect to the cell boundary. Let θ_{goal} specify the orientation of the x'-axis, and let $g_{\text{goal}} = \{x_{\text{goal}}, y_{\text{goal}}, \theta_{\text{goal}} + \Delta\theta\} \in \mathbb{R}^3$. For forward motion, the x'-axis is aligned with the robot heading, and $\Delta\theta = 0$. For reverse motion, the x'-axis is opposite the robot heading, therefore $\Delta\theta = \pm\pi$.

The remainder of this subsection describes the definition of the cells relative to this local cell frame. That is, a parameterization of $p = \{x', y', \theta'\}$ is given. Given the goal set center g_{goal}, any point $p \in \{x', y', \theta'\}$ represented in the local generic cell frame can be mapped to a point $g \in \mathcal{G}$ by

$$g(p) = \begin{bmatrix} \cos\theta_{\text{goal}} & -\sin\theta_{\text{goal}} & 0 \\ \sin\theta_{\text{goal}} & \cos\theta_{\text{goal}} & 0 \\ 0 & 0 & 1 \end{bmatrix} p + \begin{bmatrix} x_{\text{goal}} \\ y_{\text{goal}} \\ \theta_{\text{goal}} + \Delta\theta \end{bmatrix}. \tag{4}$$

Although similar to a homogeneous transform, the non-standard transformation (4) is required due to the placement of the cell within $\mathbb{R}^3$. The cell is both positioned and rotated by θ_{goal}; $\Delta\theta$ is used to position the cell along the θ-axis, but does not affect the orientation of the cell.

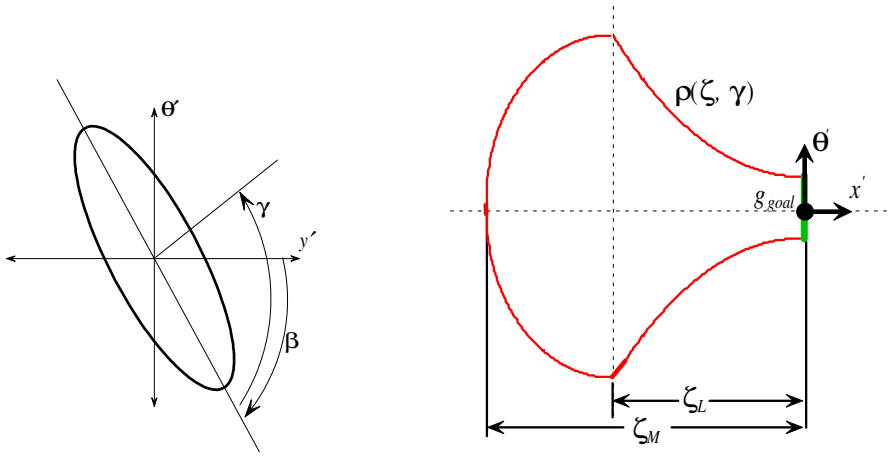
To define the generic cells, a good definition of the cell's designated goal set is needed, along with a specification of the cell boundary. In keeping with sequential composition, the general idea is to "funnel" relatively large regions of $\mathcal{G}$ into the cell's goal set. This is analogous to the fan-shaped regions of workspace defined by the optimal paths of car-like systems [56]. Therefore, we chose to define the cells such that they fan out from the goal set using a generalization of a superquadric surface [26].

We define the generic cell boundary in local coordinates with *two* smooth two-surfaces embedded in $\mathbb{R}^3$. The intersection of the cell boundary with a plane orthogonal to the central axis is a simple closed curve that may be parameterized by the orientation around the x'-axis. This suggests a cylindrical parameterization for the generic cell boundary. Let ζ be a scalar encoding the "depth" of the cell along the negative x'-axis, and let γ be a scalar encoding the angle about the local x'-axis. Define a generic cell boundary point, p, in these local coordinates as

$$p(\zeta,\gamma) = \begin{bmatrix} -\zeta \\ \rho(\zeta,\gamma)\cdot(\cos\beta\,\cos\gamma - c\,\sin\beta\,\sin\gamma) \\ \rho(\zeta,\gamma)\cdot(\sin\beta\,\cos\gamma + c\,\cos\beta\,\sin\gamma) \end{bmatrix} \in \mathbb{R}^3 . \tag{5}$$

The equations in parenthesis encode an ellipse with eccentricity $0 < c < 1$ rotated by β about the central (x') axis, as shown in Figure 8-a. With $0 < c < 1$, the velocity constraint (2) requires that $-\frac{\pi}{2} < \beta < 0$. The function $\rho(\zeta,\gamma)$ governs the radius in the local cylindrical coordinate system as the system moves by ζ along the negative x' axis, as shown in Figure 8-b. This representation is an extension to the standard superquadric representation because ρ may be a function of both ζ and γ [26]. The goal set is defined at $\zeta = 0$.

There is freedom in defining $\rho(\zeta,\gamma)$ provided the velocity constraint (2) from Section 3.2 can be satisfied for all points on the surface. We choose a ρ function that uses two continuous, piecewise-smooth segments that correspond to the two surface patches – a funnel and a cap – defining the cell boundary. The segments are shown in Figure 8-b. In the portion corresponding to the funnel, let $\rho_f(\zeta,\gamma)$ be a monotonically increasing function in ζ that governs cell growth as ζ increases away from the policy goal set; $\rho = \rho_f$ in this portion. The portion of $\rho(\zeta,\gamma)$ corresponding to the cap section monotonically decreases from the maximal value of



a) Elliptical Parameters b) ρ Parameters

Figure 8: Schematic representations of the generic cell. Some of the important parameters are labeled.

ρ_f to zero at the maximal extent of ζ. Formally, define the complete function as

$$\rho(\zeta,\gamma) = \begin{cases} \rho_f(\zeta,\gamma) & 0 < \zeta \leq \zeta_L \\ \rho_f(\zeta_L,\gamma)\ \frac{\sqrt{(\zeta_M-\zeta_L)^2-(\zeta-\zeta_L)^2}}{\zeta_M-\zeta_L} & \zeta_L < \zeta \leq \zeta_M \end{cases} . \tag{6}$$

The γ dependency in ρ_f allows radial asymmetry to be built into the cells, provided ρ_f is monotonic in ζ. This paper defines three basic cell shapes by defining three different ρ_f functions; typical cell shapes for the three functions are shown in Figure 9.

The first cell shape, as defined by $\rho_f = \rho_{f_1}$, results in the simple symmetric funnel shapes shown in Figures 4 and 9-a. Let

$$\rho_{f_1}(\zeta,\gamma) = R_o + R_e\left(\cosh\left(\frac{\zeta}{R_r}\right) - 1\right),$$

where R_o is one-half the length of the major axis of the goal set ellipse, and R_e and R_r govern the rate of expansion. The $\cosh$ function gives good results, but other monotonic functions could be used. For this choice, the cell size and shape is governed by the parameters of ρ_{f_1}, which are R_o, R_e, and R_r, along with the parameters of the ellipse c and β, the length of the cell ζ_L and ζ_M, and the location and orientation of the goal set given by g_{goal}. At $\zeta = 0$, the goal set is an ellipse like that shown in Figure 8-a. The restrictions from Section 3.2 will dictate how much the cell can grow, and its shape by limiting the parameter values. Choosing the parameter values during deployment will be discussed in Section 5.

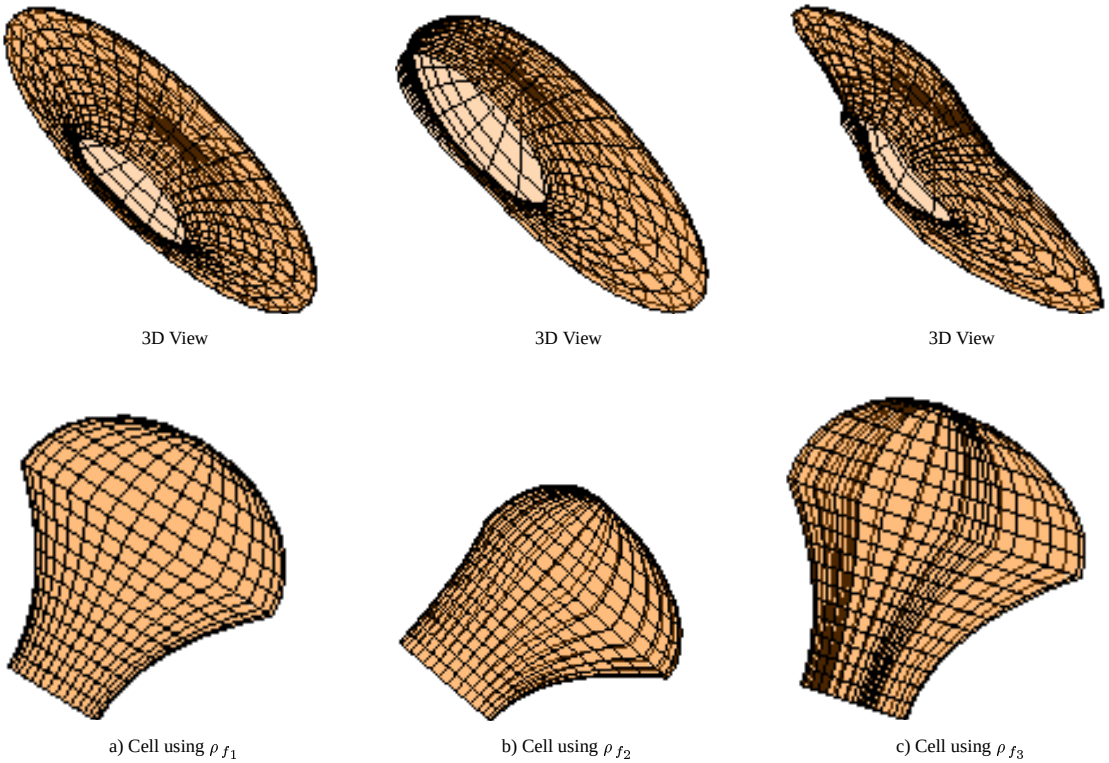


Figure 9: Example cells for each ρ_f showing 3D and x-y views.

The second cell shape, as defined by $\rho_f = \rho_{f_2}$, is used to generate a one-sided asymmetric cell, as shown in Figure 9-b. Let

$$\rho_{f_2}(\zeta,\gamma) = R_o + R_e\left(\cosh\left(\frac{\zeta}{R_r}\right) - 1\right) \cdot \frac{1+\cos(\gamma-\gamma_a)}{2},$$

where R_o, R_e, and R_r are as defined above, and γ_a is used to localize the asymmetry relative to the angle about the central axis. As with the first function, the goal set at $\zeta = 0$ is an ellipse.

The third cell shape, as defined by $\rho_f = \rho_{f_3}$, is used to generate a two-sided asymmetric cell, as shown in Figure 9-c. This cell is designed to generate a more aggressive turn than the first two cells allow under condition (2). Let

$$\begin{aligned}\rho_{f_3}(\zeta,\gamma) = &\left(R_o + R_e\left(\cosh\left(\frac{\zeta}{R_r}\right) - 1\right)\right) \cdot \\ &\left(1 + K_{g_1}\exp\left(-\frac{(\cos(\gamma-\gamma_{g_1})-1)^2}{\sigma_{g_1}}\right)\right. \\ &\left. + K_{g_2}\exp\left(-\frac{(\cos(\gamma-\gamma_{g_2})-1)^2}{\sigma_{g_2}}\right)\right),\end{aligned}$$

where R_o, R_e, and R_r are as defined above, K_{g_1} and K_{g_2} specify the relative size for the two wings of the cell, σ_{g_1} and σ_{g_2} specify the width of the wings, and γ_{g_1} and γ_{g_2} localize the wings relative to the angle about the central axis. Unlike the first two functions, the goal set of ρ_{f_3} is not an ellipse.

Given the classes of generic cells defined by these three functions, it must be shown that instantiations of the cells satisfy the necessary conditions given in Section 3.2.

4.1 Inclusion Test

Given a cell, the system must check if the current robot position and orientation $g \in \mathcal{G}$ is inside the cell. The inclusion test makes use of the cylindrical representation of the cell. Given $q = \{g, r\} \in \mathcal{Q}$ and a selected cell, represent g in the local cylindrical coordinate frame of the cell as $\{\zeta_q, \gamma_q, \rho_q\}$, where ζ_q is the distance from g_{goal} along the central axis, γ_q is the angle relative to the ellipse major axis, and ρ_q is the radial distance from the cell's central axis. Since the cell is defined in an $\mathbb{R}^3$ chart of $SE(2)$, we must also test for inclusion based on $g = \{x, y, \theta + 2n\pi\}$ where $n \in \{-1, 0, 1\}$.

The inclusion test uses two steps. The first step tests that $0 < \zeta_q < \zeta_M$; if $\zeta_q \leq 0$ or $\zeta_q \geq \zeta_M$ then the point is outside the cell. If the point passes the first test, then find the cell boundary point, $p_b = \{\zeta_q, \gamma_q, \rho_b\}$ that corresponds to $g = \{\zeta_q, \gamma_q, \rho_q\}$, where ζ_q and γ_q are the same and ρ_b is the radius of the boundary point along the vector defined by ζ_q and γ_q. These points are shown in Figure 10. From (5), the value of ρ_b is given as

$$\begin{aligned}\rho_b(\zeta_q,\gamma_q) &= \left\| \begin{bmatrix} \rho(\zeta_q,\gamma_q)\cdot(\cos\beta\,\cos\gamma_q - c\,\sin\beta\,\sin\gamma_q) \\ \rho(\zeta_q,\gamma_q)\cdot(\sin\beta\,\cos\gamma_q + c\,\cos\beta\,\sin\gamma_q) \end{bmatrix} \right\| \\ &= \rho(\zeta_q,\gamma_q)\,\frac{\sqrt{1+c^2-(c^2-1)\cos(2\gamma_q)}}{\sqrt{2}}. \qquad (7)\end{aligned}$$

If $\rho_q < \rho_b$ and $0 < \zeta_q < \zeta_M$, the configuration is within the cell.

4.2 Free Configuration Space Test

We verify that the cell is contained in the free space of positions and orientations, that is $\Xi_i \subset \mathcal{FS}_{\mathcal{G}}$, using the approach outlined in Appendix A. The test is for a specific cell using a particular choice of cell parameter

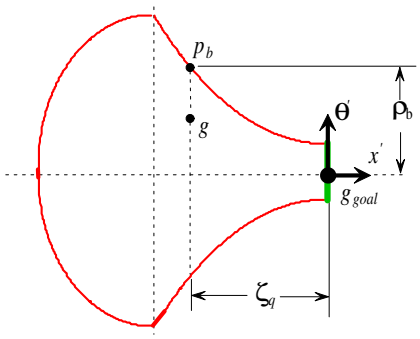


Figure 10: Corresponding boundary points. Given a point g, determine its local cylindrical coordinates, $\{\zeta_q, \gamma_q, \rho_q\}$, and find the corresponding point $p_b = \{\zeta_q, \gamma_q, \rho_b\}$ on the cell boundary.

values, $\{R_o, R_e, R_r, c, \beta, \zeta_L, \zeta_M\}$ and optionally $\{\gamma_a\}$ or $\{K_{g_1}, \gamma_{g_1}, \sigma_{g_1}, K_{g_2}, \gamma_{g_2}, \sigma_{g_2}\}$. Using the parameterized representation of the cell boundary given in (5), the approach described in Appendix A generates a representation of R_{Ξ_i}. The set R_{Ξ_i} is tested for intersection with any obstacle. If intersection occurs, the cell parameter values must be modified.

From (5), a point on the cell boundary, $g(\zeta, \gamma) = g(p(\zeta, \gamma))$, and surface normal, $n(\zeta, \gamma) = \frac{D_\zeta g \times D_\gamma g}{\|D_\zeta g \times D_\gamma g\|}$, can be calculated for the given set cell parameter values. Both $g(\zeta, \gamma)$ and $n(\zeta, \gamma)$ are piecewise smooth functions. Given an implicit representation of the robot body boundary, the cell boundary point is analytically mapped to a point in R_{Ξ_i}. By determining points along the boundary of R_{Ξ_i}, intersection with obstacles can be tested.

Appendix A describes an approach based on a triangulation of the$\{\zeta, \gamma\}$ parameter space that leads to a projection of the triangulated surface into the workspace. For the deployment presented in this paper, a surface visualization tool was used to visually inspect the projection for intersection with an obstacle; this is shown in Figure 5. Future work will automate this test by using the brute force method described in Appendix A.

4.3 Conditional Invariance Test

During instantiation, the cell parameter values must be selected so that the control system can generate velocities that enforce conditional positive invariance as described in Section 3.2.3. For the models used in this paper, where $\dot{g} = A(g)\, u$, condition (2) can be rewritten as

$$n(g(\zeta, \gamma))^T A(g(\zeta, \gamma))\, u < 0\,.$$

This condition must hold over the entire cell boundary, $\partial\Xi_i$. Let $\mathcal{U}$ be the bounded input set associated with this policy, for each point $g(\zeta, \gamma) \in \partial\Xi_i$ define

$$L(\zeta, \gamma) = \min_{u \in \mathcal{U}} n(\zeta, \gamma)^T A(g(\zeta, \gamma))\, u\,. \tag{8}$$

For a valid cell, condition (2) must hold, such that $L(\zeta, \gamma) < 0$ over the entire cell boundary. In other words, at each point on the cell boundary, the system must be able to generate a velocity that is inward pointing with respect to the cell boundary. Thus, in the worst case over the boundary, a valid cell satisfies the constraint

$$\max_{\zeta, \gamma} L(\zeta, \gamma) < 0\,. \tag{9}$$

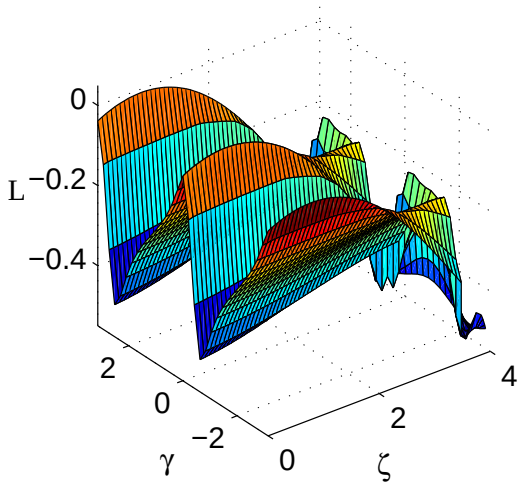


Figure 11: Constraint surface for $L(\zeta, \gamma)$ from (8).

Although non-linear, the function $L(\zeta, \gamma)$ is piecewise smooth and generally "well-behaved" for the mapping $A(g)$; therefore, it is feasible to verify that (9) is satisfied for a given set of cell parameter values. Figure 11 shows a typical constraint surface for the cell shown in Figure 4. The ridges shown in the figure are due to switching behavior in the minimization that occurs when the cell boundary normal is parallel to the x-y plane; that is the component in the θ direction is zero.

If condition (9) is satisfied for a given cell, then it is possible to define a control law over that cell that enforces conditional positive invariance. Designing such a control law, and proving that it provides finite time convergence to the cell goal set, is the next topic.

4.4 Vector Field Definition and Convergence Test

To define the control law, we use a family of level sets based on the cell boundary parameterization given in (5). This family of level sets is used to define a control vector field that flows to the policy goal set. For $g \in \Xi_i$, the corresponding level set that passes through g must be determined; represent g in the local cylindrical representation of Ξ_i by $\{\zeta_q, \gamma_q, \rho_q\}$.

Recast the cell definition equations given in (5) and (6) in terms of ζ'_M and ζ'_L to differentiate the control level sets from the cell boundary. These parameters control the relative length and size of the internal level set. As $g \in \Xi_i$, the values will have the following relationship $0 \leq \zeta'_L \leq \zeta_q \leq \zeta'_M \leq \zeta_M$. The family of level sets used for control are defined by (5) with $\rho(\zeta_q, \gamma_q) = \rho_L(\zeta_q, \gamma_q)$ where

$$\rho_L(\zeta_q, \gamma_q) = \rho_f(\zeta'_L, \gamma_q) \frac{\sqrt{(\zeta'_M, -\zeta'_L,)^2 - (\zeta_q - \zeta'_L,)^2}}{\zeta'_M, -\zeta'_L,}. \tag{10}$$

That is, the control level set is governed the by (6), with $\rho_f(\zeta'_L, \gamma_q)$ using the same parameter values as those defining the cell boundary.

First consider the case $\zeta'_M, = 0$ and $\zeta'_L, = 0$, the level set defined by $\zeta_q = 0$ and $\gamma_q \in (-\pi, \pi]$ corresponds to the policy goal set. Increasing ζ'_M, while fixing $\zeta'_L = 0$, generates a family of level sets for $0 \leq \zeta_q \leq \zeta'_M$ that grow out from the goal; these are termed the *inner* level sets, as shown in Figure 12. By fixing ζ'_M at its maximum value, $\zeta'_M = \zeta_M$, and increasing ζ'_L, the *outer* family of level sets grows; these are also shown in

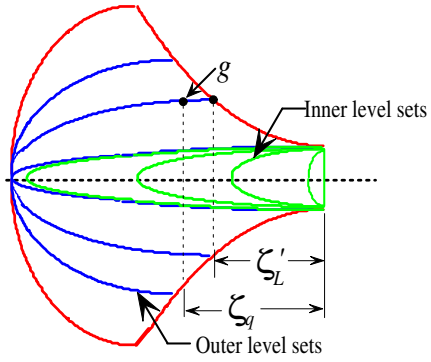


Figure 12: Level Set Definition for control

Figure 12. Thus, given $g = \{\zeta_q, \gamma_q, \rho_q\}$, the values for ζ_M' and ζ_L' must be determined such that the level set passes through g. For this to be the case, $\rho_L(\zeta_q, \gamma_q) = \rho_q$; thus values for ζ_M' and ζ_L' that satisfy

$$\rho_f(\zeta_L', \gamma_q) \frac{\sqrt{(\zeta_M' - \zeta_L')^2 - (\zeta_q - \zeta_L')^2}}{\zeta_M' - \zeta_L'} - \rho_q = 0 \tag{11}$$

must be found.

For configurations within the inner family of level sets, $\zeta_L' = 0$ and ζ_M' can be determined in closed-form from (11). If the configuration is within the outer family of level sets, as shown in Figure 12, then $\zeta_M' = \zeta_M$ and we must determine the value of ζ_L' that satisfies (11). Unfortunately, ζ_L' cannot be found in closed form. Fortunately, (11) is a monotonic function of ζ_L', which admits a simple numeric root finding procedure.

Given the values of ζ_M' and ζ_L', the level set normal is used to define a constrained optimization over the input space. The level set normal $n(\zeta_q, \gamma_q)$ is defined as in Section 4.2 using (5) and (10). The simplest constrained optimization is

$$u^* = \arg\min_{u \in \mathcal{U}} \left[n(\zeta_q, \gamma_q) \cdot A(g(\zeta_q, \gamma_q)) u \right] \quad \text{s.t. } n(\zeta_q, \gamma_q) \cdot A(g(\zeta_q, \gamma_q)) u < 0. \tag{12}$$

For a convex polygonal input set $\mathcal{U}$, the solution to this optimization will lie at the vertices of $\mathcal{U}$. To smooth u^*, the cost function, $[n(\zeta_q, \gamma_q) \cdot A(g(\zeta_q, \gamma_q)) u]$, can be augmented by a simple quadratic term.

Using u^* as the control input drives the system from the outer level sets to the inner level sets, and then continuously on to the goal. We force all the inner and outer level sets to satisfy (9) at every point in the cell, which guarantees that a solution to (12) exists. The body velocity $\dot{g} = A(g) u^*$ will bring the system configuration to a "more inward" level set; thus, the system moves a finite distance closer to the goal. Although a formal analytic proof is lacking, experience shows that if the outermost level set corresponding to the cell boundary satisfies (9), all interior level sets will also satisfy (9). This can be checked for various values of ζ_M and ζ_L during deployment.

Given the input u^*, the reference vector field over the cell Ξ_i is simply $A(g) u^*$. In reality, only u^* is important; the vector field is induced by the flow of the system given the control inputs, and is never explicitly calculated.

5 Policy Deployment Technique

To deploy a policy in this sequential composition framework, we must specify the parameters that govern the location, size, and shape of the cell. Given a specification of these parameters, the requirements from Section 3.2 must be verified. Currently, we lack an automated process for deploying the policies; therefore, we use a manual, trial and error, process to specify the policy parameters. Future work will develop automated deployment methods. While there are many possible deployment approaches, this section focuses on the approach used in the experiments described in Section 6.

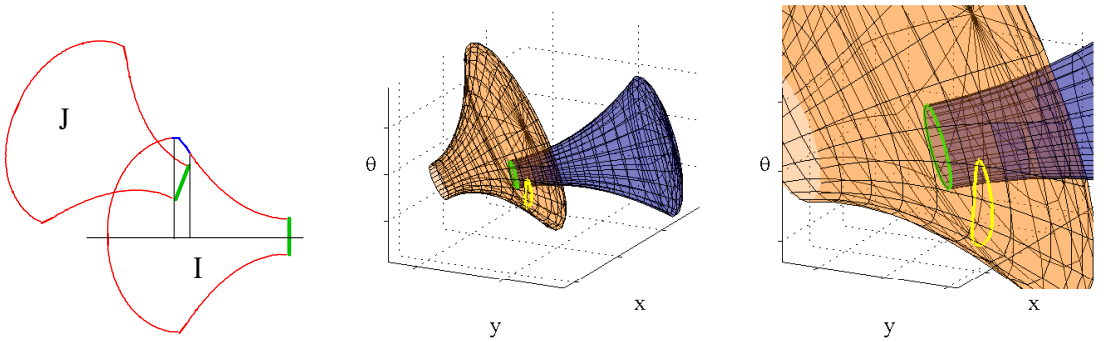
The policies are generally deployed sequentially using a backchaining procedure; that is, we test the prepares relationship to ensure the policies prepare at least one policy. We begin instantiating a policy by specifying that the policy goal set is completely contained within the domain of a set of existing policies. Assume at least one safe policy is instantiated; that is, initially $\Lambda = \{\Phi_0\}$ where Φ_0 stops the system in a safe configuration.

For the policies presented here, the prepares test can be done by verifying that all points on the policy goal set boundary are contained in the domains of neighboring cells. First, we specify g_{goal} to lie within the domain of an existing cell, or set of cells; this is verified automatically using (7). Next, we initialize the policy goal set by specifying the elliptical parameters β and c from (5) and the function $\rho(0, \gamma)$ by specifying the parameters in ρ_f. We test that the policy goal set boundary does not intersect the boundary of the neighboring cell, or the boundary of the union of neighboring cells. For this test we use a automated test based on (7) for a dense sampling of points around the goal set boundary. The check can also be done visually using Figure 13. We iteratively adjust the parameters, including the policy goal set center g_{goal}, as needed until the policy goal set is fully contained in the neighboring cell or set of cells. During this iterative process, we also test that the policy goal set is valid with respect to the mapping $A(g)$ and chosen input set; that is

$$\min_{g \in \mathscr{G}(\Xi_i)} \max_{u \in \mathcal{U}} n(g)^T A(g) u > 0 .$$

For $g \in \mathscr{G}(\Xi_i)$, $n(g)$ is aligned with the local x'-axis. Assuming that the previously deployed policies are contained in free configuration space, a valid goal set will lie in $\mathcal{FS}_G$.

If the goal set is not contained in a single policy, a simplified version of the extended prepares test ($\Phi_j \succeq \{\Phi_i\}$) is used to test for inclusion in the union of neighboring policy domains. Here, we require that each neighboring policy in the union contain the goal center of the policy being tested. This is shown in Figure 14, where the goal center for Φ_D is contained in the cells labeled A, B, and C. Then, for each



a) Schematic view of prepares test

b) Prepares test in 3D

c) Prepares test in 3D - close up

Figure 13: Prepares test for policy deployment. In the 3D figures, the goal set of Φ_J and the corresponding points on the surface of Φ_i are traced in lighter colors.

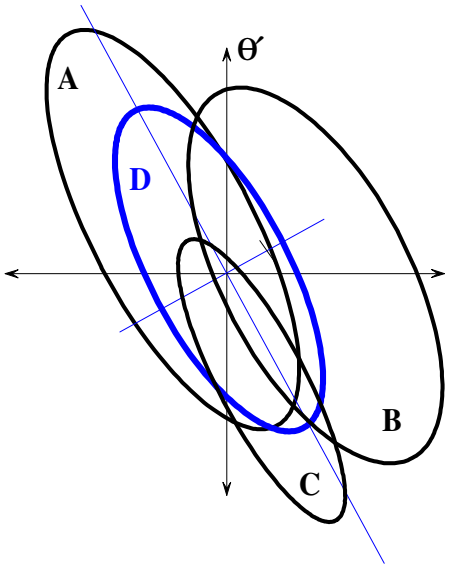


Figure 14: Prepares test for union of policy domains. Here $\Phi_D \succeq \{\Phi_A, \Phi_B, \Phi_C\}$.

point on the goal boundary of Φ_j, the maximum of the minimum distance to the cell boundaries is found. For all $\gamma \in [-\pi, \pi]$ find $g_j(0,\gamma)$, a point on the goal set boundary of Φ_j in the world frame. For each Φ_i find $\zeta_{q_i}(g_j(0,\gamma))$, $\gamma_{q_i}(g_j(0,\gamma))$, and $\rho_{q_i}(g_j(0,\gamma))$; then, using (7) find $\rho_{b_i}(\zeta_{q_i}(g_j(0,\gamma)), \gamma_{q_i}(g_j(0,\gamma)))$. The goal set prepares the union of neighboring cells if

$$\min_{\gamma} \max_{\{\Phi_i\}} \left(\rho_{b_i}\left(\zeta_{q_i}\left(g_j(0,\gamma)\right), \gamma_{q_i}\left(g_j(0,\gamma)\right) - \rho_{q_j}\left(\zeta_{q_i}\left(g_j(0,\gamma)\right), \gamma_{q_i}\left(g_j(0,\gamma)\right)\right)\right)\right) > 0\,.$$

This test is performed automatically based on a dense sampling of points around the the goal set boundary; the result can also be checked visually as in Figure 13.

Given a valid policy goal set, we grow the cell by specifying the parameters that determine the rate of cell expansion and cell shape, including R_e, R_r, ζ_L and ζ_M. During this process, we must confirm that (9) is satisfied and that the cell remains in the free configuration space. Given these conditions, the requirements that we can reach the goal in finite time and have a test for cell inclusion are automatically satisfied given the methods for defining the cell and vector field.

The test that (9) is satisfied is automated by using numerical optimization to find the maximum value of (8) over the $\{\zeta, \gamma\}$ parameter space. Given a violation of (9), we iteratively adjust the parameters as needed to ensure the requirements are met. Figure 11 can be used to guide the process of modifying parameters by showing where the violation occurs on the cell boundary.

To verify that a policy is safe with respect to obstacles, we use the expanded cell described in Section 3.2. In these experiments, we graphically verify that the projection of the expanded cell is free of intersection. We sample the $\{\zeta, \gamma\}$ parameter space with a fine resolution mesh, and calculate the corresponding points on the expanded cell surface. We project the resulting expanded cell surface mesh into the workspace and visually check for obstacle intersection, as shown in Figure 22-b. If collision occurs, the parameters governing cell size and shape are adjusted as needed, and both the free space and (8) tests are repeated.

Once an instantiated policy has been validated, it may be added to the collection of instantiated policies, Λ.

Given a collection of instantiated policies, the final step is to specify the prepares graph, Γ_Λ; this step is completely automated. The automated graph builder checks the prepares relationship for each policy against all other policies in the deployment; that is we test for policy goal set inclusion within other cells or sets of cells as described above. For the deployment used in this paper, collections of up to three cells were tested for the extended prepares. For policies that have a prepares relationship, we assign an edge cost heuristically based on the path length between the goal center and the prepared cells associated goal center, and a factor for the policies relative complexity. By assigning a relatively high transition cost between forward and reverse policies, changes in direction are reduced.

Currently, the trial and error process of policy instantiation takes between two and 30 minutes. The time depends mainly on the tightness of the corridors and sharpness of the turns being navigated. The final graph generation step is fully automated; it took several hours to run for the collection of 271 policies Section 6. The up-front cost of generating the deployment is mitigated by the ability to reuse the policies by planning multiple scenarios on the discrete graph.

6 Current Results

The control techniques and cells described in this paper have been validated experimentally on a real mobile robot. This section describes the robot system used to validate the work, and describes the deployment of policies used in the experiments. Several navigation experiments on the physical robot are discussed. These experiments demonstrate planning and re-planning in the space of deployed control policies. The section concludes with simulation results of planning using temporal logic specifications and model checking to address multiple task scenarios.

6.1 Robot System and Implementation

The techniques described in Sections 3 and 4 have been validated on our laboratory robot. This standard differential-drive robot, shown in Figure 15, has a convex, roughly elliptical body shape. To simplify calculations of R_{Ξ_i}, the composite set of points occupied by the robot body over all positions and orientations in the cell, the robot body and wheels were tightly approximated by an analytic ellipse centered in the body coordinate frame. This is shown in Figure 16. The length of the major and minor axes of the bounding ellipse are 1.12 and 0.68 meters respectively.

The control policies use the kinematic unicycle model, with the standard form

$$A(q) = \begin{bmatrix} \cos\theta & 0 \\ \sin\theta & 0 \\ 0 & 1 \end{bmatrix} .$$

The inputs $u = \begin{bmatrix} v & \omega \end{bmatrix}^T$ are forward velocity, v, in meters per second, and turning rate, ω, in radians per second.

The policies in the deployment are instantiated by choosing among a collection of four different input sets to account for local conditions. The system changes direction by switching between "Forward" and "Reverse" input sets; each having an "Aggressive" and "Cautious" set of values. The numerical values, shown in Figure 17, are based on the velocity limits of the motors, and scaled back for cautious modes and the reverse input sets. Although the robot is capable of zero-radius turns, the input bounds are chosen to model a conventional car-like system with bounded steering. This choice limits the rate of expansion of the cells based on the conditional invariance condition given in (9); the cell shapes would only be limited by the obstacles if zero-radius turns were allowed. This paper uses convex polygonal bounds on each input set; this

Figure 15: Laboratory robot

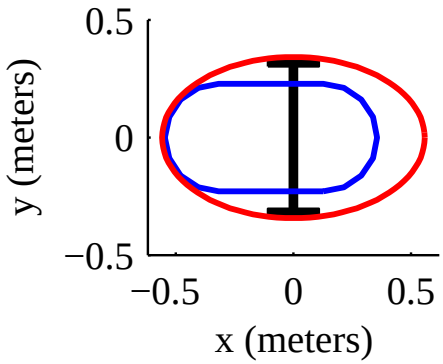


Figure 16: Bounding ellipse

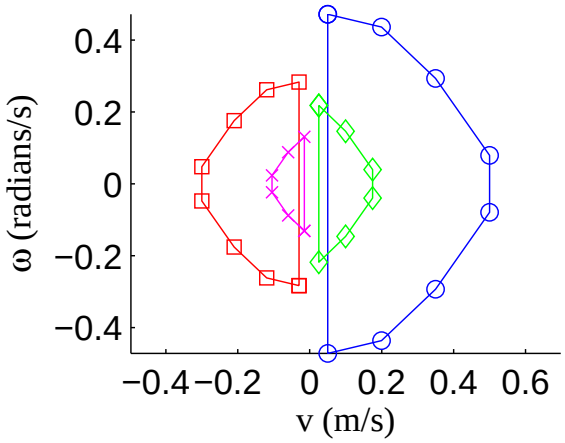
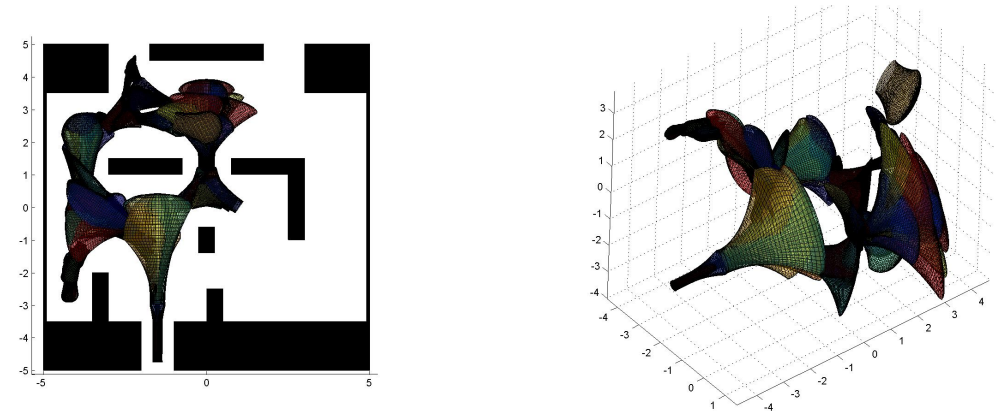


Figure 17: Four sets of bounded steering inputs. Forward Aggressive - ◯, Forward Cautious - ♢, Reverse Aggressive - □, Reverse Cautious - ×

restriction is for computational convenience, and is not fundamental. Each cell is evaluated with respect to a specific choice of input set.

For these experiments, a set of polygonal obstacles defines the 10 meter ×10 meter world shown in Figure 18-a. The world includes several narrow corridors and openings. The width of the narrow corridors is approximately one meter. This provides a clearance of approximately 16 centimeters on either side, but prevents the robot from turning around within the corridor.

For these experiments, a total of 271 basic policies of the type described in Section 4 are deployed using the techniques described in Section 5. A partial deployment is shown in Figure 18. Each policy is associated with a specific input set from Figure 17. From the collection of instantiated policies, Λ, the discrete graph representation, Γ_Λ, is determined. The deployment uses the extended definition of prepares described in Section 3.3.1, so that 15 separate policies prepare 31 different unions of policy domains.



a) Projection of cells into workspace with obstacles

b) Representation of cells in 3D C-space

Figure 18: Partial deployment of cells in environment.

6.2 Navigation Experiments

Given an implementation of the policy deployment and graph structure on the robot, the navigation task is specified as bringing the robot to a designated goal set located in the middle of the lower corridor. The designated goal set is associated with a specific policy, and therefore, a specific node in the policy graph. This node is specified as the overall goal of these experiments. An ordering of the policy graph from this overall goal node is generated using a Mini-max version of D* Lite [39].

D* Lite [39] resolves the indeterminacy described in Section 3.3.1, and provides an efficient way to re-plan in the case where a policy status is changed to invalid after the initial planning stage. Initially, the planner generates a partial order, Γ'_Λ, of the policy graph Γ_Λ by assigning a cumulative node cost based on the edge costs determined during graph generation. If a valid path through the ordered graph from a given policy node to the designated goal node does not exist, then the node (and corresponding policy) is flagged as invalid. As the planner executes, the effects of invalid nodes propagate through the graph; as a result, a path to the goal from a given node is guaranteed to exist if the node remains valid.

During execution, the hybrid policy tests the current estimate of body position and orientation for inclusion in the cells associated with successors of the currently active policy. If the current configuration is included in one of the successor domains, the associated policy is activated. If not, the currently active policy is checked for validity, and executed if it is still valid. As a final check if this fails, the robot is temporarily stopped and Γ'_Λ is searched for a valid policy. If this final search fails to find a valid policy, the robot terminates execution and reports failure.

The policy switching is completely automatic once the initial start command is given. The system stops when it reaches the policy designated as the goal or, if the configuration is not contained within a valid policy domain. If a valid policy is found, the control is calculated as described in Section 4.4. On our robot, the forward velocity and turning rate calculated by the control policy is converted to wheel speeds used by the low-level PID motor control loop.

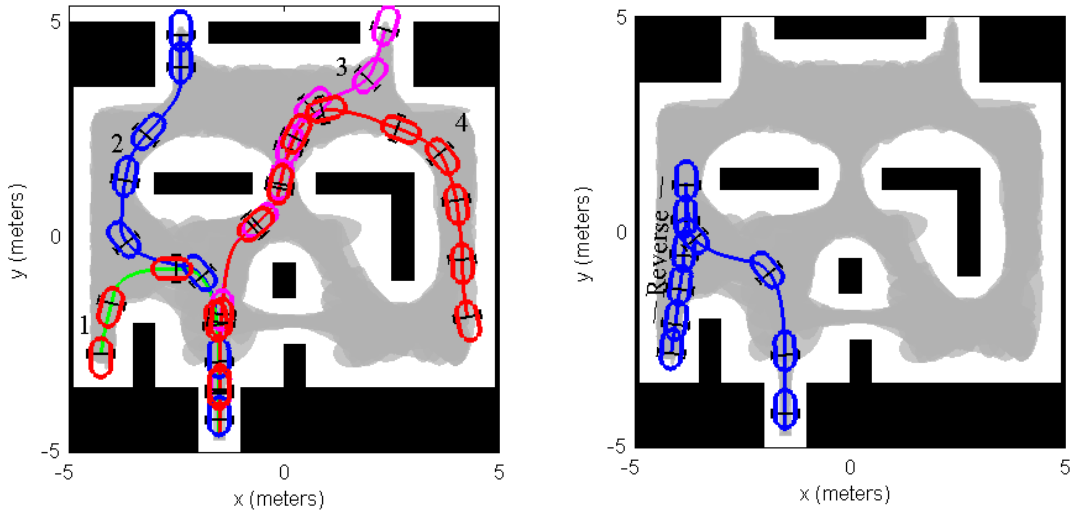
Seven representative tests on the robot are presented; a total of 19 tests were conducted, all in the same environment. The actual data was logged every 0.1 seconds during the robot experiment, and is plotted with the obstacles shown in the figures. On the plots, the robot symbols are shown every five seconds of travel time. Although the policies were initially validated in simulation, the results presented here are for actual robot runs.

For the experiments described in this paper, the robot lacks an external localization system, therefore the configuration estimates are based on pure dead-reckoning. As a result of dead-reckoning error, inherent in all wheeled-mobile robots, the robot would have crashed into some obstacles had they been physically present. Therefore, we run the robot in an open space, but use the policies that consider the obstacles shown in Figure 18-a.

Figure 19-a shows the results of four of the robot runs from four different initial conditions and the same goal node. To emphasize, the paths shown are plotted from dead-reckoned estimates of body position and orientation from an actual robot run. The induced paths are the result of policy ordering that is based on assigned edge costs in the discrete graph; an explicit desired path is never calculated. The paths are labeled (#1 - 4) clockwise from the lower left.

The path labeled #5, shown in Figure 19-b, begins near the same position as for path #1 (shown in Figure 19-a); however, the orientation is approximately 180 degrees different. As the deployment also includes policies that allow the robot to go in reverse, the robot backs up, stops, and then moves forward to the goal. To avoid damaging the real robot, the system was required to come to a stop before switching between a forward and reverse policy. The policy switching is automatic, with no operator intervention.

Paths #6 and #7, shown in Figure 20, have two features that demonstrate the flexibility of the approach. First, like path #5, they demonstrate automatic forward/reverse switching. In this case, the deployment does not include policies that are expressive enough to turn the lower right corner in one continuous motion. The robot initiates a K-turn, again with the policy switching occurring automatically as dictated by the policy



a) Experimental Runs #1-4

b) Experimental Run #5

Figure 19: Experimental results of five robot runs using the proposed hybrid control framework. The experimental runs use the same ordering determined by a discrete planner, only the initial conditions vary. The projection of the individual policy domains are shown in light gray. Runs #1 and #5 start at the same position, but 180 degrees apart in orientation. Run #5 automatically backs out of the narrow corridor, and then switches to forward motion.

ordering. While this points to the need for future research in designing more expressive policies, it also validates the basic approach as the combination of simple policies with discrete planning is still capable of generating expressive motions.

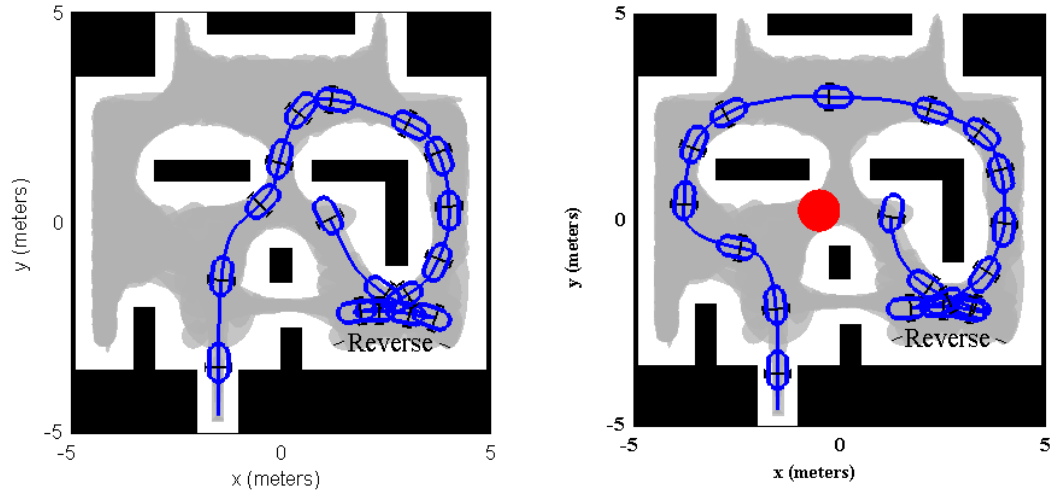
The second feature demonstrated by paths #6 and #7 is that of re-planning in the discrete space of available policies. The robot starts from slightly different initial conditions, but begins to converge to the same vector field flow near the K-turn. Just after the K-turn during run #7, the two policies crossing the circular obstacle shown in Figure 20-b are flagged as invalid. This triggers a re-planning step using D* Lite that reorders the policies, thereby inducing the robot to take "the long way around" via path #7. This re-planning occurred in real time, while the hybrid control policy was executing on the robot.

6.3 Task Level Planning Simulations

To demonstrate the utility of planning in the space of control policies, a multiple task scenario is simulated using the same robot model, obstacle environment, and policy deployment described above. The scenario is modeled on a mail delivery robot operating in an office environment. The robot, which begins in a given region, is tasked with picking up a package at a designated region, and dropping the package off at the mail-room. The robot is also tasked with picking up two packages at the mail-room, and delivering them to two separate locations. The robot is to finish at a designated region.

Each pickup or delivery point is associated with the goal set of a specific policy. Using the labeling of policies in the deployment, the specification may be given as

- Start in location BT
- Pickup (visit) at either DE or FO, then deliver to GO (mail-room)



a) Path #6

b) Path #7

Figure 20: Paths #6 and #7 show the execution of a K-turn to allow motion around a sharp corner. The paths diverge in response to two policies that are invalidated during run #7; the online re-planning is performed using D*-Lite.

Note, policies DE and FO have goal sets in approximately the same position, but the goal sets are 180 degrees apart in orientation.

- Pickup GO (mail-room), then deliver (visit) CR
- After CR, deliver CF
- Avoid BA, BB, BH, and HI
- Finish in location A

The plan must generate motion that navigates between the regions BT and A in a way that satisfies the other specifications. These specifications can be encoded in temporal logic by

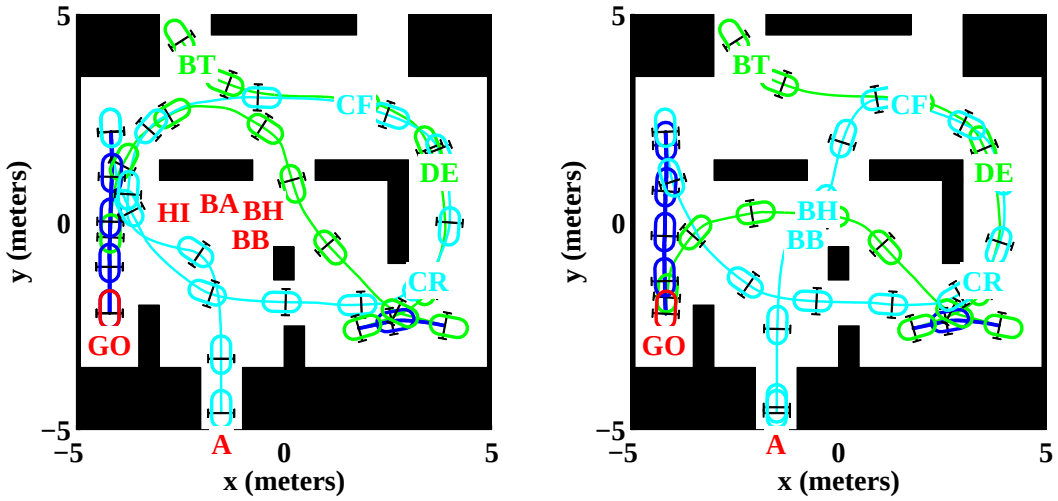
$$(\diamond((DE \mid FO) \wedge \diamond((GO) \wedge \diamond((CR) \wedge \diamond((CF) \wedge \diamond(A))))) \wedge \\ \square(\neg BA \wedge \neg BB \wedge \neg BH \wedge \neg HI)) \ ,$$

where $\diamond$ denotes the *EVENTUALLY* temporal operator, $\square$ denotes the *GLOBALLY* temporal operator, $\neg$ denotes the *NOT* logic operator, $\mid$ denotes the *OR* operator, and $\wedge$ denotes the *AND* logical operator [19, 20].

Given the temporal specification and the policy graph Γ_Λ, model checking techniques are used to generate an open loop sequence of policies that satisfy the specification[2] [19, 20]. Given an initial position and orientation of the robot that lies within the domain of policy BT, executing the policies according to the given sequence satisfies the specification. The simulation of this plan is shown in Figure 21-a. Alternately, we can change the specification so that instead of avoiding the regions (BA, BB, BH, and HI), we visit either BB or BH. The results of this later result is shown in Figure 21-b.

This task level planning worked well in simulation, but limited attempts to execute the plans on the actual robot exposed some limitations. The attempted runs ended in failure when a perturbation took the robot

[2]Many thanks to Hadas Kress-Gazit at UPenn for assistance in encoding the specification and executing the planner.



a) Path #1

b) Path #2

Figure 21: Simulation of open loop policy sequences derived from temporal logic specifications.

outside the domain of one of the policies in the open loop sequence. This can be addressed by modifying the policies to have a larger domain, thereby making it less likely to escape a policy under perturbation, or by rerunning the model checking based planner with the current configuration as the new initial condition [19]. Fundamentally, the open-loop sequences lose the robustness inherent in the partial order approach.

One reason for the system being forced through policies with limited domains is that the current implementation of the planner does not handle the indeterminacy due to the extended prepares; therefore, edges connecting unions of policies were removed from the graph. This forced the system through some policies with limited domains, making the system much more sensitive to localization disturbances.

7 Conclusion

The results of simulations and actual robot experiments presented in this paper demonstrate the validity of the approach, and provide a powerful demonstration of the inherent flexibility of planning in the space of control policies. Policies of the type outlined in this paper enable a large class of discrete planners, including those that can handle temporal logic specifications, to work with real, non-ideal, convex-bodied systems operating in real environments. The approach provides consistent results and maintains the provable guarantees of the local control policies, while enabling the flexibility to replan in real time. The sequential composition techniques advocated in this paper are generally applicable to a wide variety of control policies — beyond the flow-through policies described in this paper — provided the basic requirements that we outline are met.

One shortcoming of this set of experiments is the lack of integrated localization, which prevented the robot from running in an actual cluttered environment due to the dependence on dead reckoning. As the ultimate success of the strategy depends on accurate estimates of configuration, future experiments will incorporate vision-based localization that allows the system to estimate its configuration based on the position of known landmarks. This will allow testing the control strategy with a fully integrated autonomous system operating amongst physical obstacles.

Another issue that must be addressed is the time required to instantiate the policies. Currently, the trial and error method used to specify the policy parameter values is too time consuming. Future work will incorporate improvements to the deployment scheme that allow faster, more automated deployment. One of these improvements, automated methods for testing the inclusion in the free configuration space, was discussed in Section 3.2.2. Work remains in developing automated techniques for specifying the policy parameter values.

A major theoretical issue with the approach is the lack of completeness. The work on automating the deployment of local policies will address the completeness issue by automatically deploying additional policies on-line if the current configuration is not contained in the current deployment. That said, there are valid reason for not seeking to be complete. By not trying to cover "every nook and cranny" of the free configuration space, the number of policies is kept reasonable . Limiting the size of the discrete planning problem enables faster planning in the space of control policies. By allowing for online deployment when new policies are needed, the benefits of fast planning can be maintained, while addressing the lack of completeness.

Our immediate research plans are to extend the results to more complicated configuration spaces and system models. Specifically, the 4-variable Ackermann steered car, where the mapping $A(q) \neq A(g)$. This structural difference, which necessitates a change in the control laws, is due to $A(q)$ depending on an internal shape variable that specifies the steering angle. Additionally, more expressive cell definitions will be developed that allow more aggressive maneuvers relative to the robot capabilities.

Acknowledgments

The first author would like to acknowledge Sarjoun Skaff and Hyungpil Moon for their assistance with the robot experiments, Maxim Likhachev and Dave Ferguson for helpful discussions and D* Lite code, and Hadas Kress-Gazit (UPenn) for generating the task level plans.

A Free Space Testing

This appendix presents an approach to determining the composite set of points R_{Ξ_i} occupied by a robot body over all configurations within a cell Ξ_i; that is we determine the swept volume of $R(g)$ over all $g \in \Xi_i$. This enables testing that a given cell is contained in the free space of positions and orientations of the robot body. The approach presented here is complementary to the approach of expanding obstacles; here we expand the cell.

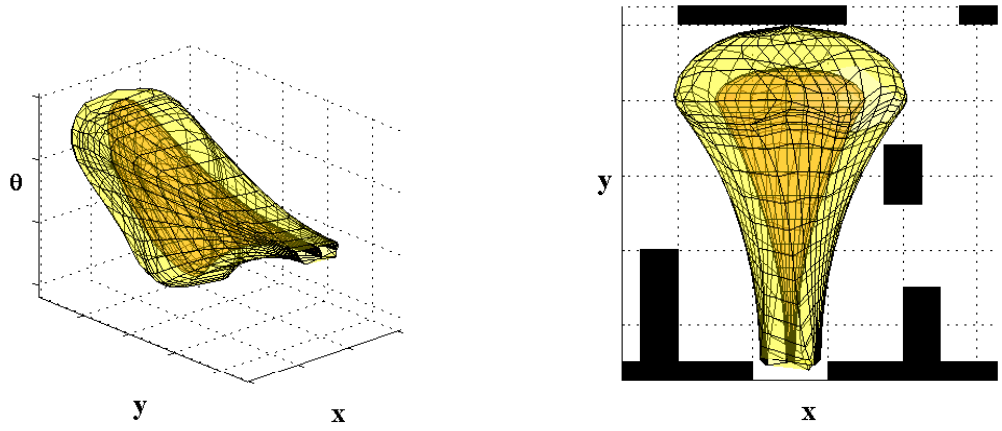
First, identify the body configuration space $\mathcal{G}$ with $\mathcal{W} \times \mathbb{R}$ and map the cell Ξ_i to $\mathcal{W} \times \mathbb{R}$; in an abuse of notation, let Ξ_i denote the cell in $\mathcal{W} \times \mathbb{R}$. The cell $\Xi_i \subset \mathcal{W} \times \mathbb{R}$ is expanded to account for the extent of the robot body (Figure 22-a), then the expanded cell is projected to $\mathcal{W}$ to yield R_{Ξ_i} (Figure 22-b). It is sufficient to show that no obstacles intersect the boundary of R_{Ξ_i}, denoted ∂R_{Ξ_i}, and that no obstacles are completely contained in the interior of R_{Ξ_i}. The expansion and subsequent tests depend only on the robot body shape, the collection of obstacles, and the specific cell shape.

Loosely speaking, the expanded cell is found by calculating the Minkowski sum[3] of $R(g)$ and Ξ_i, denoted $R(g) \oplus \Xi_i$. In this case, however, $R(g)$ is a 2D set in $\mathcal{W}$ and both Ξ_i and $R(g) \oplus \Xi_i$ live in $\mathcal{W} \times \mathbb{R}$. Another complication is that $R(g)$ varies with $g \in \Xi_i$. Thus, the Minkowski sum must be calculated based on a planar slice of the cell at a given orientation θ (Figure 23-a) and the robot body oriented at that orientation. We extend the Minkowski sum definition as

$$R(g) \oplus \Xi_i = \left\{ \begin{bmatrix} x \\ y \\ \theta \end{bmatrix} + \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix} p \mid g = \{x, y, \theta\} \in \Xi_i \text{ and } p \in R(\{0, 0, \theta\}) \right\}, \tag{13}$$

where $R(\{0, 0, \theta\})$ represents the set of robot body points at the origin but rotated by θ. This extended definition maps the 2D point p into the 3D space $\mathcal{W} \times \mathbb{R}$. This continuous mapping is equivalent to the standard Minkowski sum of a planar slice of Ξ_i at constant orientation with the robot at the same orientation (see Figure 23).

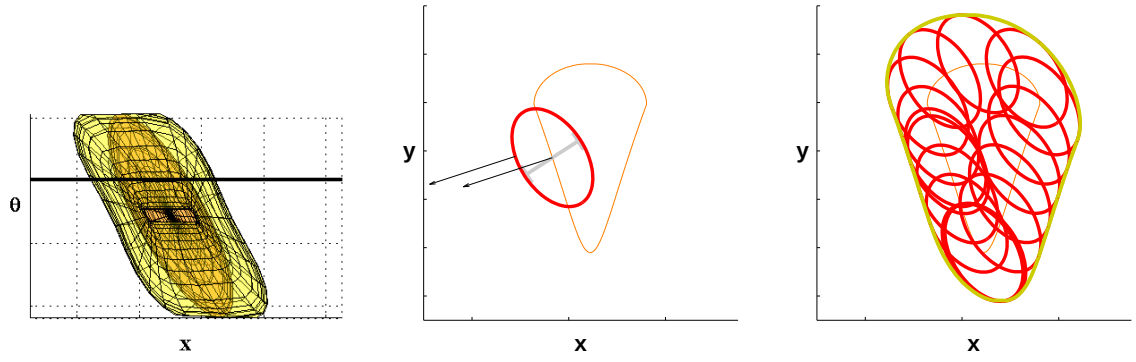
[3]The Minkowski sum is in contrast to the Minkowski difference, which is used to expand obstacles.



a) Cell expanded to represent robot body extent

b) R_{Ξ_i} - Extent of robot body in workspace

Figure 22: The cell boundary (dark inner surface) may be expanded to account for the non-trivial robot shape. The minimum signed distance from the boundary ∂R_{Ξ_i} to the closest obstacle is the closest approach of the system to an obstacle under the control of the associated policy. Note, the open face of the goal set is not shown. The goal set will be contained within the domain of another policy, so expanding this set is unnecessary.



a) Cell cut at constant orientation b) Robot body at cut boundary point c) Boundary of extended cell for cut plane

Figure 23: The calculation of the extended surface is based on a planar Minkowski sum. a) Consider cutting the cell at constant orientation. b) The robot body is placed along the boundary of the cell for this constant orientation. The corresponding point on the extended surface is found by matching normals for the planar cell boundary and the robot body. c) The boundary of the extended cell is calculated.

The composite set R_{Ξ_i} is found by projecting the boundary of $R(g) \oplus \Xi_i$ into the workspace using the trivial projection $\pi_{xy}\{x, y, \theta\} = \{x, y\} \in \mathcal{W}$. To determine the boundary, use the convolution surface of $\partial R(g)$ and $\partial \Xi_i$. For Minkowski sums of two sets A and B, the boundary of $A \oplus B$ is a subset of the convolution of the boundaries ∂A and ∂B [45, 46, 49]. That is $\partial(A \oplus B) \subset \partial A * \partial B$, where

$$\partial A * \partial B = \{a + b \mid a \in \partial A, b \in \partial B, \text{ s.t. } n_{\partial A}(a) \parallel n_{\partial B}(b)\} \tag{14}$$

with $n_{\partial A}(a)$ the boundary normal to A at a and $n_{\partial B}(b)$ the boundary normal to B at b. If both sets, A and B, are convex, then $\partial(A \oplus B) = \partial A * \partial B$.

The convolution surface definition in (14) is extended analogous to the Minkowski extension in (13). Given a robot configuration point on the cell boundary, $g \in \partial \Xi_i$, the corresponding outward pointing normal to the cell boundary, $n(g) = \begin{bmatrix} n_x & n_y & n_\theta \end{bmatrix}^T$, is projected to the x-y plane using the trivial projection $\pi_{xy} n(g) = \begin{bmatrix} n_x & n_y \end{bmatrix}^T$. Given a point $p \in \partial R(\{0, 0, \theta\})$, let $n_R(p)$ denote the outward pointing normal to the robot body at orientation θ. Then, the robot boundary points p such that $n_R(p)$ is parallel to $\pi_{xy} n(g)$ are found (see Figure 23-b). The convolution surface $\partial R(g) \oplus \partial \Xi_i$, which contains the boundary of $R(g) \oplus \Xi_i \subset \mathcal{W} \times \mathbb{R}$, is given by

$$\partial R(g) * \partial \Xi_i = \left\{ \begin{bmatrix} x \\ y \\ \theta \end{bmatrix} + \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix} p \mid g = \{x, y, \theta\} \in \Xi_i, p \in \partial R(\{0, 0, \theta\}), \text{ s.t. } \pi_{xy} n(g) \parallel n_R(p) \right\} \tag{15}$$

For realistic body representations, this will give a finite number of possible points on the expanded surface. For convex robot bodies, this result gives two potential points; the correct choice is the robot boundary point whose normal points in the same direction as the cell boundary normal. Thus, given a piecewise analytic representation of the robot body boundary and a parametric representation of the cell boundary, an exact piecewise parametric representation of the cell/robot convolution surface, $\partial R(g) * \partial \Xi_i$, can be determined. While the convolution surface may define some points on the interior of the expanded cell, the convolution surface will contain the expanded cell surface.

Example A.0.1 *Elliptical Robot Body.*

Consider the elliptical robot body shown in Figure 23-b. In the body reference frame, the robot body boundary is defined by $\left\{p_{rb} = \{x_{rb}, y_{rb}\} \in \mathbb{R}^2 \mid f^{-1}(p_{rb}) = 0\right\}$, where the function $f(p_{rb}) = p_{rb}^T M p_{rb} - 1$ with M defined as a 2×2 matrix that encodes the elliptical shape.

Use the extended convolution operator (15) to find the point on the expanded cell boundary given a point on the cell boundary. Let $g = \{x, y, \theta\} \in \partial\Xi_i \subset \mathcal{W} \times \mathbb{R}$ represent the configuration of the body reference point on the cell boundary, and let the cell surface normal $n(g)$ be given. Define the projected unit vector $\hat{n}_\pi(g) = \frac{\pi_{xy} n(g)}{\left\|\pi_{xy} n(g)\right\|}$. To determine the expanded cell, the point $p = \{x_p, y_p, \theta\} \in \partial R(g) * \partial\Xi_i \subset \mathcal{W} \times \mathbb{R}$ that corresponds to g must be found.

In the body reference frame, the point p is given by $p_{rb} = \mathrm{Rot}(\theta)^T \pi_{xy}(p - g)$ with $\mathrm{Rot}(\theta)$ the 2×2 rotation matrix. Let $v_p = \pi_{xy}(p - g)$, and rewrite the body function as

$$f(p) = v_p^T \mathrm{Rot}(\theta)\ M\ \mathrm{Rot}(\theta)^T\ v_p - 1\,. \tag{16}$$

The robot boundary normal in the workspace frame is given by

$$n_R(p) = 2\mathrm{Rot}(\theta)\ M\,\mathrm{Rot}(\theta)^T\ v_p\,.$$

Since p, and thus $n_R(p)$, is unknown, let k represent the unknown $\|n_R(p)\|$. To satisfy the matching normal requirement of the convolution surface, equate $n_R(p)/k$ and $\hat{n}_\pi(g)$ and solve for v_p as follows:

$$\begin{aligned}
\frac{n_R(p)}{k} &= \hat{n}_\pi(g) \\
2\mathrm{Rot}(\theta)\ M\,\mathrm{Rot}(\theta)^T v_p &= k\hat{n}_\pi(g) \\
v_p &= \frac{k}{2}\left(\mathrm{Rot}(\theta)\ M\,\mathrm{Rot}(\theta)^T\right)^{-1}\hat{n}_\pi(g) \\
v_p &= \frac{k}{2}\mathrm{Rot}(\theta)\ M^{-1}\,\mathrm{Rot}(\theta)^T\,\hat{n}_\pi(g)\ .
\end{aligned}$$

Substitute v_p into (16), and solve $f(p) = 0$ for

$$k = \frac{2}{\sqrt{\hat{n}_\pi(g)^T\,\mathrm{Rot}(\theta)\ M^{-1}\,\mathrm{Rot}(\theta)^T\,\hat{n}_\pi(g)}}\,.$$

Substituting k into the solution for v_p yields

$$\begin{aligned}
p &= g + \begin{bmatrix}1 & 0\\ 0 & 1\\ 0 & 0\end{bmatrix} v_p \quad \in \partial R(g) * \partial\Xi_i \\
\begin{bmatrix}x_p\\ y_p\\ \theta\end{bmatrix} &= \begin{bmatrix}x\\ y\\ \theta\end{bmatrix} + \begin{bmatrix}1 & 0\\ 0 & 1\\ 0 & 0\end{bmatrix} \frac{\mathrm{Rot}(\theta)\ M^{-1}\,\mathrm{Rot}(\theta)^T\,\hat{n}_\pi(g)}{\sqrt{\hat{n}_\pi(g)^T\,\mathrm{Rot}(\theta)\ M^{-1}\,\mathrm{Rot}(\theta)^T\,\hat{n}_\pi(g)}}
\end{aligned}$$

Thus, the point $g = \{x, y, \theta\}$ on the cell boundary is mapped to the point $p = \{x_p, y_p, \theta\}$ on the convolution surface that contains the expanded cell boundary.

To verify that the set does not intersect an obstacle, the boundary ∂R_{Ξ_i} is found, where $R_{\Xi_i} = \pi_{xy}(\partial R(g) * \partial\Xi_i)$. Finding ∂R_{Ξ_i} is equivalent to the silhouette extraction problem in computer graphics. Fortunately, there are several existing algorithms that may be used to determine the silhouette of $\partial R(g) * \partial\Xi_i$ [23].

Given a triangulated surface representation of the expanded cell, a common "brute force" approach is to determine the set of edges that contribute to the projection boundary by examining the adjacent facet normals [23]. The resulting finite set of line segments can be tested for intersection with a obstacle boundary edge or inclusion of an obstacle vertex inside the projection boundary. The minimum distance from a boundary line segment to an obstacle may also be determined. The complexity of this approach is linear in the number

of edges in the triangulation. While this brute force approach to silhouette extraction is not appropriate for real-time calculations, it is easy to implement and is suitable for cell validation during policy instantiation. The accuracy of the technique is determined by the accuracy of the surface triangulation; thus, more triangles implies more accuracy. Provided the closest approach to an obstacle is greater than the maximum error in the triangulation approximation, the cell is contained in the free configuration space.

References

[1] A. M. Bloch *et al. Nonholonomic Mechanics and Control*. Springer, 2003.

[2] Kwok Wai Au and Yangsheng Xu. Path following of a single wheel robot. In *IEEE Conference on Robotics and Automation*, pages 2925–2930, San Francisco, CA, USA, April 2000.

[3] J. Barraquand and J.C. Latombe. Nonholonomic multibody robots: Controllability and motion planning in the presence of obstacles. *Algorithmica*, 10:121–155, 1993.

[4] Calin Belta, Volkan Iser, and George J. Pappas. Discrete abstractions for robot planning and control in polygonal environments. *IEEE Transactions on Robotics*, 21(5):864–874, October 2005.

[5] Michael S. Branicky. Stability of switched hybrid systems. In *Proceedings of the 33rd Conference on Decision and Control*, pages 3498–3503, 1994.

[6] Michael S. Branicky. *Studies in Hybrid Systems: Modeling, Analysis, and Control*. PhD thesis, MIT, Dept. of Elec. Eng. And Computer Sci., June 1995.

[7] Michael S. Branicky. Multiple lyapunov functions and other analysis tools for switched and hybrid systems. *IEEE Transactions on Automatic Control*, 43(4):475–482, April 1998.

[8] Michael S. Branicky. Behavioral programming. In *Working Notes AAAI Spring Symp. on Hybrid Systems and AI*, Stanford, CA, March 1999.

[9] Michael S. Branicky and Gang Zhang. Solving hybrid control problems: Level sets and behavioral programming. In *Proc. American Control Conference*, pages 1175–1180, Chicago, IL, June 2000.

[10] R. W. Brockett. Asymptotic stability and feedback stabilization. In Roger W. Brockett, Richard S. Millman, and Hector J. Sussmann, editors, *Differential Geometric Control Theory*, pages 181–191. Birkhauser Boston, 1983.

[11] R. R. Burridge, A. A. Rizzi, and D. E. Koditschek. Sequential composition of dynamically dexterous robot behaviors. *International Journal of Robotics Research*, 18(6):534–555, 1999.

[12] Howie Choset, K. M. Lynch, S. Hutchinson, G. Kantor, W. Burgard, L. E. Kavraki, and S. Thrun. *Principles of Robot Motion: Theory, Algorithms, and Implementations*. MIT Press, 2005.

[13] David C. Conner, Alfred A. Rizzi, and Howie Choset. Composition of Local Potential Functions for Global Robot Control and Navigation. In *IEEE/RSJ Int'l. Conf. on Intelligent Robots and Systems*, pages 3546 – 3551, Las Vegas, NV, October 2003.

[14] C. I. Connolly and R. A. Grupen. Nonholonomic path planning using harmonic functions. Technical Report 94-50, UMass Computer Science, 1994.

[15] C. Canudas de Wit and R. Roskam. Path following of a 2-DOF wheeled mobile robot under path and input torque constraints. In *IEEE International Conference on Robotics and Automation*, pages 1142 – 1146, April 1991.

[16] R. DeCarlo, M. Branicky, S. Pettersson, and B. Lennartson. Perspectives and results on the stability and stabilizability of hybrid systems. *Proceedings of the IEEE, Special Issue on Hybrid Systems*, 88(7):1069–1082, July 2000.

[17] F. Diaz del Rio, G. Jimenez, J. L. Sevillano, S. Vicente, and A. Civit Balcells. A generalization of path following for mobile robots. *Journal of Robotic Systems*, 18(7):325 – 342, 2001.

[18] A. Deluca, G. Oriolo, and C. Samson. *Robot Motion Planning and Control*, chapter Feedback Control of a Nonholonomic Car-Like Robot, pages 171–254. Springer-Verlag, 1998.

[19] George Fainekos, Hadas Kress-Gazit, and George J. Pappas. Hybrid controllers for path planning: A temporal logic approach. In *IEEE Conference on Decision and Control*, Seville, Spain, 2005.

[20] George Fainekos, Hadas Kress-Gazit, and George J. Pappas. Temporal logic planning for mobile robots. In *IEEE Conference on Robotics and Automation*, Barcelona, Spain, 2005.

[21] Th. Fraichard. Motion planning for autonomous car-like vehicles. Ercim News (42):26-28, July 2000.

[22] E. Frazzoli, M.A. Dahleh, and E. Feron. Robust hybrid control for autonomous vehicle motion planning. In *Proceedings of the 39th IEEE Conference on Decision and Control*, volume 1, pages 821–826, Sydney, Austrailia, 2000.

[23] Bruce Gooch. Silhouette extraction. In *Course Notes for Theory and Practice of Non-Photorealistic Graphics: Algorithms, Methods, and Production Systems*. SIGGRAPH, 2003.

[24] Thomas A. Henzinger. The theory of hybrid automata. In *Proceedings of the 11th Annual Symposium on Logic in Computer Science (LICS)*, pages 278–292. IEEE Computer Society Press, 1996.

[25] T. Ikeda, M. Fukaya, and T. Mita. Position and attitude control of an underwater vehicle using variable constraint control. In *Proceedings of the 40th IEEE Conference on Decision and Control*, volume 4, pages 3758 –3763, 2001.

[26] Aleš Jaklič, Aleš Leonardis, and Franc Solina. *Segmentation and Recovery of Superquadrics*, volume 20 of *Computational imaging and vision*. Kluwer, Dordrecth, 2000. ISBN 0-7923-6601-8.

[27] P. Jiménez, F. Thomas, and C. Torras. *Robot Motion Planning and Control*, chapter Collision Detection Algorithms for Motion Planning, pages 255–304. Springer-Verlag, 1998.

[28] George Kantor and Alfred A. Rizzi. Feedback control of underactuated systems via sequential composition: Visually guided control of a unicycle. In *Proceedings of 11th International Symposium of Robotics Research*, October 2003.

[29] George A. Kantor and Alfred A. Rizzi. Sequential composition for control of underactuated systems. Technical Report TR-03-23, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, December 2003.

[30] Ilya Kolmanovsky and N. Harris McClamroch. Developments in nonholonomic control problems. *IEEE Control Systems*, 15:20–36, December 1995.

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

[32] J. P. Laumond. *Nonholonomic Motion Planning for Mobile Robots*. Centre National de la Recherche Scientifique, Laboratoire d'Analyse et d'Architecture des Systemes, 1998.

[33] J. P. Laumond, editor. *Robot Motion Planning and Control*. Springer-Verlag, 1998.

[34] J. P. Laumond, S. Sekhavat, and F. Lamiraux. *Robot Motion Planning and Control*, chapter Guidelines in Nonholonomic Motion Planning for Mobile Robots, pages 1–54. Springer-Verlag, 1998.

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

[36] Steven M. LaValle. From dynamic programming to RRTs: Algorithmic design of feasible trajectories. In A. Bicchi, H. I. Christensen, and D. Prattichizzo, editors, *Control Problems in Robotics*, pages 19–37. Springer-Verlag, Berlin, 2002.

[37] Steven M. LaValle and James J. Kuffner. Randomized kinodynamic planning. *International Journal of Robotics Research*, 20(5):378–400, May 2001.

[38] Daniel Liberzon and A. Stephen Morse. Basic problems in stability and design of switched systems. *IEEE Control Systems*, pages 59–70, October 1999.

[39] Maxim Likhachev and Sven Koenig. Speeding up the parti-game alorithm. In *Advances in Neural Information Processing Systems 15*. MIT Press, 2003.

[40] Stephen. R. Lindemann and Steven M. LaValle. Smoothly blending vector fields for global robot navigation. In *IEEE Conference on Decision and Control*, Seville, Spain, 2005.

[41] Gabriel A.D. Lopes and Daniel E. Koditschek. Level sets and stable manifold approximations for perceptually driven nonholonomically constrained navigation. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2004.

[42] Brian Mirtich and John Canny. Using skeletons for nonholonomic path planning among obstacles. In *Proceedings of IEEE International Conference on Robotics and Automation*, pages 2533–2540, May 1992.

[43] Richard M. Murray, Zexiang Li, and S. Shankar Sastry. *A Mathematical Introduction to Robotic Manipulation*. CRC Press, 1994.

[44] Sarangi Patel, Sang-Hack Jung, James P. Ostrowski, Rahul Rao, and Camillo J. Taylor. Sensor based door navigation for a nonholonomic vehicle. In *IEEE International Conference on Robotics and Automation*, pages 3081–3086, Washington,DC, May 2002.

[45] M. Peternell, H. Pottmann, and T. Steiner. Minkowski sum boundary surfaces of 3D-objects. Technical report, Vienna Univ. of Technology, Geometry Preprint Series No 140, 2005.

[46] Martin Peternell and Friedrich Manhart. The convolution of a paraboloid and a parametrized surface. *Journal for Geometry and Graphics 7*, pages 157–171, 2003.

[47] Arthur Quaid and Alfred A. Rizzi. Robust and efficient motion planning for a planar robot using hybrid control. In *IEEE International Conference on Robotics and Automation*, volume 4, pages 4021 – 4026, April 2000.

[48] Alfred A. Rizzi. Hybrid control as a method for robot motion programming. In *IEEE International Conference on Robotics and Automation*, volume 1, pages 832 – 837, May 1998.

[49] Maria Lucia Sampoli. Computing the convolution and the minkowski sum of surfaces. In *SCCG '05: Proceedings of the 21st spring conference on Computer graphics*, pages 111–117, New York, NY, USA, 2005. ACM Press.

[50] Nilanjan Sarkar, Xiaoping Yun, and Vijay Kumar. Control of mechanical systems with rolling constraints: Applications to dynamic control of mobile robots. *The International Journal of Robotics Research*, 13(1):55–69, February 1994.

[51] S. Sekhavat and M. Chyba. Nonholonomic deformation of a potential field for motion planning. In *Proceedings of IEEE International Conference on Robotics and Automation*, pages 817–822, May 1999.

[52] Anthony (Tony) Stentz. The D* algorithm for real-time planning of optimal traverses. Technical Report CMU-RI-TR-94-37, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, October 1994.

[53] Anthony (Tony) Stentz. The focussed D* algorithm for real-time replanning. In *Proceedings of the International Joint Conference on Artificial Intelligence*, August 1995.

[54] P. Svestka and M. H. Overmars. *Robot Motion Planning and Control*, chapter Probabilistic Path Planning, pages 255–304. Springer-Verlag, 1998.

[55] A. Tayebi and A. Rachid. A unified discontinuous state feedback controller for the path-following and the point-stabilization problems of a unicycle-like mobile robot. In *IEEE International Conference on Robotics and Automation*, pages 31–35, October 1997.

[56] M. Vendittelli, J.P. Laumond, and C. Nissoux. Obstacle distance for car-like robots. *IEEE Transactions on Robotics and Automation*, 15(4):678–691, 1999.

[57] Libo Yang and Steven M. Lavalle. The sampling-based neighborhood graph: An approach to computing and executing feedback motion strategies. *IEEE Transactions on Robotics and Automation*, 20(3):419–432, June 2004.

