Integrating Planning and Control for Constrained Dynamical Systems

David C. Conner

CMU-RI-TR-08-01

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Robotics

Robotics Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania

December, 2007

Thesis Committee:
Howie Choset, Chair
Alfred A. Rizzi, Chair
Jeff Schneider
Vijay Kumar, University of Pennsylvania

Abstract

This thesis develops an approach to addressing the coupled navigation and control problem for wheeled mobile robots. Instead of using a top-down decoupled approach that does not respect low-level constraints, or a bottom-up approach that cannot guarantee satisfaction of high-level goals, our approach is middle-out. We develop local feedback control policies that respect the low-level constraints. The approach then uses a collection of these policies with existing formal discrete planning methods to either produce a hybrid feedback control policy that guarantees high-level goals are satisfied, or in the worst case, verifies that the high-level specification is not realizable. Our approach enables existing formal symbolic planning methods to be applied to highly constrained systems.

We extend the sequential composition of local feedback control policies to wheeled mobile robots in a way that enables the automated synthesis of hybrid control policies. The thesis defines four basic "composability" requirements that guide our design of local policies. We develop two families of generic feedback policies that induce low-level behaviors in a way that enables their formal composition. The thesis also develops a novel approach for guaranteeing that a given control policy is collision free. By design, the policies respect multiple interacting constraints including large non-circular body shapes, nonholonomic constraints, and input bounds. Given a collection of the local policies and a task specification, our approach uses existing symbolic planning methods to automatically synthesize a switching strategy among the policies. Executing the switching strategy induces continuous motion that satisfies the high-level behavioral specification. This thesis demonstrates the approach on real mobile robots.

While wheeled mobile robot navigation is the chosen domain in this thesis, our future work will develop composable policies that extend these formal methods to other constrained dynamical systems.

Acknowledgments

I am indebted to my advisers, Howie Choset and Alfred Rizzi, for their support, patience, and guidance throughout the thesis process. I am grateful for their time in smoothing the rough edges, and challenging me to think more deeply about the fundamental questions underlying the basic problems. I would also like to thank my committee members, Jeff Schneider and Vijay Kumar, for their useful comments and discussions along the way.

My long term office mates, Sarjoun Skaff and Jonathan Hurst, have helped keep me sane during my stay at Carnegie Mellon. I thank you for the laughs, debates, and general good humor during my stay. You have been true friends, and I shall always value our time together.

Several of the simulations and experiments discussed in this thesis are the result of an ongoing collaboration with Hadas Kress-Gazit and George Pappas at UPenn. Thank you for your help and assistance with the automata synthesis. Your help, as well as your friendship, is valued.

Along with Sarjoun, I would like to thank Steve Tully, Hyungpil Moon, and Gorkem Erinc for their invaluable assistance with the robot experiments. I'd also like to thank Bart Nabbe for helpful conversations about the vision-based localization. Thanks to Maxim Likhachev and David Ferguson for sharing their D*-lite code and discussing aspects of discrete planning.

I would like to thank my friends and colleagues at the Robotics Institute, especially those in the Bio-robotics Lab, Microdynamic Systems Lab, and Manipulation Lab, including Aaron, Amir, Bertram, Clark, Devin, Elie, Prasad, Ravi, Sidd, and Uluc. Thanks for encouragement, and many enlightening discussions. Thanks to Ross and Matt for giving me useful comments on some thesis chapters. Thanks also to Bernardine, Chris, James, and Joel for helpful conversations.

Aaron Greenfield and David Steck, along with Sarjoun and Jonathan, helped me stay somewhat healthy and burn off some frustration in the weight room. My wife thanks you for that.

My thanks to Suzanne, Peggy, Jean, and Stephanie for their assistance and support over the years. Thanks for keeping the ship upright, and the faculty in line.

Thanks to the many professors and staff at CMU who were generous of their time, and freely willing to discuss ideas with no apparent payoff for them. I especially thank Chris Atkeson, Ed Clarke, John Dolan, Geoff Gordon, David Handron, Ralph Hollis, George Kantor, Matt Mason, and Reid Simmons. I appreciate the collaborative spirit fostered at CMU in general, and the Robotics Institute in particular.

Finally, but most of all, a big thank you to my Family. To my parents, I thank you for the sacrifices that you made, and for your love and support. To my children, Matthew and Ian, you are a constant source of inspiration and joy in my life. Never be afraid to follow your dreams; I love you so very much. To Cody, thanks for being my furry "best friend" for 13 1/2 years[1]. To my wife Karen, I could not have done this without your support, and probably would not have undertaken this task without your encouragement. Thank you for your love, sacrifices, and constant support. I love you. I think of you every time I type my password.

I am thankful for all that I have been given. X

This work was sponsored in part by the U.S. Army Research Office, under MURI DAAD19-02-01-0383. The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Office or the U.S. Government.

[1]This statement requested by Ian.

Contents

List of Figures

Chapter 1

Introduction

One of the most basic problems in robotics is moving around an environment towards a designated goal while avoiding obstacles. Determining how to avoid obstacles and reach the designated goal is a *navigation problem*; moving is a *controls problem*. In other words, for a real system to avoid obstacles and reach its goal, it must determine the control inputs that induce the required motion. Thus, at its most basic, this is a *coupled* navigation and control problem. Designing a single, globally-convergent, feedback control policy that addresses this coupled problem is generally intractable due to the complexity of non-linear system modeling and satisfying multiple interacting constraints.

Conventionally, this coupled navigation and control problem is addressed by using a decoupled approach. First the goal is defined, then tractable planning techniques specify a safe path through the environment, and a control law that causes the system to follow the desired path is designed. This decoupled approach can be problematic as constraints on the inputs and vehicle dynamics may make following a given path unrealizable; that is, there are no inputs that realize the desired velocity. This can render the goals unreachable without replanning. This thesis advances a robot control paradigm that integrates planning and control by considering the constraints up front. The low-level system behavior is guaranteed by local feedback control policies, and the high-level behavior is guaranteed by formally composing these simple behaviors. This method, termed *sequential composition*, is shown to be robust to disturbances and perturbations [21]. This thesis serves to extend basic sequential composition to nonholonomic systems and incorporate additional planning techniques.

1.1 Motivation

This thesis focuses on the domain of wheeled mobile robots navigating among obstacles in a planar workspace, as shown in Figure 1.1. We use this navigation problem to demonstrate the policy composition approach. Although navigation and control appears to be extensively addressed in the literature [22, 70, 75, 118], further inspection reveals that this problem demonstrates a variety of nonholonomic velocity constraints, input bounds, and configuration limits due to body shape and obstacles, that are often ignored or simplified. While each of these constraints is challenging in and of themselves, the combination and interaction of the constraints makes this a particularly difficult challenge. Where previous research has simplified this problem by eliminating one or more of these constraints, this thesis treats the constraints holistically. Thus, the wheeled mobile robot navigation and control problem provides an excellent contrast between existing planning and control approaches, and the paradigm advocated here.

Existing approaches to addressing the coupled navigation and control problem can be broadly classified as either “bottom-up” or “top-down” approaches [14]. The *bottom-up* approach depends

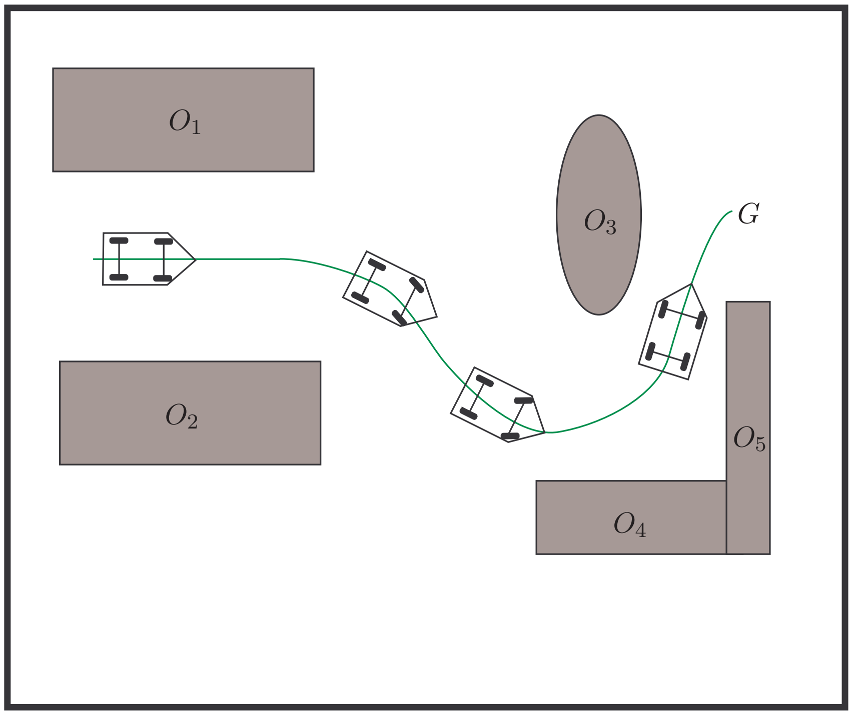
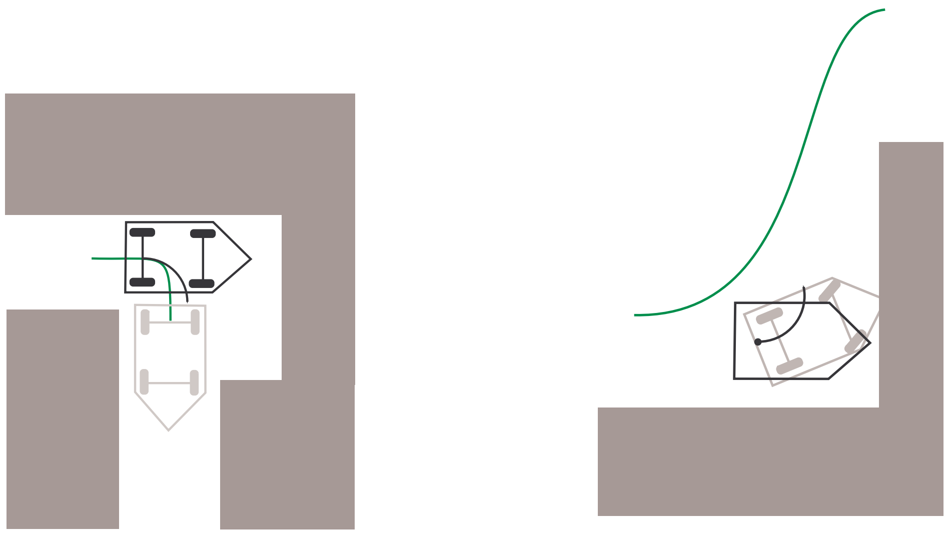


Figure 1.1: Navigation problem: control a mobile robot so that it moves through its environment and reaches its goal without colliding with any obstacles, O_i.

on emergent behaviors that are induced by applying low-level primitives based on reactions to sensor inputs [19]. This *behavior-based* approach is generally easy to implement, and has been used to demonstrate moderately complex behaviors. The low-level behaviors respect the low-level constraints by construction. This bottom-up approach attempts to avoid high-level constraints such as obstacles and satisfy high level goals by switching among the low-level behaviors. The approach is fundamentally not verifiable, and is not capable of guaranteeing that complex behaviors are correctly performed. A contrasting approach used in mobile robots is the “top-down” approach. Here, a reasoning system defines and schedules intermediate goals or tasks, a planning system defines a path to goal, and a feedback control policy attempts to follow the path [22, 75]. Provided each subtask is satisfactorily executed, the overall behavior is realizable. As stated earlier, this decoupled approach can be problematic because the high-level reasoning typically ignores the low-level system constraints in order for the planning problem to be tractable. This may result in goals that are unreachable, or plans that are not robust to disturbances along the way; consider the illustrations in Figure 1.2.

This thesis seeks to enable a “middle-out” [14] approach that combines the best of the “bottom-up” and “top-down” approaches. The low-level behaviors are implemented using feedback control policies that are designed to satisfy all of the system constraints over a limited domain. The key difference with respect to conventional bottom-up approaches is that convergence guarantees are required for these behaviors. Instead of identifying sub-tasks or sub-goals from the top down, this middle-out approach uses the available local policies to define what sub-tasks are realizable. These



(a) Curvature constraint violation (b) Significant disturbance

Figure 1.2: Decoupled planning and control can lead to difficulties for constrained systems. a) The implemented control law cannot follow the planned path due to a curvature constraint, shown by darker line. b) Trying to reacquire a valid workspace path after a disturbance may lead to collision because the control law is ignorant of obstacles. The question then arises, how much error can be tolerated without requiring replanning?

sub-tasks are then composed to address the overall goal. This technique has been called "behavioral programming" [14]. It then becomes more natural to plan symbolically over a discrete collection of realizable behaviors than by specifying sub-goals in the continuous workspace. One goal of this thesis is to extend the types of planning available, while preserving the systems ability to react to changing environmental conditions.

1.2 Approach

The middle-out approach requires local *feedback control policies* that map vehicle states to valid control inputs. The *domain* of the policy is the region over which the mapping is valid. To be valid, the feedback control policy must respect the interacting system constraints over its domain, and

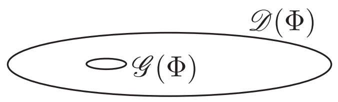


Figure 1.3: The policy has a domain, $\mathscr{D}(\Phi)$, over which it is valid, and a designated goal set $\mathscr{G}(\Phi)$.

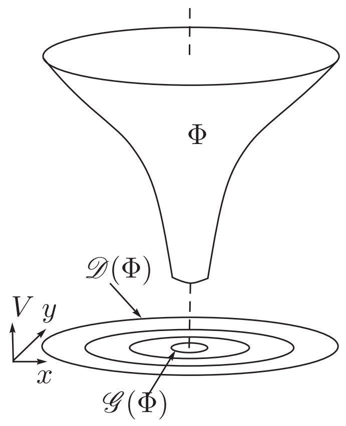


Figure 1.4: The funnel metaphor for a feedback policy can be viewed as an idealized version of a Lyapunov function. The funnel "mouth", represented by the largest ellipse at the maximum Lyapunov value, specifies the policy domain $\mathscr{D}(\Phi)$; the planar ellipses represent level sets of the Lyapunov value. The system flow induced by the closed loop dynamics of the feedback policy moves the system to ever lower Lyapunov values, V, toward the goal set, $\mathscr{G}(\Phi)$, represented by the projection of the funnel's small end.

guarantee convergence to its goal set. Figure 1.3 shows a schematic representation of the domain and goal set of a policy over a planar region. Throughout this thesis, we use the "funnel" metaphor to represent the closed-loop action of a policy over its domain [21, 86]. With reference to Figure 1.4, the height, V, of the simple funnel represents the value of an idealized Lyapunov function. Under the influence of the policy, the closed-loop system dynamics act to decrease the height (Lyapunov value) while bringing the system towards the goal set. Given certain properties of the feedback control policy, the closed loop behavior flows from the policy domain to its associated goal set [21].

Ideally, the coupled navigation and control problem could be addressed with a single global feedback control policy that respects the system constraints. In this case, the ideal global control policy would have a Lyapunov function whose level sets resemble those of Figure 1.5. Instead of a "thin" path defined through the workspace, the global policy has a "thick" domain that covers the workspace. This approach is robust to disturbances, and mitigates the need for some re-planning. For unconstrained systems navigating in open spaces, the global control policy design is simple. Unfortunately, designing a single, provably correct, globally convergent control policy for realistic constrained systems navigating in cluttered environments is thus far intractable.

To get around the difficulty of designing a global feedback control policy, this thesis advocates addressing the coupled navigation and control problem by decomposing the global problem into a series of intermediate tasks, or behaviors, where each intermediate task is solvable by a memoryless[1] state feedback control policy with a local domain. Each intermediate behavior takes the system safely to its intermediate goal, and brings the system state closer[2] to the final goal. Put simply, the policies are sequentially composed to approximate the global policy; hence the name *sequential composition*. The benefit is that local policies are easier to define in a way that satisfies the system constraints.

[1]Memoryless policies depend only on the current state, and not on previous states.

[2]Here we define "closer" in the sense of remaining actions, and not necessarily closer in the Euclidean sense.

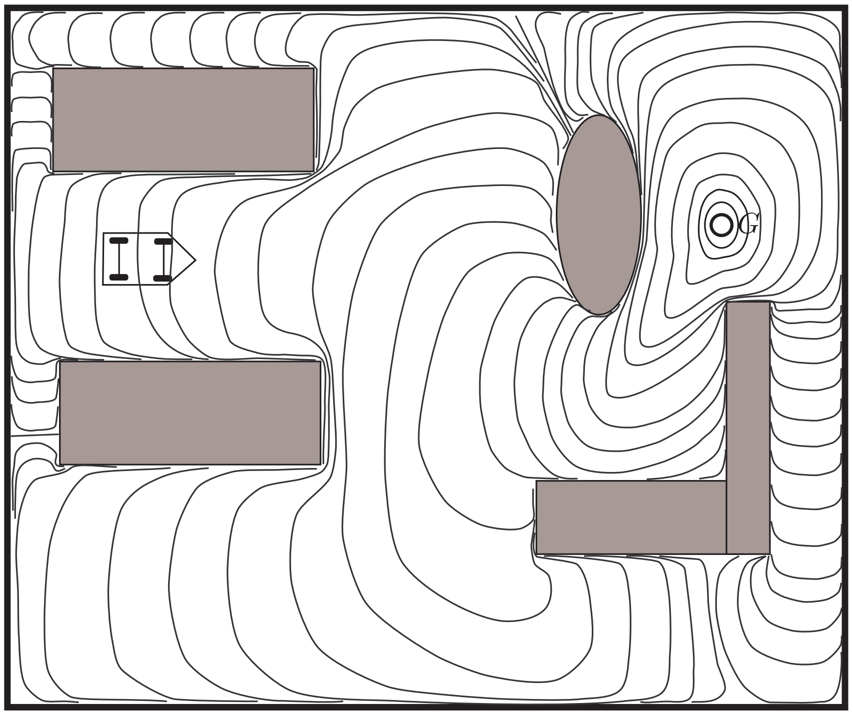


Figure 1.5: The ideal solution is a global control policy that induces the desired behavior. Here, the contours represent an iconic view of level sets for some idealized potential function.

The feedback control policies defined in this thesis are specifically designed to satisfy the low-level constraints of the system over its local domain, while retaining performance and safety guarantees. Consider the two-dimensional iconic policies represented in Figure 1.6. The small funnels are ordered so that the goal set of one empties into the domain of another, until the final funnel's goal set corresponds to the overall goal.

By examining the relationship among domains for a collection of policies, the resulting transitions between policy domains can be represented as a graph. As the closed-loop behavior of the policy moves the system from its domain to its associated goal set, if the goal set of one policy is contained in the domain of another, the first policy induces a transition from its domain to the next. Figure 1.7 shows a trivial example of this transition relation. Given the local behaviors encoded by the local policies, planning becomes a problem of ordering the discrete graph. Thus, while the low-level behaviors are encoded "bottom up" using policies that respect the system constraints, the planning and reasoning steps can be applied "top down" on the discrete graph. The discrete transitions are realizable by the system because of the guarantees provided by the local feedback control policies. Planning in this discrete space of policies is easier than planning over the continuous configuration space, and much more flexible with respect to high-level task specifications.

This approach leverages the strengths of symbolic planning methods and feedback control approaches, while preserving the guarantees of both. The result is a *hybrid* control system that exhibits both continuous dynamics and discrete events and/or logic. The key challenge faced by this thesis

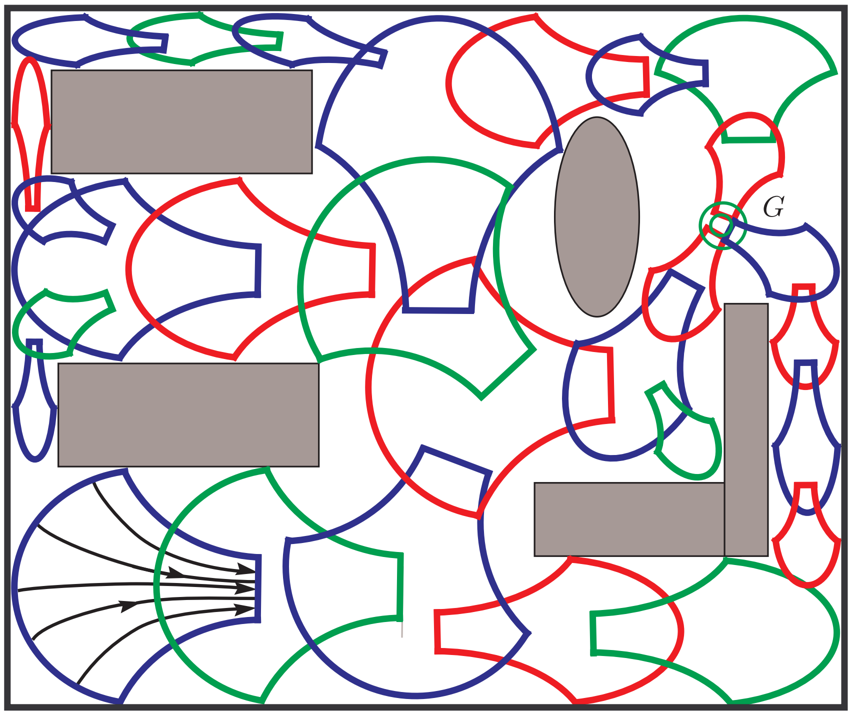


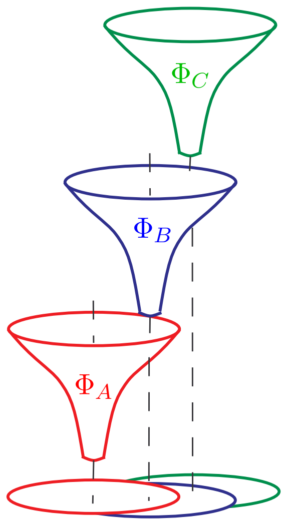
Figure 1.6: Policy composition gives a formal method of approximating the convergence of a global policy. The two-dimensional iconic funnels represent policies that induce flow over the robot configuration space. As shown in the lower left funnel, the close loop action induces flow from the larger domain into the narrow portion that serves as the goal set. Note, the colors are to help differentiate different policy domains; they have no special meaning.

in developing these hybrid control systems is to define control policies that respect the system constraints and are "composable."

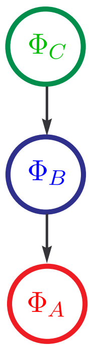
The benefit of developing suitable policies, and the real power in sequential composition, is the flexibility of planning in the space of control policies. Because the planning occurs on the discrete graph, it becomes tractable to plan for multiple goals that depend on information gathered at run time [24]. The aim is to move beyond simple navigation from point 'A' to point 'B', towards a higher level symbolic specification of tasks and goals, while retaining the robustness and guarantees of feedback control. Instead of requiring each small detail to be specified, we would like to describe the task at a high level, and have the system autonomously execute in a manner that satisfies that desired task. This thesis develops techniques that enable expressive and flexible planning with real systems, operating under real world constraints. We seek to advance symbolic *behavioral programming* techniques, and extend their application to highly constrained systems [14, 105].

1.3 Thesis Contributions

The first contribution of this thesis is to refine the idea of "composability" as it relates to the policy design. In order to realize the benefits of sequential composition, there must exist a collection of



(a)



(b)

Figure 1.7: A more complex behavior can be induced by the composition of relative simple policies: a) policy composition, b) discrete transitions between policy domains represented as a graph.

local policies that respect the local constraints of the system, while guaranteeing performance over the local domain. To that end, we enumerate four necessary properties that local policies must satisfy to be composable. These properties are generally applicable to any dynamic system, and extend the basic convergence and invariance properties defined in [21]. This thesis extends the types of policies allowed under sequential composition to include "flow-through" policies in addition to conventional convergent policies, and develops minor extensions to the allowed relationship among the policies.

Second, we demonstrate a policy design approach that satisfies the necessary properties for wheeled-mobile robots moving in cluttered environments. Two parameterized policy designs are presented; one based on level sets and one based on path-following. The thesis defines tractable techniques for testing that the defined policies satisfy the necessary properties. This includes a new technique for testing that a given continuous feedback policy is collision free over its domain. Other policy designs are certainly possible, and may be readily incorporated into the policy composition framework provided they satisfy the four necessary properties defined in this thesis.

Third, a strategy for partially automating the policy deployment is presented. Leveraging invariance properties of the robot model, a limited number of basic maneuvers may be instantiated at various locations in the environment via rigid body transformation. As this approach only approximates the ideal global policy, the approach is not necessarily complete; therefore, the thesis defines a sampling-based approach to assess the relative completeness of the policy deployment. That is, determine what fraction of the free space is captured by the hybrid control system.

Fourth, the thesis demonstrates a variety of planning approaches over the discrete graph. This thesis puts several existing approaches to discrete planning into the context of sequential composition, and discusses their relative strengths and weaknesses. The approaches range from simple graph-based Dijkstra's search, to reactive automata-based approaches that satisfy high level temporal specifications [68]. Thus, we extend sequential composition to more flexible planning techniques, while enabling these advance planning techniques to be applied to more complex and realistic systems. While the discrete planning approaches are not contributions, this thesis allows these discrete planning approaches to be applied to more complex systems. The planning techniques are demonstrated via simulations of several robot models and experiments on a real mobile robot.

Finally, the thesis concludes with a discussion of some open problems that would allow even more expressive and flexible planning over the policies.

1.4 Thesis Overview

Before presenting the extensions to the sequential composition approach, the thesis addresses the existing literature on the subject. First, to put this work in context, we provide a brief description of contrasting approaches that address the navigation and control problem. Next, we provide an overview of the work that inspires this approach. The related work concludes with a discussion of some discrete planning techniques that may leverage our approach.

Chapter 3 provides an overview of our technical approach. This includes an enumeration of the generic policy requirements, as well as our extensions to the basic sequential composition approach. The chapter concludes with a discussion of several approaches to planning in the space of control policies.

Chapter 4 describes work on fully actuated idealized systems. This serves to solidify the ideas, and highlight some of the issues. From there, Chapter 5 extends the basic approach to single bodied nonholonomically constrained mobile robots. This chapter discusses policy design and deployment approaches, as well has the approach to measure the completeness of the deployment. Chapter 5 makes reference to several appendices that provide details about the specific policy designs.

Chapter 6 presents several advanced demonstrations of planning in the space of control policies. These demonstrations serve to show the flexibility of the approach, and motivate further research. Chapter 7 concludes with an overview of some open problems that remain, including some possible approaches to discrete planning that seek to combine the strengths of the discrete planning approaches described in the thesis.

Chapter 2

Related Work

The work related to this thesis comes in three general areas: contrasting approaches, sequential composition, and discrete planning. The first section provides contrast for the approach advocated by this thesis by giving a broad overview of other approaches to addressing the navigation and control problem for wheeled mobile robots. As our approach is motivated by the sequential composition technique advocated in [21], the second section provides an overview and presents work directly related to sequential composition. The final section describes existing work in discrete planning that can be used to plan in the space of control policies; leveraging this works allows us to expand the type of planning used with policy composition,

2.1 Conventional Approaches

Numerous techniques have been developed over the years in an attempt to address the problem of moving a robot or other dynamical system from one point to another in a cluttered environment; see [22, 70, 71, 75, 118] for details. Researchers have typically broken the problem into different parts, only focusing on one part, and leaving the rest to others. Some techniques work only in ideal conditions; while others solve local problems, but not global problems. This section provides an overview of three approaches: path planning approaches that consider nonholonomic constraints, control approaches that attempt to follow paths, and attempts to couple planning and control.

Nonholonomic Motion Planning and control This thesis specifically addresses single-bodied, wheeled mobile robots subject to nonholonomic constraints, which limit the instantaneous velocity of a system and complicate the coupled navigation and control problem. This subsection provides an overview of approaches that specifically address nonholonomic constraints.

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 [18]. It is well known that most nonholonomic systems, although small-time locally controllable, fail Brockett's test [65, 94]. Several classes of stabilizing feedback control policies have been developed: discontinuous time invariant, time varying, and hybrid control policies [65]. Given the complexity of this "simple" control task, it is no surprise that the coupled navigation and control task for nonholonomic systems is more complex than for holonomic[1] systems.

[1]We yield to common usage and refer to systems subject to nonholonomic constraints as "nonholonomic systems." Systems without nonholonomic constraints are called "holonomic systems." To be technically correct, it is the constraints that are classified as holonomic and nonholonomic, not the systems.

Ignoring obstacles for the moment, there are several methods – including sinusoidal inputs, piecewise constant inputs, optimal control, and differentially flat inputs – that solve the point-to-point steering problem between two positions using open-loop controls [73, 94]. These methods, which are sometimes incorporated into the discrete planning systems, are open loop control methods that do not respect obstacles [65, 72, 94]. Collision detection must be performed by simulating the system response to determine feasibility in a cluttered environment [53]. These point-to-point steering methods are strictly open-loop, and not suitable for feedback control. The inevitable errors that arise during execution necessitate repeated applications of the algorithms to induce convergence to the goal point.

For a cluttered environment, there are numerous planning techniques for holonomic systems, but fewer that simultaneously address nonholonomic constraints and obstacles [22, 75]. A common planning approach is to pretend the system is holonomic and use a standard planning system such as grid-based planning or Voronoi diagrams. If a nonholonomic system is small-time locally controllable, any continuous path can be approximated arbitrarily well [73]. 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 control reversals to follow the desired path. Another approach is to perturb the planned path to respect nonholonomic constraints [111].

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 [89]. The approach uses the SFP metric to define the largest ball around the current configuration, and then selects the shortest feasible path to the point on the holonomic path that intersects the ball [120]. Methods based on dynamic programming determine an optimal path for a discrete set of controls (e.g. hard left, soft left, straight, soft right, hard right) [5, 39, 76]. Probabilistic roadmaps (PRM) and rapidly-exploring random trees (RRT) are other discrete approaches to determining feasible paths for nonholonomically constrained systems [78, 115]. The approaches look for safe feasible paths between the current point and a chosen sample point. With the exception of dynamic programming, these discrete methods are not feedback based, and require re-planning if the system deviates from the desired path.

Path Following Control Laws Given a path through the cluttered environment, the system requires a control policy that causes the system to follow the designated path. A path following control policy is used to determine the control inputs that cause the system to converge to a desired path if the initial condition is off the path, and to follow the path in spite of disturbances.

The presence of nonholonomic constraints renders the design of path-following control law a non-linear controls problem. 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 [29]. The design of the control laws is often based on Lyapunov analysis [32] or feedback linearization [33, 110].

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 [29]. This may fail in cluttered environments as the designated point may exactly follow a safe path, yet still allow another point on the robot to collide with an obstacle. The second formulation attempts to have the robot track position and orientation of a path in the free configuration space [32]. 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 [3, 32, 29, 116].

In addition to path-planning, trajectories that specify when the system arrives at points along the path may be planned [32, 110]. Trajectory-tracking problems can be problematic if the system is subject to a constraint that delays the tracking [32]. 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 require reverse motions along the path to match the specific time and position [110]. Unless the time matching along the trajectory is crucial, path-following is often a better formulation [32, 110].

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 as shown in Figure 1.2-b. Path-following may be coupled with local obstacle avoidance [9, 38, 60, 69, 113], but this may invalidate the convergence guarantees. Thus, if the errors are large enough, the paths must be re-planned, starting from the current location.

Coupled Planning and Control Some attempts have been made to integrate planning and control, most notably potential methods and optimal control techniques [93, 104].

Potential functions, which are used despite the well known local minima problem, address the coupled navigation and control problem by using the potential function's negative gradient vector field to determine control inputs [64]. For idealized, holonomic, kinematic systems, the negative gradient vector, or any positive scalar multiple thereof, may be used directly as control inputs. For nonholonomic systems, most potential functions do not have gradients that respect the nonholonomic constraints, which makes direct usage of the gradient infeasible.

For idealized holonomic second-order dynamical systems, the addition of a dissipative term in the control law results in convergence to a local minimum for any system whose total energy is less than or equal to the potential on the boundaries of the free configuration space [64, 63]. Most potential methods used in control do not account for control input bounds of second-order dynamical systems. Many of these methods have unbounded potential at the obstacle boundary [104]. Even with bounded potentials, the control laws may not respect arbitrary dynamic constraints on control inputs if the potential function does not account for the total energy [64]. For example, a system near a boundary moving towards the boundary may not stop before collision with the boundary under gradient control if the total energy is not respected. The magnitude of the potential gradient may also vary greatly, and therefore, be unsuitable for direct control.

Optimal control techniques are closely related to potential-based navigation techniques. Locally, by following the negative gradient, a system maximally reduces the potential. Optimal control techniques build a special potential function, called a *value function*, such that a local decision induces the optimal trajectory for a given cost function. Given running and terminal cost functions, the value function is the solution to the Hamilton-Jacobi-Bellman (HJB) partial differential equation [35].

A defined cost structure is fundamental to the use of optimal control techniques [47]. If this cost structure is not given *a priori*, the cost structure must be designed to generate the desired behavior, while guaranteeing some measure of safety and robustness. The design of such a cost/reward structure is difficult because the induced behavior is only known after the result is calculated. Therefore, the design of a suitable cost function is an iterative process – define a cost function, compute the controller, run experiments, evaluate the results, and modify the cost function as necessary. Given the value function, the optimal control formulation results in a global control policy that specifies the optimal control action for a given state.

Solving this global control problem is one of finding the appropriate value function for a given cost function by solution of the HJB; however, several problems arise. In general, global C^1 smooth solutions to the HJB do not exist [35, 92]. A well known consequence of the lack of global C^1 continuity is that the optimal solutions are *fragile*, and may be unstable for minor perturbations [57]. As HJB equations do not generally have a closed form solution, they are typically solved numerically using finite element, finite difference, or dynamic programming methods [35, 92].

Dynamic programming (DP) can be used to solve the HJB numerically using a cost-to-go iteration scheme by discretizing the state and control action space [6]. The discrete action space can be used to model the impact of nonholonomic constraints. Although DP is extremely powerful, it suffers from the well known *curse of dimensionality*, and is limited to low dimensional state spaces or coarse approximations. Numeric solutions to the HJB equation often require adaptive discretization to yield an acceptable solution [92].

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 [27]. 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 invariant sub-manifolds in configuration space that contain the goal [50, 56, 85]. While the nonholonomic constraints are generally not integrable, constraints can be added that render the system integrable on a configuration space sub-manifold. On this sub-manifold, the control naturally respects the constraints and it is possible to steer towards a designated goal contained in the sub-manifold. The hybrid control approach drives the system to some point on 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 sub-manifold can be intractable; particularly for systems where the invariant sub-manifold is not given in closed form, but must be approximated through an iterative process. The approaches also involve designing several functions that require insight into the specific problem and system constraints. While suitable for feedback over local domains, they are generally not applicable to cluttered environments due to the difficultly of defining the sub-manifold that avoids obstacles.

2.2 Policy Composition Approaches

Due to the limitations of existing approaches to addressing the coupled global navigation and control problem, this thesis advocates using *sequential composition* of local control policies, which are easier to define in a way that satisfy system constraints [21]. Sequential composition enables the construction of switched control policies with guaranteed behavior and provable convergence properties. Since this thesis uses sequential composition as a tool to construct hybrid control policies, this section begins by describing the basic sequential composition approach defined in [21]. This enables a better understanding of how our work extends and compliments sequential composition. The section provides examples of applications of sequential composition to existing systems, and concludes with a discussion of some related approaches.

2.2.1 Basic Sequential Composition

The 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. Burridge *et al.* [21] build on this simple idea by formally defined what composition means, and defining an algorithm for constructing a hybrid control policy using this idea. As later chapters will extend the basic approach, this section presents a formal overview of their work.

Sequential composition is based on a formal notion of *prepares* defined among policy domains [21]. The prepares concept is built upon the idea of "pre-image back chaining" [86]. For a given control policy, define the *safe domain of attraction* as the largest region of state space that does not intersect an obstacle and where the closed-loop behavior does not allow the state to exit the region, and induces convergence to the policy's goal. In other words, the safe domain of attraction is *positive invariant*; that is, for any initial condition within the safe domain of attraction, the state does not exit the domain under the influence of the policy. Henceforth, the term *domain* is synonymous with *safe domain of attraction*. For a given policy, the domain is the pre-image of the goal set in the sense that any state in the domain is mapped to the goal set by the action of the policy.

Sequential composition defines a formal relationship between policy domains. Let a finite collection of 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's goal corresponds to the overall goal. Given two control policies from Λ, with domains $\mathscr{D}(\Phi_i)$ and goal sets $\mathscr{G}(\Phi_i)$, Φ_2 is said to *prepare* Φ_1, denoted $\Phi_2 \succeq \Phi_1$, if $\mathscr{G}(\Phi_2) \subset \mathscr{D}(\Phi_1)$.

By properly prioritizing the collection of 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 [21]. The domain of attraction of the switched policy is equal to the union of the domains of the component policies, as shown in Figure 2.1; the domain over which a given policy is active is determined by the switching strategy. It is easier to define policies that respect the system constraints over a limited local domain; by composing local policies that respect constraints, the hybrid control policy defined by the local policies and switching strategy also respects the constraints. By adding local policies that capture additional regions of the free state space, an almost global control policy can be defined.

Prioritizing the policies is done in relation to an overall goal and the prepares relationship between policy domains. The prepares relationship between any two policies in the collection Λ induces a directed graph, Γ_Λ, over the collection of instantiated control policies. A directed edge

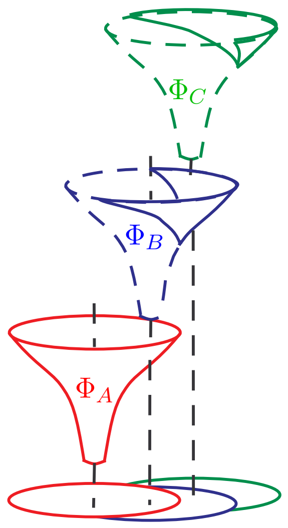


Figure 2.1: Burridge-Rizzi-Koditschek defined a switching strategy based on a total ordering of the policies.

connecting two nodes in Γ_Λ corresponds to a transition that may occur when the state of the system enters the domain of the other policy; this transition is guaranteed by the prepares relationship. The graph Γ_Λ, which we term the *prepares graph*, defines a *transition relation* between control policies [23]. The prepares graph approximates the continuous transitions as a set of discrete transitions between nodes that represent the local policy domains. The act of assigning a priority to each policy is a form of discrete planning using the prepares graph.

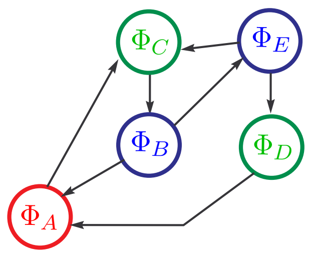
In general, the prepares graph Γ_Λ is cyclic, which can lead to limit cycles that do not reach the goal policy; however, a directed acyclic graph, $\Gamma'_\Lambda \subset \Gamma_\Lambda$, may be generated over the collection of policies. Figure 2.2 shows a simple example. In [21] this is accomplished by searching Γ_Λ 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 directed acyclic graph Γ'_Λ can be viewed as an ordering over the collection of control policies. By construction, the directed acyclic graph is a connected graph containing a node corresponding to the policy that stabilizes the overall goal. Switching between policies in Γ'_Λ is guaranteed to bring the system to the overall goal. Under the transition map induced by the prepares relationship, Γ'_Λ represents a finite state automata [48].

Given the collection of policies Λ, the switching strategy defined by Γ'_Λ induces an overall switching control policy Φ. The union of the domains of the policies included in Γ'_Λ gives the domain of the overall policy; that is

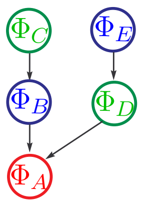
$$\mathscr{D}(\Phi) = \bigcup_{\Phi_j \in \Gamma'_\Lambda} \mathscr{D}(\Phi_j) \ . \tag{2.1}$$

The collection of policies in Λ and the switching strategy defined by the ordering Γ'_Λ is called a *deployment*.

The overall control policy induced by sequential composition is fundamentally a hybrid control policy [12, 14, 48]. The composition of these local policies in a hybrid systems framework enables analysis on the discrete representation of the transitions between policy domains [14, 21]. Given knowledge of policy domains containing the current state, one may analyze whether another policy's goal is reachable using Γ'_Λ, without the need to re-analyze the underlying continuous system [48]. This gives a simple discrete approach to deciding if a given navigation problem is solvable with a particular collection of policies. This 'reachability' analysis may be done on Γ_Λ prior to planning, or implicitly during construction of the acyclic graph Γ'_Λ. That is, starting from the overall



(a) Simple prepares graph Γ_Λ with 3 cycles



(b) Γ'_Λ tree

Figure 2.2: The graph on the left has 3 cycles; the tree on the right obeys the sample prepares relationship while protecting against limit cycles. In this case, the goal node is Φ_A.

goal, any policy added to the ordering Γ'_Λ can drive the configurations in its domain to the goal by construction.

The stability of the underlying control policies guarantees the stability of the overall switched policy because the ordering results in monotonic switching [21]. That is, the policies switch from lower priority to higher priority policies. This obviates the need for complex hybrid stability analysis of the form given in [11, 13, 30, 79]. Disturbances, which may carry the state to the domain of a lower priority policy, are robustly handled provided their magnitude and rate of occurrence is small compared to the convergence of the individual policies [21]. The overall control policy resulting from the partial order covers the largest region of the free state space for a given collection of control policies, while guaranteeing that any state in the union of the individual domains is ultimately brought to the goal.

Burridge *et al.* [21] demonstrate sequential composition on a robot that juggles a ping-pong ball by repeatedly batting the ball with a paddle. The robot is tasked with moving the ball by juggling through its environment while avoiding obstacles. In this case, the obstacles are sensor limits (camera field of view) and a physical obstacle in the workspace. Burridge *et al.* define a policy with free parameters; changing the parameter values changes the bouncing ball's steady state horizontal position and apex height above the horizontal plane. A policy with free parameters is called a *generic policy*; assigning specific parameters values results in an *instantiation* of the generic policy. By making slight modifications to the basic policy, they define a collection of generic policies termed a *palette* [21] .

Burridge *et al.* [21] use a manual approach to ordering the policies. Starting with a single policy that stabilizes the overall goal, multiple policies are instantiated in the system's free space by specifying a collection of set-points and other control policy gains for generic policies chosen from the palette. These instantiated policies form the collection Λ. The instantiation of the policies is performed manually.

The policies are added in sequence to create a total order Γ'_Λ of the policies while the collection Λ is being defined; the prepares test is performed against the composition of all higher priority policies. Given the current state estimate, the complete list of policies is searched from highest priority to lowest priority for a policy that contains the current state; that policy is then executed. The experimental results demonstrate that the sequential composition technique repeatedly brought the ball to the overall goal in spite of perturbations, thus demonstrating the inherent robustness of the technique [21].

2.2.2 Applications of Sequential Composition

The idea of sequential composition has been used for several robotic systems. In this sub-section, we provide an overview of these related works.

Rizzi [105] uses sequential composition to simplify motion programming for the case of an idealized holonomic second-order dynamical robot, $\ddot{q} = u$ with both the control inputs u and configuration q in $\mathbb{R}^n$. The robot is subject to velocity and acceleration constraints in the form of Euclidean norm bounds. Sequential composition guarantees the overall behavior of the system by using control calculations specified over a convex polytope in the configuration space. Rizzi specifies the global motion by specifying a goal point for a single policy to lie within an overlapping convex polytope.

By specifying a goal point within the boundary of an overlapping polytope, the control policies over each polytope can be composed to move the idealized system through space into the domain of another switched policy defined over the adjacent polytope [105]. Since the goal point of one polytope is at rest, the policy of the first polytope trivially prepares the policy of the next polytope, provided the goal point is included in the interior of the next polytope. By chaining a series of overlapping polytopes, with appropriate goal points, the system is induced to move to an overall goal based on the sequential composition of the individual policies. Thus, if the initial state lies within the savable set of any policy in the deployment, the system is guaranteed to be brought to rest at the overall goal.

Quaid and Rizzi [103] extend this basic approach to more complicated bounds on acceleration and velocity found with planar motors. Safety is enforced in a dynamic multi-robot environment by only activating policies whose domain does not overlap with a polytope corresponding to a valid policy of another robot.

Yang and Lavalle [123] develop a similar approach to Rizzi [105], except their version is restricted to kinematic systems and does not consider input bounds. They define a potential function over a ball in configuration space. The balls are then distributed throughout configuration space using a graph-based sampling technique. The overlapping balls serve the same function as the overlapping polytopes in Rizzi's approach. In parallel work, Brock and Kavraki [17] use balls in workspace to define a connected tunnel through workspace, and then use a potential-based control policy to drive the system through the tunnel. Pathak and Agrawal [98] apply Brock and Kavraki's method to circular wheeled mobile robots by defining convergent switching control policies over circular regions of obstacle free space.

Sequential composition has also been used to control wheeled mobile robots. Kantor and Rizzi [55] define visual servoing control policies for a nonholonomic unicycle with a limited field of view. Their approach uses variable constraint control to define and parameterize individual control policies. Patel *et al.* [97] use sequential composition to define switching policies for a nonholonomic wheelchair that navigates through a doorway using visual servoing with a limited field of view. In both cases, the control policies are designed based on careful analysis of the system, its constraints, and the problem at hand; the deployment is carefully constructed by hand to enforce the prepares relationship.

Both Kantor's and Patel's 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 [97]. It is assumed that another control policy will become active after the vehicle passes through the doorway. This thesis expands upon this idea of *flow-through* policies, and formalizes some extensions to the basic sequential composition technique [25].

Lindemann and LaValle [83, 82, 84] follow our approach [25], and define flow-through vector fields over disjoint regions of free space. Their approach uses a different vector field generation technique, and is extended to cylindrical algebraic decompositions. The work is applied to point nonholonomic systems with bounded steering, but unbounded control inputs [84]. Their work is fundamentally an application of the sequential composition techniques advocated by this thesis. Their approach differs from this work in that their focus is on theoretical completeness and smoothness for simpler systems, while this thesis explicitly considers the interaction of robot body shape and input bounds, and discusses the planning aspects of the work in more detail.

Related Approaches There has been other work in control policy composition techniques that are not directly derived from sequential composition, even though the approaches are similar or in some cases an extension of the basic idea.

Branicky [14, 15] describes a *behavioral programming* technique in a hybrid systems formalism that is congruous with the sequential composition approach advocated in this thesis. Fast marching methods were used to define local policies for fully actuated systems. These local policies obey a prepares relationship, which allow them to be composed. Branicky focuses on the high-level view of the approach, and its application as a "middle-out" approach to planning. Instead of a top-down conventional planning approach, or a bottom-up reactive approach, these papers advocate a "middle-out" approach where there is a systematic way of predictably translating symbolic task descriptions into feedback control policies.

The "flow-through" policy design approach has been applied to piecewise affine systems using policies defined over simplices, which are triangles in the plane or pyramids in $\mathbb{R}^3$ [7, 44, 45, 107]. Habets *et al.* [44, 45, 46] define necessary and sufficient conditions for piecewise affine control policies that drive the system to a designated facet or set of facets in order to address the reachability problem for a hybrid system defined over a collection of simplices. This leads to the synthesis of switching control policies that flow from simplex to simplex toward an overall goal. While typically described in a hybrid systems formalism, these approaches are instances of sequential composition of local policies. Roszak and Broucke [107] provide new necessary and sufficient conditions for n-dimensional linear affine systems with $n-1$ inputs. These conditions reduce the general problem to a set of at most n linear programming problems.

Belta *et al.* [7] use piecewise affine control policies defined over simplices to synthesize a hybrid control policy. In their work, the focus is on planning in the discrete abstraction and not defining a global policy. Their approach defines a sequence of simplices that must be navigated, and then defines policies over each simplex that induces the desired closed-loop motion. These methods were originally developed for idealized holonomic systems, but may be applied to point nonholonomic systems using feedback linearization [7]. 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.

Frazzoli *et al.* [41] uses language similar to sequential composition to describe a *maneuver automaton*. There the focus is on defining the relationship between open-loop motion primitives, and specifying which motion primitives may be concatenated based on a prepares-style relationship; this contrasts with conventional dynamic programming methods which implicitly assume that all motions in the discrete set are always feasible. The maneuver automaton is used in an open-loop motion planning strategy that uses optimal control over the finite set of motion primitives to determine the values of certain free parameters. The previously selected maneuvers constraint the set of admissible maneuvers for the next step. By obeying the relationships specified in the maneuver automaton, aggressive maneuvers can be incorporated into the planning framework. The maneuver automaton is fundamentally an open-loop planning method, and does not directly specify domains or feedback control policies.

Several existing control paradigms use a more general form of policy composition. One example is variable structure control [31]. More closely related to sequential composition is work on "patchy vector fields" [2]. These approaches generally consider stabilization of nonlinear systems, and are not concerned with the planning across the composed policies, nor navigation problems.

2.3 Discrete Planning Methods

The works described in the previous section have been applied to solving a particular navigation problem, that is defining a global control policy that brings the system to a designated goal point. This does not not fully exploit the power of sequential composition. In this section, several discrete planning approaches are described at a high level; these approaches allow flexible symbolic planning on the prepares graph defined by policy composition. Chapter 3 describes the approaches in more detail, and discusses their pros and cons as they relate to specific applications of sequential composition.

The most basic approach, as followed by [21], is to define a total order of the policies. Here, each policy is assigned a priority ordering based on the prepares relationship. The ordering may be searched from highest priority to lowest, with the highest priority policy whose domain contains the current state being executed. This ordering may be constructed without explicitly constructing the entire prepares graph [21]. This approach is useful for bringing the system to a single overall goal.

Given the explicit prepares graph, which may very well be cyclic, the graph may be converted to a tree using basic graph search algorithms such as Dijkstra's algorithm or A^*[108]. Variants of A^* such as D^*, D^*-lite, and DD^*-lite are used to rapidly reorder policies when some policies become invalid due to additional information gathered during execution [114, 81, 88]. The D^* algorithm, originally developed for grid based path planning, facilitates fast re-planning based on changes in the graph cost structure [114]. A similar, but algorithmically different version, called D^*-lite has been applied to Markov Decision Processes, which are similar to graph structures but support non-deterministic outcomes [81]. The approach uses a Mini-max planning algorithm to plan for the best action considering the worst outcome of each transition.

Given an appropriate transition relation, like the prepares graph, recent work has focused on symbolic planning that satisfies high-level specifications, and tasks with sub-goals that temporally depend on each other. Model checking tools [23] have been used to generate sequences of policies whose invocation induce behaviors that satisfy high-level specifications given in linear temporal logic [36, 37, 61]. Linear temporal logic (LTL) [34] combines the standard logic operators 'NOT', 'AND', and 'OR' with temporal connectives such as 'NEXT', 'ALWAYS', 'EVENTUALLY', and 'UNTIL'. This allows specifications such as "visit region A after region B, but never region C." Using the prepares graph, a sequence of policies is defined that induces the correct behavior. This model checking-based planning produces a sequence of policies and not a global policy; the planning step must be rerun in the face of disturbances [36].

As a step toward a feedback-based temporal planning, Kress-Gazit *et al.* [68] use the automata synthesis algorithm of [102] to generate an automaton from the prepares graph. Using the policies described in Chapter 4, [68] generates an automaton that executes local feedback control policies in order to satisfy temporal specifications. As the system is an automaton, and not just an open loop sequence of policies, the system is able to respond to environmental information gathered during run time. The closed-loop behavior satisfies the high-level specifications encoded in a subset of LTL.

Chapter 3

Overview of Technical Approach

This chapter presents extensions to the basic sequential composition technique described in [21]. These extensions allow for more general policy types and prepares relationships, thereby increasing the flexibility of the approach. General definitions that help formalize the discussion are given; model specific details are withheld until Chapters 4 and 5. This chapter defines the requirements for “composable” policies that later guide the development of the feedback policies.

The chapter’s first section gives general definitions and notation used throughout the thesis. The second section briefly describes the use of *flow-through* policies. The chapter’s third section describes four policy requirements that are necessary for composable policies. The focus is on the high-level requirements; later chapters deal with the specific requirements of various robot models and specific policy designs. The fourth section discusses the basic approaches to planning in the space of control policies. This section serves to highlight the issues and trade-offs involved in several approaches by focusing on a simple example.

3.1 Basic Definitions

In order to develop our approach in a formal sense, we present a series of definitions. The robot is a single rigid body that moves on a bounded planar workspace $\mathcal{W} \subset \mathbb{R}^2$. The workspace is cluttered with a finite number of obstacles, which are represented as unions of convex regions O_i. The robot *configuration*, denoted q, is the minimum size set of variables required to specify the position of every point on the robot [22]. The number and type of variables that are required varies with different systems. The configuration space $\mathcal{Q}$ is the space of all possible configurations. Let $R(q) \subset \mathcal{W}$ denote the workspace area occupied by the robot at configuration q. A configuration is said to be collision free if, for all obstacles, $R(q) \bigcap O_i = \emptyset$. Thus, the obstacles in the environment constrain the set of admissible collision free robot configurations; the set of collision free configurations, or *free configuration space*, is denoted $\mathcal{Q}_{\text{free}} \subset \mathcal{Q}$, where

$$\mathcal{Q}_{\text{free}} = \left\{ q \in \mathcal{Q} \mid R(q) \bigcap \bigcup_i \mathcal{O}_i = \emptyset \right\}.$$

For a more in depth presentation of these definitions, refer to Appendix A.

The *state* of the system is the minimum information necessary to specify the motion of the system. For *kinematic*, or first-order systems, the state is simply the configuration q. By definition, the state of second-order systems is $\{q, \dot{q}\}$. Denote the state space of the system, whether kinematic or second-order, as $\mathcal{X}$. For kinematic systems, the free state space is simply $\mathcal{X}_{\text{free}} = \mathcal{Q}_{\text{free}}$. For second-order systems, the naive definition is $\mathcal{X}_{\text{free}} = T\mathcal{Q}_{\text{free}}$, the tangent bundle of the free configuration space. However, for systems with bounded accelerations, there are velocities at points in the free configuration space that make collision unavoidable; thus, $\mathcal{X}_{\text{free}} \subset T\mathcal{Q}_{\text{free}}$, with *regions of inevitable collision* excluded [40, 77].

The focus in this thesis is on defining memoryless state feedback control policies. A feedback policy, denoted $\Phi : \mathcal{X} \to \mathcal{U}$, is a mapping between the system state and its allowable control inputs, where $\mathcal{U}$ denotes the bounded input space. The policy generally has a limited domain $\mathscr{D}(\Phi) \subset \mathcal{X}$. We consider the general nonlinear equation of motion $\dot{x} = f(x, u)$ where $f : \mathcal{X} \times \mathcal{U} \to T\mathcal{X}$ for $x \in \mathcal{X}$ and $u \in \mathcal{U}$. Therefore, the closed loop system dynamics are given by $\dot{x} = f(x, \Phi(x))$, which defines a vector field $X : \mathscr{D}(\Phi) \to \mathcal{T}\mathcal{X}$ over the policy domain with $X = f \circ \Phi$.

Define the closed-loop *flow*, $X_t(x)$, of the vector field, where the parameter t specifies motion along integral curves of the vector field from initial condition x; that is, how the system moves as time evolves. Figure 3.1 provides a schematic picture of this definition. The flow has the following properties: $X_0(x) = x$ and $\frac{d}{dt}X_t(x) = X(X_t(x)) = f(X_t(x), \Phi(X_t(x)))$. Implicit in this definition is the assumption that composition of the policy and the equations of motion satisfy the requirements for the existence of a solution to the ordinary differential equation encoded in the vector field [8, 117]. For *convergent policies*, there is a designated goal set $\mathscr{G}(\Phi) \subset \mathscr{D}(\Phi)$ such that for all $x \in \mathscr{D}(\Phi)$, there exists $t \in [0, \infty)$ such that $X_t(x) \in \mathscr{G}(\Phi)$. First, unlike the prior work that considered only point goals, this definition allows for full dimensional goal sets. That is, the goal can be a neighborhood and not just a single goal point. Second, note that this definition does not require the state to remain in the goal set, which opens the door to *flow-through* policies.

3.2 Flow-through Policies

Where conventional sequential composition techniques [21, 105] used asymptotically stable state feedback control policies, this thesis allows what we term as *flow-through* policies.

Definition: Flow-through policy: A *flow-through policy* is a policy whose goal set is on the boundary of the domain. Invoking the policy will cause any initial state in the policy domain to exit the domain by passing through the goal set. The system does not stop in the goal set.

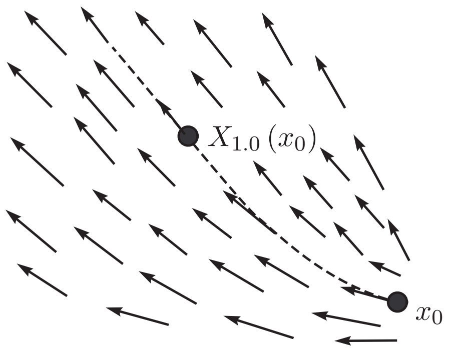


Figure 3.1: The vector field flow for an initial point x_0 is shown by the dotted line. The point indicated by $X_{1.0}(x_0)$ indicates the point obtained by flowing along the vector field from the initial point for one unit of time.

In other words, the policy eventually brings all states within its domain to the goal set, but the state does not necessarily remain in the goal set.

Flow through policies have several benefits. First, flow-through policies naturally encode certain high-level behaviors such as "leave this room via the doorway" [97]. Second, flow-through policies allow the policy designer to put off the implications of Brockett's theorem, which provides necessary conditions for the existence of smooth stabilizing control laws [18]. This gives the control designer more flexibility by allowing the local control policies to be smooth and time invariant, while relying on the switching strategy of the overarching hybrid control policy to reconcile the constraints of Brockett's theorem. Finally, flow-through policies give the control designer the freedom to match vector fields at policy boundaries.

The major drawback to flow-through policies is the more complicated prepares test. Whereas asymptotically stable policies have a trivial prepares test based on a single configuration at rest [105], flow-through policies require a prepares test based on the full state. For kinematic systems the test remains a configuration-based test; however, for second order systems the test is based on configuration and configuration velocity.

3.3 Composability Requirements for Local Policies

As this thesis seeks to compose local feedback control policies as illustrated in Figure 3.2, the question arises, "what are the necessary properties of policies that allow composition within the sequential composition framework?" In this section, four necessary properties are defined that make the feedback control policies composable. Here the focus is on basic requirements; Chapters 4 and 5 specialize these general requirements to the specific system models and control policies developed therein. To satisfy the requirements, a policy must be realizable on a given system; that is, a given policy may satisfy the requirements for one system but not another.

3.3.1 Collision Free

For a policy to be valid within the sequential composition framework, it must be safe. That is, over its entire domain it must be collision free. Consider Figure 3.3, the same policy domain can be safe

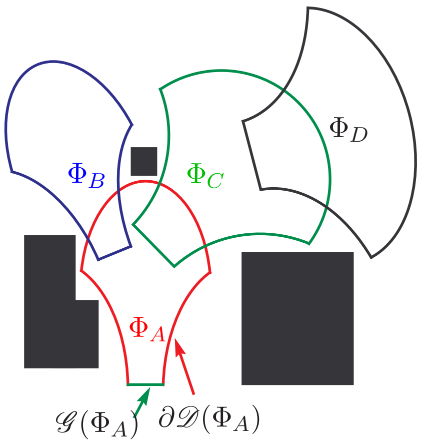
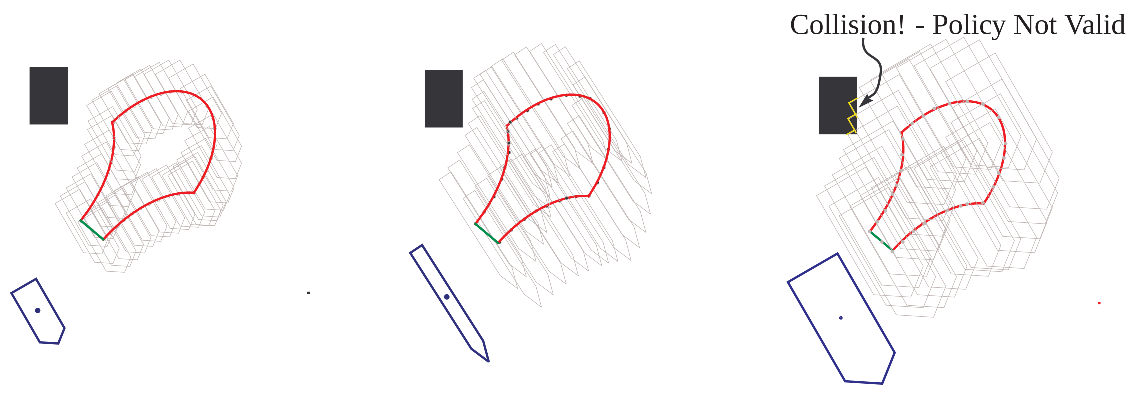


Figure 3.2: Composition of multiple policies. The two-dimensional iconic funnels represent the boundaries of multiple control policies; the goal set and domain boundary of Φ_A are labeled.



(a) (b) (c)

Figure 3.3: The policy domain must lie within the free state space of the vehicle. That is, all states in the domain must be free of collision for a specific vehicle size and shape. Here, the iconic funnels represent *the workspace projection of a slice* of a idealized policy domain. The policy is defined for the (x, y, θ) reference on a rigid body; the slice is taken at a fixed body orientation. Three different vehicle body sizes are shown as dark polygons; the light gray polygons shown in the figures represent the convolution of each vehicle body along the domain boundary. The vehicles shown at (a) and (b) are collision free at this orientation; vehicle (c) will collide for some states in this domain.

or unsafe based on the size and shape of the vehicle. It is therefore imperative that any proposed policy design approach have a tractable method of verifying the safety of an instantiated policy; that is, a policy whose free parameters are assigned specific values that give the policy domain a particular shape. Chapter 5 presents a method that maps points on the policy domain boundary to the full body extent in workspace. This allows for direct intersection tests with the workspace obstacles. This approach greatly simplifies the collision tests, relative to the alternative of constructing the free state space boundary by enlarging the obstacles, and excluding the regions of inevitable collision, and then testing for intersection with the policy domain.

3.3.2 Convergent in Finite Time

For each policy deployed within the sequential composition framework, it must be shown that the policy induces convergence to its goal set in finite time. In order to do discrete planning on the graph, it is necessary to guarantee that the desired discrete transition is eventually enabled. If the system does reach its goal set, and the two policies have a prepares relationship, then the transition is enabled. This contrasts with a system that stops inside the policy domain, or engages in a limit cycle, and thus never enters the goal set. In these cases, the system may not reach the domain of the next policy. Note that the behavior within the goal set is unrestricted; stopping within the goal set is allowed, as is limit cycle behavior.

Formally, this requirement is for all states $x \in \mathscr{D}(\Phi)$, there exists a finite time $T \in [0, \infty)$ at which $X_T(x) \in \mathscr{G}(\Phi)$ for $X = f \circ \Phi$. Obviously, for good performance, the maximum elapsed time taken for any point in the policy domain to reach the goal set should not be relatively large.

3.3.3 Conditionally Invariant

To guarantee that a policy is safe to invoke, the system must remain in the safe domain of the policy until the time at which the state enters the goal set. This property, termed *conditional invariance* [56], requires that once a policy becomes active for any state in the domain, the system state does not exit the domain of a policy except via the designated goal set as long as that policy remains active. Formally, the domain $\mathscr{D}(\Phi)$ is *conditionally-positive invariant* under the influence of policy Φ with goal set $\mathscr{G}(\Phi)$, if for all states $x \in \mathscr{D}(\Phi)$, $T \in [0, \infty)$ is the smallest time such that $X_T(x) \in \mathscr{G}(\Phi)$, and $X_t(x) \in \mathscr{D}(\Phi)$ for all $t \in [0, T]$. For flow-through policies, once the system state exits the domain via the goal set, the overall safety of the approach is dependent on switching to another safe policy. By composing only safe policies according to the prepares relationship, the overall hybrid control strategy is safe.

To enforce conditional invariance, the policy is subject to the necessary restriction that for each state on the domain boundary there exists a control input that induces an inward pointing velocity along the domain boundary, excluding the goal set for flow-through policies. For a state x on the domain boundary, and outward pointing normal $n(x)$, the induced velocity is constrained such that $n(x) \cdot \dot{x} < 0$; as shown in Figure 3.4. Given the equations of motion, this requirement can be rewritten $n(x) \cdot f(x, u) < 0$. Therefore, the minimal necessary condition for conditional positive invariance is that for all boundary points $x \in \partial\mathscr{D}(\Phi)$ the set $\{u \in \mathcal{U} \mid n(x) \cdot f(x, u) < 0\}$ is not empty. Given a bounded input set, this constraint has the effect of limiting the size and shape of the policy domain boundary. Chapter 5 shows how to map this boundary normal constraint for each state into a half-space constraint on the bounded input space; this enables a simple test for validity across the policy domain boundary.

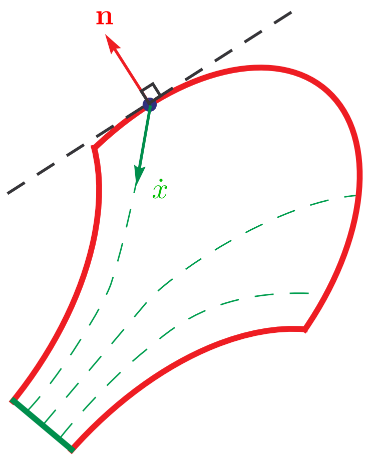


Figure 3.4: The integral curves of the closed-loop system can cross the domain boundary only at the goal set. Thus, for points along the domain boundary, the induced velocity must be inward pointing. That is $\dot{x}(t) \cdot \mathbf{n}(x(t)) < 0$, where $\mathbf{n}(x)$ is an outward pointing surface normal at $x(t)$ on the boundary and $\dot{x}(t)$ is the system velocity under the influence of the policy.

3.3.4 Efficient Inclusion Tests

As this hybrid control scheme is to be executed in real time, and the method is based on testing for transition from one domain to another, the system must have efficient tests for domain inclusion. The need for efficient tests, which may be calculated over many policies during a given control calculation, guides the choice of policy representation. Consider the states illustrated in Figure 3.5. State x_1 is in the domain of Φ_A but not Φ_B, state x_2 is in the domain of both, and state x_3 is outside the domain of both Φ_A and Φ_B. There are several possible domain representations for a given policy. Chapters 4 and 5 describe simple geometric shapes used to define the policy domains in this thesis.

In summary, policies that respect the system constraints, have simple inclusion tests, are completely contained in the free state space, are conditionally invariant, and have a vector field flow that converges to a well defined goal set in finite time may be deployed in this sequential composition framework. Given a specific system model, workspace obstacles, and bounded input set $\mathcal{U}$, these conditions limit the size and shape of the policy domains. Chapters 4 and 5 present the design of several *generic* policies that satisfy these requirements for a variety of system models. These policies serve as examples; any policy that satisfies the requirements given here may be incorporated into the hybrid planning and control framework described in this thesis.

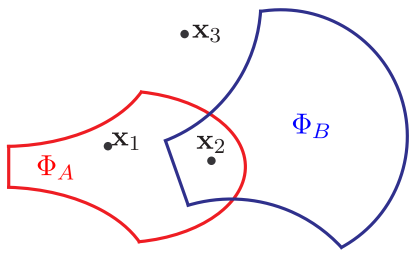


Figure 3.5: Successful operations depends on simple and efficient domain inclusion tests. Point $\mathbf{x}_1$ is in the domain of iconic policy A only, point $\mathbf{x}_2$ is the domain of both policies A and B, and point $\mathbf{x}_3$ is outside of both domains.

3.4 Extended Prepares Definition

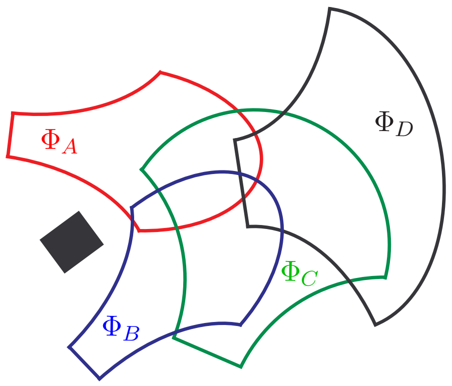
Sequential composition, as defined by [21], specifies a relationship among the policies. Finite time convergence coupled with conditional positive invariance induces a transition relation between a given policy domain and the domain of another policy which contains the goal set of the first policy. Recall from Section 2.2.1, that this pairwise relation between policies is called a *prepares* relationship, denoted $\Phi_j \succeq \Phi_i$ [21]. In order to induce a prepares relationship with Φ_i, the size of the goal set of Φ_j is necessarily limited because the goal set must be contained in the domain of Φ_i, which is a bounded region.

To provide more flexibility in planning, it is useful to consider larger goal sets not covered by a single policy domain. Therefore, we extend the conventional definition of *prepares* from a relation between two policies, to a relation between a policy and a set of policies.

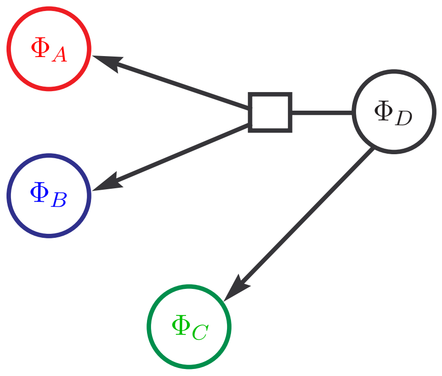
Definition: Prepares: A selected policy, Φ_i, *prepares* a set of policies if the goal set of the selected policy, $\mathscr{G}(\Phi_i)$, is contained in the union of the domains of the policies in the set. That is, $\Phi_i \succeq \{\Phi_j\}$ if $\mathscr{G}(\Phi_i) \subset \bigcup_j \mathscr{D}(\Phi_j)$.

An example is shown in Figure 3.6-a.

This added flexibility in the definition of prepares introduces added complexity to the discrete transition relation encoded in the prepares graph. Thus, the ability to define larger goal sets via the extended prepares definition bears a cost that is borne by the discrete planning. The flow along a vector field is mathematically determinate; therefore, from any initial condition within the single policy the flow will result in a transition specific policies in the union. On the other hand, the discrete transition relation encoded by the extended prepares relationship is an approximation. From the point of view of the discrete relationship, the transition is non-deterministic, and cannot be represented by a simple graph. The nondeterminacy can be represented as an *action* with multiple *outcomes*, as shown in Figure 3.6-b. This representation is common with Markov Decision Processes (MDP) [81, 122]. From the perspective of the discrete planning system, the choice of outcome is *externally* imposed on the discrete transition relation by the closed-loop system dynamics.



(a)



(b)

Figure 3.6: Example of the extended prepares definition using iconic funnels. a) Policy Φ_D prepares neither Φ_A nor Φ_B, but does prepare $\Phi_A \bigcup \Phi_B$ and Φ_C. b) Transition relation. From the discrete planning perspective, the choice of transition from Φ_D to Φ_A or Φ_B is non-deterministic; that is, it is imposed *externally* by the closed-loop system dynamics. The discrete planning method must account for either possibility.

So long as the planning system takes each possible outcome into account, the transition relation is valid for planning. We will abuse notation and continue to refer to the transition relation as a "prepares graph", even for the case of non-deterministic outcomes induced by the extended prepares relationship.

3.5 Policy Space Planning

This section highlights several issues involved in planning over a collection of control policies that satisfy the above requirements. Several approaches to defining switching strategies among the policies are discussed; these approaches make use of existing discrete planning techniques. The discussion extends the basic partial order approach presented in [21]. This thesis work enables advanced planning techniques to be applied to systems with more complex dynamics and constraints.

The planning takes place in the space of instantiated local feedback control policies. Recall from Section 2.2.1, that the *palette* is a collection of generic policies; that is policies with free parameters. Policies are *instantiated* by assigning specific parameter values to a generic policy chosen from the palette. Given a collection of instantiated policies, which we call a *suite* of policies, planning involves defining a switching strategy among those policies to address a given task. The suite of policies and the switching strategy is called a *deployment*.

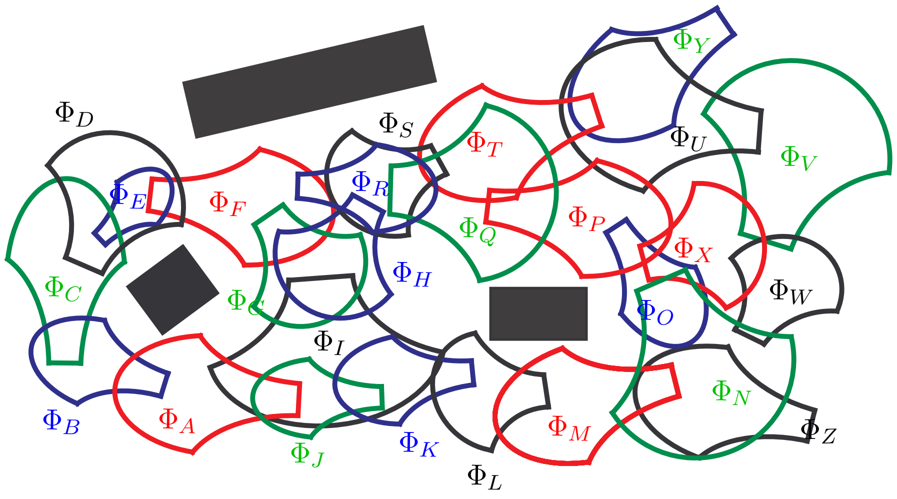
To illustrate the types of planning that are possible on the discrete prepares graph, consider the "toy" example shown in Figure 3.7. Here 26 policies are instantiated over the workspace with three obstacles; these policies make up the suite $\Lambda = \{\Phi_A, \ldots, \Phi_Z\}$. The policy domains are shown in Figure 3.7-a, and the associated prepares graph, Γ_Λ, is shown in Figure 3.7-b. The figure shows one example of the extended prepares definition with $\Phi_V \succeq \{\Phi_X, \Phi_W\}$. Two policies, Φ_Y and Φ_Z, do not prepare any others. As is generally the case, the prepares graph is cyclic.

In the discussion that follows, some planning methods use a cost associated with each transition to facilitate policy ordering. In this example, a heuristic cost has been assigned to each edge in the graph shown in Figure 3.7-b. This section does not address how the costs are assigned.

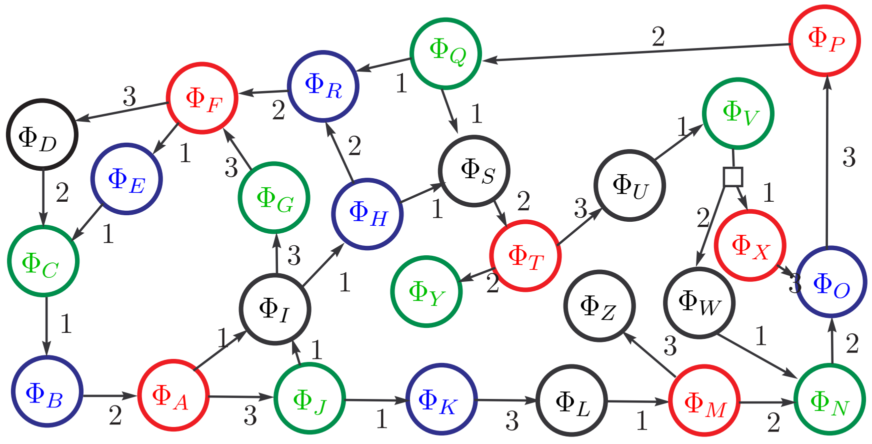
In the subsequent discussion, the switching strategies are often modeled as finite automata, with nodes and transitions between nodes. For each node there is an associated policy that is executed upon transition into the node. Initially, there is a one-to-one correspondence between nodes and policies; later, as temporal dependencies are incorporated, the automata will have multiple nodes that map to a single policy. For this reason, this discussion will enforce a distinction between a node in the automata representation and its associated control policy. Transitions represent switches between policies governed by the continuous state evolution into the domain of a policy associated with a child node; that is, the transitions are enabled with the state enters the domain of a policy associated with a child node.

For transitions based on the extended prepares definition, the transition will be associated with a non-deterministic outcome, and the finite automata may more properly be modeled as a Markov Decision process. Here the transition is an action, with multiple outcomes. The action represents a desired transition to a set of nodes associated with the set of policies in the extended prepares. The transition is enables as soon as the system state enters the domain of any policy associated with an outcome node. This thesis will use the term finite automata to encompass the non-deterministic outcomes.

For simple navigation to an overall goal, we assume the existence of a single stabilizing policy, which will be referred to as the "goal policy." The node in the automaton that is associated with the goal policy will be the termed the "goal node." In some cases, where the system has a known initial



(a)



Φ_A Φ_B Φ_C Φ_D Φ_E Φ_F Φ_G Φ_H Φ_I Φ_J Φ_K Φ_L Φ_M Φ_N Φ_O Φ_P Φ_Q Φ_R Φ_S Φ_T Φ_U Φ_V Φ_W Φ_X Φ_Y Φ_Z
1 2 3

(b)

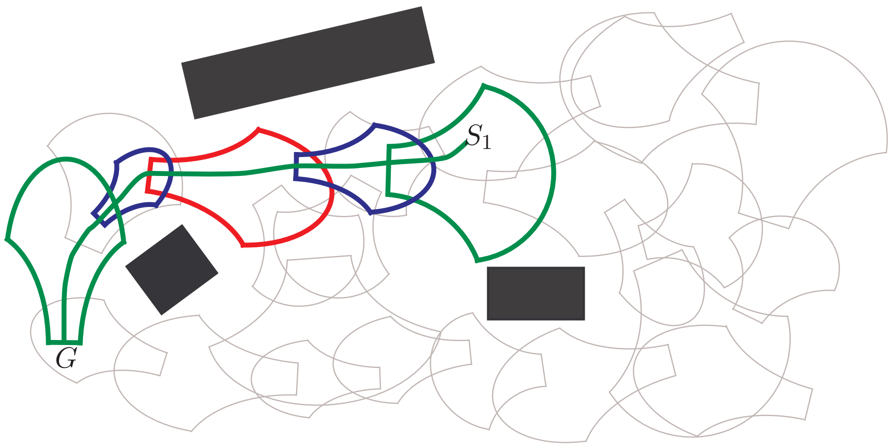
Figure 3.7: Given the collection of iconic policies Λ shown in (a), the discrete transition relation Γ_Λ shown in (b) encodes the "prepares" relationship between policies. Thus the continuous behavior of the system is abstracted as a discrete set of transitions between policy domains. Each transition shows an associated heuristic cost that may be used in planning.

condition, references to the "initial policy" and "initial node" are made as appropriate, where the initial state is contained in the domain of the initial policy.

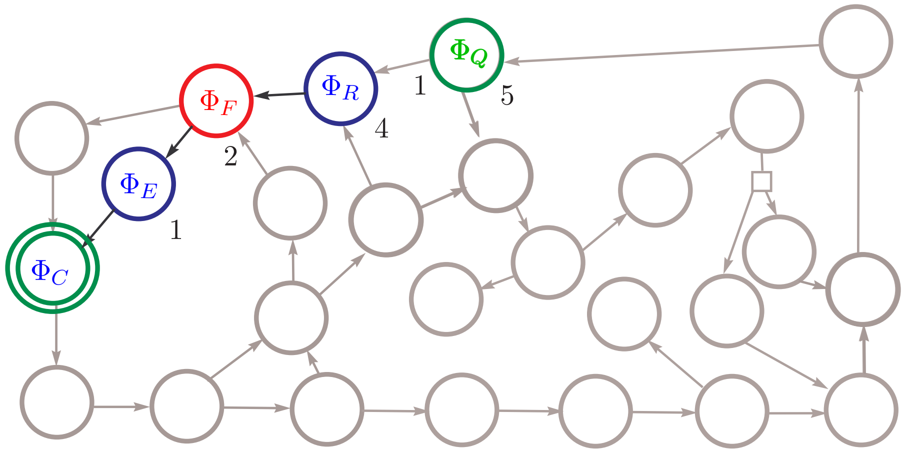
3.5.1 Sequence-based Planning

The most basic type of planning in the space of policies is to define a sequence of policies that drive the system to the goal. This is accomplished by determining a sequence of nodes, also called a "walk"[1] , through graph Γ_Λ that connects the goal node with the initial node. In the toy example shown in Figure 3.8, Φ_C is designated as the goal policy, while Φ_Q is the initial policy; that is, Φ_Q's domain contains the initial state. The path induced by invoking policies along the ordered walk

[1]Many modern texts refer to this as a "path." We chose to reserve the term "path" to refer to the sequence of configurations of the system, and use the alternate term "walk" instead [121]



(a)



(b)

Figure 3.8: Given a goal, G, corresponding to the goal set of policy Φ_C in Figure 3.7, and an initial condition contained in the domain of Φ_Q, the discrete planning system can specify a sequence of policies to invoke by searching the prepares graph. Figure 3.8-a shows a path through the workspace that may be induced by executing the policies according to the sequence shown in (b). The numbers below each node denote the cumulative cost based on the edge costs in Figure 3.7.

$\Phi_Q \rightarrow \Phi_R \rightarrow \Phi_F \rightarrow \Phi_E \rightarrow \Phi_C$ flows from the initial state to the goal set of Φ_C. The workspace path is never explicitly defined, but is induced by the closed loop dynamics of the system. We note that the choice of Φ_E over Φ_D was made based on the heuristic costs; other criteria, such as robustness, might lead to Φ_D being chosen.

Sequential planning has several advantages over conventional path planning. Instead of the "thin" path through configuration space defined by many path planning methods, this graph walk corresponds to a "thick" path that corresponds to the policy domains. Minor disturbances do not require replanning provided the perturbation remains in the current policy's domain. As the safe domains are explicitly defined, the system can readily test to see if replanning is necessary.

Numerous graph planning tools, such as Dijkstra's algorithm, A^*, and D^* variants, are available for determining a valid walk [108]. The planning methods can determine an optimal graph walk based on heuristic costs assigned to the edges connecting nodes. The induced path will not necessarily be the optimal path as the heuristic costs assigned to the prepares graph represent some average cost of activating a policy over its entire domain, and not a cost specific to the induced trajectory. For large perturbations, or if additional information received during execution invalidates certain policies, approaches such as D^* allow for rapid replanning on the discrete graph [114, 81, 88]. As replanning occurs on the discrete graph it has the potential to be much faster than for conventional grid-based approaches.

In addition to basic graph walks, temporal specifications can be satisfied by adding discrete states to an automaton that specifies the allowable transitions in the prepares graph [36, 37, 61]. The search problem in the automaton becomes exponential in the number of temporal specifications, therefore model checking approaches are commonly used. Most approaches considering temporal specifications in use at this time do not consider heuristic costs, and only consider the discrete transitions. The sequence must be re-planned if the temporal specification changes [36].

Sequence-based approaches have two fundamental drawbacks. First, the overall domain is smaller than for the order-based approaches described next. As these sequences are open loop walks, the walk must be re-planned if significant disturbances take the system out of the domains of policies in the sequence; thus, the sequence-based hybrid system loses some robustness to disturbance. While the policy domains represent a "thicker" path, the domain is still not "global" because not all policies are used. Second, as the policies must be executed in sequence, the system cannot take advantage of opportunistic jumps to higher priority policies. Thus, the sequences restrict the system to invoking the policies according to discrete transitions, which are at best a coarse approximation of the closed loop behavior.

3.5.2 Order-based Planning

For navigation to a single goal, order-based approaches offer a "plan once, execute many times" strategy. The decision regarding which specific policy is executed is deferred until run time; policies whose domains contain the current state are executed according the to predefined ordering. Given any initial state in the domain of any policy within the ordering, the hybrid policy will bring the system to the designated goal; that is, hybrid policy approximates the desired global policy. Fundamentally, order-based approaches have a larger domain than a specific policy sequence that solves a single navigation problem.

With order-based approaches the entire collection of policies is considered; therefore, the order-based approach is more robust than the sequence-based approach where only some policies are used in the deployment. In the face of a disturbance, as long as the system state remains in the domain of at least one policy in the ordering, the execution continues.

In this subsection three types of orderings are considered: totally ordered lists, finite automata, and partial orders. We begin with the more concrete examples of ordered lists and finite automata, and then describe the more abstract partial order.

By considering heuristic costs assigned to the prepares graph, a discrete planner can order of all the policies in the suite based on the cumulative cost to goal. In general, the conversion from a generally cyclic transition relation encoded by the prepares graph, to an acyclic transition relation with a single goal is not unique. While the choice between some transitions may remain arbitrary, the cost-based ordering provides a systematic approach defining a policy switching strategy.

Dijkstra's algorithm, A^*, D^*, and other variants may be used to convert the cyclic prepares graph into an acyclic directed transition relation with cumulative costs assigned to each node. Each node in this tree-like structure maps to a node in the prepares graph, and hence to a particular policy. Each transition maps to one edge in the prepares graph, such that the transition points to a node, or nodes in case of extended prepares, with the minimum cost to goal. The transitions between associated nodes must have an associated edge in the prepares graph. In symbolic planning terms, the transitions in this tree-like structure encode a "policy" for each node in the prepares graph; that is, at a given node with associated policy, the transition points to a node associated with a prepared policy that represents the best choice to minimize cost. This thesis reserves the term policy to mean "continuous feedback control policy", and will use the term "action" to denote the desired transition.

D^* and its variants allow for rapid reordering of the tree-like structure as new information is obtained that changes the cost structure of the graph. For example, if a policy becomes invalid based on a newly discovered obstacle, D* allows the relevant nodes of the ordering to be rearranged without required a complete re-plan.

Figure 3.9 shows the tree-like representation with cumulative node costs assigned; the acyclic transition relation Γ'_Λ is constructed from the prepares graph from Figure 3.7. Φ_C is the goal node. Note that in constructing the transition relation in Figure 3.9, Φ_V is at lower priority than either Φ_X or Φ_W. This is required because of the external choice imposed upon the extended prepares relationship. Also, note that the designated goal cannot be reached from Φ_Y and Φ_Z; therefore, these policies are removed from Γ'_Λ and the domains of these policies are not included in the over all hybrid policy domain.

Given the assigned costs to each node in the tree-like structure, the ordering can be executed as a finite automata model that corresponds to the tree-like structure, or as a totally ordered list based on the assigned costs. We begin with the discussion of the list-based total order.

One execution strategy is to convert the tree-like structure to an ordered list of policies based on the cumulative costs assigned to each node. While there may be some arbitrary choice involved if nodes have the same cumulative costs, the list results in a total order of the policies. If the domain inclusion tests are relatively fast, and the number of policies relatively small, then it is possible to search an ordered list of policies from highest to lowest priority at each controller time step[2], and execute the first policy whose domain contains the current state.

Consider the following examples shown in Figure 3.10. State S_1 is contained in the domains of both Φ_Q and Φ_T; Φ_Q is executed based on the total ordering induced by the costs shown in Figure 3.9. Φ_E is assigned higher priority than Φ_D [3]. During execution, a disturbance causes the trajectory to exit the domain of Φ_E as shown in Figure 3.10. As Φ_D is also a valid policy, a search over the total order chooses Φ_D and continues execution on its way to the overall goal. Thus, the system can automatically react to disturbances using the total order. As long as the disturbances

[2]The policies are designed as continuous policies, but execution of the hybrid controller on a computer introduces a discrete time step.

[3]Graphs are often drawn with the root node at the top, therefore, lower priority policies are on lower layers. Here, the graphs are drawn horizontally with the root on the left to save space. We will retain the higher/lower terminology.

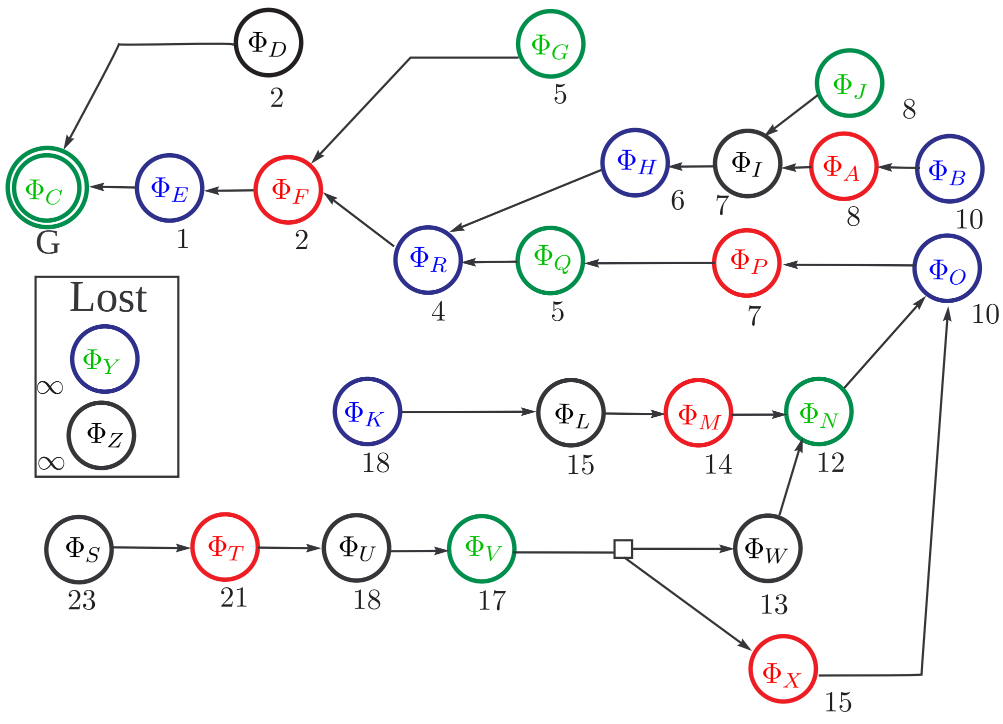


Figure 3.9: The prepares graph Γ_Λ is converted from a cyclic graph to an acyclic tree-like structure Γ'_Λ. This structure is not a true tree due to the non-deterministic transitions encoded by the extended prepares relation. The cumulative path cost is shown at each node.

are infrequent relative to the overall convergence rate, this method has been proven robust [21]. The states S_2 and S_3 demonstrate the non-determinism inherent in the extended prepares definition; the induced trajectories from both states pass through Φ_V, but take different routes through the graph and workspace because of different policies that are invoked as the induced trajectories enter different domains.

The benefit of the list-based total order approach is that it allows opportunistic switching; if a disturbance or induced trajectory takes the system state into the domain of a higher priority policy, then that policy can be executed immediately, without waiting for transitions through intermediate child nodes. For example, consider the paths starting at S_4 in Figure 3.10. The policies Φ_G and Φ_H overlap and are both prepared by Φ_I. In Figure 3.9, Φ_I is assigned to transition to Φ_H based on the higher cost of the edge from Φ_I to Φ_G. The path labeled 'a' illustrates the path induced by following the this policy switching strategy. On the other hand, a list-based execution strategy allows the opportunistic switch to node Φ_G as soon as the state enters the domain of Φ_G; this switch is based on the lower cumulative node cost at Φ_G. Path 'b' represents the path taken by using opportunistic switching. In the case of opportunistic switching, Γ'_Λ does not represent a guaranteed transition relation. The list-based method cannot guarantee that a node will not be skipped, only that the system will inevitably transition to some higher priority policy.

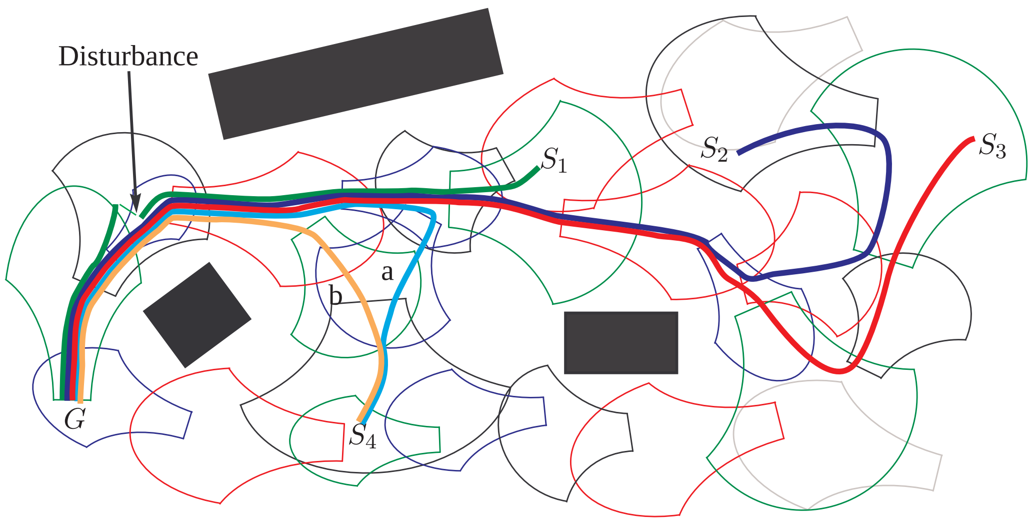


Figure 3.10: Given the ordering from Figure 3.9, this figure shows the execution for several different initial conditions, labeled S_i. The thick lines represent the integral curves induced by the local policies. The lines labeled ‘a’ and ‘b’ represent two different flows induced by different policy switching strategies. The line from S_4 suffers a disturbance, that is captured by another policy domain. The flow lines terminate at the overall goal set G.

The ordering encoded in the tree-like transition relation can also be executed as a finite automaton. In this mode, the software governing policy execution monitors the current node, which is stored as an additional internal state variable. The inclusion tests only need to check the children and current policy during runtime. The transitions to a new policy domain may occur as soon as the state enters the domain of a policy associated with a child node.

The chief benefit of the finite automaton-based execution strategy is faster execution time, because fewer inclusion tests are required. This is because the testing of policy domains is limited to the current node and its children in the finite automaton. The approach is advised if the inclusion tests are relatively slow, or the number of policies relatively high. If a disturbance takes the state outside the domain of both the current policy and its children, then the system should revert to a search over the entire tree to preserve robustness. In this case, depending on required search time, it may be prudent to pause the vehicle motion while conducting the total order search.

The finite automata approach does not allow opportunistic switching as readily as the list-based total order. Trajectory ‘a’ flowing from state S_4 represents the trajectory induced by following the actions specified by the finite automaton execution strategy. A limited form opportunistic switching can be allowed by checking nodes associated with policies that are prepared by the current policy, but do not represent the “best” action. The opportunistic switch can be enabled for any “prepared node” that has a lower cumulative cost assigned.

Another possible approach to allow opportunistic switching is run an “anytime” search algorithm in parallel with the automaton-based execution strategy; the anytime algorithm can seek opportunistic switches in the spare computing cycles between control updates [80]. The anytime algorithm should restart the search process when its cumulative cost equals that of the current node.

This approach combines the benefits of the finite automata-based execution strategy with the benefit of opportunistic switching.

One potential downside to opportunistic switching is due to the lumped approximation of the discrete transitions. While the node cost may be lower, the state may enter the policy domain in a region that requires higher than average costs; therefore, it may be prudent to evaluate a transition cost based on current state before allowing opportunistic switching. The control designer may also wish to guarantee transitions in a predictable manner. The choice of whether to allow this limited opportunistic switching could be made on a node-by-node basis.

The final order-based approach is a *partial order*. In theory, all that is needed for an order-based approach is a function that prioritizes the set of policies whose domain contains a given state and are valid for a given navigation task. This function is called a *partial order*. Consider state S_1 in Figure 3.10 again; state S_1 is contained in the domains of both Φ_Q and Φ_T. A partial order will chose one over the other. Using a partial order in place of a total order requires knowledge that the overall goal is reachable via a given policy, otherwise the policy is invalid for the task. Thus, while the partial order itself requires only knowledge of the collection of policy domains containing the current state, evaluating that the goal is reachable from a particular node requires global knowledge of the prepares graph.

One drawback to the order-based approaches are that they are limited to addressing a single navigation task. That is, order-based approaches are best suited for navigation to a specific goal, governed by a specific policy. Order-based approaches by themselves are ill suited for tasks that require visiting multiple points, or whose ultimate goal depends on information gathered a run time. A higher-level executive can re-order the policies to induce changes in navigation behavior, or switch between multiple orderings if needed. Another alternative is exploit the finite automata more fully, to generate hybrid policies that satisfy the high level specifications automatically.

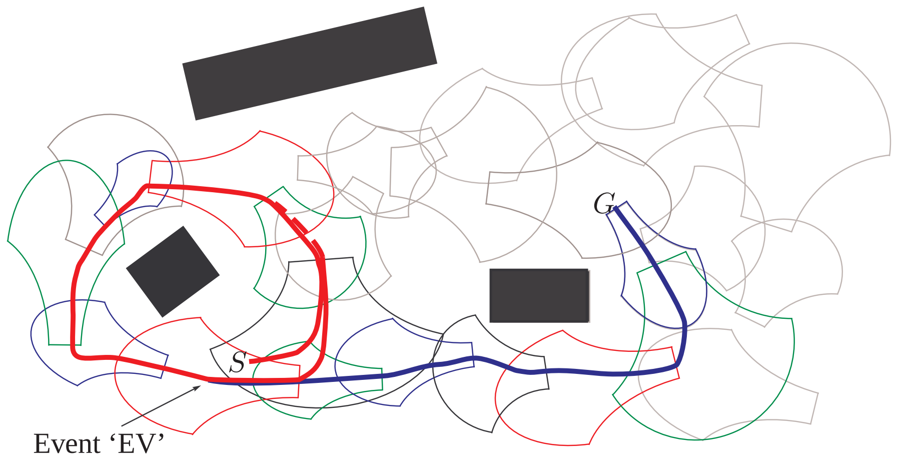
3.5.3 Automata-based Planning

In order to plan for higher-level task specification, including those with the need to respond to events or respect temporal restrictions, a more flexible planning approach is needed. Sequence-based approaches require replanning if the system needs to react to an event, and order-based approaches by themselves are only suitable for single tasks. To address this issue, recent work has focused on automatically synthesizing automata from a prepares graph [61, 68].

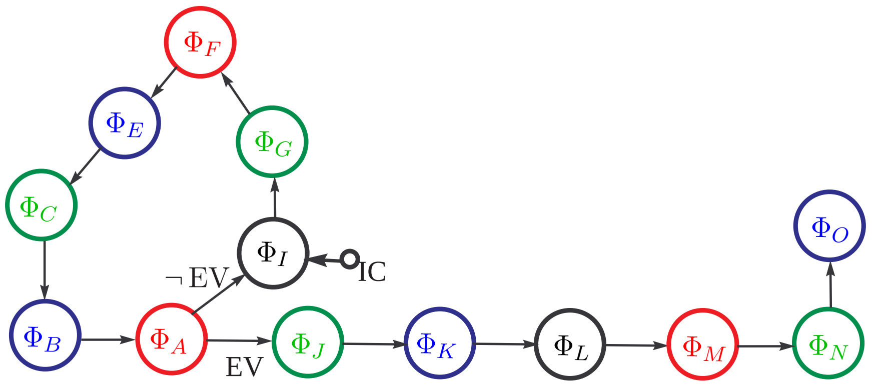
Combining policy composition with automata synthesis leverages the strengths of control theoretic and computer science approaches. Control theoretic approaches offer provable guarantees over local domains; unfortunately, the control design requires a low-level specification of the task. In contrast, discrete planning advances from computer science offer the ability to specify more general behaviors, which may react to environmental changes, and generate verifiable solutions at the discrete level; discrete planning lacks the continuous guarantees and robustness offered by feedback. Synthesizing an automaton that governs the execution of the local feedback policies provides the benefits of both feedback and discrete planning, while mitigating the drawbacks. These automata synthesis tools specify behaviors in terms of linear temporal logic (LTL) operations on the prepares graph nodes. LTL combines the standard Boolean logic operators, such as ‘AND’,‘OR’, and ‘NOT’, with temporal operators such as ’ALWAYS’ and ’NEXT’ [34].

Kress-Gazit *et al.* [68] have developed an automaton synthesis tool that use specifications encoded in a subset of the full LTL that describe behaviors on the prepares graph generated by the work in this thesis. The approach allows both discrete inputs and discrete outputs to be specified. The discrete inputs are sensed by the robot, and the discrete outputs trigger actions, such as sound an alarm. This allows the system to change high-level behavior-based on discrete events, which allows the system to react to environmental changes in a guaranteed manner. Given a specification and prepares graph, the synthesis process either extracts an automaton that satisfies the specification, or shows that the specification is not realizable on the prepares graph. Transitions in the automaton are governed by the transitions between policy domains and the discrete events sensed by the robot [24]. Thus, the combination of automata and continuous feedback control policies allows high-level specifications to be satisfied by executing the continuous feedback control policies. The work in this thesis has enabled these approaches to be applied to more complex systems.

Returning to the example of Figure 3.7, we wish to specify that the robot patrol the lower left obstacle until an event is seen, and then goes to a particular station. The first part, "patrol the lower left obstacle by visiting areas 'F' and 'A', until an event 'EV' at 'A' is seen." After seeing the event, "go to 'O', sound an and stay put." Figure 3.11 shows an example of an automaton whose execution satisfies this behavior.



(a)



(b)

Figure 3.11: Automata-based planning allows for the system to react to local conditions while satisfying a given specification. Figure (a) shows portions of the path taken while satisfying the automaton shown in (b).

There are several down sides to the current automata synthesis approaches. First, these current approaches do not consider transition costs. Thus, a heuristic cost associated with invoking a more complex policy is not taken into consideration by the current synthesis tools. Second, because the automaton does not use all of the policies in the collection, some robustness to disturbance is lost. In Chapter 6, we present one approach to combining the automata synthesis with order-based planning to improve the robustness of the automaton.

3.6 Summary

This chapter has introduced two extensions to the basic sequential composition technique. First, flow-through policies are introduced, which allow the system to encode natural behaviors for non-holonomic systems. Second, the prepares definition is extended to allow a policy to prepare a set of policies. This extension provides more flexibility in instantiating the local policies, but complicates the discrete planning. The impact of this change on the planning is discussed.

The chapter discusses the properties that are necessary for composable policies. In addition to policies that that respect the system constraints, the policy domains must be completely contained in the free state space and conditionally invariant. The vector field flow induced by the closed-loop policy must converge to a well defined goal set in finite time. Additionally, the policies should have simple and efficient inclusion tests to allow the approach to be executed in real time. Any policy with these properties can be deployed in our hybrid control framework.

The chapter discusses approaches to planning in the space of instantiated policies. Three basic approaches are presented: sequence-based, order-based, and automata-based. A "toy" example highlights the differences between the approaches. Section 3.5 discusses the relative strengths and weaknesses of each approach. In general, order-based and automata-based approaches are preferred over the sequence-based approaches for reasons of robustness and flexibility.

3.7 Glossary

As a convenience, this section reiterates several definitions given in the text of earlier chapters.

Definition: Policy: A policy is a mapping from state to the bounded input space; that is, $\Phi : \mathcal{X} \rightarrow \mathcal{U}$. In this thesis, the term *policy* is shorthand for *continuous feedback control policy*

Definition: Domain: The domain of a policy, denoted $\mathscr{D}(\Phi)$, is the region over which the state to input mapping is valid. $\mathscr{D}(\Phi) \subset \mathcal{X}$.

Definition: Flow: The *flow* of the system under the influence of a given policy, denoted X_t, is the family of integral curves induced by the closed loop dynamics of the system where $X = f \circ \Phi$. Assigning a specific initial condition identifies a specific integral curve, or flow line. Specifying an initial state and elapsed time t identifies a specific point in the state space. That is, starting from $x\,(0)$ and flowing along the integral curve passing through $x\,(0)$ for t seconds brings the system to $x\,(t) = X_t\,(x_0)$.

Definition: Goal Set: The goal set of a policy, denoted $\mathscr{G}(\Phi)$, is a subset of the domain, that is $\mathscr{G}(\Phi) \subset \mathscr{D}(\Phi)$, whereby invoking the policy over the domain will induce motion that flows to the goal set. Thus,

$$\forall\, \mathbf{x}_0 \in \mathscr{D}(\Phi) \;\; \exists t \text{ s.t. } X_t\,(\mathbf{x}_0) \in \mathscr{G}(CP)\ .$$

Definition: Generic Policy: A *generic policy* is a policy defined by free parameters. The parameter values determine the size, shape, and location of the policy domain, as well as the state-to-input mapping of the policy.

Definition: Palette: A *palette* is a collection of available generic policies.

Definition: Instantiated Policy: An *instantiated policy* is a policy with assigned parameter values. That is, it is a generic policy whose free parameter values have been assigned; therefore, the policy domain and mapping is determined.

Definition: Suite: A *suite* is a collection of instantiated policies available for planning.

Definition: Prepares Graph: The *prepares graph* encodes the prepares relationships between policies in the suite.

Definition: Deployment: A *deployment* is a suite of policies and a defined switching strategy for executing the policies. Given a suite of policies and the prepares graph, the planning system generates a deployment.

Chapter 4

Application of Policy Composition to Fully Actuated Systems

This chapter demonstrates the approach to policy composition and planning on fully actuated dynamical systems. Since the system is an idealized point with fully actuated dynamics, it has no orientation and is free of nonholonomic constraints. For these idealized systems, the configuration and workspaces are equivalent, and the equations of motion are given by either $\dot{q} = u$ or $\ddot{q} = u$, where $q, u \in \mathbb{R}^n$. We define a class of flow-through policies for these relative simple systems to demonstrates the principles behind the hybrid control approach outlined in Chapter 3.

The chapter begins with a discussion of a particular policy design approach that satisfies the requirements from Chapter 3; proofs are given in Appendix B. Designs for both kinematic and second order systems are presented. Throughout the chapter, examples are given to illustrate behavior of each policy. The chapter presents several examples to demonstrate the variety of planning techniques discussed in Chapter 3. This last section concludes with an approach to automating the deployment of policies for second order systems.

4.1 Local Control Policy Design for Fully Actuated Systems

As the basic navigation task is defined in the workspace, our approach defines cells within the workspace; by design, composable feedback control policies are relatively easy to define over each cell. For simplicity, the examples in this thesis are restricted to cells that are convex polygons in an $\mathbb{R}^2$ workspace. While this presentation focuses on convex polygons, the ideas are directly applicable to convex polytopes in $\mathbb{R}^n$. In most of the policy designs, the extension to arbitrary convex regions is obvious. Although beyond the scope of this thesis, many of the techniques can naturally be extended to regions that are homeomorphic to balls in $\mathbb{R}^n$. The approach assumes that a finite convex decomposition of the free configuration space is given. In practice, such a decomposition may be specified by a map or floor plan, or calculated automatically for low dimensional spaces [58].

The local policies defined over each cell are based on local potential functions, which are used to define one of two configuration-based velocity reference vector fields over each cell. We then design a control law for each system model that causes convergence to the reference vector fields. For the first type of vector field, the integral curves emanating from all interior points cross the cell boundary within a specified region; this type is termed a *flow-through* vector field. Integral curves of the second type converge to a designated goal point in the interior of the cell; this type is termed a *convergent* vector field. The remainder of this section provides details for this policy design approach for both kinematic and second order systems.

4.1.1 Vector Field Design

The vector fields used in the policy design approach are based on gradients of a potential function. The flow-through and convergent vector fields use two different types of potential functions. In both cases, it is generally easier to calculate a valid potential function over a unit n-ball than over a general convex polytope. For this reason, we define a mapping from the cell to the n-ball, and use this to "pull back" a potential function from the ball to the polytope, and calculate the gradient of the pulled back potential. Let $\varphi : \mathcal{P} \to \mathcal{B}$ define a mapping from an arbitrary cell, $\mathcal{P}$, to a unit n-ball, $\mathcal{B}$, centered at the origin. The mapping maps interiors to interiors, and boundaries to boundaries. For a convex polytope, Appendix B defines the mapping based on a scaled radial retraction as

$$\varphi(q) = \frac{q}{\|q\| + \beta(q)},$$

where $q \in \mathcal{P}$, and $\beta(q) : \mathcal{P} \to \mathbb{R}$ is a scaling factor based on the distance to the cell facets.

Given γ_b, a potential function defined over the n-ball, the potential function in the arbitrary cell is given as

$$\gamma = \gamma_b \circ \varphi\,. \tag{4.1}$$

By the chain rule, the potential gradient $D_q\gamma = D_{\varphi(q)}\gamma_b \cdot D_q\varphi$ for $q_b = \varphi(q)$, which lies on the interior of the n-ball by construction. The gradient, $D_q\gamma$, will be well defined if the Jacobian of the mapping, φ, is full rank over the interior of the cell; Appendix B proves that our mapping for convex polytopes is full rank on the interior.

We now discuss the particulars of the two vector field designs using the potential function mapping approach.

Flow-through Vector Field Design

The flow along a flow-through vector field exits a given configuration space cell through a designated boundary region termed the *outlet zone*. The remaining boundary is termed the *inlet zone*.

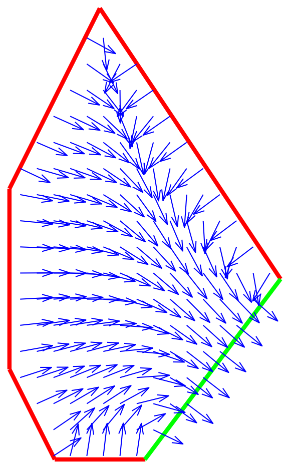


Figure 4.1: Vector field for a flow-through policy.

Figure 4.1 shows a typical flow-through vector field for a convex polytope, where one facet is defined as the outlet zone and the remaining facets are designated inlet zones.

To generate a flow-through vector field over the cell as the negative gradient of a potential function, the potential function γ must be free of local minima on the cell interior. A harmonic function, which is a solution to Laplace's equation, provides a natural way to define a potential without spurious local minima. The design approach defines a constant minimum potential along the outlet zone; a constant maximum potential is defined along the inlet zone. While a solution over an arbitrary convex polytope is not available, the solution to Laplace's equation over the n-ball is a computable integral equation.

On the unit disk with piecewise continuous boundary conditions the solution exists in closed form [35, 106]. Let $q_d = \varphi(q) = (x_d, y_d)$ be the Cartesian coordinates of a point in disk mapped from a point in the polygon. For the disk, the most natural representation is in polar coordinates, so define

$$\begin{aligned} \rho &= \sqrt{x_d^2 + y_d^2}\,, \\ \theta &= \operatorname{atan2}(y_d, x_d)\ . \end{aligned}$$

If the boundary condition is 0 along the outlet zone, and 1 along the inlet zone, and the outlet zone crosses the negative x-axis, the solution to Laplace's equation on the unit disk in $\mathbb{R}^2$ is

$$\begin{aligned} \gamma_b(\rho,\theta) = \frac{\alpha_1 - \alpha_0}{2\,\pi} &+ \frac{1}{\pi}\tan^{-1}\left(\frac{\rho\sin(\alpha_1-\theta)}{1-\rho\cos(\alpha_1-\theta)}\right) \\ &- \frac{1}{\pi}\tan^{-1}\left(\frac{\rho\sin(\alpha_0-\theta)}{1-\rho\cos(\alpha_0-\theta)}\right)\ , \end{aligned} \tag{4.2}$$

where α_i denote the angle coordinates of the vertices of the outlet face. See Appendix B for details.

Figure 4.2 shows an example of the mapping and potential used to generate the vector field in Figure 4.1. Although the potential field, γ, over the cell resulting from $\gamma = \gamma_b \circ \varphi$ no longer obeys Laplace's equation, γ is free of spurious minima as proven by

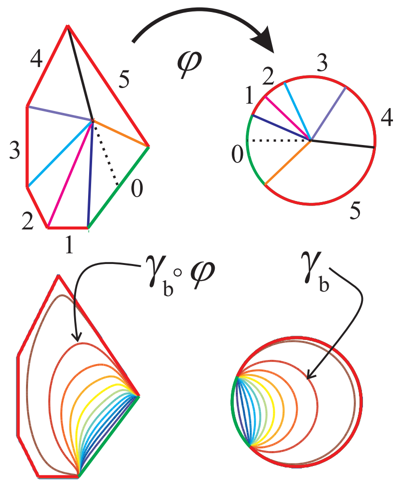


Figure 4.2: Mapping from polygon to unit disk. The contour plot on the left shows level sets of the pullback $\gamma_b \circ \varphi$ on the polygon; the contour plot on the right shows the corresponding level sets of γ_b on the unit disk.

Lemma 4.1.1 *The potential function γ defined by (4.1) is free of local minima over the interior of $\mathcal{P}$ provided γ_b is a harmonic function, with a constant minimum potential along the outlet zone and a constant maximum potential along the inlet zone, and the Jacobian of the mapping φ is full rank.*

Proof: By construction, γ_b is a harmonic function with extrema on the boundaries of the unit n-ball; therefore, the gradient $\mathcal{D}_{q_b}\gamma_b$ for $q_b = \varphi(q)$ is non-zero on the interior of the n-ball since there are no interior critical points [35].

The Jacobian of the mapping φ is assumed to be full rank, meaning that $D_q\varphi$ is a full rank matrix for all $q \in P$; therefore, $D_q\gamma = D_{\varphi(q)}\gamma_b \cdot D_q\varphi \neq \mathbf{0}$ for non-zero $\mathcal{D}_{q_b}\gamma_b$.

Since $D_q\gamma \neq \mathbf{0}$ for all $q \in P$, we conclude that there are no critical points on the interior of P [66]; therefore, we conclude that a local minimum does not exist on the interior.

□

Flowing along the gradient vector field provides the correct behavior, but induces some undesirable variability in speed. The gradient vector field has large magnitudes in some portions of the cell and small gradients in others. Therefore, define the unit reference vector field, $\hat{X}(q)$, for $q \in \mathcal{P}$ by taking the negative normalized gradient of the potential; thus,

$$\hat{X}(q) = -\frac{D_q\gamma^T}{\|D_q\gamma\|} = -\frac{D_q\varphi^T\, D_{\varphi(q)}\gamma_b^T}{\left\|D_{\varphi(q)}\gamma_b\, D_q\varphi\right\|}\ , \tag{4.3}$$

where $q_b = \varphi(q) \in \mathcal{B}$. The flow along the unit vector field captures the desired behavior, without being bound to the speed specified by the norm of the potential function gradient. The vector field $\hat{X}(q)$ is orthogonal to the cell boundaries because the potential extrema are along the boundaries of the cell by virtue of the mapping φ that maps boundary to boundary.

It is worth noting that Lindemann and LaValle [82, 83, 84] adapted this approach by using a different vector field generation approach over convex polytopes. Their vector fields have similar properties; the vector field derivatives are smooth almost everywhere and can be defined orthogonal to the cell boundaries. As such, their vector field design approach can be used with the control laws defined below.

Convergent Vector Field Design

The flow-through style policies are useful for driving the system from cell to cell, but not for converging to an overall goal. The policies developed by Rizzi [105] are appropriate for converging to a goal, but do not have vector fields that are orthogonal to the cell boundaries. To simplify the prepares test between policies defined over adjacent cells, and to provide some control continuity across cell boundaries, we consider a modification that generates a vector field that is orthogonal to the cell boundaries.

First, map the goal cell to the unit ball, $\mathcal{B}$, using the map φ as before, and let

$$q_g^b = \varphi(q_g)\ ,$$

where q_g is the goal configuration, and q_g^b is the mapped goal point. Let $\psi : \mathcal{B}\backslash q_g^b \to \mathcal{B}\backslash 0$ define a diffeomorphism such that $\psi(\partial\mathcal{B}) = \partial\mathcal{B}$ and $\lim_{q\to q_g} \psi(\varphi(q)) = 0$.

For the unit disk, a mapping based on complex numbers serves the purpose. Let $z = q^b = \varphi(q)$ be an arbitrary point in the unit disk represented in complex plane. Let $z_g = q_g^b = \varphi(q_g)$ be the goal

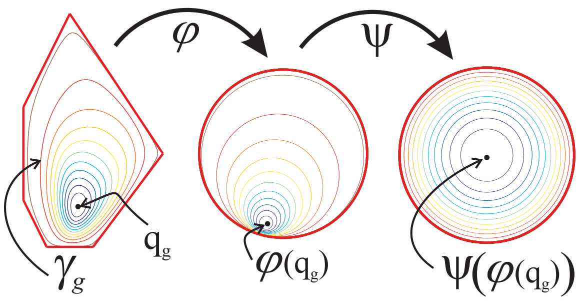


Figure 4.3: Equipotential contours for a convergent potential function found by mapping the goal cell to a unit ball centered at the goal, and letting $\gamma_g = \frac{1}{2}\left\|\psi \circ \varphi\right\|^2$.

point in the complex plane. Then

$$z_\psi = \psi(z) = \frac{z - z_g}{1 - \bar{z_g} \cdot z} , \tag{4.4}$$

where $\bar{z_g}$ is the complex conjugate of z_g. Clearly, $z_\psi = 0$ if $z = z_g$. Simple algebraic calculations show that the boundary maps to the boundary. Figure 4.3 shows an example of this mapping for a polygonal cell in the plane.

Define the potential function $\gamma_g : \mathcal{P} \to \mathbb{R}$ such that

$$\gamma_g = \frac{1}{2}\left\|\psi \circ \varphi\right\|^2 ,$$

where $0 \le \gamma_g \le 1$. For the goal policy, define the convergent reference vector field, $\hat{X}(q)$, as

$$\hat{X}(q) = -\frac{D_q\gamma_g^T}{\|D_q\gamma_g\|} \frac{\|q - q_g\|^2}{\|q - q_g\|^2 + \alpha} , \tag{4.5}$$

where $\alpha > 0$ is a scalar parameter that regulates the rate of deceleration near the goal. The vector field is orthogonal to the boundary and has decreasing magnitude near the goal point.

4.1.2 Control Law Design

Given either flow-through or convergent reference vector fields, $\hat{X}(q)$, this section derives a family of control laws that cause the system to converge to the vector field integral curves in such a way that the properties outlined in Chapter 3 are satisfied. The section defines control laws for kinematic and second-order dynamical systems. The second-order systems consider both unbounded and bounded acceleration.

Kinematic Control Law

For an idealized kinematic system of the form $\dot{q} = u$, the control inputs follow directly from the reference vector fields with

$$u = X(q) = s(q)\ \hat{X}(q) , \tag{4.6}$$

where $s(q) \in (0, \mathrm{V}_{\max}] \subset \mathbb{R}$ and $\hat{X}(q)$ is defined in (4.3) or (4.5). The scaling function $s(q)$ is used to respect speed limits imposed upon the system. For ideal kinematic systems, a constant speed scaling can be defined.

The control law (4.6) with vector fields (4.3) or (4.5) define a generic class of composable policies defined over convex cells. As the system is an idealized point robot, the requirement that the domain lie within the free configuration space is trivially satisfied since the cells are assumed to be fully contained in the free configuration space.

The control in (4.6) induces the system trajectory to follow the integral curves of $\hat{X}(q)$ by definition; thus, by construction, the kinematic control policy is conditionally positive invariant for both flow-through and convergent vector fields.

The policy induces finite time convergence provided $s(q)$ is bounded above zero over the entire cell; that is, $s(q) > \epsilon > 0$, for some finite ϵ. The flow-through vector field will cause the system to exit the cell in finite time because the velocity component along the vector field is always strictly positive , and every flow line exits the cell because there are no local minima. Thus, the system always makes finite progress along the flow line toward the exit face. For convergent vector fields, the system will converge to some arbitrarily small neighborhood of the goal in finite time.

The configuration-based test for inclusion into the kinematic domain is easily calculated for polytopes defined by half-space constraints. Thus, these policies satisfy all of the composability requirements given in Chapter 3.

Dynamical Control Law: Unbounded Acceleration

Given an idealized second-order dynamical system of the form

$$\ddot{q} = u \,, \tag{4.7}$$

subject to the velocity constraint $\|\dot{q}\| \leq \mathrm{V}_{\max}$, define a reference vector field $X(q) = s(q)\ \hat{X}(q)$, for some positive scalar function $s(q) \in (0, \mathrm{V}_{\max}] \subset \mathbb{R}$. In addition to enforcing the velocity limit, $s(q)$ is used to enforce the prepares relationship among neighboring policies.

Following Rizzi [105], define a velocity reference control policy of the form

$$u = K\ (X(q) - \dot{q}) + D_q X\, \dot{q}\,, \tag{4.8}$$

where $K > 0$ is the “velocity regulation” gain that acts to decrease the error $(X(q) - \dot{q})$, and $D_q X\, \dot{q}$ is a feed-forward term that enables the system to follow the changing vector field.

Lemma 4.1.2 *In the absence of acceleration constraints, with sufficiently large K and initial velocities such that $\|\dot{q}\| \leq \|X(q)\|$, and $\dot{q}^T X > 0$ or $\|\dot{q}\| = 0$, the trajectories of the closed-loop system defined by (4.7) under the influence of (4.8), converge to the integral curves of the vector field $X(q)$ in a way such that the trajectory never exits the cell except by the outlet zone. Furthermore, the system speed remains less than the reference speed while the system remains in the policy domain; that is $\|\dot{q}\| \leq \|X\|$. For flow-through vector fields, the system trajectory exits the cell in finite time.*

See Lemma B.2.4 in Appendix B for a detailed proof. Appendix B also includes details on calculating a lower bound for K.

For unbounded dynamical systems, control law (4.8) with vector fields (4.3) or (4.5) define a generic class of policies defined over convex cells. The domain test requires three calculations, a configuration test for inclusion in the polytope and two velocity tests for $\|\dot{q}\| \leq \|X(q)\|$ and $\dot{q}^T X > 0$. Lemma 4.1.2 proves conditional invariance and finite time convergence.

For policies defined over disjoint cells with unbounded acceleration, the prepares tests are satisfied if the speed profiles at the exit zone of one cell is less than the speed profile along the inlet zone of the next cell. Given that the reference vector fields are orthogonal at the boundaries, the requirement that $\dot{q}^T X > 0$ is satisfied by any velocity that crosses the shared boundary. Therefore, as long as the reference speed of the "upstream" policy is less than or equal to the reference speed of the "downstream" policy, the prepares test is satisfied. In the unbounded acceleration case, $s(q)$ can be a constant less than $\mathrm{V}_{\max}$ for all policies over a disjoint decomposition; this, the policies satisfy the velocity bound.

Figure 4.4 shows a simulation of a variety of initial conditions for the dynamical system given in (4.7) under the influence of (4.8). The policy deployment is defined and ordered by hand to ensure prepares based on the adjacency relations among cells. For each initial condition, the system converges to the goal configuration using the hybrid control strategy induced by the underlying decomposition into disjoint polygonal cells. The simulation, where the policies are executed based on the total ordering, demonstrates the global control policy that is induced by our policies and deployment method. The resulting trajectories induced by the feedback control policies are dependent upon the underlying decomposition; that is, the shape of the integral curves are determined by the shape of the polygon and the selected outlet zone. Different decompositions yield different trajectories; however, the overall convergence to the goal is guaranteed provided the prepares relationships are obeyed.

Dynamical Control Law: Bounded Acceleration

As a more realistic system, consider (4.7) with the following dynamic constraints,

$$\|\dot{q}\| \leq \mathrm{V}_{\max}\,, \tag{4.9}$$

$$\|u\| = \|\ddot{q}\| \leq \mathrm{A}_{\max}\,. \tag{4.10}$$

The velocity limit is taken to be a safety limit as before. In the presence of the acceleration constraint, the control law in (4.8) is insufficient. In regions where the vector field is changing, just tracking the vector field with $\dot{q} = X(q)$ will violate the constraints if $\|u\| = \|D_q X\,\dot{q}\| > \mathrm{A}_{\max}$.

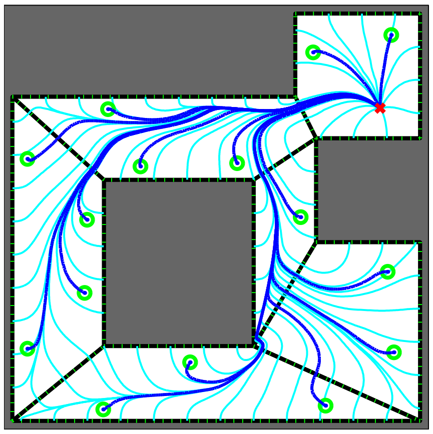


Figure 4.4: Simulation of a dynamical system using the velocity reference control policies based on flow-through and convergent vector fields. Light colored lines represent integral curves of the $X(q)$, while the dark colored lines represent trajectories of the system for various initial conditions.

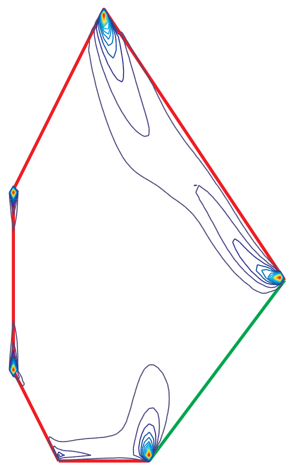


Figure 4.5: Spectral norm of the derivative of the negative normalized gradient vector field ($\left\| D_q\hat{X} \right\|$) for a polygonal cell, with the cell boundary shown. The largest norms, which remain finite due to approximation, are located near the polygon vertices.

To avoid violating the constraints, scale the reference vector field to encode the idea of slowing down while turning and define the variable speed vector field $X(q) = s(q)\ \hat{X}(q)$, where

$$s(q) = \min\left(\frac{s^*}{\left\| D_q\hat{X} \right\| + \lambda}, \mathrm{V}_{\max}\right) . \tag{4.11}$$

with s^* a constant defined for each policy as described below, and $\lambda > 0$ an arbitrary constant that prevents divide by zero. The term $\left\| D_q\hat{X} \right\|$ is the spectral norm [1] of $D_q\hat{X}$, which increases in relation to the change in the vector field, thereby decreasing the speed. Figure 4.5 shows a plot of $\left\| D_q\hat{X} \right\|$ for the example polygonal cell shown in Figure 4.1. The constant s^* is chosen so that

$$s^* \leq \min_q \frac{\sqrt{\mathrm{A}_{\max}}\ \left(\left\| D_q\hat{X} \right\| + \lambda\right)}{\sqrt{\left\| D_q\hat{X} - \frac{\hat{X}\ D_q\left\| D_q\hat{X} \right\|}{\left\| D_q\hat{X} \right\| + \lambda} \right\|}} , \tag{4.12}$$

which provides a conservative bound that guarantees the system will not exceed the acceleration bound so long as the speed at q does not exceed $\frac{s^*}{\left\| D_q\hat{X} \right\| + \lambda}$.

Although this form allows the reference velocity control policy to make use of the dynamical capabilities of the system, the form is still not sufficient to prevent constraint violations when $\dot{q} \neq X(q)$. In the case when the initial velocity is not aligned with the vector field, the proportional term of (4.8) may cause the acceleration constraint to be violated.

[1]The spectral norm of a matrix M, denoted $\| M \|$, is defined as

$$\| M \| = \max_{\|\mathbf{x}\|=1} \|M\,\mathbf{x}\| .$$

To prevent this, we define a switched (hybrid) control policy over each cell; the component policies are called 'Save', 'Align', and 'Track', denoted Φ_S, Φ_A, and Φ_T respectively. The component policies are designed to cause the system to converge to the integral curves of $X(q)$ without violating the constraints or exceeding the specified speed. This section gives an overview of each policy; refer to Appendix B for implementation details .

The Save policy for polytope cells, Φ_S, is based on the policy presented in [105]. The policy is designed to use the maximum available acceleration applied orthogonal to the boundary point of first collision based on the open loop dynamics. This is guaranteed to bring the system to rest within the cell if there exists any policy capable of doing so [105]. The domain of Φ_S is termed the *savable set*, and corresponds to any initial condition in the state space associated with a given cell that can be brought under control and avoid collision. Appendix B defines the *collision avoidance ratio*, $\zeta_c > 0$ as the ratio of the braking distance to the collision distance. The braking distance is the distance the system would move toward the point of imminent collision while maximally braking; the collision distance is the distance to the closest collision point. If $\zeta_c < 1$, collision can be avoided and the system brought safely to rest; therefore, the savable set for a given polytope, $\mathcal{P}$ is

$$\mathscr{D}(\Phi_{S_\mathcal{P}}) := \{(q\,,\dot{q}\,) \mid q \in \mathcal{P},\, \zeta_c < 1\}\ .$$

The goal set of this policy, $\mathscr{G}(\Phi_{S_\mathcal{P}})$, is some configuration within the cell $\mathcal{P}$, where the system is at rest; that is, $\mathscr{G}(\Phi_{S_\mathcal{P}}) = \{(q, \dot{q}) \mid q \in \mathcal{P}\ , \|\dot{q}\| = 0\}$.

Lemma 4.1.3 *For a given convex polytope and initial velocity such that* $0 \le \zeta_c < 1$, *the Save policy acts to decrease* ζ_c. *Therefore,* ζ_c *remains less than one, collision is avoided, and the system remains in the savable set* $\mathscr{D}(\Phi_{S_\mathcal{P}}) := \{(q\,,\dot{q}\,) \mid q \in \mathcal{P},\, \zeta_c < 1\}$ *and eventually comes to rest.*

See Lemma B.3.2 in Appendix B for proof.

The Align policy, Φ_A, is designed to apply maximum acceleration to the system in order to quickly bring the velocity vector into alignment with the vector field $X(q)$ defined by either (4.3) or (4.5). Φ_A transitions from performing a Save control action, that is applying maximum acceleration orthogonal to the point of boundary collision, to using a portion of the available acceleration to reduce the angle between the reference velocity and current velocity, while also decreasing the current speed. A user defined constant, $0 < \mu < 1$, determines the transition between the "saving" and "aligning" actions. Define

$$\upsilon = \max\left(0, \frac{\mu - \zeta_c}{\mu}\right)\ ,$$

and let $\sigma : \upsilon \to [0, 1]$ define the transition function such that $\sigma(0) = 0$ and $\sigma(1) = 1$. The Align policy is given by

$$\Phi_A\ :\ u = \begin{cases} \mathrm{A}_{\max} \frac{(1-\sigma(\upsilon))\,\Phi_S + \sigma(\upsilon)\,\hat{e}}{\|(1-\sigma(\upsilon))\,\Phi_S + \sigma(\upsilon)\,\hat{e}\|} & \dot{q}^T X \le \dot{q}^T \dot{q} \\ \mathrm{A}_{\max} \frac{(1-\sigma(\upsilon))\,\Phi_S - \sigma(\upsilon)\,\hat{\dot{q}}}{\left\|(1-\sigma(\upsilon))\,\Phi_S - \sigma(\upsilon)\,\hat{\dot{q}}\right\|} & \text{otherwise} \end{cases}\ , \tag{4.13}$$

where $\hat{e} = \frac{X(q) - \dot{q}}{\|X(q) - \dot{q}\|}$ is the unit vector along the velocity error, $\hat{\dot{q}} = \frac{\dot{q}}{\|\dot{q}\|}$ is the unit direction of current speed, and Φ_S denotes the input defined by the save policy. The demonstrations in this chapter use $\sigma(\upsilon) = \sqrt{\upsilon}$.

The Align policy is hybrid (switched) policy. At $\sigma(\upsilon) = 0$, that is when $\zeta_c \ge \mu$, the Align policy is "saving" with $u = \Phi_S$. When $\sigma(\upsilon) > 0$ the policy transitions to aligning, but switches behavior based on the system velocity. In the normal mode, Φ_A uses a transition function to combine the Save action with acceleration along the velocity error vector $\hat{e}$. If the acceleration along $\hat{e}$ would increase the velocity, as when $\dot{q}^T X > \dot{q}^T \dot{q}$, then Φ_A switches to apply maximum acceleration along the negative of the current velocity. The Align policy always acts to decelerate the system.

The domain of the Align policy over a given cell is

$$\mathscr{D}(\Phi_{A_{\mathcal{P}}}) := \{(q\,, \dot{q}\,) \mid q \in \mathcal{P},\, \zeta_c < 1\}\ ,$$

which is equivalent to the domain of the Save policy over the same cell. The goal set is

$$\mathscr{G}(\Phi_{A_{\mathcal{P}}}) = \{(q, \dot{q}) \mid q \in \mathcal{P}\ , \|\dot{q}\| = 0\}\ ,$$

which is also the same as for the Save policy. Since Align is equivalent to Save for $\zeta \geq \mu$, the domain is conditionally invariant by Lemma 4.1.3.

The Track control policy, Φ_T, is designed to bring the system velocity into alignment with the vector field $X(q)$ using the maximum available acceleration and then track the vector field according to (4.8). The domain of the Track control policy over a given cell, $\mathcal{P}$, is

$$\mathscr{D}(\Phi_{T_{\mathcal{P}}}) = \left\{(q\ ,\ \dot{q}) \mid q \in \mathcal{P}\ ,\ \dot{q}^T X > 0\ ,\ \|\dot{q}\| \leq \|X(q)\|\right\}\ .$$

That is, the speed is less than or equal to the reference speed, and the orientation error between the current velocity and the desired velocity is less than $\frac{\pi}{2}$. For flow-through vector fields as in (4.3), the Track control policy guarantees that the system trajectory does not exit the cell, other than by the outlet zone, which is the goal set, $\mathscr{G}(\Phi_{T_P}) = \left\{(q\ ,\ \dot{q}) \mid \|\dot{q}\| \leq \|X(q)\|\ , \dot{q}^T X > 0\,, q \in \partial\mathcal{P}_{\text{outlet}}\right\}$. For convergent vector fields, $\mathscr{G}(\Phi_{T_P}) = \{q_g\ , 0\}$, where q_g is the goal state.

The Track control policy monotonically decreases the orientation error between the current velocity and the desired velocity. The approach uses some of the available acceleration to prevent the orientation error from increasing as the trajectory evolves, and uses the remainder of the available acceleration to decrease the error. If the speed, $\|\dot{q}\|$, is less than the desired speed, $\|X(q)\|$, then the speed scaling chosen in (4.12) guarantees that there will be acceleration remaining to reduce the error. In the limit, as the velocity error approaches zero, the Track control policy is identical to (4.8).

Lemma 4.1.4 *Under the influence of the Track control policy, the system (4.7), with constraints given in (4.9) and (4.10), and initial condition $\{q, \dot{q}\} \in \mathscr{D}(\Phi_{T_{\mathcal{P}}})$, converges to the integral curves of $X(q)$, defined in (4.11), in a way such that $\|\dot{q}\|$ remains less than $\|X(q)\|$ and the trajectory never exits the cell except by the outlet zone. For flow-through vector fields, the system trajectory exits the cell in finite time. For convergent vector fields, the system converges to an arbitrarily small neighborhood of the goal in finite time.*

See Lemma B.3.3 in Appendix B for proof.

The Save policy may be used by itself over a given cell provided that other deployed policies cover the entire cell; that is, that other cells overlap the entire Save cell as shown in Figure 4.6. In other words, no matter where the system comes to rest within the cell, another policy should capture that state. The Save policy is typically used over relatively large regions to capture fast initial conditions that cannot be captured by the smaller cells that cover the larger region. This allows the deployment to capture more adverse initial conditions in the free state space than possible with a policy over a smaller configuration-cell. Since the Align policy subsumes the behavior of the Save policy for high collision avoidance ratios, the Save policy is not needed if an Align policy is deployed over the same cell.

The Align and Track policies are designed to work together. The intention of the Align policy is to bring the system into alignment with the reference vector field, while also slowing the system;

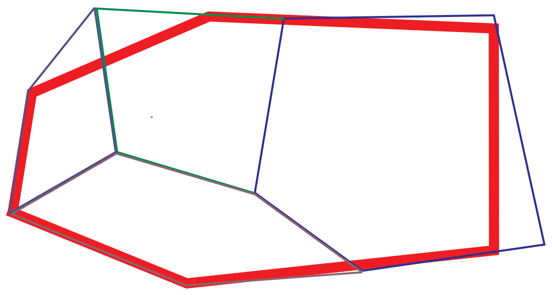


Figure 4.6: The Save policy is used to capture more adverse initial conditions. In this example, the large Save cell, denoted by the thick line, is covered by four other cells.

thus, guaranteeing that Align prepares the Track policy; that is, $\Phi_{A_\mathcal{P}} \succeq \Phi_{T_\mathcal{P}}$. This is trivially verified since the Align policy can bring the system to rest in the cell; thus, the Save policy is not needed over the same cell. For flow-through style policies defined by the vector field given by (4.3), define the conditionally positive invariant switched *Flow* control policy composed of Align and Track policies as $\Phi_F = \{\Phi_T, \Phi_A\}$. For the convergent policy using the vector field defined by (4.5), define the switched *Goal* control policy $\Phi_G = \{\Phi_T, \Phi_A\}$. In both Flow and Goal policies, the Φ_T policy has highest priority. The Flow and Goal policies are *meta-policies*.

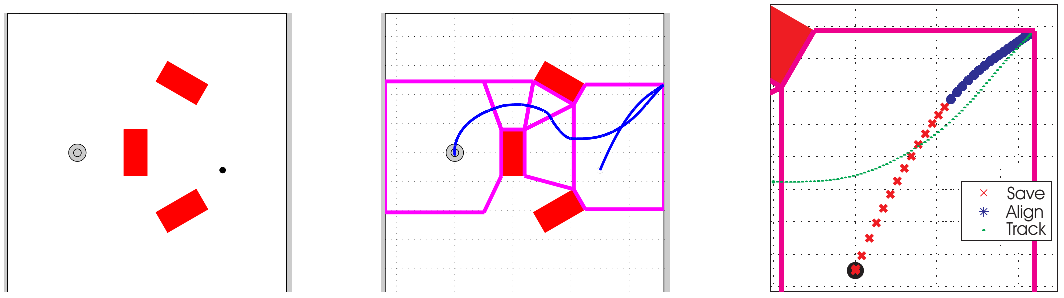
Definition: Meta-policy: A *meta-policy* is a control policy over a local domain that is made up of component policies and a switching strategy among the component policies.

For planning purposes, each meta-policy is treated as a single policy in the prepares graph.

The Flow policy causes the system to exit a given cell, therefore the policy parameters must be tuned to respect the prepares relationship with a nearby policy domain. Let k denote the relative priority of Φ_{F_k} with 1 being the highest. In order for $\Phi_{F_{k+1}}$ to prepare Φ_{F_k}, the exit zone of its component policy $\Phi_{T_{k+1}}$ must be contained in the closure of $\mathscr{D}(\Phi_{F_k})$, and the speed profile along the exit zone of $\Phi_{T_{k+1}}$ must be less than or equal to the same speed profile in Φ_{F_k}. This is done by choosing s^*; therefore, (4.12) represents an upper bound on the speed scaling. The s^* scaling may need to be reduced when considering the prepares relationship. Thus, to ensure prepares, s^* is reduced from (4.12) until the speed profile along the exit zone is below the speed profile of the adjacent policies. The speed profile tests may consider either Φ_{T_k} or Φ_{A_k} for determining the speed profile that must be matched; considering only Φ_{T_k} leads to smoother paths, but is more conservative with respect to domain.

The simulation shown in Figure 4.7 is based on a disjoint convex polygonal decomposition of the free workspace. The approach applies the switched meta-policy $\{\Phi_T, \Phi_A\}$ to each polygon; the policy ordering is specified manually based on the given goal location. The policy free parameters $\alpha > 0$, $0 < \mu < 1$, and $\lambda > 0$ are defined as constants, and applied to all policies. The free parameters $K > 0$ and $s^* > 0$ are manually chosen for each policy to enforce the prepares relationship with adjacent policies and to enforce constraints as described in Appendix B. This is done during the process of ordering the policies by checking the speed profile for a fine sampling of points along the exit boundary, against the speed profiles evaluated for the same points in the policy being prepared.

The simulation demonstrates policy switching, both among the meta-policies defined over cells, and among the component policies of the individual meta-policies. The initial velocity, which points toward the upper right corner of the initial cell, is chosen to just miss the cell boundary. The overall hybrid policy activates the Align component policy first, followed by the Track policy of the same



a) Course layout b) Decomposition and path c) Close-up of switched behaviors.

Figure 4.7: Simulation of a constrained dynamical system showing the result of hybrid switching policies. The initial velocity is to the upper right. The Align policy is activated first, with saving action preceding the transition to aligning action. The Track policy then takes over and drives the system out the first cell, and through the entire region by composing the local control policies.

meta-policy. In this example, the initial velocity is such that $\zeta_c > \mu$, and the system must first activate the "saving" action of the align control policy; this reduces ζ_c. Figure 4.7-c differentiates between the saving action ($\zeta_c \geq \mu$) and the aligning action ($\zeta_c < \mu$) during execution of the Align policy. After the system switches to the first Track component policy, the system exits the domain of the first meta-policy, and enters the domain of the Track component policy in the adjacent cell. The induced trajectory converges to the goal as desired, while avoiding the obstacles; the overall hybrid control policy switches meta-policies as the system moves from cell to cell.

4.2 Policy Space Planning and Control

The vector field definitions, coupled with the kinematic control law and Save, Align, and Track policies for second-order systems, form a palette of generic policies. The policies are instantiated in the system workspace by specifying a convex polytope, and the necessary policy parameters, K, α, s^*, μ, λ, and q_g, as needed. This section presents examples and techniques for instantiating policies in the system state space for both the kinematic and dynamical systems defined in Section 4.1.2. The examples demonstrate a variety of approaches to planning and control design using the generic policies defined above in a hybrid control framework.

4.2.1 Basic Scenarios

For the basic scenarios, we assume a bounded workspace with polygonal obstacles, and that a disjoint, finite convex decomposition of the free workspace is given. The collection of polygons covers the free workspace. An undirected adjacency graph, which encodes the relationship between cells, is given.

A simple planning approach is to define a sequence of cells that must be navigated, as described in Section 3.5.1. The policies are specified as needed. Given the adjacency graph, deploying a hybrid policy is as simple as planning a walk through the adjacency graph that connects the cells containing the start and goal points. First, a convergent policy is deployed over the cell containing the goal. Then, neighboring cells in the walk define flow-through policies such that the outlet zones coincide with the common boundary of neighboring cells; this is specified based on the adjacency relationship between neighboring cells along the adjacency graph walk. The choice of exit faces turns the undirected adjacency graph into a directed prepares graph that is built during planning. The policy parameters are specified as needed to enforce the velocity and acceleration bounds and prepares relationships.

Figure 4.8 shows an example simulation of this technique for a kinematic system. Policy switching is based on domain inclusion tests for the sequence of policies. This approach results in a complete navigation method for kinematic systems provided the available cells cover the free space, but only uses some cells in the sequence; as with all sequence-based approaches, this approach is less robust than order-based approaches.

In order to provide a global feedback policy that does not require replanning, all cells in the disjoint covering should be used for the total order-based hybrid control policy described in Section 3.5.2. Since the cells cover the free space, the hybrid control policy is complete for kinematic systems; the resulting global hybrid control policy brings any state within the domain of the local policies to the overall goal. Figure 4.4 shows an example of this technique for second-order systems with unbounded acceleration. The policy parameters for each cell are chosen to enforce the prepares relationship. This method is complete for any initial condition in the domain of one of the policies.

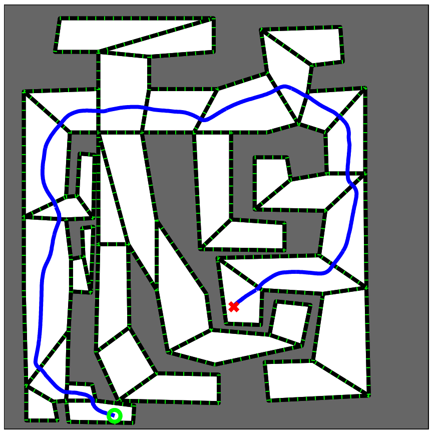


Figure 4.8: Simulation of a kinematic system. The dark line shows the path taken, dark region denotes the boundary of the free space, and dotted lines show the decomposition into convex polygons.

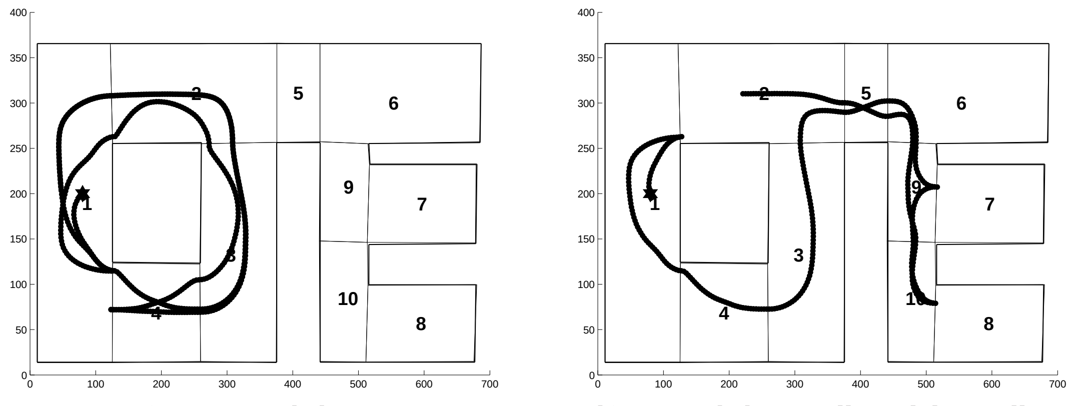
4.2.2 Reactive Automaton Based Planning

The work in this thesis enabled Kress-Gazit *et al.* [68] to generate reactive automata with policies defined over convex polytopes for fully actuated systems. They use the kinematic policies defined in Section 4.1 as the foundation for hybrid control synthesis technique. The low-level continuous behavior is governed by the continuous execution of the local feedback control policies; the high-level behavior is governed by the discrete transitions in the finite automaton. This approach allows the system to change behaviors based on information gathered at run time.

The low-level policies are defined based on a disjoint decomposition of the planar workspace into convex polygons. For each cell, a set of control policies is defined such that the system can exit the cell and enter any of the neighboring cell. The undirected adjacency graph associated with the decomposition is converted into a directed prepares graph, which encodes all of the transitions the system can make between cells.

The automaton synthesis algorithm takes as input the possible transitions encoded in the prepares graph, the allowable discrete inputs sensed by the robot, and behavior specification given in LTL. The automata synthesis extracts an automaton that specifies the policy switching based on the discrete sensor inputs; this allows the user to specify behaviors at a high-level, with the low-level motion induced by the policy composition governed by the automaton.

Figure 4.9 shows an example scenario where a robot is tasked with patrolling a nursery listening for crying babies. When crying is detected, a discrete sensor signals the automaton, and the hybrid control policy induces the robot to search for an adult to alert. This simulation demonstrates the how the local feedback control policies can be combined with automaton synthesis tools to generate reactive hybrid control policies. During execution, the automaton transitions are governed by changes to discrete inputs based on sensor measurements and the continuous transitions between policy domains. The hybrid control policy induces continuous behavior, which changes based on the sensed inputs, that satisfies the high-level specification. The controller synthesis is enabled by the composable policies and discrete transition relationship encoded by the prepares graph.



a) No crying babies b) Crying baby in cell 4, adult in cell 8

Figure 4.9: Nursery simulation using automaton that encodes "check for crying babies in cells 2 and 4, when crying detected, search for and alert adult in cells 6, 7, or 8." Low-level behavior governed by kinematic policies described in Section 4.1. Simulation and figures are courtesy of Hadas Kress-Gazit and George J. Pappas, GRASP Lab, University of Pennsylvania.

4.2.3 Global Policy Design: The "Dynamical P" Problem

As a demonstration of the power and flexibility of the hybrid control approach for second order systems, this section presents what we term the "dynamical P" problem. Figure 4.10 shows a collection of cells that cover a workspace, which is not simply connected. The goal is designated in the lower portion of the loop inside region 'a'; a decision about which way to travel around the loop to get to the goal must be made. For kinematic systems, this choice is typically made based on path length from the initial configuration to the goal. For constrained dynamical systems, the choice must take into consideration the initial velocity of the system. The hybrid control approach taken in this thesis allows the decision to be made at execution time based on which policy in a predetermined deployment contains the current state.

This section describes a simulation that assumes an idealized dynamical robot, $\ddot{q} = u$ with $q, u \in \mathbb{R}^2$, subject to both velocity and acceleration constraints in the form of (4.9) and (4.10). The

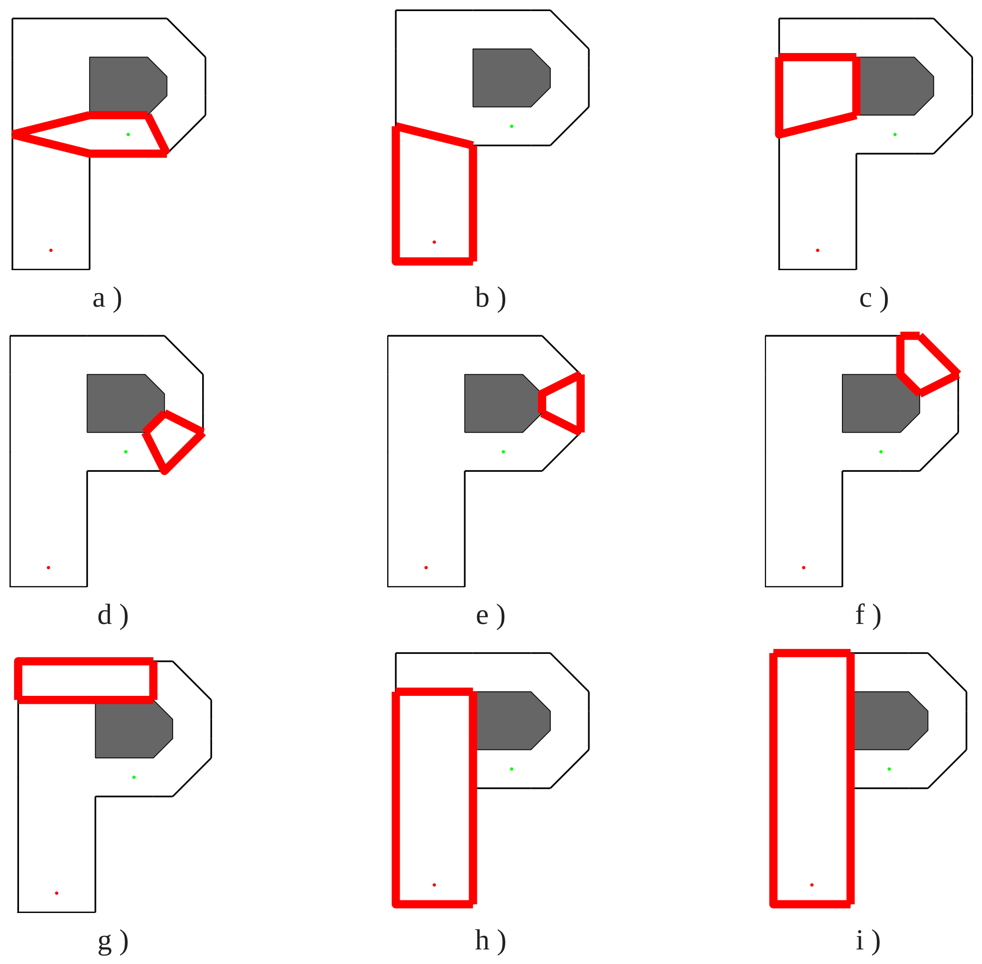


Figure 4.10: Configuration space cells used to define local control policies for "dynamical P" simulation.

policy deployment,

$$\mathcal{U}' = \left\{\Phi_{G_a}, \Phi_{F_b}, \Phi_{F_c}, \Phi_{F_d}, \Phi_{F_e}, \Phi_{F_f}, \Phi_{F_g}, \Phi_{S_h}, \Phi_{S_i}\right\} \ ,$$

is generated by hand. The first subscript refers to the 'Goal', 'Flow', and 'Save' hybrid policies defined in Section 4.1.2; the second subscript corresponds to the configuration-cells shown in Figure 4.10. The policies are executed according to the ordered list $\mathcal{U}'$, with Φ_{G_a} being the highest priority. At each time step, the list is searched from highest to lowest priority until a domain that contains the initial state is found as described in Section 3.5.2. This example allows the demonstration of switching behavior in simulation, while remaining simple enough to present in detail; additional deployments and configuration-cells are possible.

The deployment uses one Goal policy (Φ_{G_a}) and six Flow policies ($\Phi_{F_b}, \Phi_{F_c}, \Phi_{F_d}, \Phi_{F_e}, \Phi_{F_f}$, and Φ_{F_g}) to provide a disjoint cover of the free configuration space. The Flow policies are configured to prepare the adjacent policy of higher priority. The 'Goal' and 'Flow' policies use 'Align' and 'Track' component policies; therefore, there are actually fourteen total component policies deployed for this group. Together, these policies induce a piecewise potential-based navigation function for any state within the domains of the Goal or Flow policies.

To capture faster initial velocities, the two Save policies are deployed in the large corridor. The Save policy Φ_{S_h}, whose cell is shown in Figure 4.10, covers the corridor from the bottom to the top of the obstacle forming the "P"; thus, $\Phi_{S_h} \succeq \bigcup \{\Phi_{G_a}, \Phi_{F_b}, \Phi_{F_c}\}$. Any state savable by this policy prepares the Flow or Goal policies in a way that causes the system to enter the lower half of the loop from the left. The second Save policy, Φ_{S_i}, encompasses the entire large corridor on the left side; thus, $\Phi_{S_i} \succeq \bigcup \left\{\Phi_{G_a}, \Phi_{F_b}, \Phi_{F_c}, \Phi_{F_g}\right\}$. Anything savable by Φ_{S_i} that comes to rest in Φ_{F_g} will enter the lower loop from the right by traveling above the obstacle.

Figure 4.11 shows three initial conditions that demonstrate the change in behavior as a response to changes in the initial conditions. The first case, which starts slowly straight up, activates Φ_{F_b} and Φ_{G_a}. The second case, which starts to the right, activates Φ_{S_h}, Φ_{F_c}, and Φ_{G_a}, in that order. The final case, which starts to the left with a high initial velocity, activates Φ_{S_i}, Φ_{F_g}, Φ_{F_f}, Φ_{F_e}, Φ_{F_d}, and Φ_{G_a}, in that order. The switching is automatic based on the state being within the domain of the highest priority control policy as specified by the deployment.

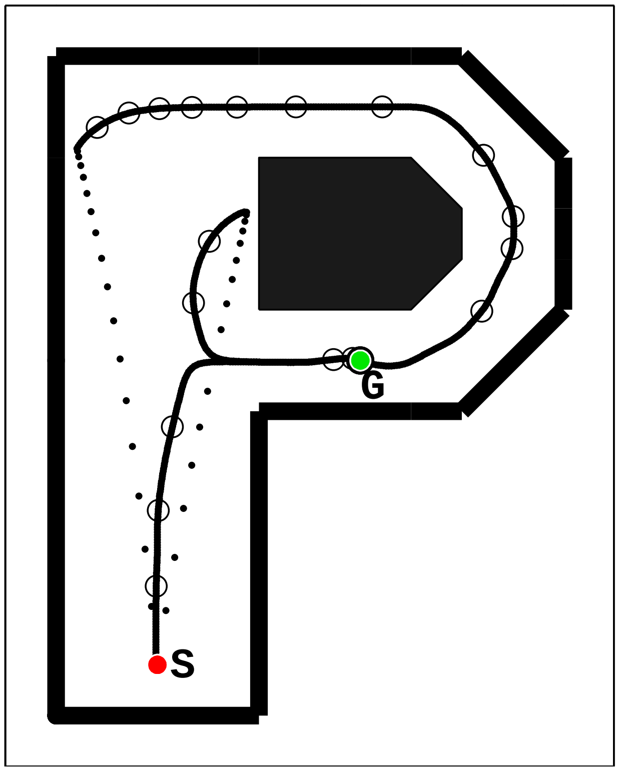


Figure 4.11: Deployment of local policies induces a change in the route taken given three different initial conditions. The spacing of the dots and circles corresponds to different intervals of elapsed time.

4.2.4 Automated Policy Instantiation and Deployment

This subsection explores an approach to automating the policy deployment for second-order systems. In the above examples, the policy parameters are specified by hand to enforce the required prepares relationship among policies. In this section, the selection of cells and specification of policy parameter values is automated. The simulations of the resulting deployment provide additional demonstrations of the flexibility of planning in the space of control policies.

We assume a cell decomposition that contains a rich collection of polygons – large ones and small ones, disjoint and overlapping – is given. Overlapping cells allow the second-order policy domains to cover a large fraction of the free state space. The collection of cells forms the foundation of the automated instantiation process, which automatically chooses a cell from the collection, and then specifies the policy parameters.

In this case, automatic instantiation means specifying the goal set, that is which facet, and free parameters – K, α, s^*, μ, and λ – over a chosen cell. The policy ordering is determining during instantiation to guarantee the global behavior. This demonstration focuses on building a hybrid global control policy that addresses a single navigation task using a total order of policies defined over convex polygons.

Several choices are made to simplify the deployment strategy. First, policies are instantiated one at a time. Second, the automated deployment algorithm only uses each cell once; that is, only one facet is chosen as an exit for flow-through policies. Third, the algorithm does not test that a given policy increases the size of the domain of the hybrid policy defined by previously deployed policies. Therefore, it is possible that existing policies may *dominate* [88] the current policy, meaning that the previously instantiated policy domains may completely contain the policy domain being considered. Although retaining dominated policies leads to policies that are never invoked, the overall correctness of the hybrid policy remains since the highest priority policy is always used. As a simpler version, a cell is discarded if the cell is completely contained within another cell.

To reduce the computational expense, the extended prepares tests are not conducted. Recall, that the extended prepares tests is based on a policy goal set being contained in the union of several policy domains Instead, the automated approach uses the simpler prepares relation between two policies. One exception is that Save policies are deployed over large regions of configuration space in order to capture extreme initial velocities and enlarge the fraction of free state space in the overall domain. The decomposition used to deploy flow policies covers all of the free configuration space; therefore, the Save policy trivially prepares the other policies since the goal state is at rest. With these caveats, Algorithm 1 addresses the automated deployment problem for the specific policies developed in this section in a way that integrates the planning and policy specification stages.

The algorithm begins by choosing a cell from the collection of convex polygonal cells to serve as the goal cell (line 2). The cell is chosen according to some heuristic from those cells that contain the designated goal. For this demonstration, the heuristic is calculated by triangulating each polygon by including the goal point as a vertex. The polygon with the largest ratio of minimum triangle area ($A_{\min}$) and maximum triangle area ($A_{\max}$) weighted by the total polygon area ($A_{\mathcal{P}}$) is chosen; that is $h = A_{\mathcal{P}} \frac{A_{\min}}{A_{\max}}$. This results in choosing a relatively large polygon, while avoiding elongated ones; see Figure 4.12. The Goal policy parameters are defined to satisfy the system constraints. The deployment (line 4) and list of unprepared policies (line 5) are initialized to include only the Goal policy. The goal cell is used, so it is removed from the list of available cells (line 6); that is, it is taken out of the collection of convex polygonal cells available for instantiation.

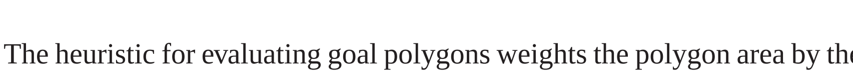


Figure 4.12: The heuristic for evaluating goal polygons weights the polygon area by the area ratio of smallest and largest triangles in a triangulation that includes the goal point. In this case, the policy on the left is preferred.

```
Algorithm 1: Automated Deployment for Fully Actuated Systems
Input: Finite collection of convex polygonal cells $\mathcal{K} = \{\mathcal{P}_1, \ldots, \mathcal{P}_M\}$
       that covers the free configuration space, and a goal configuration $q_g$
Output: Ordered collection of instantiated control policies
        $\mathcal{U}' = \{\Phi_1, \ldots, \Phi_{O(M)}\}$
(1)   Let $\mathcal{K}' = \{\mathcal{P}_i \in \mathcal{K} \mid q_g \in \mathcal{P}_i\}$
(2)   Chose $\mathcal{P}_g \in \mathcal{K}'$ according to a heuristic
(3)   Parameterize the goal policy $\Phi_G = \{\Phi_T, \Phi_A\}_g$ based on $\mathcal{P}_g$, $q_g$, and con-
      straints
(4)   Set $\mathcal{U}' = \{\Phi_G\}$ (the deployment)
(5)   Set $\mathcal{V} = \{\Phi_G\}$ (list of unprepared policies)
(6)   Set $\mathcal{K} = \mathcal{K} \backslash \mathcal{P}_g$ (don't reuse cells for simplicity)
(7)   while $\mathcal{V} \neq \emptyset$
(8)       Choose $\Phi_c \in \mathcal{V}$ based on a heuristic (e.g., highest priority or minimum
          distance to $q_g$)
(9)       Let $\mathcal{V} = \mathcal{V} \backslash \Phi_c$ (remove policy from list of unprepared)
(10)      Let $\mathcal{P}_c$ be the cell associated with $\Phi_c$
(11)      Let $\mathcal{K}' = \{\mathcal{P}_i \in \mathcal{K} \mid \text{there is a face of } \mathcal{P}_i \text{ contained in } \bar{\mathcal{P}}_c, \text{ the closure of } \mathcal{P}_c\}$
(12)      Let $\mathcal{K} = \mathcal{K} \backslash \mathcal{K}'$ (don't reuse cells for simplicity)
(13)      $\forall \mathcal{P}_i \in \mathcal{K}'$ set parameter values for $\Phi_i = \{\Phi_T, \Phi_A\}_i$ such that $\Phi_i \succeq \Phi_c$
(14)      Let $\mathcal{V}' = \{\Phi_i \mid \mathcal{P}_i \in \mathcal{K}' \text{and } \Phi_i \succeq \Phi_c\}$
(15)      Order $\mathcal{V}'$ based on some heuristic (for example  average cost to execute
          or distance to $q_g$)
(16)      $\mathcal{U}' = \{\mathcal{U}', \mathcal{V}'\}$, i.e. add ordered $\mathcal{V}'$ to the end of $\mathcal{U}'$
(17)      $\mathcal{V} = \{\mathcal{V}, \mathcal{V}'\}$ (update list of unprepared policies)
(18)  endwhile
(19)
(20)  Deploy 'Save' policies over any unused cells
(21)  while $\mathcal{K} \neq \emptyset$
(22)      Choose largest $\mathcal{P}_i \in \mathcal{K}$
(23)      Let $\mathcal{K} = \mathcal{K} \backslash \mathcal{P}_i$
(24)      Deploy Save policy $(\Phi_S)_i$ based on $\mathcal{P}_i$
(25)      $\mathcal{U}' = \{\mathcal{U}', (\Phi_S)_i\}$
(26)  endwhile
```

The bulk of the algorithm deploys policies to prepare previously deployed policies, building a total order list in the process. The algorithm is greedy in the sense that it uses cells as soon as an facet is completely contained in the domain of the previously deployed policies; it is possible that delaying the use of a particular cell would allow a larger domain. The prepares test uses the more restrictive test that a cell facet be completely contained in the cell closure of the policy it is preparing (line 11). The deployment occurs in "layers" as each deployed policy is evaluated to see what unused cells can prepare the deployed policy. Each layer is ordered (line 15) before the new policies are added to the deployment (lines 16 and 17). An alternative scheme would be to swap lines 15 and 17, only add Φ_c to $\mathcal{U}'$ in line 16 (instead of $\mathcal{V}'$), and reorder $\mathcal{V}$ instead of just $\mathcal{V}'$ in the new line 17.

The final **while** loop in Algorithm 1 is used to deploy individual Save policies to capture larger regions of the free state space for these second-order systems than is possible with the cells that prepare other cells. Implicit in this coding is the assumption that the cells associated with previously deployed Track policies cover the free configuration space.

To fully demonstrate automated deployment, Algorithm 1 is implemented in simulation. The simulation is applied to the environment shown in Figure 4.13. A convex decomposition with 603 cells is given. The large number of cells includes overlapping cells and multiple disjoint coverings of the free configuration space. Given specification of an arbitrary goal point, the algorithm automatically selects the individual cells and calculates the required parameters during policy deployment.

The simulation results of the global switched control policy are shown in Figure 4.13. Four different initial conditions, starting from two separate configurations, are simulated. The simulation demonstrates the policy-induced decision making that is inherent in the policy composition approach.

This chapter developed a policy design approach for fully actuated systems with input constraints. We design policies for both flow-through and convergent goals over convex polytopes; both kinematic and second-order systems are considered. These policies satisfy the composability requirements, enabling the policies to be deployed in our hybrid control framework. These composable policies enable a variety of discrete planning techniques; this demonstrates the flexibility of the policy composition approach. The examples range from simple sequence-based planning, to automatic synthesis of reactive automata based on high-level behavioral specification. The composition of the local policies induces different behaviors based on the initial conditions and the specific policy domains, without requiring re-planning. The drawback to these policy designs is that because the systems are idealized, the results are not directly transferable to more constrained systems.

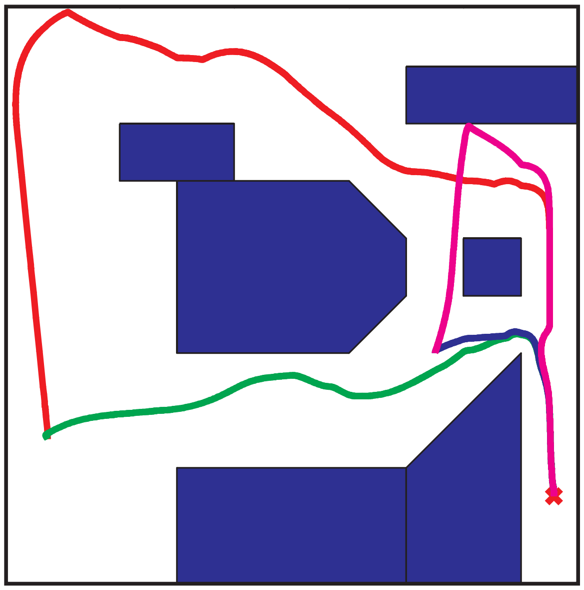


Figure 4.13: Automated deployment simulation for four separate initial conditions.

Chapter 5

Application to Single-bodied Wheeled Mobile Robots

This chapter extends our approach to integrating planning and control to single-bodied wheeled mobile robots. While the results from Chapter 4 show the promise of our hybrid control approach, the numerous constraints found in real mobile robots render the techniques for fully actuated systems inapplicable. This chapter presents the design of composable policies that apply to single-bodied wheeled mobile robots while satisfying the composability requirements of Chapter 3. This opens up the many symbolic reasoning techniques described in earlier chapters to more realistic systems with multiple interacting constraints.

This chapter begins with a discussion of some of the system constraints and the modeling framework used to address them. Our basic policy design approach is presented in the chapter's second section. The third section presents two specific policy designs that follow the basic approach and address the constraints for three wheeled mobile robot models. The chapter's fourth section discusses techniques that are used to verify that the policy designs satisfy the composability requirements. The chapter's fifth section discusses approaches to instantiating the generic policies, including a semi-automated approach. The sixth section discusses techniques for generating the prepares graph. The chapter concludes with a discussion of an approach to quantifying the relative completeness of the collection of policies.

5.1 System Constraints and Modeling Framework

Mobile robots introduce many complications that are not addressed by the policies for fully actuated idealized systems. This chapter focuses on policy designs that address three of these complications: body size/shape, nonholonomic constraints induced by wheels, and input constraints. This section discusses the implications of these constraints, and provides an overview of the modeling framework used in our control design. Appendix A provides a detailed overview of the framework, including formal definitions and derivations for the specific systems used in the this thesis.

5.1.1 Pose Space Constraints

The robot is a single convex body that moves in the planar workspace. The location of every point in the robot can be expressed relative to a single reference frame attached to the robot body. The robot *pose*, is locally represented by $g = (x, y, \theta)$, which specifies the position (x, y) and orientation ($\theta \in (-\pi, \pi]$) of this body-fixed frame relative to the global fixed reference frame in the workspace; see Figure 5.1. The pose space $\mathcal{G}$ is the space of all possible poses. In differential mechanics terms, the pose evolves on the $SE(2)$ manifold; see Appendix A for details.

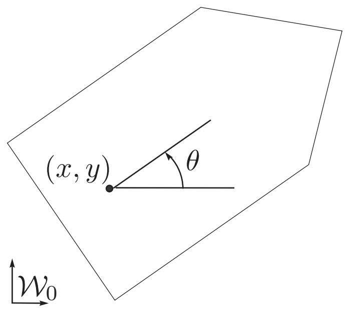


Figure 5.1: The robot *pose* is the position and orientation of the robot body, denoted $g = (x, y, \theta)$, relative to a global workspace frame $\mathcal{W}_0$. The *configuration* of the robot is $q = \{g, r\}$ where r represents internal variables specifying such things as steering angles and wheel rotations.

The obstacles in the environment constrain the set of admissible robot poses. For a pose to be collision free, the body must not intersect any obstacles in the environment. Let $R(g) \subset \mathcal{W}$ denote the workspace area occupied by the robot at pose g[1]. Stated formally, the requirement that the robot not intersect any obstacles at a given a pose is, for all obstacles, $R(g) \bigcap O_i = \emptyset$. The set of collision free poses, or *free pose space*, is denoted $\mathcal{G}_{\text{free}}$, where

$$\mathcal{G}_{\text{free}} = \left\{ g \in \mathcal{G} \mid R(g) \bigcap \bigcup_i O_i = \emptyset \right\} \subset \mathcal{G}.$$

The free pose space is the region that the robot must navigate through to reach its goal; that is, the control policy must induce pose velocities that keep the system within the free pose space during travel.

5.1.2 Nonholonomic Constraints

The robot is driven by wheels in contact with the ground; thus, the system is subject to nonholonomic constraints induced by rolling without slipping or sliding sideways[1, 22, 87, 94]. For example, the systems in this thesis cannot move instantaneously sideways relative to the forward facing pose. Many systems are also subject to steering bounds that require translation in order to rotate. These constraints limit the pose velocities to a sub-manifold of the full pose tangent space. The control policy design must induce pose velocities that respect the nonholonomic constraints, otherwise the control is not realizable on a given system.

To fully specify the position and motion of the robot, the drive wheel systems must be specified with variables in addition to the pose. These variables, denoted by r, are termed *shape*[2] variables [96]. Examples of shape variables include drive wheel rotation angles and steering wheel positions for Ackermann steered cars. Therefore, the robot configuration is fully specified by $q = \{g, r\}$; that is, the robot pose plus any necessary shape variables, r. Denote the shape space as $\mathcal{R}$. The shape variables do not directly impact the robot pose, and do not cause intersection with the workspace obstacles. Thus, the robot free configuration space is $\mathcal{Q}_{\text{free}} = \mathcal{G}_{\text{free}} \times \mathcal{R}$.

[1]Note, that $R(g) = R(q)$, where $R(q)$ was defined in Chapter 3. Only the pose portion of configuration impacts the set R.

[2]In differential mechanics, *shape* variables are also called *base* variables and the *pose* variables are termed *fiber* variables; see Appendix A for details.

This thesis is concerned with so called *purely kinematic systems* where there are sufficient independent nonholonomic constraints to constrain the equations of motion to a first order, or purely-kinematic, relationship between shape velocities and pose velocities [1]. That is, there exists a mapping $A(q) : T_r R \to \mathcal{T}_g G$ derived from the nonholonomic constraints such that $\dot{g} = A(q)\,\dot{r}$.

For purely kinematic systems, the control of pose velocities is induced via the shape variable velocities through the mapping $A(q)$. This thesis focuses on systems that directly control the shape velocities, that is $\dot{r} = u$ for $u \in \mathcal{U}$ a bounded input set; therefore, the pose velocities are $\dot{g} = A(q)\,u$. The mapping $A(q)$ is nonlinear, which renders the control problem of choosing inputs to generate a given pose velocity also nonlinear.

Example: Differential-drive robot

As an example, consider the case of a standard differential drive robot shown in Figure 5.2. The robot body pose is $g = (x, y, \theta)$. The robot is driven by two wheels in contact with the ground; denote the rotation of each wheel about its axle as $r = \{\psi_L, \psi_R\}$ for left and right wheels respectively. The vehicle is prevented from sliding sideways relative to its instantaneous heading, and each drive wheel is assumed to roll without slipping. These three independent nonholonomic constraints are represented as

$$\begin{bmatrix} \sin\theta & -\cos\theta & 0 & 0 & 0 \\ \cos\theta & \sin\theta & -c & -R & 0 \\ \cos\theta & \sin\theta & c & 0 & -R \end{bmatrix} \cdot \dot{q} = \mathbf{0}\,,$$

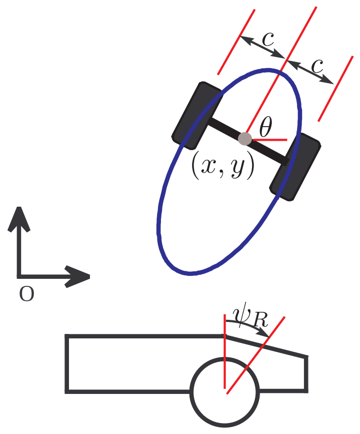


Figure 5.2: Differential drive robot with two drive wheels $\{\psi_L, \psi_R\}$ as shape variables and body pose given by $g = (x, y, \theta)$. The full configuration is $q = (x, y, \theta, \psi_L, \psi_R)$.

where $\dot{q} = \begin{bmatrix}\dot{x} & \dot{y} & \dot{\theta} & \dot{\psi_L} & \dot{\psi_R}\end{bmatrix}^T$, R is the drive wheel radius, and $2c$ is the vehicle *track*[3]. The derived mapping between shape velocities, $\dot{r} = \begin{bmatrix}\dot{\psi_L}, & \dot{\psi_R}\end{bmatrix}^T$, and pose velocities, $\dot{g} = A(q)\,\dot{r}$, is

$$A(q) = \begin{bmatrix} \frac{R}{2}\cos\theta & \frac{R}{2}\cos\theta \\ \frac{R}{2}\sin\theta & \frac{R}{2}\sin\theta \\ -\frac{R}{2c} & \frac{R}{2c} \end{bmatrix} . \tag{5.1}$$

5.1.3 Input Bounds

The shape variables and their derivatives, that is the shape velocities, may be bounded based on safety or mechanical limitations. For example, the electric motors commonly used to drive mobile robots have a top speed based on the motor characteristics and available voltage. Speed limits may also be imposed to provide safe operation in given environment. These speed limits may be direct limits to the shape velocities, or limits on the pose velocities, which are in turn mapped to shape velocity limits. In either case, the limits are ultimately applied to the control inputs. Shape velocity bounds map directly to input bounds for systems with $\dot{r} = u$.

Other limits may be imposed on the shape positions; for example, steering wheel positions may be mechanically limited. These constraints limit the shape space directly, and therefore the set of admissible shape velocities along the shape space boundary. That is, the shape velocities must not be outward pointing along the shape space boundary. Together, these constraints act to limit the allowable inputs available to the system at a given configuration.

The constraints interact to complicate the control problem. Since the induced pose velocities must keep the system in the free pose space, the pose velocities are constrained along the free pose space boundaries. These constrained pose velocities constrain the set of allowable shape velocities via the mapping $A(q)$; for kinematic systems, these constrain the set of admissible inputs. In defining the mapped pose velocity constraints, the policy must consider both pose velocities and “important” shape variables that modify $A(q)$. For some system models, such as the differential-drive system highlighted above, $A(q)$ only depends on the pose; other systems, such as automobiles, have $A(q)$ mappings that depend on both pose and shape variables. In these later cases, pose velocity constraints may induce constraints on the shape variables, which in turn will induce constraints on the shape velocities. That is, the shape velocities will be limited to those that do not lead to violations of the shape variables. Input bounds limit the set of achievable velocities, determined by the mapping $A(q)$.

5.2 Basic Design Approach

In the context of this framework, navigation refers to motion in the free pose space, while control is effected in the shape space. The mapping $A(q)$, which is derived from the nonholonomic constraints, connects the navigation and control problems. This section presents a basic policy design approach that makes use of the $A(q)$ mapping to build policies that satisfy the composability requirements set forth in Chapter 3. The policies specify inputs from the bounded input set that induce

[3]In mobile robotics, the distance between drive wheels is sometimes referred to as the *wheelbase*. In automotive terms the *track* is the distance between drive wheels and the *wheelbase* is the distance between the front and back wheels [16]. This thesis considers multiple robot models, including automobiles; therefore, the standard industrial terms are used.

a pose velocities $\dot{g} = A(q)\,u$ such that the system avoids obstacles and reaches a designated local goal.

Our approach defines the policies over collision free regions of pose space. We redefine the term *cell* to specify a region of free pose space that is used to define a local policy. The cells are designed such that under the influence of an associated control policy, the system is induced to move from any pose in the cell to a relatively smaller subset of the cell designated as the goal set. In this way, the cell defines a "funnel" that takes a relatively large region to a relatively small goal set. Figure 5.3 shows a conceptual example of a cell. The policy design specifies shape velocities that induce the desired convergence to a designated goal set within the cell. As the basic navigation problem addressed in this thesis is completely specified in terms of motion in the free pose space, specific trajectories in the shape space are not a primary concern, provided they satisfy the necessary constraints.

The cells, denoted $\Xi \subset \mathcal{G}_{\text{free}}$, are restricted to compact, connected, full dimensional subsets of $\mathbb{R}^3$ without holes. That is, the cells are defined in the local $\mathbb{R}^3$ chart of the pose space. The cell goal set is a subset of the closure of the cell; for flow-through policies the goal set is on the cell boundary. We assume the cell boundary is composed of parameterized surface patches defined by differentiable functions, such that an outward pointing normal is well defined almost everywhere. The surface patches define a continuous cover of the cell boundary; we refer to this boundary surface as a piece-wise parameterized surface.

In defining the local cells, there are several competing objectives. First, we desire cells with simple representations that are easy to parameterize. Additionally, we desire cells that have simple tests for inclusion so that the system can determine when an associated policy may be safely used. On the other hand, for a given policy goal set, we want the cell to capture as much of the free pose space as possible given the system constraints. That is, we want the policies to be "expressive." Given the system constraints, there is a trade-off between the size of the goal set and the size of the cell. Larger goal sets tend to allow larger cells; however, the goal sets should be small enough to be contained in another cell. This interplay between the goal sets and cells is one of the basic design challenges of defining composable policies. The cells should also admit feasible policy designs

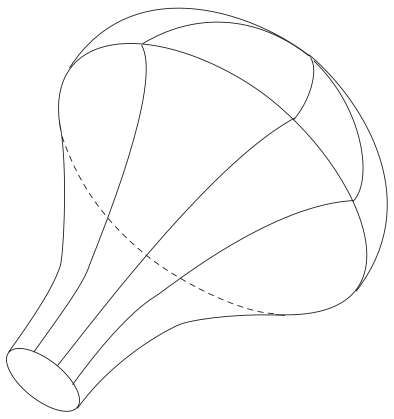


Figure 5.3: A schematic representation of a cell defined in the three dimensional body pose space. In this case, the goal set is the two-dimensional set of points at the small end of the cell.

over the entire pose space region. This thesis looks at a set of relatively simple cell designs, and demonstrates what can be accomplished with them.

The local policy is defined over the cell; that is, the cell defines the pose space domain of the local policy. Thus, the composability properties that hold for the policy domain must also hold for the cell. That is, the cell must be contained in the free pose space, and must be conditionally invariant under the influence of a local policy that brings any pose in the cell to its goal set in finite time. Furthermore, the cell should admit a simple inclusion test. These requirements on the policy domains provide limits on the freedom to define cells that result in valid policies.

Recall from Chapter 3 that the velocities on the domain boundary must point inward to induce conditional invariance, except at the goal set for flow-through policies. Thus, for all poses g in the cell boundary, there must exist an achievable pose velocity $\dot{g}$ such that $\mathbf{n}\left(g\right)^T \dot{g} < 0$, where $\mathbf{n}\left(g\right)$ is the outward pointing normal. Using the mapping $A\left(q\right)$, where $q = \{g, r\}$, this pose velocity constraint is mapped to a half space constraint on the shape velocities; that is $n^T A\left(q\right)\dot{r} < 0$. If there does not exist such a valid $\dot{r}$ in the bounded shape tangent space, then the cell is not valid. For kinematic systems, where $\dot{r} = u$ with $u \in \mathcal{U}$ a bounded space, $n^T A\left(q\right) u < 0$ provides an additional input constraint along the cell boundary.

Arbitrary cell shapes are not allowed; a cell can only be valid, if there exists a valid input that keeps the velocity inward pointing on the cell boundary. Let $\omega = \mathbf{n}\left(g\right) \cdot A\left(q\right)$ and ω^- be the negative half space in $\mathcal{U}$ defined by ω and the origin of $\mathcal{U}$; this is shown in Figure 5.4. Along the cell boundary, any $u \in \omega^- \bigcap \mathcal{U}$ renders the cell conditionally invariant. Because the ω constraint is a function of both the pose along the boundary and the boundary normal, the boundary constraint that $\omega^- \bigcap \mathcal{U} \neq \emptyset$ limits the size and shape of the pose space cell.

The constraints described previously provide absolute limits on the size and shape of a given cell. Additional limitations will be imposed by the specific policy designs. Given a particular cell, the aim of control policy design is to specify a mapping from any pose in the cell to an input in the bounded input space that induces the desired behavior. That is, we seek a mapping $\Phi : \Xi_i \rightarrow \mathcal{U}$ such that the induced flow enters the designated cell goal set in finite time while satisfying conditional

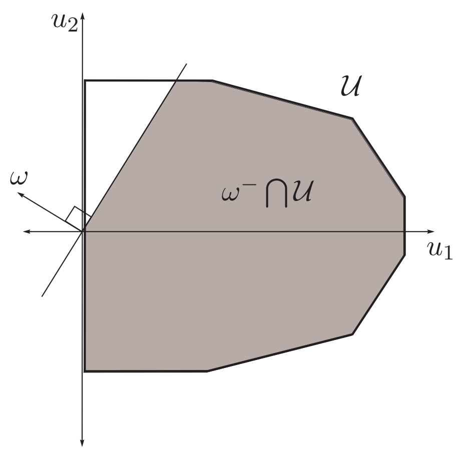


Figure 5.4: The conditional invariance requirements for kinematic systems induce constraints on the input set. The cell size and shape is limited such that $\omega^- \bigcap \mathcal{U} \neq \emptyset$ for $\omega = \mathbf{n}\left(g\right) \cdot A\left(q\right)$ for all g in the cell boundary.

invariance. Defining a valid policy requires verifying that the policy design being applied over a particular cell satisfies the composability requirements of Chapter 3. In other words, a given policy design may fail to take full advantage of the capabilities of the system and further limit the size and shape of the cell.

For the fully actuated idealized policies described in Chapter 4, the policy design was accomplished by defining a reference vector field over the cell such that the vector field flow entered the goal set. The mapping from configuration to input was based on the reference vector at a given configuration. The vector field serves to encode the desired behavior over the entire cell. For nonholonomic systems, the nonlinear relationship between inputs and pose velocities makes it difficult to define a reference vector field over a given pose space cell.

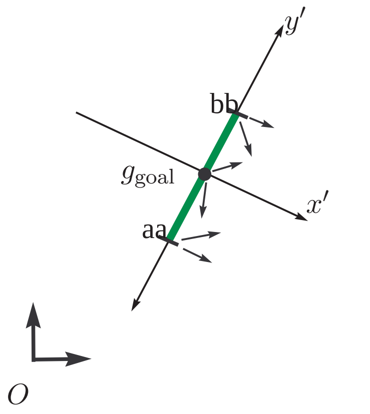
Our approach for purely kinematic systems is to define a set of input constraints for each pose in the cell. By carefully defining the constraints, any input chosen from the constrained input set $\mathcal{U}$ induces a behavior that satisfies the requirements of Chapter 3. We take the constraint definition approach for two reasons. First, defining a continuous vector field that satisfies the nonholonomic constraints is difficult due to the complex cell shapes and nonlinear constraints. It is easier to define a set of pose velocity constraints; $A(q)$ provides a convenient mapping between pose velocity constraints and input constraints. For example, consider the boundary constraints $n^T A(q)\, u < 0$. As the inputs are not subject to the differential constraints, the selection of a valid input is generally easier than defining the specific pose velocity. Second, numerous constrained optimization techniques allow additional information to be incorporated into the choice of a control input from the valid set. For example, we can construct a cost function that penalizes rapid input changes but rewards moving fast and maximizing the distance from the defined constraint. Thus, while any $u \in \omega^- \bigcap \mathcal{U}$ is valid for a given ω constraint, constrained optimization allows us to select the “best” input according to some cost function.

5.3 Generic Policy Designs

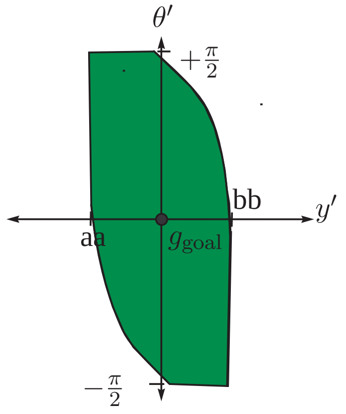
This section presents two generic policy design approaches that satisfy the composability requirements for purely kinematic systems. This section gives an overview of the specific designs; the following section discusses how the composability requirements are verified.

Both generic policy designs defined below are flow-through style policies defined over pose space cells. As stated above, each cell Ξ_i is a subset of $\mathbb{R}^3$, which represents a local chart of the pose space 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. See Figure 5.5 for details.

Given a generic cell specified in the local coordinate frame, the cell is instantiated in the pose 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 xy-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 goal. Thus, the local x'-axis is normal to the cell goal set, and outward pointing with respect to the cell boundary; we refer to the orientation of the x'-axis as the *goal set heading*. Although these choices constrain the policy freedom, they make specifying the free parameter values more tractable by limiting the choices. Let $\theta_{\mathrm{goal}} \in (-\pi, \pi]$ specify the orientation of the x'-axis, and let $g_{\mathrm{goal}} = \{x_{\mathrm{goal}}, y_{\mathrm{goal}}, \theta_{\mathrm{goal}} + \Delta\theta\} \in \mathbb{R}^3$. For forward motion, the x'-axis is aligned with the



a) Goal set projected onto plane



b) Goal set plane.

Figure 5.5: Definition of goal set. a) projection onto xy-plane with some reference velocity vectors shown, b) goal set plane demonstrating restriction imposed by continuity. The six smaller arrows denote valid velocities that cross the goal set. Points 'aa' and 'bb' are shown in both figures for reference.

robot heading, and $\Delta\theta = 0$. For reverse motion, the x'-axis is opposite the robot heading, therefore $\Delta\theta = \pm\pi$.

The required invariance properties for flow-through policies lead to some guidelines when defining a valid goal set. The goal set defined as described above appears as a line segment when projected onto the xy-plane. Consider Figure 5.5-a; the system crosses the goal set as indicated by the smaller arrows. For a flow through policy, the orientation of the body velocity vector must be within $\pm\frac{\pi}{2}$ of the goal set heading. For systems whose instantaneous forward body velocity is along the axis defining the body orientation, the restriction is $\theta \in \left[\theta_{\text{goal}} - \frac{\pi}{2} + \Delta\theta, \theta_{\text{goal}} + \frac{\pi}{2} + \Delta\theta\right]$. On the goal set boundary, the instantaneous pose velocity cannot be transverse to the goal set boundary; that is, any velocity component not orthogonal to the goal set must be pointed into the goal set. For the systems considered in this thesis, the instantaneous pose velocity is along the body heading, which puts a restriction on the velocities at the endpoints, as shown in Figure 5.5-b. The system constraints and continuity restrictions will impose additional constraints on the goal set; these are conceptually represented by the arcs that bound the goal set in Figure 5.5-b. The exact shape depends on the specific policy design. In general, the goal set is "tilted" relative to the xy-plane.

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

$$g(p) = \begin{bmatrix} \cos\theta_{\text{goal}} & -\sin\theta_{\text{goal}} & 0 \\ \sin\theta_{\text{goal}} & \cos\theta_{\text{goal}} & 0 \\ 0 & 0 & 1 \end{bmatrix} p + \begin{bmatrix} x_{\text{goal}} \\ y_{\text{goal}} \\ \theta_{\text{goal}} + \Delta\theta \end{bmatrix} . \tag{5.2}$$

Although similar to a homogeneous transform, the non-standard transformation (5.2) 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.

A key feature of these policies is that a valid policy may be relocated to a new goal point within the free pose space without impacting the conditional invariance, finite time convergence, or simple inclusion test properties; the safety of the policy with respect to collision must be verified a the new reference point. The proofs of these properties are based on simple calculations that are invariant under rigid body transformation within the pose space. This means that once these properties are verified for a given set of policy parameters, the policy may be relocated by re-specifying the goal set center. As a policy cell is relocated, the requirement that the cell lie within the free pose space must be verified. For the simple inclusion tests, special consideration must be taken as the cell orientation dimension does not wrap around at $\pm\pi$ and can be place at arbitrary g_{goal} poses; for these cells in this thesis, it is normally sufficient to tests inclusion based on $g = (x, y, \theta + 2n\pi)$ where $n \in \{-1, 0, 1\}$.

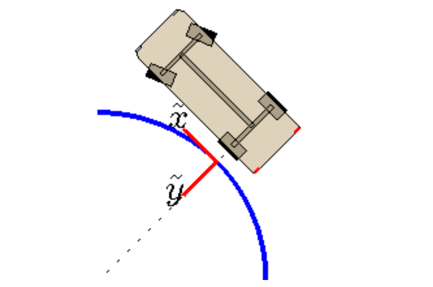
We now present a high level overview of two families of generic policies. The policies have free parameters that govern the specific size and shape of the policies; for valid parameter values, the policies are composable within our hybrid control framework.

5.3.1 'PF' Policy Design

The first family of generic policies is based on workspace path segments, which are used to define a parameterized cell in pose space. The path segment is lifted to a curve in pose space by considering the orientation of the path tangent as the desired system orientation, with the goal set center at one end of the pose space curve. A 'tube' is defined around the pose space curve to define the cell boundary. Figure 5.6 shows an example of this cell.

The policy design is based on a variable structure control approach to path following, which gives the policy its 'PF' name [4]. The PF policy defines a "sliding" control surface within the cell boundary tube. For a given robot model, the policy defines a control strategy that causes the system to steer toward the sliding surface, then along the sliding surface towards the pose space curve and the goal set center. For kinematic systems, the sliding surface defines a velocity constraint at each pose; the constraint is then mapped to a constraint on the input space, where a simple optimization is used to chose a valid input.

In addition to the goal set center, the policy free parameters include the width of the tube, the curvature of the workspace path, arc length of the path segment, and shape of the sliding surface. Valid values of the free parameters, that is values that induce composable behavior, are limited by the input constraints and specific robot system model. Thus, the composability requirements must be verified for a given set of parameter values



a) Workspace path with local frame defined

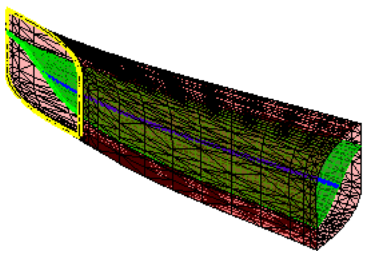

b) Cell boundary and "sliding" control surface.

Figure 5.6: Control policy based on path following control law given in [4].

See Appendix D for details about the functions used to define the cell boundary and sliding surface, derivations to show correctness, and techniques to verify that PF policies satisfy the three composability requirements.

5.3.2 'SQ' Policy Design

The second family of policies are defined using level sets of super quadric functions, which gives the policy its 'SQ' name. Two functions are used: the first expands from the goal set along the central axis, and the second caps the cell. Figure 5.7 shows a schematic representation; Figure 5.9-a shows a 3D representation of a cell with specific parameter values. The cells are naturally funnel shaped. Free parameters govern the size of the goal set ellipsoid, overall length (ζ_M), and expansion along the central axis defined by the $\rho(\zeta, \gamma)$ function, where $\zeta \in [0, \zeta_M]$ and $\gamma \in [-\pi, \pi]$ are parameters that specify a given point on the cell boundary surface.

Given the basic cell definition, we define two different control strategies for inducing convergence to the goal. The first is the variable structure control approach described in Section 5.3.1. The second, which works for systems where $A(q)$ does not depend on shape variables, is based on a family of super quadric functions that define level sets as shown in Figure 5.8. The normal, $n(g)$, of the level set passing through the pose g is used to define a constraint $\omega(g) = n(g)^T A(g)$ on the input space. Using any $u \in \omega(g)^{-1} \bigcap \mathcal{U}$ as the control action drives the system towards the inner level sets, and then towards the goal. The size and shape of the cell is limited by the constraint that $n(g)^T A(q) \bigcap \mathcal{U} \neq \emptyset$ for all level sets.

See Appendix E for details about the policies, discussion of correctness, and techniques used to verify the composability requirements.

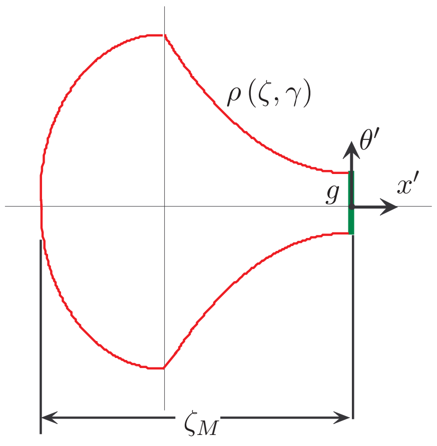


Figure 5.7: Schematic of curves used to define super quadric based cell.

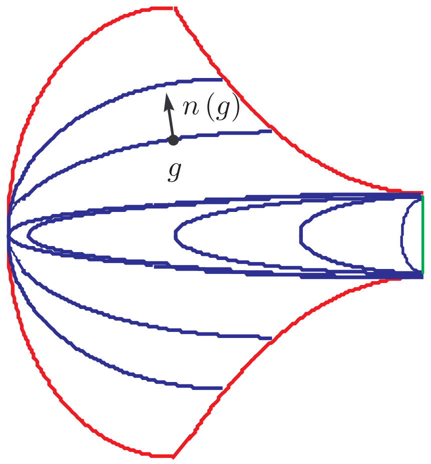


Figure 5.8: Schematic of super quadric level sets used in control policy.

5.4 Policy Validation

For a policy to be valid, the specific policy design and specific parameter values that define cell size and shape must satisfy the composability requirements from Chapter 3. That is, the cell that defines the policy domain in pose space must be contained in the free pose space. The policy domain must be conditionally invariant under the influence of a local policy that brings any pose in the cell to its goal set in finite time; that is, the induced pose velocities cannot cause the system to exit the cell except via the goal set, and the shape variable velocities cannot cause the system to exceed the shape variable constraints. Furthermore, the policy domain should admit simple inclusion tests; this requires that the cell has simple inclusion tests with respect to a local pose. The methods used to validate the policy/cell combination necessarily vary with the specific policy design, cell definition, and the system model under consideration. This section provides an overview of some of the approaches used to validate specific policy instances.

First, the cell must be completely contained in the free pose space so that it is collision free. One can imagine constructing the free pose space boundary given the size and shape of the robot body and known work space obstacles, and then checking for intersection between the free pose space boundary and the poses within a given cell. We avoid this complexity by inverting the problem, and verifying that the cells are completely contained in the free pose space based only on workspace measurements; that is, the free pose space is never explicitly defined. Let $R(\Xi_i)$ denote the union of points in workspace occupied by the robot body for all poses within a given cell Ξ_i; that is $R(\Xi_i) = \bigcup_{g\in\Xi_i} R(g)$. In other words, $R(\Xi_i)$ is the swept volume of $R(g)$ over all $g \in \Xi_i$. If $R(\Xi_i)$ does not intersect any obstacles, then the cell is free of collision; that is, the cell is contained in the free pose space. $R(\Xi_i)$ can be determined by expanding the cell boundary to account for the body shape, and mapping the expanded surface to the workspace; see Appendix C for details and a proof or correctness. Figure 5.9 shows an example cell, its expansion for a particular robot body size and shape, and its mapping into the workspace. The size and shape of the cell is limited by the obstacles in the environment.

This thesis develops a tractable approach to expanding the cell and testing for collision given a mesh representation of the cell boundary surface. The approach uses an analytic mapping from a cell boundary point to the corresponding point on the expanded cell boundary, and a mixture of

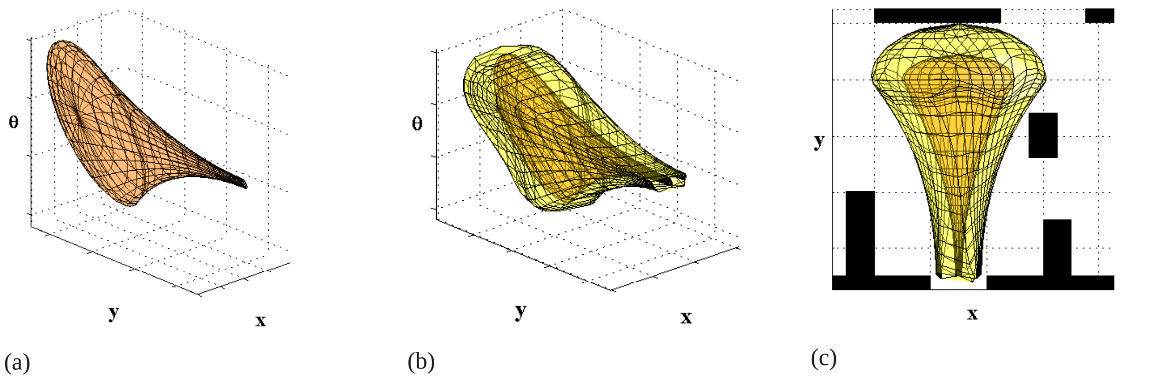


Figure 5.9: Testing that cell is contained in the free pose space. a) cell, b) expanded cell that accounts for body shape, c) projection into workspace as $R(\Xi_i)$. The projection of the cell boundary is shown as the darker inner surface.

exact tests and approximate tests to ensure the safety of a given cell without being overly conservative. First, the cell surface is represented with a surface mesh. Second, each vertex in the mesh is expanded using the point-by-point analytic mapping. This results in a mesh of the expanded cell surface, which is then projected to the workspace. For a triangular surface mesh, the projection results in a collection of overlapping workspace triangles. If a workspace triangle vertex is contained in an obstacle then collision occurs, and the cell is unsafe. This is an exact test, based on the point-wise analytic mapping. If an obstacle vertex (assuming a polygonal workspace) is inside a workspace triangle, then collision may occur, and we assume that the cell is not within the free pose space. If all the workspace triangles and obstacle vertexes are collision free, then the cell may be free. If the workspace obstacles are expanded by the maximum error of the expanded cell mesh approximation, then the approach guarantees the safety of the cell. The point-by-point analytic mapping, proof of correctness, and details about the approach to collision testing are presented in Appendix C.

A policy defined over the free space cell is safe provided the cell is also conditionally positive invariant under the influence of the policy. Conditional invariance requires that velocities along the cell boundary, excluding the goal set, are inward pointing. For a given policy specification, this property must be verified.

As an example, consider a kinematic differential-drive system such that $\dot{g} = A(q)\,u$. The mapping $A(q)$ does not depend on shape variables; so we will use the shorthand notation $A(g) = A(q)$. Let $u = \Phi_{(\Xi_i,\mathcal{U})}(g)$ denote the chosen input of a control policy defined over a specific cell with a specific input set $\mathcal{U}$. Assume the cell boundary is defined by a surface parameterized by ζ and γ. The conditional invariance requirement is that over the entire cell boundary, that is for all ζ and γ,

$$n(g(\zeta,\gamma))^T A(g(\zeta,\gamma))\,\Phi_{(\Xi_i,\mathcal{U})}(g(\zeta,\gamma)) < 0\,. \tag{5.3}$$

For each point $g(\zeta,\gamma) \in \partial\Xi_i$ define

$$L(\zeta,\gamma) = n(\zeta,\gamma)^T A(g(\zeta,\gamma))\,\Phi_{(\Xi_i,\mathcal{U})}(g(\zeta,\gamma))\ , \tag{5.4}$$

where $\mathcal{U}$ is the bounded input set associated with this policy. The more negative the value of L, the better the input respects the constraint given in (5.3). Although non-linear, the function $L(\zeta,\gamma)$ is piecewise smooth and generally "well-behaved" for the mapping $A(q)$. For a valid cell, condition (5.3) must hold over the entire boundary; therefore, $L(\zeta,\gamma) < 0$ for all ζ and γ. 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. In the worst case over the entire boundary, a valid cell satisfies the constraint

$$\max_{\zeta,\gamma} L(\zeta, \gamma) < 0 \,. \tag{5.5}$$

If condition (5.5) is satisfied for a given cell, under a given control policy, then the cell is conditionally invariant.

Figure 5.10 shows a typical $L(\zeta, \gamma)$ surface for the cell shown in Figure 5.9-a. This surface is constructed by sampling the $\{\zeta, \gamma\}$ parameter space, and calculating L. 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 xy-plane; that is the component in the θ direction is zero. As the θ component of the boundary normal crosses zero, the value of L depends only on the instantaneous heading. On either size of the horizontal normal, the θ component changes sign; thus, the rate of turn that minimizes L flips between positive and negative steering, which induces the ridges. As shown, this policy results in a conditionally invariant domain.

While this example is for a specific control policy, similar approaches exist for the control policies defined in this thesis. Appendices D and E discuss the validation approaches taken for the specific policy designs and systems used in this thesis. The evaluation functions are piecewise continuous, which lends itself to sample based and numeric optimization techniques for validation. The sample based tests are exact on a point by point basis; therefore, a coarse sampling is used for preliminary tests, and a fine resolution sampling used for final validation of a given policy specification.

5.5 Policy Instantiation

The generic policies must be instantiated in the free pose space before they can be used in the hybrid control framework. This involves specifying the free parameters and verifying that the composability requirements are satisfied. The policies must then be tested for the prepares relationship with other policies in order to construct the prepares graph used by the discrete planning. In Chapter 4, this process was automated given a decomposition of the free workspace into polytopes; the fully actuated idealized point systems makes this possible. In this chapter, the non-circular body shape

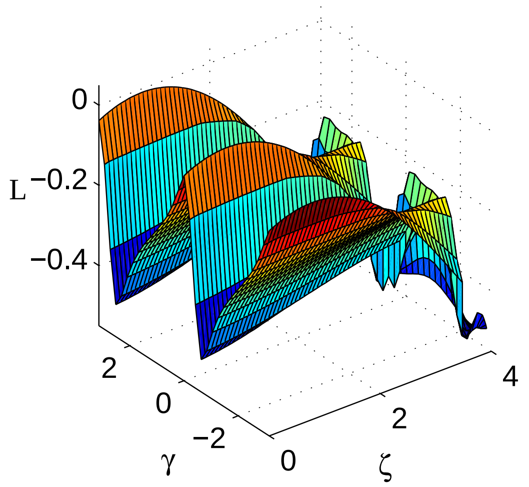


Figure 5.10: Constraint surface for $L(\zeta, \gamma)$ from (5.4).

means that a simple decomposition is not available, and the constrained dynamics preclude the use of many simple cell shapes.

Our approach for nonholonomic systems is to define the policies over cells in the pose space. The cells must be placed so that they are completely contained within the free pose space. The placements must also guarantee that the policy domains are sufficiently interconnected via the prepares relationship to address a given navigation problem.

The approach taken in this chapter is to define some policy domains first, and then attempt to fill the free pose space by incrementally adding policies that capture more free space and prepare the existing policies. This section describes two approaches to addressing this challenge: a manual approach and a semi-automated approach.

The most basic approach is to instantiate each policy individually by specifying values for its free parameters. Given a palette of generic policies, a specific generic policy is chosen. A reference point, typically the goal set center, is then assigned. From there, the parameters that determine the goal set size, and cell size and shape are specified. This approach is based on trial and error. Given a set of parameter values, the validity of the policy must be verified. This involves testing that the cell is contained in the free pose space and demonstrating that the conditional invariance constraints are satisfied on the cell boundary. Policy parameter values are then modified on an as needed basis. Using Matlab$^{\text{TM}}$code developed for this thesis, the policy validation steps generally took several minutes per policy and parameter set combination. Thus, this trial-and-error based manual approach can be time consuming.

By taking advantage of the invariance of the policies under rigid body transformation, we develop a semi-automated approach to policy instantiation. This is possible because multiple copies of a policy can be instantiated once the conditional invariance and convergence properties are satisfied; only collision must be checked with each transformation.

First, using the manual approach described above, we instantiate a collection of policies relative to a common reference pose.

Definition: Cache of Policies: A collection of policies instantiated relative to a common reference pose.

The policy cache should contain policies with a variety of domain sizes and shapes that have been validated for conditional invariance and finite time convergence. The policies are defined relative to a common reference pose that is not necessarily the goal set center. In this way, policies within the cache can prepare one another. The policies within the cache are instantiated in the free pose space by rigidly transforming them relative to the reference pose placed at a given free pose. Figure 5.11-a shows a schematic example of a cache.

Policies from the cache are placed in the pose space relative to specific reference points. The goal set center of each policy is transformed based on the relative transformation between the cache reference point and the pose space reference point. Collision testing using the expanded cell approach, which is automated, is used to discard cells that intersect an obstacle. The cache should contain a variety of cell sizes to cover small regions when the reference is near an obstacle, and large regions when the reference point is away from an obstacle. Figure 5.11 shows a schematic representation of the instantiation process. Given a cache of valid policies and a collection of reference poses, the instantiation within the free pose space is automated. The resulting collection of instantiated policies is the suite of policies, first introduced in Chapter 3. In addition to policies deployed from the cache, policies may be added to the suite manually.

By carefully defining the policy domains in the cache, and deploying the policies on a regular grid of pose space reference points, the policies can be made to systematically prepare one another. In this way, the instantiated policies are guaranteed to satisfy the prepares relationship in predictable

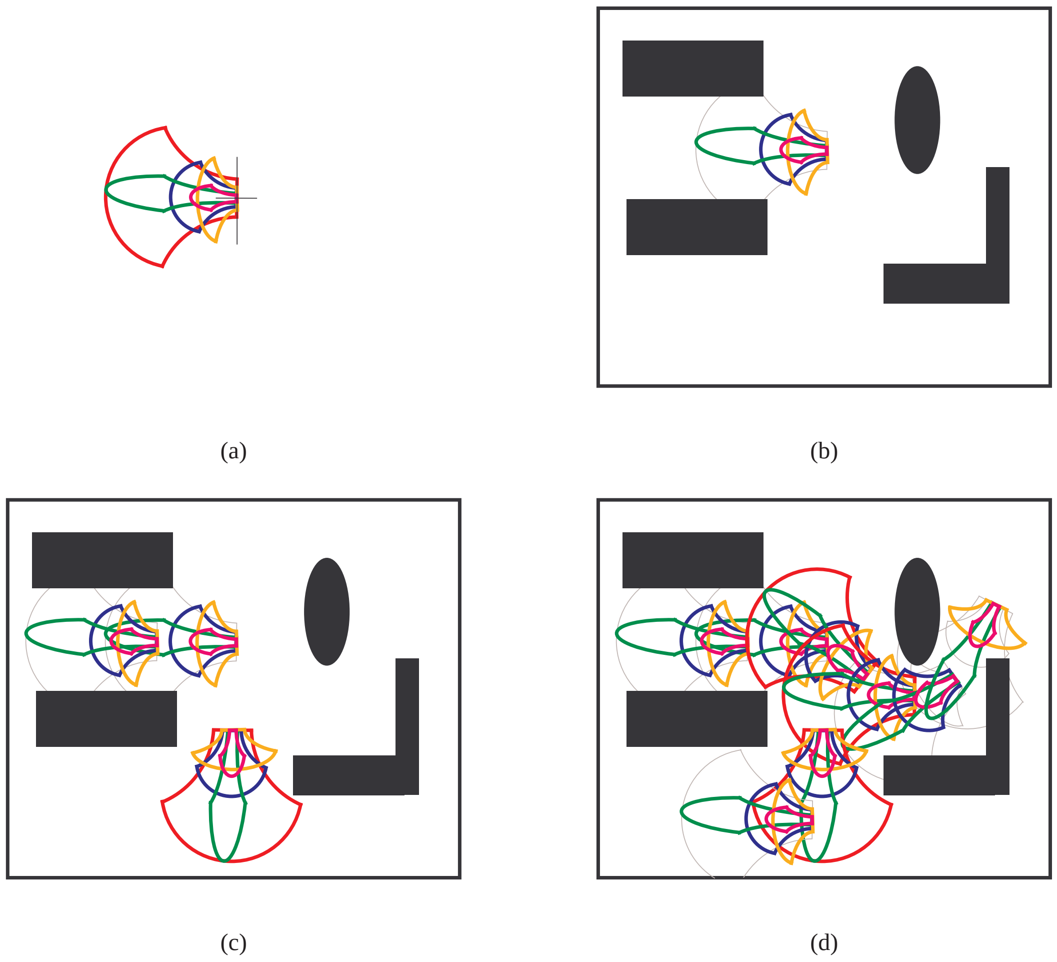


Figure 5.11: a) Cache of policies - a collection of policies instantiated relative to a common reference pose. b) The cache of policies are instantiated a given reference pose. Policy domains that collide with obstacles are discarded, as shown by the thin light gray lines. c) Policies are instantiated at three reference poses by copying and transforming the cached policies. d) Suite of policies - the collection of collision free policies instantiated in the pose space of the robot. In this final example, eight local reference poses are used.

manner. The reference pose grid spacing is chosen relative to the size of the domains in the cache. Finer reference pose grid spacings lead to better pose space coverage because the cells can be placed closer to obstacles without colliding; this results in more policies with significant cell overlap. The overlapping cells can result in more options for the planning system, thereby providing more flexibility; however, the extra flexibility comes with an increased computational burden both in the instantiation phase as well as the planning phase.

5.6 Prepares Graph Generation

In order to do discrete planning with the suite of policies, our hybrid control approach requires the prepares graph that encodes the discrete transitions between policy domains. Given a policy suite, this section discusses approaches to defining the prepares graph that are tractable. The approaches can be automated, making them suitable for use with our semi-automated instantiation technique.

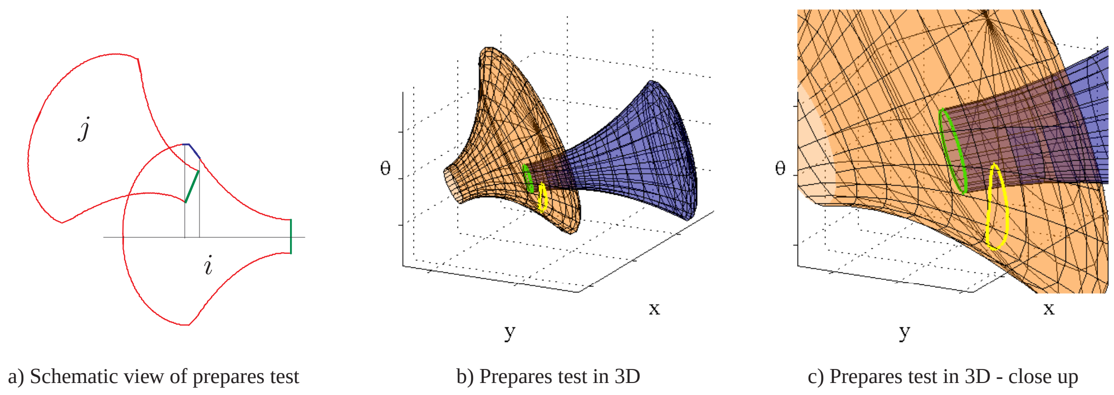
For kinematic systems, the prepares test is based on verifying that the configuration goal set of one policy is contained in the domains of other policies. The policies defined in this thesis for purely kinematic systems are defined across the range of shape variable values; thus, the prepares test only needs to consider the pose variables. Stated differently, at any pose in a cell, the policy is defined for any value of the shape variables. This allows the prepares test for the policies in this thesis to be evaluated by comparing the cell goal set of one policy with the cell boundaries of other policies. A policy prepares another policy if the cell goal set in pose space is completely contained in the other policy's cell; the extended prepares relationship holds for the union of cells.

The prepares tests for purely kinematic systems requires that every point in the goal set lie within a cell or set of cells. As the cells are defined to be compact connected regions that are isomorphic to a ball, and the boundaries of the cells are piecewise smooth functions, this can be verified by comparing the distances from goal set points to the boundary of other cells. This section describes three sample based approaches, and then discusses their extension to more accurate numerical approaches. For the policies defined in this thesis, the first two approaches were reliable, so the third approach, which is more thorough, was not implemented. This section first discusses the approach for testing one cell goal against a single cell; the extension of the techniques to sets of cells is discussed later.

Consider testing two cells to verify whether Φ_j prepares Φ_i. All of the prepares tests begin by verifying that g_{goal_j} lies within the domain of Φ_i; this is a simple inclusion test that $g_{\text{goal}_j} \in \Xi_i$, where Ξ_i is the cell associated with policy Φ_i. If this single point test fails, then Φ_j cannot prepare Φ_i. Next, we sample the goal set boundary based on the parameterization of the specific policy under consideration. The piecewise smooth functions allow an error bound to be determined, which can be used to guide the sampling resolution and determine a safety threshold for the second and third test procedures.

Given the sampling of goal set boundary points, we now discuss the three procedures used to verify the prepares relationship between two policies. The first test procedure simply verifies that all the sample points on the boundary of Φ_j's goal set are included in Φ_i. For the relatively simple cell shapes considered in this thesis, this test proved reliable for reasonable sample counts. The second prepares test casts each boundary point ${}^{j}g_k$ into a local frame of Ξ_i along the central axis. An example of this test is shown in Figure 5.12. Given the local coordinates of ${}^{j}g_k$, the corresponding point on $\partial\Xi_i$ is found. Provided the distance from the central axis to ${}^{j}g_k$ is less than the distance from the central axis to the corresponding point on $\partial\Xi_i$, the point ${}^{j}g_k$ is in the cell Ξ_i. The benefit of this second test is that it provides a distance from the goal set boundary to the cell boundary, which can allow the approach to define a safety margin. This second test can use numerical techniques to find the minimum distance from goal set boundary to the other cell boundary. This approach does not guard against an oddly shaped cell whose boundary bends in a way that removes a portion of the goal set, without intersecting the goal set boundary; while unlikely, this shortcoming motivates the third approach.

The third procedure checks to see if each point along a ray, $\vec{r}\left({}^{j}g_k\right)$, from the goal set center g_{goal_j} to the sample boundary sample point ${}^{j}g_k$ lies in the cell of Φ_i, as shown in Figure 5.13. Let ${}^{j}\ell_k = \left\|\vec{r}\left({}^{j}g_k\right)\right\|$ be the distance form the goal set center to the goal set boundary point. The approach determines the point of intersection of the ray along $\vec{r}\left({}^{j}g_k\right)$ with the boundary of Φ_i; let the distance of this point from g_{goal_j} be ${}^{i}\ell_k$. In the case of multiple intersections, the minimum distance is chosen. Since g_{goal_j} is contained in Φ_i, the intersection will exist, so that ${}^{i}\ell_k$ is well defined. If ${}^{j}\ell_k < {}^{i}\ell_k$ then every point on the ray is in Φ_i. If $\min_k \left({}^{i}\ell_k - {}^{j}\ell_k\right) > \epsilon$, where $\epsilon > 0$



j
i
θ
y
x
θ
y
x
a) Schematic view of prepares test
b) Prepares test in 3D
c) Prepares test in 3D - close up

Figure 5.12: Prepares test for policy deployment. In the three-dimensional figures, the goal set of Φ_j and the corresponding points on the surface of Φ_i are traced in lighter colors.

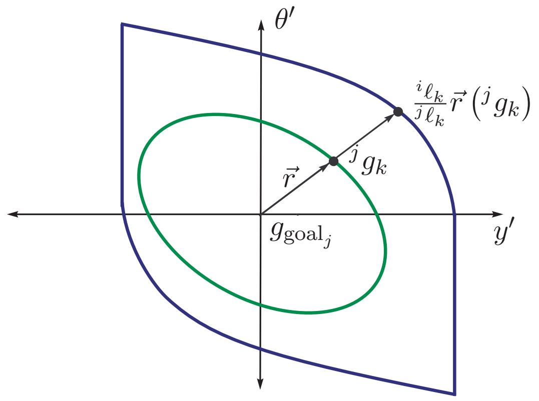


Figure 5.13: Schematic of the third prepares test, where ${}^j g_k$ is the k^{th} point on the goal set boundary of policy Φ_j, $\vec{r} = ^j g_k - g_{\text{goal}_j}$, and ${}^j\ell_k = \|\vec{r}\|$. The intersection of the ray along $\vec{r}$ with the boundary of Φ_i occurs a distance ${}^i\ell_k$ from g_{goal_j}.

is the safety threshold, then $\Phi_j \succeq \Phi_i$. This test depends on a tractable method of find the boundary intersection with $\vec{r}$, which necessarily depends on the specific cell designs. Given the intersection point, and piecewise smooth boundary surfaces, the test lends itself to numerical procedures to find the minimum distance between the goal set boundary and $\partial\Xi_i$. The specific numerical details are beyond the scope of this thesis.

If the goal set is not contained in a single policy, a restricted version of the extended prepares test ($\Phi_j \succeq \{\Phi_i\}$) is used to test for inclusion in a set of policy domains. The restriction, which simplifies implementation, requires that each policy in the union contains the goal set center. Then, as long as each goal set boundary sample point passes a prepares test for at least one cell in the union, we assume that $\Phi_j \succeq \{\Phi_i\}$, where $\{\Phi_i\}$ denotes a set of policies. When two consecutive sample points switch between policy domains that contain them, the sampling can be refined to guarantee that no gaps are found.

These tests for inclusion can be automated so that the prepares graph can be generated for a given collection of cells. To limit the algorithm complexity, we chose to limit the number of cells considered in the union to three. The automated algorithm first defines a set of possible prepared policies for a particular policy Φ_j as the set of policies containing the goal set center of Φ_j. For this subset of policies, the sampled boundary points are tested for inclusion. Any policy that contains all boundary points according to one the three test procedures is prepared by the given policy. Next, subsets of two policies that contain all boundary points are collected; that is, the given policy prepares the union of these policies. Finally, subsets where three policies contain all goal points are collected. By considering all policies in the suite in turn, the prepares graph is automatically generated. During this process a heuristic cost can be assigned to each prepares graph edge based on some defined cost metric.

The computation cost and accuracy are related to the number of sample points along the goal boundary. During manual deployment, a coarse sampling is used to guide the initial parameter specifications[4]. For final prepares graph generation, a fine sampling along the boundary is used. With several hundred policies, the generation of the prepares graph takes hours using code developed in Matlab™. Suites of tens of thousands of policies sometimes took days to generate the prepares graph; the time varied depending on the relative interconnectedness of the resulting prepares graph. Thus, both the manual instantiation and prepares graph generation represent substantial time and computational investments. This upfront cost is justified based on the planning flexibility demonstrated in the next chapter.

5.7 Relative Completeness Quantification

As described above, this approach is not complete; that is, it does not guarantee that the free pose space is covered. There is a trade-off between relative completeness and the number of policies used for planning. Increasing the number of policies increases both the upfront computational costs and the cost of planning and replanning. A relatively small number of simple policies is unlikely to provide good coverage of a complicated workspace. The question then arises, "how does one quantify the relative completeness in order to evaluate one deployment over another?" As a measure of relative completeness, we define the *coverage fraction*.

Definition: Coverage Fraction: The fraction of free pose space covered by the suite of policies.

While the definition is straight forward, calculating this in closed form is intractable using current techniques, if not impossible in general. In this section we demonstrate an effective sample-based method for estimating the coverage fraction, and discuss opportunities for using the coverage fraction to guide policy instantiation.

The sample-based approach is based on Monte Carlo methods [91]. To begin, a regular sampling grid is defined over the $\mathbb{R}^3$ representation of pose space. The sampling grid spacing is chosen based the relative size of the features in the world and the robot. A randomly chosen sample point is taken from the grid and the robot is tested for collision at that pose. In order to avoid over sampling one region, we define points on a regular grid so that each sample can be tracked and only used once. If the robot is free of collision at the sample pose, then that pose is in the free space. The sampling continues until a user-determined number of free pose samples is collected. This process gives a sampling of the free pose space. From the collection of free pose samples, a smaller number is randomly sampled and tested for inclusion in the domain of any cell in the suite of policies. The coverage fraction is then estimated as

$$C_f = \frac{\text{\# included free poses}}{\text{total \# sampled free poses}} .$$

[4]During manual instantiation, the three-dimensional cells can also be visualized to provide visual confirmation of the prepares relationship and guide parameter selection.

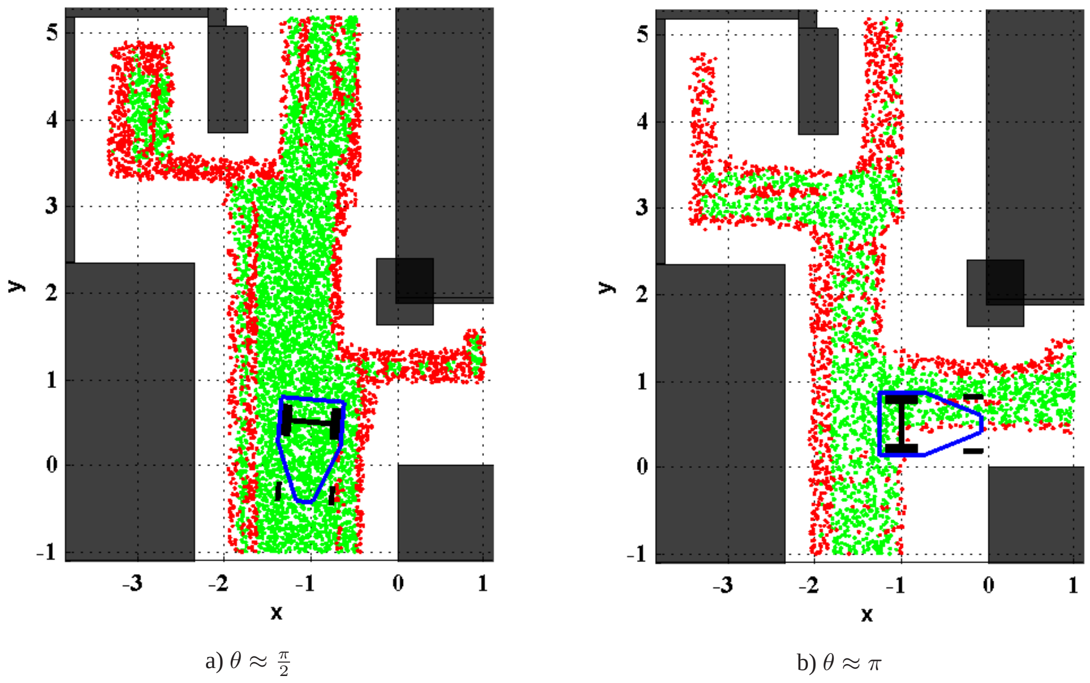


Figure 5.14: Plots showing the sampled points for a narrow range of orientations shown by the robot. The dots represent free poses; lighter green represents poses included in at least one policy and darker red represents poses that are not captured by at least one domain. The policies are deployed on a regular 0.30 meter grid spacing in this example.

Figure 5.14 shows an example of the sampling by projecting the poses for narrow bands of orientation into the workspace. Most of the missed poses appear along the boundary of the free pose space.

For the results described in this subsection, a specific policy cache is deployed on a regular reference pose grid using the semi-automated approach described in Section 5.5. The reference grid points are placed at regular intervals in (x, y) as shown in Figure 5.15. The experiments consider a square grid in (xy) and a *staggered* grid. The grids are referred by their nominal spacing, Δ, along

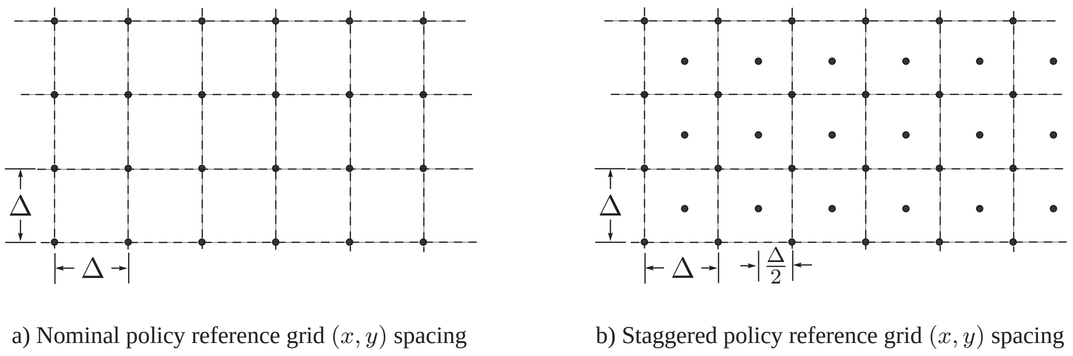

Δ
Δ
$\frac{\Delta}{2}$
a) Nominal policy reference grid (x, y) spacing
b) Staggered policy reference grid (x, y) spacing

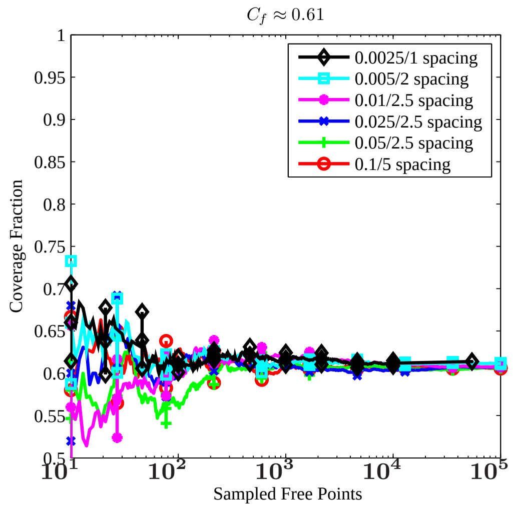
Figure 5.15: Figures show the nominal and staggered reference grid spacings used for policy instantiation. These (x, y) grid points are crossed with a regular spacing along the θ dimension to make a three-dimensional reference grid in pose space.

a single row or column. The regular workspace grids are crossed with a set orientations to give a regular reference pose grid. For the figures and tables below, the notation '& 45' is used to denote orientations of $\left\{-\frac{3\pi}{4}, -\frac{\pi}{2}, -\frac{\pi}{4}, 0, \frac{\pi}{4}, \frac{\pi}{2}, \frac{3\pi}{4}\pi\right\}$; otherwise, the policy reference grids are placed at $\left\{-\frac{\pi}{2}, 0, \frac{\pi}{2}, \pi\right\}$. During the semi-automated deployment, each policy from the cache is instantiated at a reference pose; the policy is added to the suite if and only if the policy is collision free.

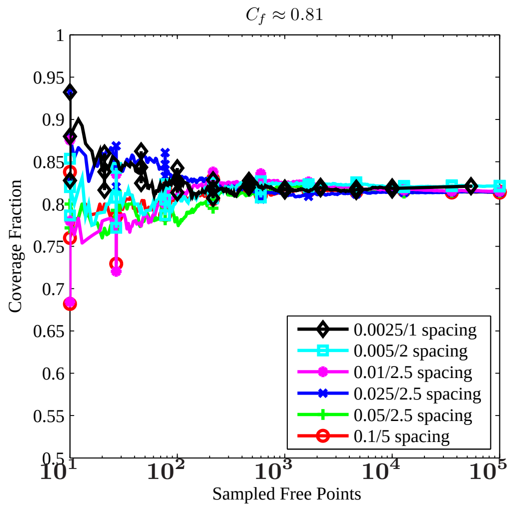
Figure 5.16 shows examples of the coverage fraction estimates for two suites of policies deployed in the environment shown in Figure 5.14. The samples are chosen from a variety of sampling grid spacings. The coarsest uses a spacing of 0.1 meters in the x and y dimensions, and 5 degrees along the θ dimension; the finest sampling grid uses .001 meters and 1 degree. As Figure 5.16 shows, the coverage fraction estimate converges as the number of samples increases for a variety of grid spacings. The figures show the mean estimates from five different samplings taken from the collection of free pose samples taken at a given sampling resolution. The error bars indicate wide variance for fewer samples, but show negligible variance for larger sample counts. Table 5.1 shows the standard deviations obtained for the sampling grid of 0.0025 meters and 2 degree resolution sampling for ten different suites. A reasonable estimate, with less than one percent standard deviation over the coverage fraction range, is consistently obtained after 5,000 samples.

Our expectations, which are confirmed by the experiments shown in Figure 5.17, is that increasing the number of deployed policies tends to increase the coverage fraction. In the experiment, a specific cache of policies is defined and then instantiated on regular reference grids defined in the environment. Figure 5.17 shows the results for eight grid spacings. The first number in the Figure 5.17 legend refers to the nominal reference grid spacing in (x, y); an 'S' is used to denote the staggered grid as shown in Figure 5.15-b. As the reference grid spacing decreases, more policies are deployed and the coverage fraction increases as expected.

The increase in number of policies may be prohibitive in terms of computational cost because many of the additional cells overlap others, without increasing the coverage fraction. This suggests



a) 0.30m reference grid spacing



b) 0.15m & 45 reference grid spacing

Figure 5.16: Coverage fraction estimate converges as the number of sample points goes up. The plots show a variety of pose space sampling grid resolutions for each policy reference grid spacing; the legend refers to the $(xy)\ /\ \theta$ sampling grid spacings in meters and degrees respectively. The graphs represent the mean of five sample-based estimates; error bars are shown. The results converge to a reasonable estimate after 5,000 samples.

Table 5.1: Coverage fraction and standard deviations for various policy grids and sample counts using 0.0025 meter and 2 degree sampling resolution.

		σ^2@Sample Count			
Deployment Reference Grid	C_f	50	500	5000	50000
0.30	0.6054	0.0627	0.0313	0.0064	0.0024
0.30 & 45	0.6359	0.0466	0.0322	0.0075	0.0021
0.30S	0.7354	0.0650	0.0217	0.0033	0.0027
0.30S & 45	0.7540	0.0637	0.0201	0.0041	0.0023
0.15	0.7976	0.0700	0.0211	0.0037	0.0017
0.15 & 45	0.8146	0.0599	0.0216	0.0035	0.0012
0.15S	0.8540	0.0480	0.0154	0.0054	0.0011
0.15S & 45	0.8669	0.0456	0.0150	0.0057	0.0010
0.075	0.8831	0.0388	0.0130	0.0047	0.0012
0.075 & 45	0.8952	0.0335	0.0137	0.0051	0.0010

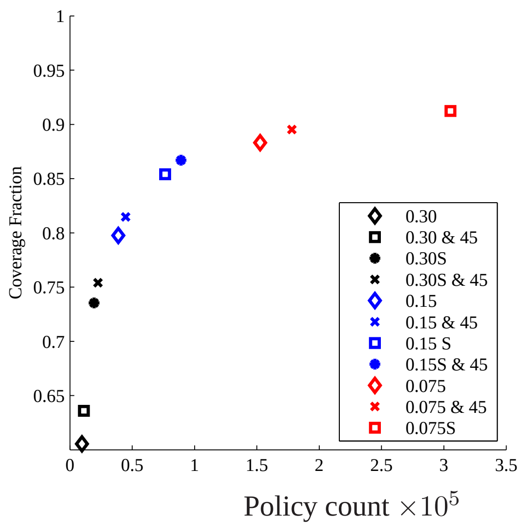


Figure 5.17: Plot showing estimated coverage fraction for a number of policy grid spacings that result in varying numbers of deployed policies.

using the coverage fraction to guide incremental deployment strategies where the policy to be added is chosen as one that captures the most un-included poses from the sample. This greedy strategy preferentially chooses those policies that overlap the least. This approach increases the computational burden up front in order to lower the computational burden at planning time. By decreasing the overlap and number of policies, this greedy approach may have the unintended consequence of reducing the flexibility of the suite during discrete planning. This is a trade off that must be evaluated by the system designer on a case by case basis depending on the planning scenarios under consideration and the computational resources available.

These tests were conducted for a single cache of policies. To increase the coverage fractions, one could also consider adjustments to the cache of policies. By including slightly larger and slightly smaller policies, the policies can capture domains that are otherwise missed for a given policy grid. By considering policy dominance, that is policies that completely contain the domains of other policies in the cache, and only instantiating the policy that contains the largest (most dominate) policy, the total number of instantiated policies can be reduced.

The real test of the coverage fraction is relative to a specific navigation problem; that is, how "global" is the hybrid control policy. Thus far, this discussion has focused on evaluating the coverage fraction for a given suite of policies. For a specific navigation problem, some policies in the suite may be discarded if they do not prepare other policies that reach the goal. The coverage fraction should be evaluated for the *deployment*, that is the suite of policies and the switching strategy, for the navigation problems being considered. Providing more policies, with more interconnections in the prepares graph, increases the likelihood that a policy will be included in the deployment, at a cost of increased computation up front and during planning.

5.8 Conclusion

This chapter provides an overview of our approach to extending policy composition techniques to single-bodied wheeled mobile robots. After discussing the modeling framework for purely kinematic system, the chapter presents a policy design approach based on defining cells in the free pose space, and feedback control policies over those cells. Two specific families of generic policies that follow this approach are introduced. The chapter discussed techniques for validating specific instances of the generic policies, and verifying that the policies satisfy the composability requirements of Chapter 3. The cells have explicit boundaries in the pose space, which allows the safety of the policies to be guaranteed provided the cells are complete contained in the free pose space, and the associated control policy renders the cell conditionally invariant. The chapter introduces a novel approach to verifying that the cell are collision free based purely on workspace measurements, in a way that does not require the construction of the free pose space boundaries. The policies have simple inclusion tests by design.

Approaches to policy instantiation and prepares graph generation are discussed. The chapter introduces a semi-automated approach to policy instantiation based on a collection of policies in a cache. The cache policies are specified manually, and validated for conditional invariance and finite time convergence. The semi-automated approach takes policies from the cache and places the cells at specific reference poses via a rigid body transformation. The invariance properties of the policies guarantee that the transformed policies retain the conditional invariance and finite time convergence properties. The transformed policies are tested for collision using the expanded cell approach introduced in this chapter; only collision free policies are retained in the policy suite. Given the suite policies, this chapter introduced three procedures for determining the prepares relationship among policies in the suite. The policy instantiation and prepares graph generation processes incur an upfront cost; the upfront cost is offset by the planning flexibility of this hybrid control approach, which is demonstrated in the next chapter.

Finally, this chapter introduced a sample based method for evaluating the relative completeness of the suite of of policies. The results demonstrate that the sample based approach converges to a reasonable estimate of the coverage fraction; that is, the portion of free pose space covered by the instantiated policies.

Given these policies and validation tools, it is feasible to deploy composable policies for wheeled mobile robots in a way that enables symbolic planning on these constrained systems. The next chapter explores various planning techniques in simulation and experiment on real mobile robots using policy suites defined using these techniques.

Chapter 6

Demonstrations of Coupled Planning and Control

This chapter demonstrates the coupled planning and control approaches advocated in this thesis; experiments and simulations of several different robot models validate the approaches discussed in Chapter 3. The experiments are designed to demonstrate the sequence-based, order-based, and automata-based planning approaches using the policies developed in Chapter 5. The approach is applied to real systems that exhibit the imperfections and model uncertainty of real world applications, operating in confined environments. The results demonstrate that the composition of simple policies allows more complex behaviors to emerge; unlike other behavior-based approaches [19], these emergent behaviors are guaranteed to induced the desire global behavior. The experiments also exhibit the robustness of feedback control to model uncertainty and disturbances. In spite of the overall success, several issues arise during the testing. We discuss these issues, how they impact the relative strengths and weaknesses of the different approaches, and present our methods of addressing the issues.

The chapter is divided into sections based on the planning methods used in the demonstration. First, order-based approaches to building a global policy are discussed. Here the task is navigation to an overall goal using an ordering of the instantiated policies. Experiments using both SQ and PF policy types, and two different robot models are presented. Next, the sequence-based approach that uses model checking to satisfy temporal constraints is discussed. A sequence of policies is generated such that the composition of the policies in sequence moves the robot through a series of tasks. Simulations of one particular robot model are given, and the limitations of the sequence-based approach on a real robot are discussed. Finally, we present examples using automata synthesis to build formally correct reactive hybrid control policies. Results from both real robots and simulations are given. The robots demonstrate navigation tasks that change based on different sensed inputs; that is, the robots react to changes in their environment.

6.1 Order-based Planning

The first tests follow the conventional sequential composition approach by using an ordering of the suite of policies to address a single navigation and control task. Here, the attempt is to build a near global control policy using the instantiated local policies. The basic task is to navigate through an environment without collision to a designated goal; the goal is chosen to correspond to the goal set of a policy in the suite. This section presents scenarios for two different robot/environment combinations; several initial conditions are shown for each environment.

Each scenario is set up as follows. First, we assume the environment is fully known. The control policies are instantiated using the generic policies defined in Chapter 5; Matlab$^{\text{TM}}$code developed for this thesis is used to test requirements for composable policies. The policies are verified for the particular robot model and input set; collision is tested based on the workspace model. We then determine the prepares graph using code that implements the ideas discussed in Chapter 5. Heuristic costs are assigned to each edge based on a cost function related to the size and complexity of the given policy transition. The deployments are verified in simulation prior testing on the robot. Afterwards, a list of policies with their assigned parameter values is written to a text configuration file; another configuration file defines the control input bounds used for the particular robot. The policies are then transferred to the robot for execution.

The robot control is governed by a software *executive* program that coordinates the pose estimation, policy switching, and motor control output. The executive, developed for this thesis, is written in C++. During execution, the control inputs specified by the local policy are sent to a low-level motor controller. The policy control input is specified as a forward velocity and rate of turn, which is mapped to desired wheel velocities. A low-level motor controller attempts to move the drive wheels according to the velocity command by specifying voltages to the motor. As this control is never perfect, and is subject to delay, error, and second-order dynamics, the actual robot is an imperfect match to the kinematic model assumed in the policy design. Appendix F discusses the particulars and limitations of the particular robot models used in these experiments in more detail.

During the robot startup, the robot software executive first reads the suite of policies from the configuration file, along with the specified control input bounds appropriate for the given robot. The goal, which is assigned according to the particular navigation task, is specified as a particular policy. The executive then uses D*-lite [81] to order the policies according to the assigned heuristic costs; the entire prepares graph is ordered.

The output of this D*-lite implementation is a switching strategy, which we represent with a finite automaton. Each node is assigned a cumulative cost to the goal and a preferred action. The automaton is tree-like; that is, there are no cycles. During the experiments, the policies are executed using the finite automaton model to provide faster execution time due to fewer inclusion tests, as discussed in Chapter 3. The ordering is also stored as a totally-ordered list of nodes according to the cumulative costs assigned to each node in the automaton. This list is for recovery after perturbations. After D*-lite is finished with the initial ordering, the robot executive searches the ordered list until a node whose policy that contains the initial pose is found. If the search fails to identify a valid policy, that is an ordered node with finite cost, then execution terminates.

The robot executive program executes the hybrid controller based on the finite automaton. The current automaton node is stored during execution. The executive accepts the current local pose estimate, and checks if the current pose estimate is contained in the domain of the policy associated with a child node of the current node [1]. That is, the policy associated with the child node of the current node in the automaton. If the child policy domain contains the current pose estimate, then the child node becomes the current node and the associated policy is executed. Otherwise, the current local pose is tested against the current policy domain, and the current policy is executed if its domain still contains the current local pose estimate. If a disturbance takes the current pose estimate outside the current domain, a zero velocity command is sent to the motor controller, and the executive begins a total order search using the ordered list. If the search fails to identify a node associated with a valid local policy, then execution terminates; otherwise, the newly identified node becomes the current node, and the policy execution continues. This approach allows the robot to recover from unexpected perturbations, while preserving the speed of the local search in the finite automaton representation.

[1]Recall from Chapter 3 that we make a distinction between the nodes of the automaton used to represent the switching strategy, and the policies associated with each node.

6.1.1 'Deminer' Robot Experiments

The first experiments use the 'Deminer' robot shown in Figure 6.1. The robot is a standard differential-drive robot with a convex, roughly elliptical body shape. The control inputs are taken from one of four bounded input sets, $\mathcal{U}_i$, as shown in Figure 6.2; each local policy is associated with one particular input set. See Appendix F for details about the robot size, shape, and the input sets.

For these experiments, a total of 288 basic SQ type policies described in Appendix E are manually instantiated using the techniques described in Chapter 5. During the instantiation process, each set of policy parameter values is checked for validity for the particular input set assigned to the policy; that is, the composability properties are verified and the parameter values adjusted as necessary. Figure 6.3 shows the domains for seven policies. It is worth noting that the manual instantiation process is time consuming, especially around sharp turns which required much trial and error to get policies that appropriately prepare one another. The extended prepares definition is used most often near the corners of obstacles to allow larger policy domains to be deployed. For this suite of policies, fifteen separate policies prepare 31 different unions of policy domains, which introduces indeterminacy into the discrete prepares graph as discussed in Chapter 3.

For these demonstrations, the Deminer robot operates among a set of polygonal obstacles that define the ten meter by ten meter world. Figure 6.4 shows the projection of all 288 instantiated

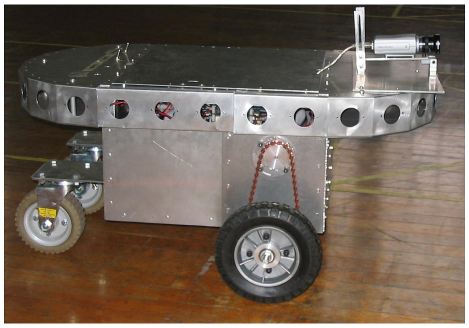


Figure 6.1: 'Deminer' laboratory robot

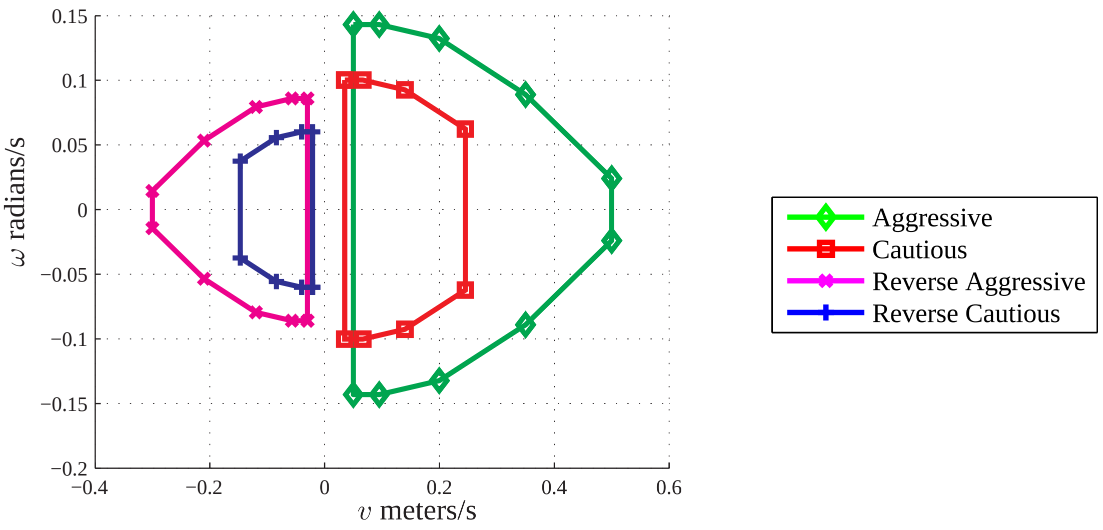
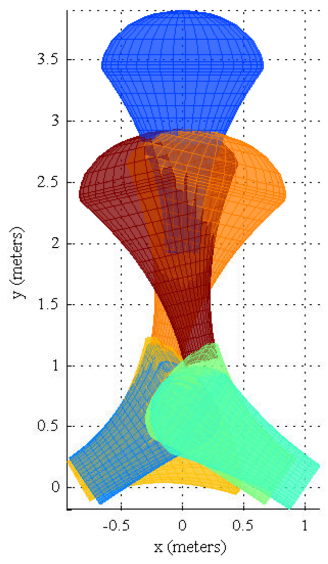
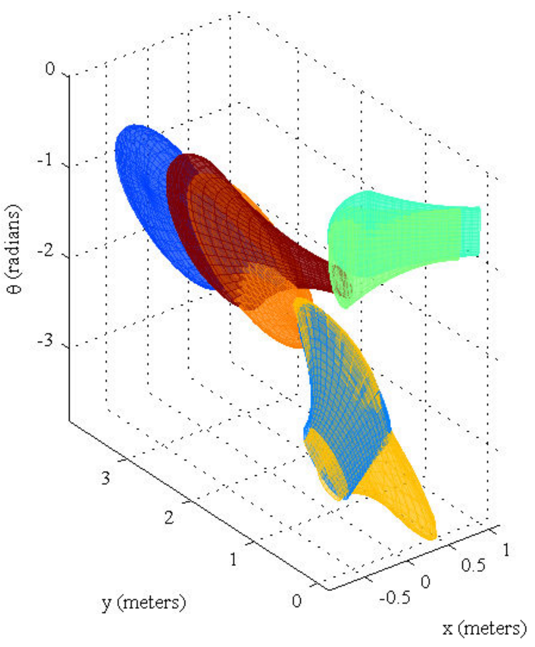


Figure 6.2: Four sets of bounded steering inputs used in Deminer experiments. .



a) Projection of cells into workspace with obstacles



b) Representation of cells in three-dimensional pose space

Figure 6.3: Detail of seven cells in environment.

policies into this workspace. The world includes several narrow corridors and openings, with the narrow corridors measuring approximately one meter. This provides a clearance of approximately 16 centimeters on either side of the robot, but prevents the robot from being able to turn around within the corridor. The policy domains projected into Figure 6.4 appear to be away from the obstacles; however, the expanded cells that account for body shape are much closer to the obstacles. The expanded cells are not shown in Figure 6.4, but an example is shown in Figure 5.9. Figure 6.5 shows a representation of these 288 policy domains in the three-dimensional pose space.

The designated navigation task is specified as bringing the robot to the middle of the lower corridor. This navigation goal corresponds to the goal set of particular local policy, which is chosen as the goal of the ordering. D*-lite is used to generate the ordering of the prepares graph associated with the suite shown in Figure 6.4. After the system state passes through the local goal set of this designated goal policy, the execution is halted.

A total of 12 different experiments were conducted on the actual robot. These experiments are based on dead-reckoning position estimates. 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 ran the robot in an open space, but used the policies that consider the obstacles shown in Figure 6.4. In this mode, the robot “hallucinates” the obstacles. With this dead-reckoned position estimate, all 12 experiments ran to completion providing a proof of concept, and demonstrating the reliability of the approach. Seven representative results from unique initial poses are presented below; the remaining five tests duplicated some initial poses, with similar results.

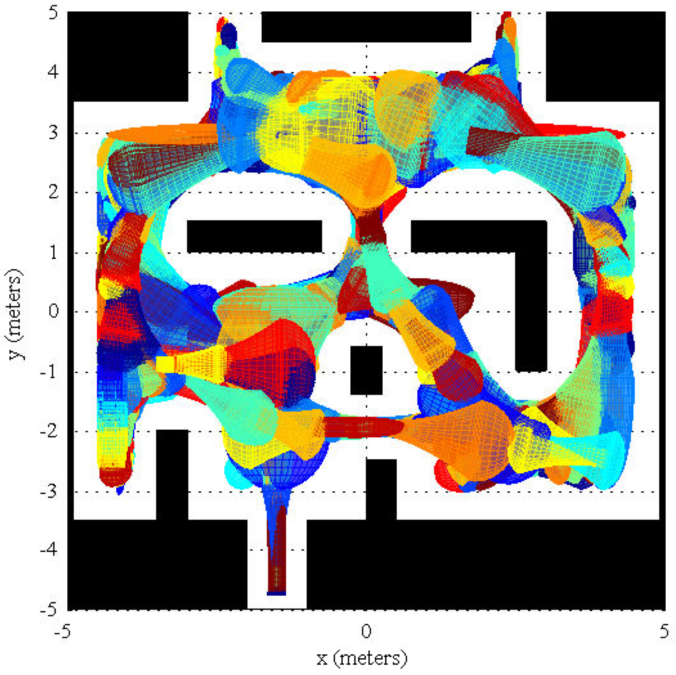


Figure 6.4: Projection of 288 cells into the workspace.

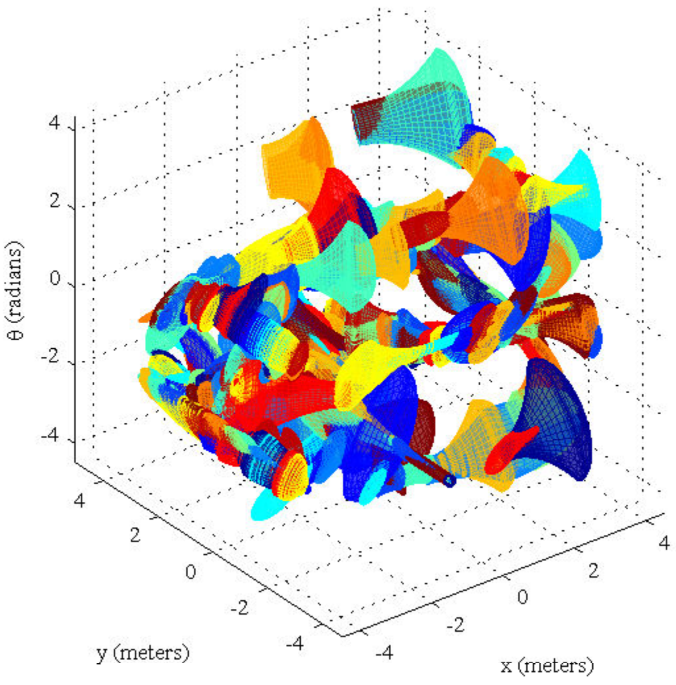


Figure 6.5: Complete suite of 288 cells in three-dimensional pose space.

Figure 6.6 shows four paths that start from four different initial conditions, but converge to the same goal. The paths are shown on the same plot to underline the global nature of the resultant hybrid control policy. It is worth emphasizing that 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 switching according to the ordering defined by D*-lite using the prepares graph for the suit of policies shown in Figure 6.4; an explicit desired path is never calculated. The paths are labeled (#1 - 4) clockwise from the lower left.

The paths labeled #5 and #6 in Figure 6.7 demonstrate the flexibility of planning in the space of control policies. Path #5 begins near the same position as path #1 (shown in Figure 6.6); however, the orientation is approximately 180 degrees different. The composition of local policies enables the robot to back up, stop, and then move forward to the goal without colliding with an obstacle. This is because the policy suite also includes policies that allow the robot to go in reverse. The policy switching between forward and reverse is automatic given the particular ordering, with no operator intervention. To avoid damaging the real robot, the system is required to come to a stop before switching between a forward and reverse policy. Similarly, path #6 demonstrates a more complex K-turn induced by the composition of the simple SQ policies shown in Figure 6.4. The

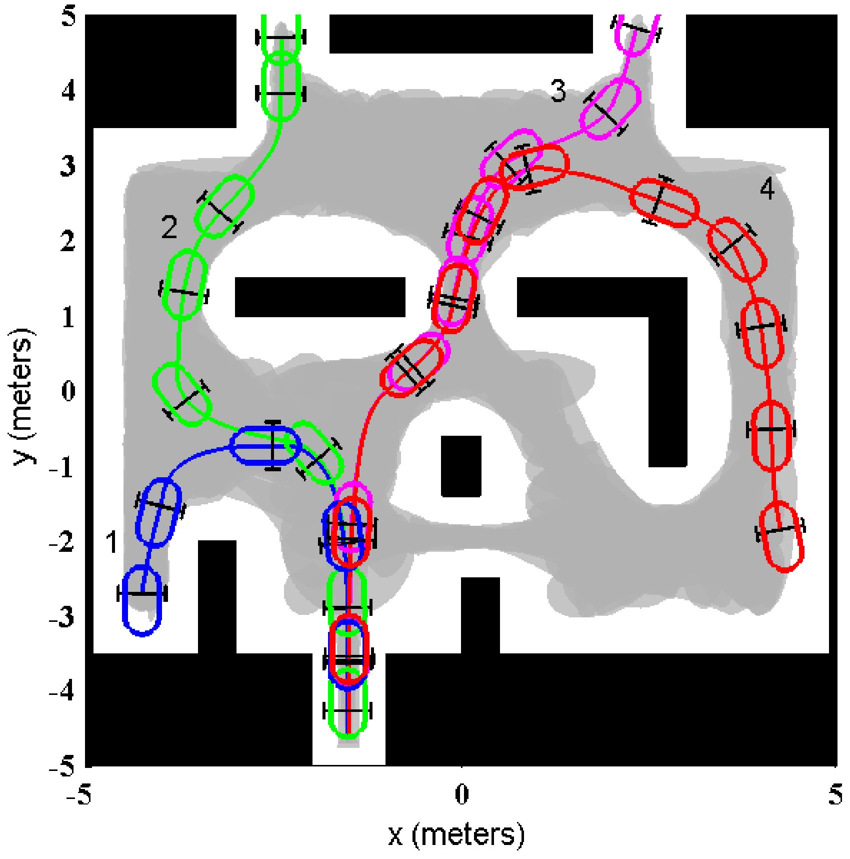


Figure 6.6: Four experimental runs demonstrate the "global" nature of the hybrid control policy induced by the ordering of the policies; only the initial conditions vary. The projection of the individual policy domains are shown in light gray. The actual data is plotted at 0.1 second intervals; robot symbols are drawn for every five seconds of travel time.

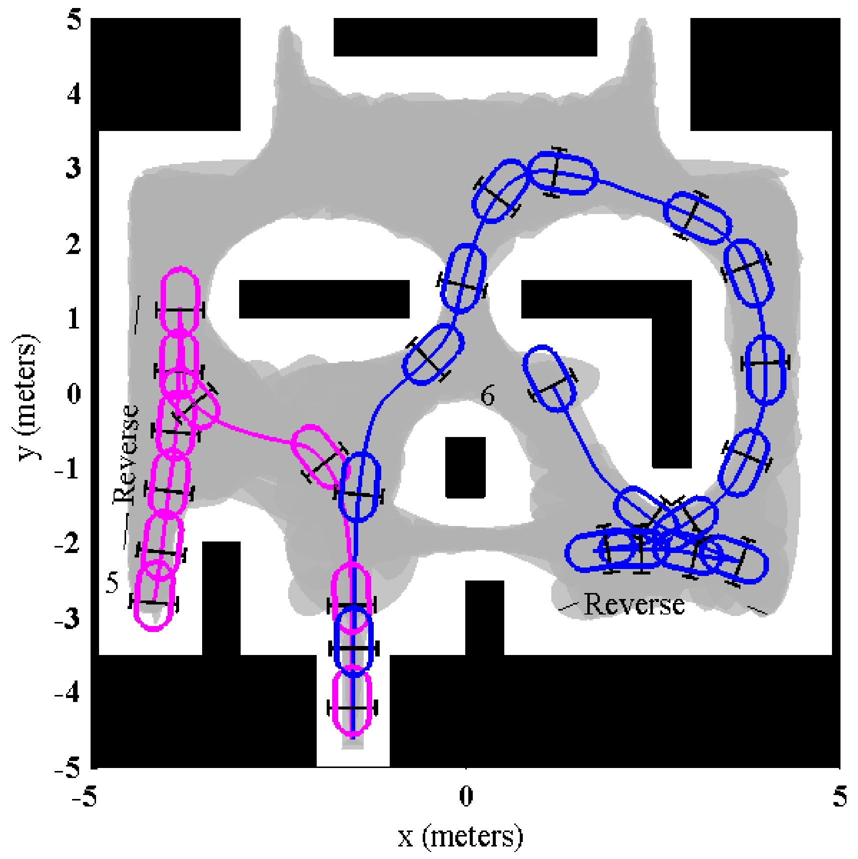


Figure 6.7: Two additional runs using the same ordering. Run #5, which starts at the same position as Run #1 but oriented 180 degrees apart, automatically backs out of the narrow corridor, and then switches to forward motion. Run #6 demonstrates a complex K-turn maneuver to get around a sharp corner. The behavior is automatically induced by the composition of local policies.

policy suite does not include policies that are expressive enough to turn the lower right corner in one continuous motion. While this points to the limitations of this particular suite of policies, it validates the basic approach. The combination of simple policies with discrete planning is still capable of generating expressive motions; thereby demonstrating the flexibility of planning in the space of control policies.

A second advantage of planning in the space of control policies is the ability to do fast planning and re-planning. In this case, the planning is over a prepares graph with only 288 nodes. Figure 6.8 shows path #7, which starts near the initial condition of path #6. Path #7 begins to converge to path #6,which is shown as a dotted line; however, just after the K-turn the two policies crossing the circular obstacle shown in Figure 6.8 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 occurs in real time, while the hybrid control policy is executing on the robot.

This experiment served as an early proof of concept. The experiment demonstrates using simple local feedback control policies to induce a global behavior on a single-body nonholonomically constrained robot. Planning and re-planning on the prepares graph carries low overhead. The policy

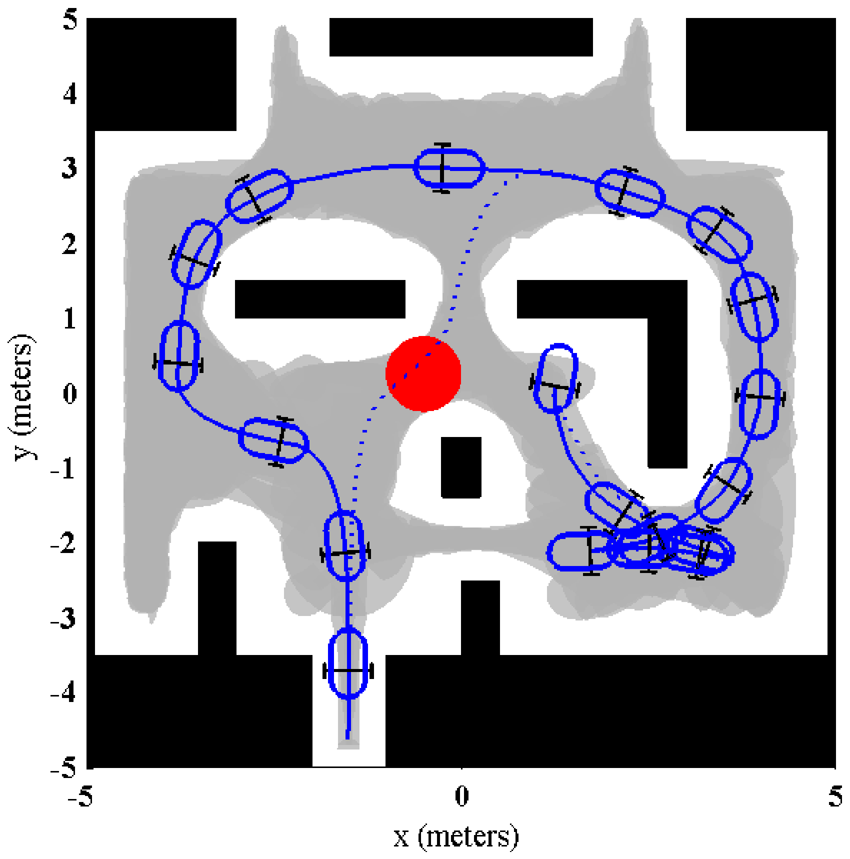


Figure 6.8: Paths #6 and #7 diverge in response to two policies that are invalidated during run #7; D*-lite efficiently re-orders the policies during the run as additional information is gathered. The dark circle represents a new obstacle that invalidates certain policies.

composition induced by the hybrid control policy allows complex behaviors to emerge in a provably correct manner through the policy ordering.

6.1.2 ‘LAGR’ Robot Experiments

The previous experiments did not have integrated localization, which limits their practical value. Using pure dead reckoning makes the experiment too much like a simulation, as the hybrid control system is not subject to disturbances due to localization error and correction. To address this shortcoming, a second set of order-based experiments uses a robot equipped with an integrated vision-based localization system. This allows the hybrid control approach developed in this thesis to be evaluated in an integrated system.

In the next set of experiments, we use the ‘LAGR’ robot shown in Figure 6.9 because it has four pairs of stereo cameras to perform vision based localization relative to known landmarks placed in the environment; the landmarks are color coded as shown in Figure 6.9. The localization system uses an extended Kalman filter to update the pose estimate based on measurements of range and bearing to the identified landmarks. The system is now subject to disturbances based on jumps in the pose estimate as new landmarks come into view.

In addition to the different robot, these experiments are carried out in a different environment with a different policy suite, which uses PF style control policies. The PF policies are more natural for specifying motion around tight corners than the SQ policies used in the Deminer experiments. This is because the PF policies can be deployed relative to an arc in workspace, where the SQ policies had a straight central axis. Figure 6.9 shows the LAGR robot successfully navigating a particularly tight spot under closed loop control.

Instead of the manual instantiation approach taken with the previous experiments, these PF policies are instantiated using the cache and reference point approach discussed in Chapter 5. A total of 313 policies are systematically defined in the cache; 156 forward, 156 reverse, and one special ‘Halt’. These policies include various widths, lengths, and arc radii. Each policy in the cache is associated with a one of thirteen bounded input sets; the inputs for each policy are taken from its associated input set. See Appendix F for specific details about the input sets. The input sets include sets for straight PF policies and arc based PF policies. Both forward and reverse sets are associated with each group, as are aggressive and cautious sets. The sets for straight policies

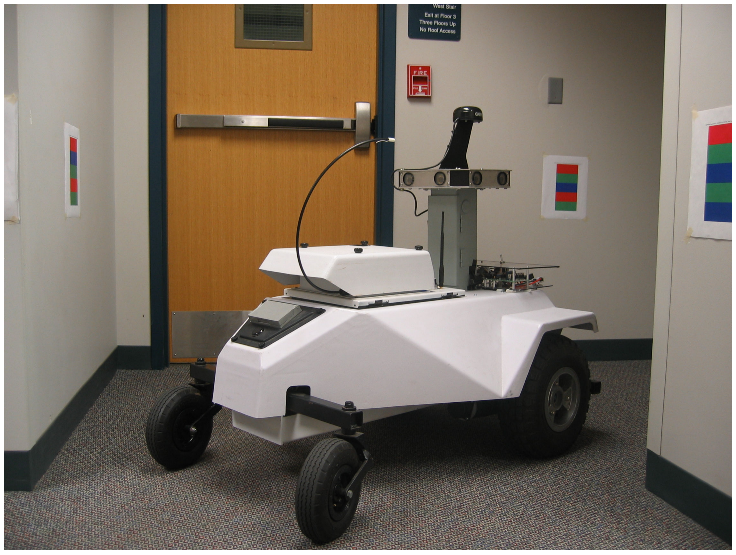


Figure 6.9: ‘LAGR’ robot navigating a corridor. Three color-coded landmarks, which are used by the vision based localization system, are visible in the image.

allow less aggressive steering, while those for tight turns use a cautious forward speed and more aggressive steering. By matching the input set to a particular policy, the feedback is tuned to the local conditions. Even though the robot is capable of zero-radius turns, the input sets are constrained to excludes zero-radius turns; this approximates the behavior of more constrained systems such as cars, and provides a greater challenge to the hybrid control approach.

In the demonstrations here, the robot maneuvers about a hallway with long thin corridors; therefore, it makes sense to instantiate the policies in "lanes." That is, the policies are instantiated in straight lines running the length of the hallways; the lines of policies are analogous to lanes on a highway.

To further simplify policy instantiation and planning, some local policies are grouped into meta-policies. Figure 6.10 shows a meta-policy associated with taking the system from one of three lanes entering from the right and moving the system to the top most lane exiting to the left. The meta-policy defines an order-based switching strategy between its component policies, which are defined with respect to a common reference point. A meta-policy is instantiated by instantiating its component policies relative to its reference point; the meta-policy can only be instantiated if all of its component policies are collision free relative to a specified reference point in the free pose space. Figure 6.11 shows the component policies to scale, along with the expanded cells defined by the robot body size and shape.

While the component policies could be instantiated individually to accomplish the same task, meta-policies simplify the deployment by grouping similar behaviors. For planning purposes, the meta-policy is treated as a single node in the prepares graph. Furthermore, the meta-policies can allow manual verification of the prepares relationships. This can allow policies that cover the goal set, but do not all contain the goal set center, to be prepared as a group; this removes the restrictions of Section 5.6 for automated verification. By design, the meta-policies only allow designated component policies to be prepared. This affords the designer more control over the meta-policy

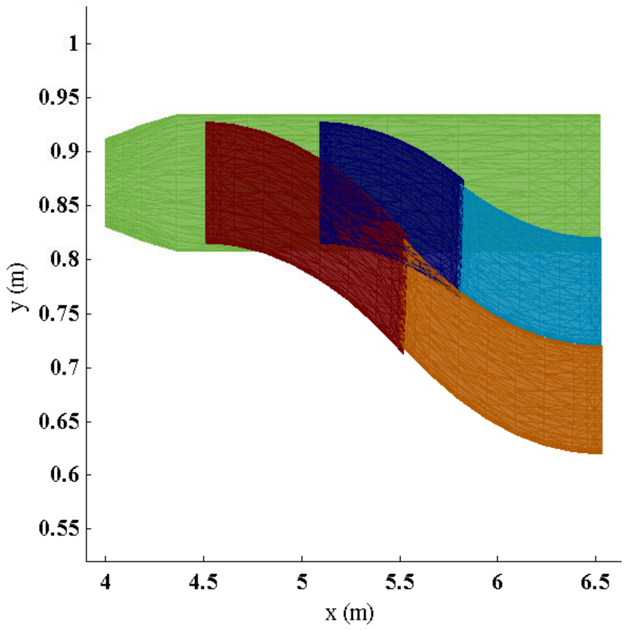
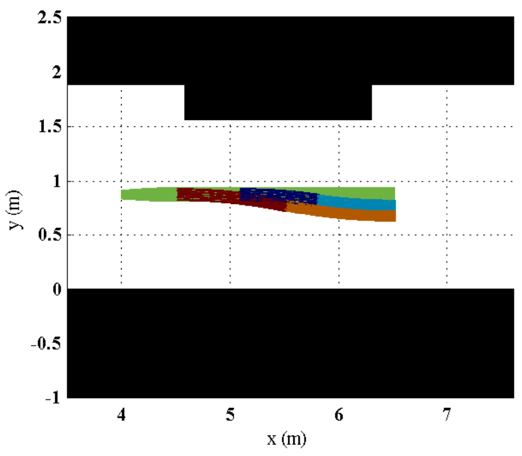
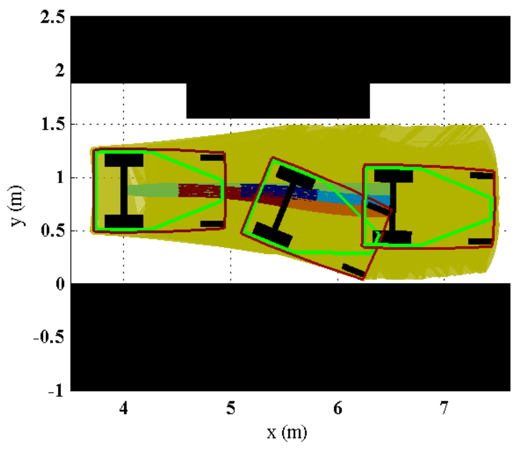


Figure 6.10: An example meta-policy that uses five PF style policies to move three lanes coming from the right into a single lane exiting to the left. The policies are shown projected into workspace. To enhance the detail, the figures axes are not equal.



a) Meta-policy projected into workspace



b) Expanded meta-policy showing body extent in workspace

Figure 6.11: Example meta-policy used for LAGR experiments. The meta-policy domains are shown relative to obstacles in the environment in the proper scale. The figure on the right shows the body extent into the workspace for this meta policy. Three robots are shown at various poses on the cell boundaries; the robots use a bounding polygon for calculating the body extent.

behavior by allowing normally unused component policies to be added for robustness, without including them in the prepares graph used for planning. These component policies can only be active after the meta-policy becomes active.

Several meta-policies are defined based on needed motion in the hallways. In addition to the "lane change right" meta-policy, meta-policies for "lane change middle" and "lane change left" are defined. Given the narrowness of the hallways, only three lanes spaced 0.1 meters apart are defined in most corridors. The lanes are defined in the forward direction along the length of the corridor.

Several meta-policies associated with turning corners are defined, including both "left turn" and "right turn". Figure 6.12 shows an example left turn. The turn meta-policies include simple arcs of various radii to blend three lanes into one orthogonal lane. To improve robustness, the turning

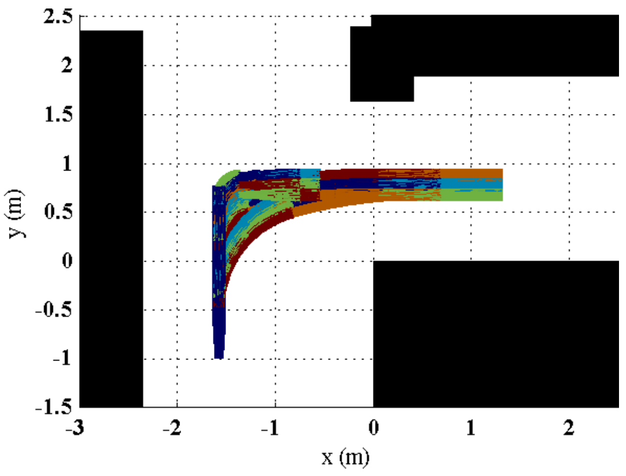


Figure 6.12: Meta-policy used to turn 3 lanes left.

policies include both short radius turns and longer radius turns. The long radius arcs blend with the short radius turns to provide a transition from straight motion to turning motions.

Many of the hallways were so narrow that a simple arc could not navigate the hallways; therefore, two different meta-policies induce K-turn motions in the narrow hallways. Figure 6.13 shows actual data from a K-turn executed by the robot during an experiment. In this case, the behavior is encoded in the meta-policy by design, where the behavior emerged as a consequence of discrete planning in the Deminer experiments.

The meta-policies are implemented in a modular fashion that makes the planning and execution transparent to the robot executive. When testing for inclusion before becoming active, the meta-policy only tests those component policies designated as inlets for the prepares definition. Once the meta-policy becomes active, all component policies are tested using the internal ordering; that is, the inclusion test resorts to a total order search over component policies if necessary once the current node is associated with the meta-policy. This approach provides robustness to the meta-policy, while given the designer control over when the meta-policy is allowed to become active initially.

The lane change meta-policies are instantiated at regular intervals along evenly spaced lanes in the corridors. The component policies in the lane-change meta-policies are associated with input sets that have positive forward velocity. Turn and K-turn meta-policies are instantiated in a way that they are prepared and prepare the basic lane policies at the appropriate junctions. In addition to the meta-policies, a few 180 degree arcs are used in the larger central hallway to allow continuous turns. By design, according to the instantiated policies, the robot is only allowed to reverse motion in certain spots through the use of a K-turn maneuver or 180 degree arc in the central corridor. Otherwise, the robot must travel in a loop.

Two additional meta-policies are defined to navigate a small “nook”; Figure 6.9 shows the robot navigating this nook. When the robot is in the nook, it is halted by a special policy that is prepared by the incoming meta-policy; the halt policy prepares a meta-policy that exits the nook.

The use of meta-policies reduces the total number of nodes in the prepares graph that is used for planning. In this case, a total of 309 meta-policies and 86 individual policies are deployed in the environment. Thus, the prepares graph used for planning has only 395 nodes, compared to the grand total of 2846 PF policies that are instantiated in the environment. The grand total includes both individual and meta-policy components. The policy suite includes 3155 policies, the 2846 PF policies and 309 meta-policies.

The robot software executive functions the same as with the Deminer experiments. The planning takes place over the 395 nodes in the prepares graph. During execution, when a meta-policy becomes active, it is treated as a switched hybrid control policy in its own right; the component policies are activated according to the meta-policies local ordering. On the executive level, the meta-policy remains active until the state enters the domain of a child node of the ordering, or exits the domain of all policies in the meta-policy’s collection of component policies.

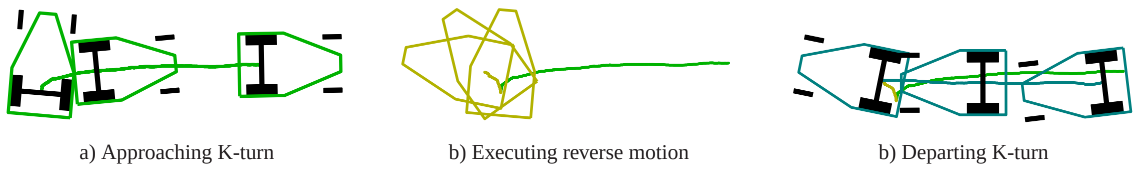


Figure 6.13: Plot of data from actual robot experiment as the robot executes a K-turn in the upper hallway. The K-turn is automatically induced by the composition of policies based on simple arcs and straight path segments.

Figure 6.14 shows the results of executing the hybrid control policy induced by the ordering of the policies. The meta-policy corresponding to stopping in the small nook was chosen as the goal. The figure shows eleven different runs from five different initial conditions. In most cases, the curves overlap and are indistinguishable, which shows the repeatability of the performance. Figure 6.15 shows the individual runs. Figure 6.16 shows a close up of the final configuration in the nook; the eleven overlapping robots closely match one another, which demonstrates the repeatability of the closed loop system.

During execution, five of the eleven runs experienced disturbances that took the system pose estimate outside the domain of the current policy. In these cases, the total order was searched, a valid policy was found, and the robot continued to the goal.

To demonstrate re-planning during execution, two additional runs are shown in Figure 6.17. During execution 24 policies that pass through the lower corridor are invalidated, which triggers a re-planning step using D*-lite. During the initial planning stage, the D*-lite takes approximately 0.050 seconds to order the 395 nodes of the prepares graph. The D*-lite re-planning step required only 0.018 seconds; this compares favorably with the 0.01 second loop time for the robot executive function. During these re-planned runs, the executive needed to search the total order twice per run. These searches were required near the K-turn in the lower left corridor, as disturbances allowed the robot to exit the policy domains due to the aggressive turning that is required.

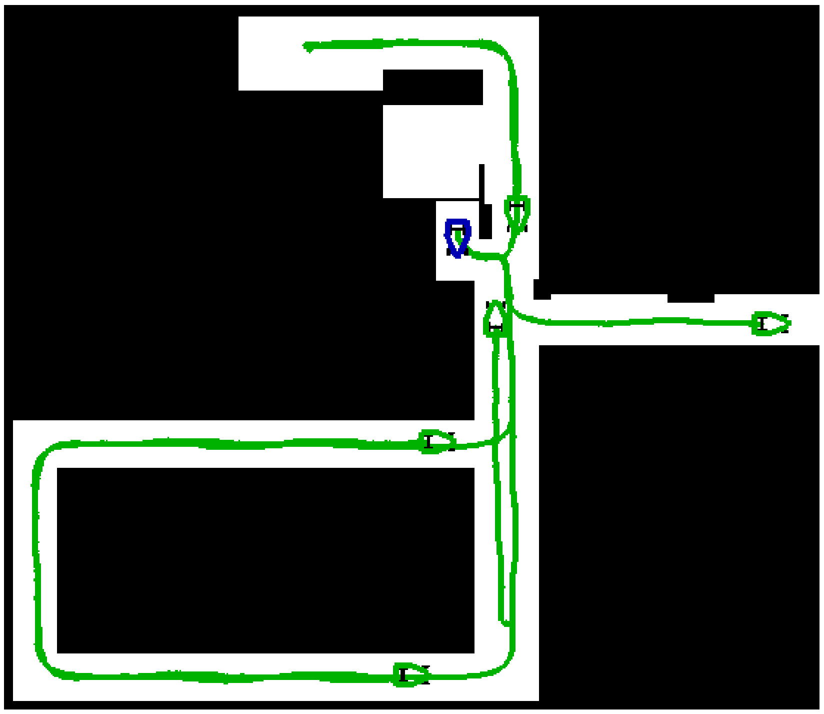
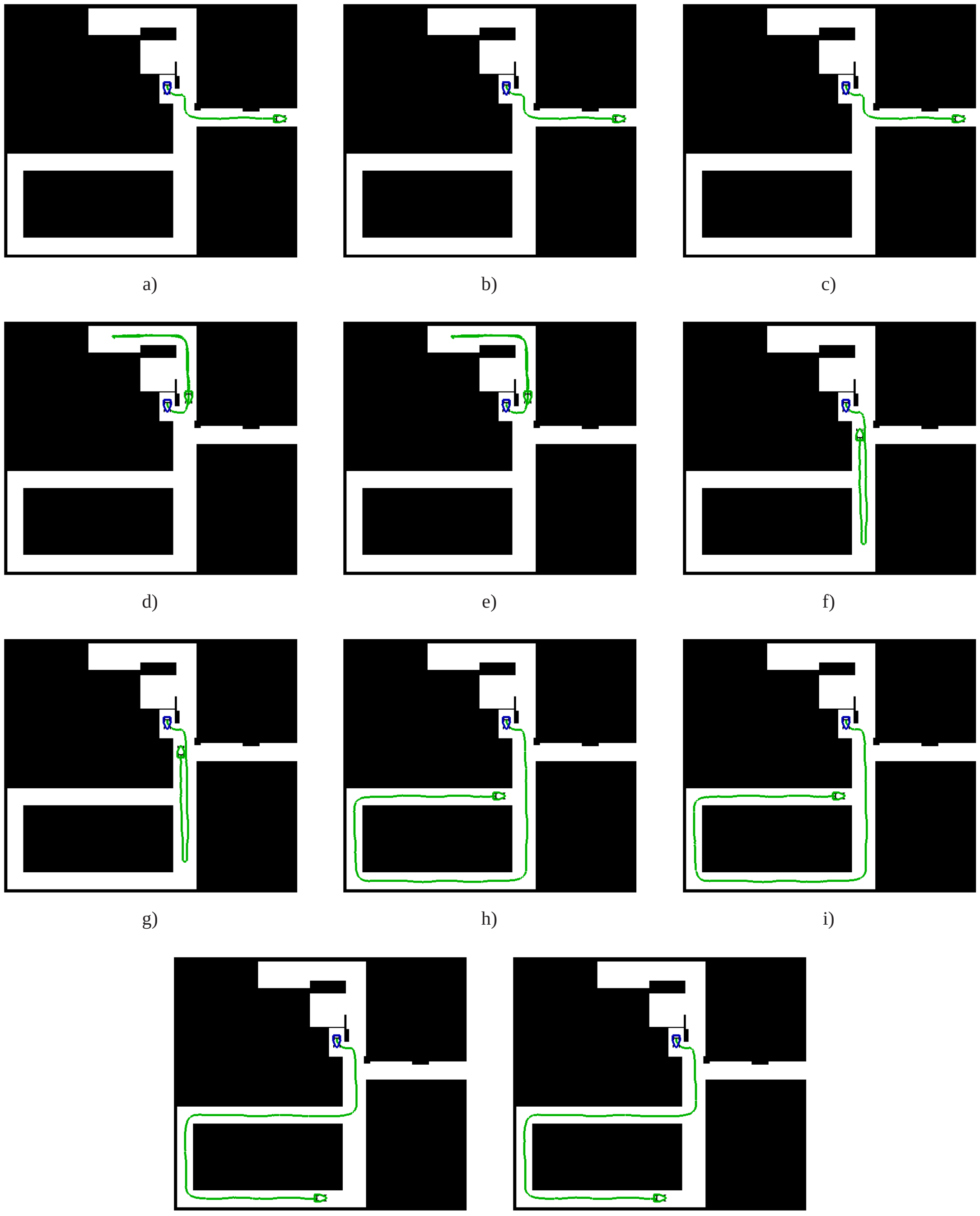


Figure 6.14: ‘LAGR’ robot navigating corridors using a ordering of instantiated policies. Eleven different runs, from five different initial conditions, converge to the designated goal set. The five lighter green robots mark the initial conditions; the darker blue robot marks the goal. The corridors shown represent an approximately 29 m x 27 m portion of Carnegie Mellon’s Newell-Simon Hall A-level.


a)

b)

c)

d)

e)

f)

g)

h)

i)

j)

k)

Figure 6.15: Details of eleven runs using the ‘LAGR’ robot navigating with the same ordering of instantiated policies.

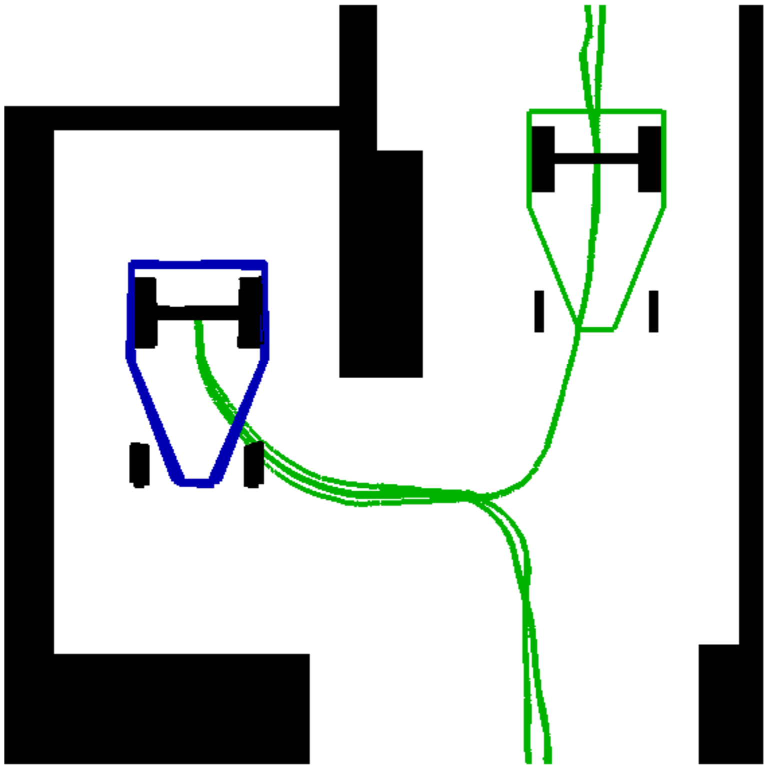


Figure 6.16: A close-up of the final poses. This figure shows the results from 11 runs; the closely overlapping robots show the repeatability of the performance, even after long runs.

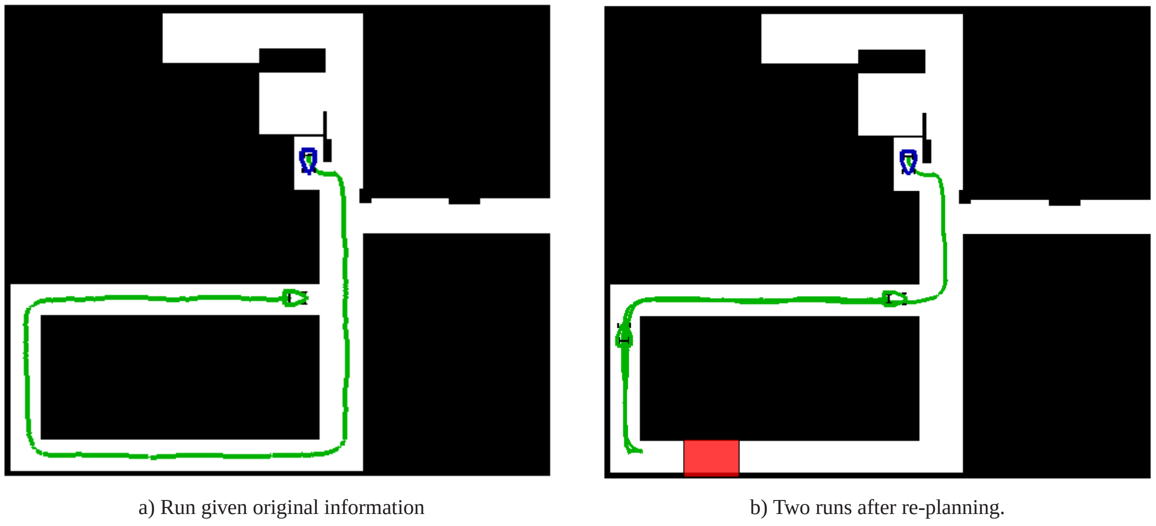

a) Run given original information
b) Two runs after re-planning.

Figure 6.17: D*-lite is used to re-order the policies after policies traversing the lower corridor are invalidated by an obstacle. The alternate route executes a ‘K-turn’ in the lower corner, as the system cannot turn around in the hallway using the policies in the cache. These figures show an approximately 19m x 19m square of the corridors.

In addition to the thirteen successful runs, four experiments ended in failure when the pose estimate exited the domains of all policies in the deployment. Two of these four failure were during execution of the K-turn during re-planning experiments. That is the disturbances encountered during the aggressive turns were significant enough to take the system outside the domains of all policies in the deployment.

These are failures in the sense that the robot ceases execution, and cannot recover with the existing suite of policies, but not in the sense of crashes or incorrect behavior. The safety of the approach is preserved, as the robot halts execution and comes to rest safely as soon as the pose estimate exits the domain. These four experiments do not invalidate the hybrid control approach; rather they demonstrate the inherent safety encoded in the explicit domains. The approach automatically recognizes when the system is outside a valid domain, and is able to halt execution. In most cases, these failures could be avoided by deploying more policies in the environment, and increasing the coverage fraction of the policy suite.

There are two main causes of these failures where the system exited the domain unexpectedly, and thus violated the conditional invariance of the local policies. The first relates to the localization system; as new landmarks come into view, the pose estimate occasionally changes faster than the robot can respond. This occurs relatively infrequent, as evidenced by the many long successful runs.

The second contributing factor is errors in commanded velocities. The LAGR robot, while appropriately designed for its intended function of outdoor navigation, lacks sufficient control resolution for fine positioning in relatively narrow environments that necessarily have thin policy domains due to the size of the robot. Its velocity control is not sufficiently responsive mainly due to limited encoder resolution, which causes noisy velocity signals and limits the control gains that can be applied, and to large disturbance forces due to the caster wheels. During aggressive turns, the combination of localization update changes and velocity controller error allows the system to exit the policy domain because the actual velocities do not match the commanded velocities, in violation of the kinematic assumption of policy design. Again, the hybrid control approach with explicit policy domains defined by the cells, recognizes that the pose is entering an unsafe region, and halts execution.

Another potential source of failure is unwanted limit cycle behavior, which occurs when a disturbance takes the system outside the domain of one policy, but the state is captured by the domain of a lower priority policy. This violates the assumption of monotonic switching assumed by the order-based approach [21]. For limited disturbances, this is not a problem. If the disturbance occurs repeatedly, the executive can become trapped in a cycle between the same policies and fail to make progress towards the goal. This failure was observed during some preliminary experiments, mainly near the aggressive K-turn. The most likely cause is jumps in the pose estimates as landmarks come in and out of view. In general, the localization is relatively consistent; however, jumps of several centimeters have been observed. This failure has not been observed since some additional policies were added to the K-turn meta-policy before the final experiments.

In spite of the limitations of the LAGR robot, the order-based hybrid control policy robustly addressed the coupled navigation and control problem in most cases. The few failures could have been addressed by adding more policies to the suite of local policies in order to capture more of the free pose space. For single navigation tasks, the order-based approach to generating a hybrid control policy proves reliable.

6.2 Model Checking-Based Sequence Planning

While the order-based approach allows a single task to be addressed, the approach does not handle multiple tasks that depend on temporal constraints. One might imagine specifying a sequence of sub-goals, and then reordering the policies for each sub-goal in turn. In this mode, a higher-level control would switch between hybrid control policies. While not infeasible, care must be taken to guarantee that each sub-goal is reachable from the previous. Furthermore, this requires the sub-goals, and their ordering to be defined prior to the policy orderings, which may lead to unrealizable sub-goals.

In this section we explore an alternative approach that automatically specifies this high-level "program" by specifying a sequence of policies that satisfy some specification. The approach is based on model checking, which is used to define a sequence of policies whose invocation will cause the system to satisfy a high-level specification [36, 37] . The planning occurs in the space of instantiated control policies using the prepares graph. The approach automatically checks that the specification is realizable, and automatically generates sub-goals as required.

To demonstrate the utility of this approach, a multiple-task scenario is simulated using the robot, obstacle environment, and policy deployment described in Section 6.1.1. 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 one or two-letter alphabetic labeling of the local policies, the specification may be given as

- Start in location BT; that is start within the domain of Φ_{BT}
- Pickup (visit) at either DE or FO, then deliver to GO (mail-room); that is, first go to either $\mathscr{G}(\Phi_{\mathrm{DE}})$ or $\mathscr{G}(\Phi_{\mathrm{FO}})$, and then go to $\mathscr{G}(\Phi_{\mathrm{GO}})$.

  Note, policies DE and FO have goal sets in approximately the same workspace position, but the goal sets are 180 degrees apart in orientation.
- Pickup GO (mail-room), then deliver (visit) CR; that is, from $\mathscr{D}(\Phi_{\mathrm{GO}})$ go to $\mathscr{G}(\Phi_{\mathrm{CR}})$.
- After CR, deliver to CF; that is, after $\mathscr{G}(\Phi_{\mathrm{CR}})$ go to $\mathscr{G}(\Phi_{\mathrm{CF}})$.
- Avoid BA, BB, BH, and HI; that is policies Φ_{BA}, Φ_{BB}, Φ_{BH}, and Φ_{HI} are invalid.
- Finish in location A; that is at the goal set of Φ_{A}.

The plan must generate motion that navigates between the regions BT and A in a way that satisfies the other specifications.

Given an LTL encoding of this temporal specification and the prepares graph, model checking techniques are used to generate an open loop sequence of policies that satisfy the specification[2] [36, 37]. The policies are executed in the same framework as the order-based approaches by encoding the sequence of policies as a tree where each node has a single child[3]. The key difference is that now the order must be maintained; the system cannot recover from a perturbation by searching the tree as an ordered list. This is because multiple nodes in the sequence may map to the same continuous

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

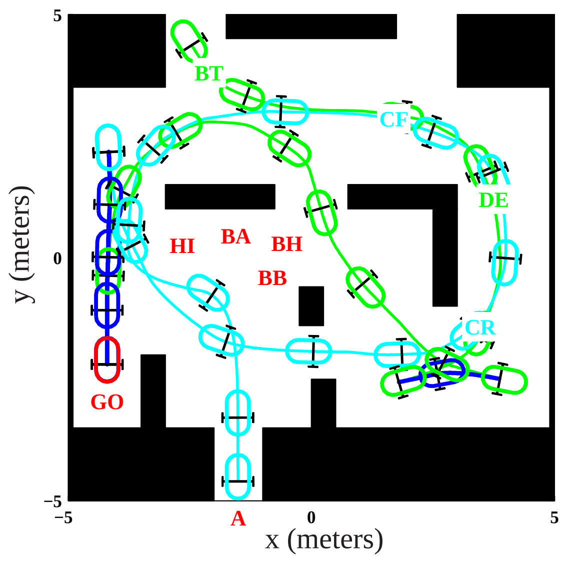
[3]The model checking-based approach used here does not allow non-deterministic transitions; therefore, transitions that depend upon the extended prepares relationship are eliminated from the prepares graph before planning.

policy. The ordering of policies is critical; in the event of a disturbance, the sequence must be re-planned with knowledge of what sub-goals have already been satisfied so that the specification can be modified accordingly.

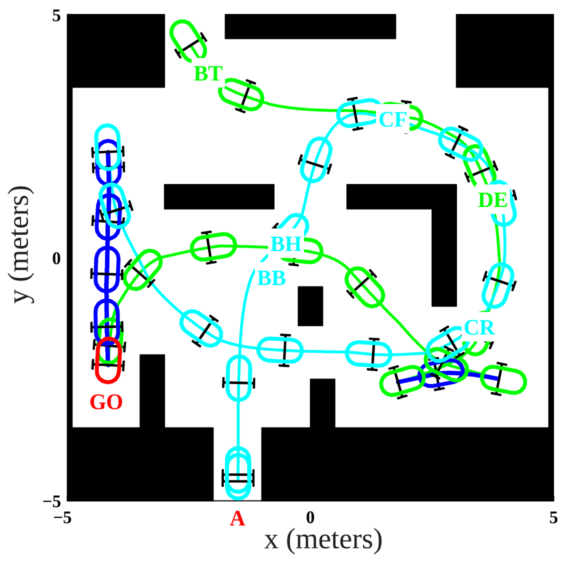
Given an initial position and orientation of the robot that lies within the domain of policy BT, executing the policies according to the given sequence satisfies the specification. Given the prepares graph and specification, the model checking procedure guarantees the result by construction. In this case, executing the sequence will first take the robot to either DE or FO, and then to GO, after reaching GO, the system reverses course and visits CR, and then CF, until finally reaching the goal at A. All the while, the system never executes policies BA, BB, BH, or HI. The simulation of this plan is shown in Figure 6.18-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 6.18-b.

This task level planning worked well in simulation, but limited attempts to execute the plans on the Deminer robot exposed some limitations. All runs ended in failure when a perturbation took the robot outside the domain of one of the policies in the open loop sequence. For the Deminer robots, these perturbations tended to be small as the localization was based on dead reckoning, and did not suffer jumps in pose estimates. The perturbations were mainly due to velocity control lag and caster wheel drag. In spite of the perturbations being small, the robot exited the domains of some local policies, primarily because the model checking-based planner chose some policies with relatively small domains.

These policies with small domains were chosen for two reasons. First, the model checking-based approach does not consider heuristic costs. In the order-based approach, policies with small domains are assigned relatively high transition costs, which means they are not high priority, and are only invoked if necessary. Second, the model checking-based approach does not consider non-deterministic transitions. Therefore, all policies that depend on the extended prepares relationship are invalid. Around tight turns, only relatively small domains remain valid. One approach to addressing this issue, is to use meta-policies with the extended prepares relationship and do model



a) Path #1



b) Path #2

Figure 6.18: Simulation of open loop policy sequences derived from temporal logic specifications.

checking on the meta-policy level. Fundamentally, the open-loop sequences lose the robustness inherent in the order-based approach because not all policies are considered; this limits the domain of the resulting hybrid control policy.

The model checking based approach is also rigid in the sense that it does not allow the system to react to changes in the environment without re-planning. To allow more flexible approaches that can react to changes, Kress-Gazit *et al.* [68] have developed an approach that uses the prepares graph defined by our approach to generate an automata that reacts to discrete sensor inputs.

6.3 Automata-based Planning

Like sequence-based approaches, automata-based switching strategies are capable of addressing multiple tasks; however, automata have the added advantage of changing behaviors during runtime based on gathered information without requiring re-planning. Combining the policy composition approach advocated in this thesis with automaton synthesis tools such as those of [68] enables a constructive approach to building a hybrid control policy whose continuous execution satisfies high level specifications, while enabling the constrained system to react to environmental changes.

This section presents several experiments and simulations using the synthesis approach given in [68]. As discussed in Chapter 4, [68] uses a disjoint workspace decomposition and adjacency graph to choose policies based on our fully actuated policies for idealized systems. In contrast, this section defines the specifications and automata synthesis in terms of the prepares graph. This approach is more flexible because it can be applied to constrained systems, and allows for overlapping policy domains.

The section presents two examples. The first uses the LAGR robot and policies from Section 6.1.2; both simulations and experiments are discussed. The second demonstration uses PF policies with an Ackermann steered vehicle to demonstrate complex traffic behaviors in simulation. The latter presentation includes a discussion of using this approach as the basis for a decentralized multi-agent control system.

6.3.1 'LAGR' Robot Experiments

The first example is termed the "timid night watchman." The LAGR robot is tasked with patrolling office corridors by visiting four checkpoints in turn. If an intruder is detected, as indicated by a binary sensor called 'Intruder', the robot is to "run and hide" in the small nook near the workspace center; after the intruder is gone, the robot should resume patrolling. The system also includes a 'Hazard' input; upon sensing a hazard, the robot should stop in place. The robot resumes motion when the hazard is clear. The robot has three outputs: 'Stop', which indicates that the robot should cease executing its local policy and stop in place, 'CheckPoint', which means the robot is at a designated checkpoint, and Φ_i, which encodes which policy is associated with the automaton node.

The desired behavior is encoded in linear temporal logic (LTL) and input to the automaton synthesis algorithm developed by [68]. The algorithm takes the initial conditions, transition relations, and goals, then checks whether the specification is realizable. A specification is *realizable* if an automaton that specifies valid transitions can be synthesized given the LTL inputs. If the specification is realizable, the algorithm extracts a possible, but not necessarily unique, automaton that implements a strategy that the system should follow in order to satisfy the desired behavior.

Using the specific "timid night watchman" task, the behaviors are encoded as follows. The Hazard input is initially False, and there are no other assumptions about the environment so both its transitions and goals are set to True. The Intruder input is allowed to be either True or False.

The system state is assumed to be in one of two initial policy domains, the initial policies are not checkpoints, and the system is not stopped by hazard. The system transitions include knowledge of the prepares graph. The system stops if and only if there is a hazard sensed. The system also encodes that the current policy reference does not change if the system stops. If an intruder is sensed, and the system is hidden in the nook, the system should stay in the nook. The system should always patrol if the intruder is not sensed. The CheckPoint output is set if and only if the robot is at a designated checkpoint policy. The desired behavior, given as the system goal, is that the system either stops or eventually visits each checkpoint in order infinitely often.

Together, the automaton and policy suite serve as a hybrid control policy. For these specifications, the extracted automaton has 2400 nodes. Executing the local control policies as specified by the automaton induces a continuous system evolution that satisfies the high level specification. At the start of execution, we search the entire automaton as a list of nodes until a node is found that has the correct input state (Hazard = False) and whose associated policy contains the initial pose. This approach, which allows starting from some arbitrary initial pose, works for this particular scenario because of the cyclic behavior of the scenario; other scenarios might require that the robot start in the domain of a policy in an explicit set of initial policies. A simulation run is shown in Figure 6.19. The intruder detector is triggered at an arbitrarily specified time.

The automaton governs the selection of local control policies. The automaton transitions between nodes as the system pose enters the domain of a policy associated with a child node of the current automaton node. In other words, from node p_i, at each time step[4], the values of the binary

[4]The policies are designed as continuous control laws; however, the implementation on a computer induces a discrete time step. We assume the time step is short compared to the time constant of the closed-loop dynamics.

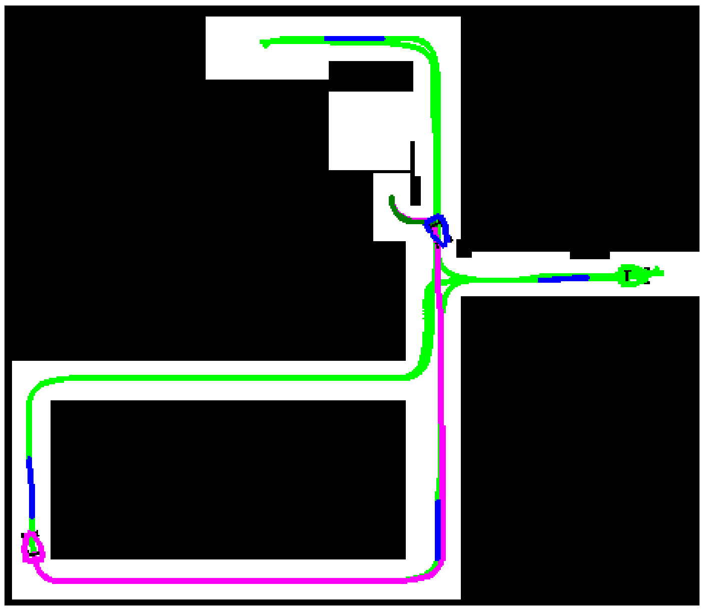


Figure 6.19: A simulation of path induced by an automaton that encodes the behavior patrol the corridors by visiting four specific policy domains is shown. Upon sensing a ‘Intruder’, the “timid night watchman” goes and hides in the corner until the intruder leaves. Three robots are shown: the initial pose to the right, the final pose when execution is terminated near the middle, and the pose at which the intruder is detected in the lower right.

sensor inputs are evaluated. Based on these inputs, all valid successor nodes are determined. If the vehicle is in the domain of policy Φ_l, which is associated with a valid automaton successor node p_j, the transition is made. Otherwise, if the vehicle is still in the domain of Φ_k, which is the active policy associated with node p_i, the execution remains in node p_i. If a node has more than one child node that represents a valid transition, the choice can be made arbitrarily. For these experiments, the first valid transition as defined by the synthesis algorithm is chosen. This execution based on continuous motion is equivalent to the discrete execution of the automaton [37, 61].

Figure 6.20 shows the progression of the system through the automaton nodes as the system moves through the environment. Note the cyclic nature as the system completes three patrols before the intruder is detected. As the automaton state transitions, so does the associated policy as shown in Figure 6.21.

In this execution strategy, the continuous evolution of the system governs the discrete transitions in the automaton; therefore, the resultant transitions are asynchronous, and not governed by a fixed time step. In this current implementation, the discrete inputs act as guards on the automaton transitions; the discrete input must match the value associated with the child node to allow transition into the child node, but does not force transition out of the current node. Another approach could check the discrete input at each update step and force transitions out of a given automaton node if the inputs do not match the reference input. This would require that each node has a child with the same policy reference, but different discrete inputs .

If the prepares graph changes, the automaton synthesis algorithm must be re-run. Figure 6.22 shows the simulated path taken when an automaton is synthesized for the prepares graph with 24 policies associated with the lower corridor invalidated. The resultant automaton contains 2580 nodes; its execution correctly satisfies the original specification by only invoking valid policies.

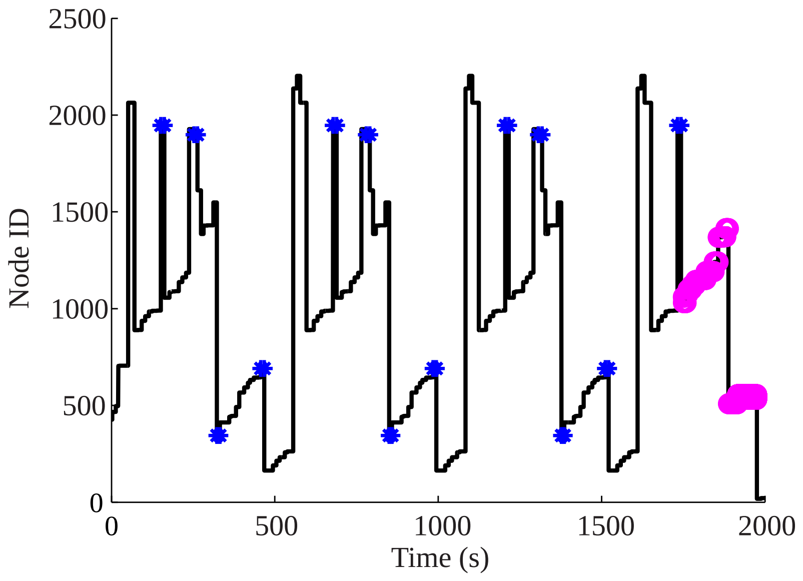


Figure 6.20: As the system executes, the automaton changes nodes based on the discrete inputs and inclusion of the current pose in a given policy domain. The graph shows three distinct phases. The thirteen points marked with '*' indicated the check points passed. The thickest portion, which is actually closely spaced '○' symbols, shows the portion where the intruder is detected. Notice that the system makes multiple passes past each checkpoint before the intruder is detected.

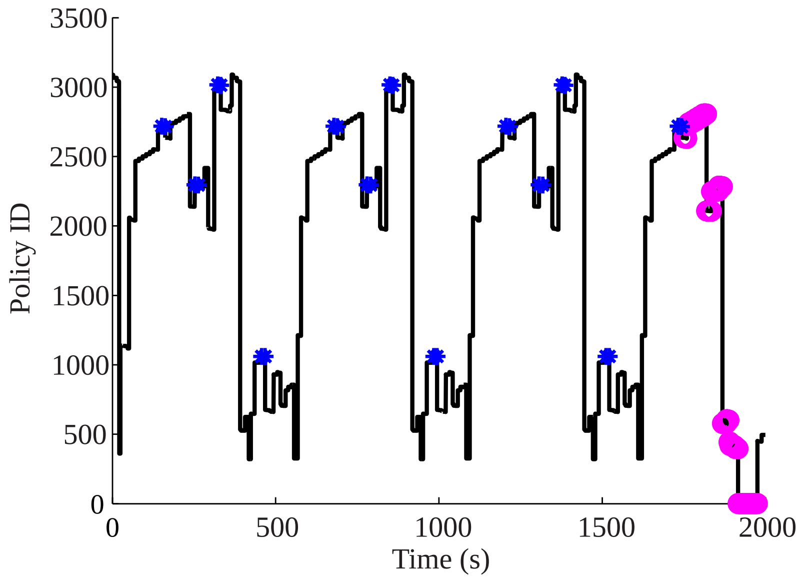


Figure 6.21: Each node in the automaton is associated with a particular policy in the suite. As the system executes, the local policies are activated by the automaton based on the local pose estimate.The graph shows the same three distinct phases as Figure 6.20.

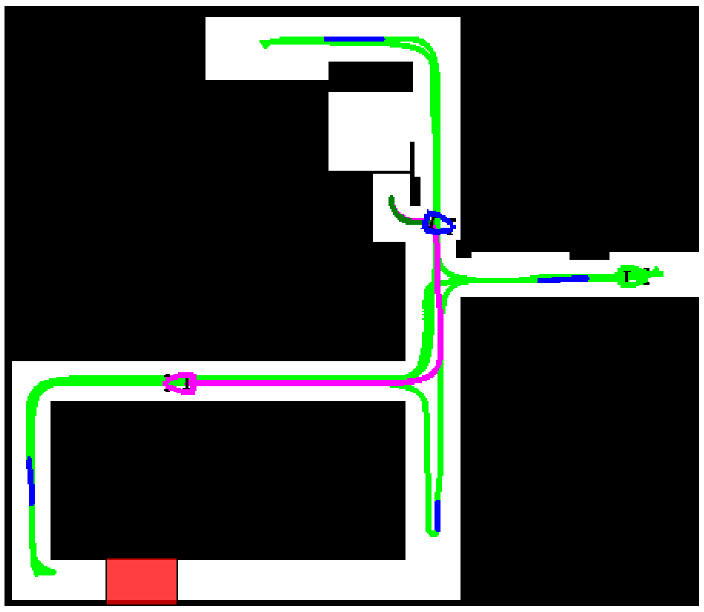


Figure 6.22: As new information becomes available, such the obstacle in the lower corridor, the automaton synthesis formulates a different automaton based on changes to the prepares graph. The new automaton preserves the correct behavior.

The automaton synthesis approach guarantees the correct behavior under very reasonable assumptions. First, the automata synthesis only returns an automaton if the specification is realizable for the given policy suite and associated prepares graph. Second, given a realizable specification, the algorithm is guaranteed to produce an automaton such that all its executions satisfy the desired behavior **if** the environment behaves as assumed. The construction of the automaton is done using LTL statements that encode admissible environment behaviors; if the environment violates these assumptions, the automaton is no longer correct. Since the specifications encode the transitions allowed by the prepares relationship, the only case in which the system pose is not in the domain of Φ_k, or in any successor Φ_l, is if the environment behaved "badly." That is, either some disturbance caused the policies to violate the prepares relationship, or the environment violated assumptions governing the allowable discrete inputs. This later case requires careful sensor design, with only those restrictions that are necessary. Either case invalidates the automaton. In the event that a valid transition does not exist, the automaton executive raises an error flag, and halts the system. A new plan must be requested.

Unfortunately, for real systems disturbances are a fact of life. Policies may be designed to be as robust as possible, but disturbances may still take the system out of the domain of a currently executing policy. Often these disturbances are simply due to pose estimation updates as described above. The hybrid control system should have a method of recovery, which will likely require some knowledge of the hybrid control system and task. For the task described in this section, our approach is to search the automaton as a list of nodes until a node whose associated policy contains the current pose estimate and whose discrete input matches the current sensor value; as is done for the initial condition. This works in this example because of the cyclic nature of the task.

A more fundamental problem occurs when the disturbance takes the system outside the domain of all policies in the automaton. Depending on the initial specification, the automaton synthesis does not necessarily use every available policy. As with sequence-based approaches, this has a negative impact on the overall robustness of the policy composition technique relative to the collection of available policies. This thesis considers two approaches to addressing the problem of unused policies.

The first approach explicitly allows the initial condition to be in any available policy and have any allowable sensor value. The assumption during synthesis that the system is in one of two initial policy domains is made to limit the size of the automaton. No assumptions about the initial policies could be made; this would force the automaton synthesis to include all policies, but would greatly increase the size of the automaton. The particular implementation of the synthesis algorithm used in this thesis precluded this approach; this is not a theoretical issue, later work will build a more robust synthesis tool to address this implementation issue [67].

The second approach, which is used in these experiments, is to augment the synthesized automaton to add nodes for each unused policy/sensor combination. If a policy is unused by the original automaton, but prepares another policy that is used for all input combinations, then a node is added to the automaton with the unused policy as a reference. This added node ignores the sensor inputs. The children of the added node are all nodes in the automaton whose associated policies are prepared by the added node's policy or have the same policy reference. Since all input combinations are covered, a valid child transition will eventually exist. This process is repeated until all policies that prepare others are added to the deployment. This approach maximizes the overall hybrid control policy domain for the given collection of domains, while adding the smallest number of nodes to the automaton. This gives the system a way to "get back on track." If the disturbance causes the system pose to exit the domains of every policy in the suite, then the hybrid control policy will stop the robot and cease execution. Only by adding additional policies, and regenerating the automaton can the system recover.

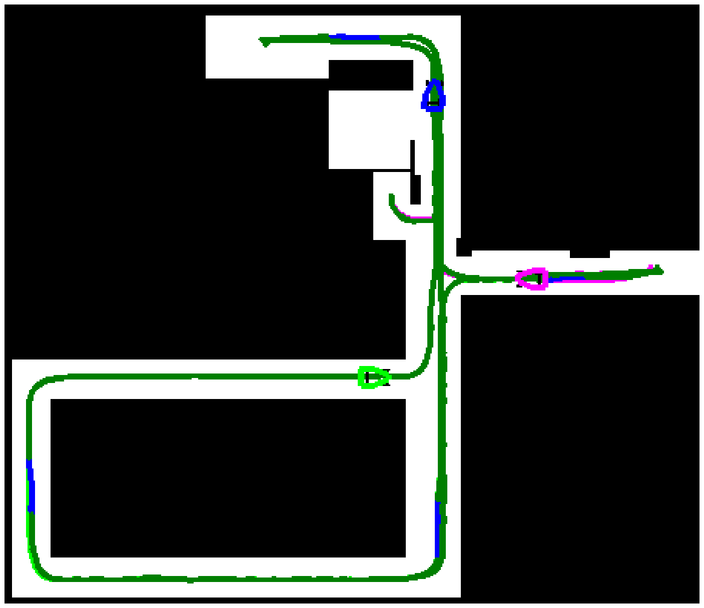


Figure 6.23: Actual run on LAGR robot. Here, the robot resumes patrolling after hiding early in the experiment.

Figure 6.23 shows an example run using the augmented automaton. During the experimental runs, the ‘Intruder’ is signaled at will via a remote switch. The experiment successfully satisfies the specification. Figure 6.24 shows the progression of nodes during execution. Note that the node ID’s above 2400 are those added during the augmentation process; without these, the execution would have ceased earlier due to disturbances. Given the augmented automaton, the system is able to search for a node whose policy contains the current pose. Eventually, the execution did quit when a disturbance finally took the system out of the domain of all the policies. Figure 6.25 shows the policy switching induced by the augmented automaton. The experiment was repeated several times; the automaton successfully induced the correct behavior each time until disturbances caused the system to terminate; this points to the need for more policies to be added to the policy suite.

The drawback to the augment and search approach is that there is no history; therefore, the system will sometimes repeat an earlier portion of the patrol loop, prior to visiting the other nodes. This problem could be addressed by adding an output that encodes which “downstream” check point will be encountered next, and using this information to guide the search for a valid node. This requires associating each policy with the closest checkpoint before the synthesis. One possible approach is to choose the checkpoint that generates the least cumulative cost for a given policy from a set of costs generated by considering each checkpoint as the goal of a policy ordering. During disturbance recovery, the system searches for a node whose associated policy domain contains the current pose and whose “ClosestCheckpoint” output matches the assigned checkpoint for that policy.

The automata-base approach is capable of producing complex behaviors, which allow the system to react to changes in the environment via the binary environmental inputs. Additionally, the automata-based approach allows the system to exhibit desirable limit cycles; in this example, repeatedly patrolling a hallway. Thus automata-based approaches are more suitable for repetitive

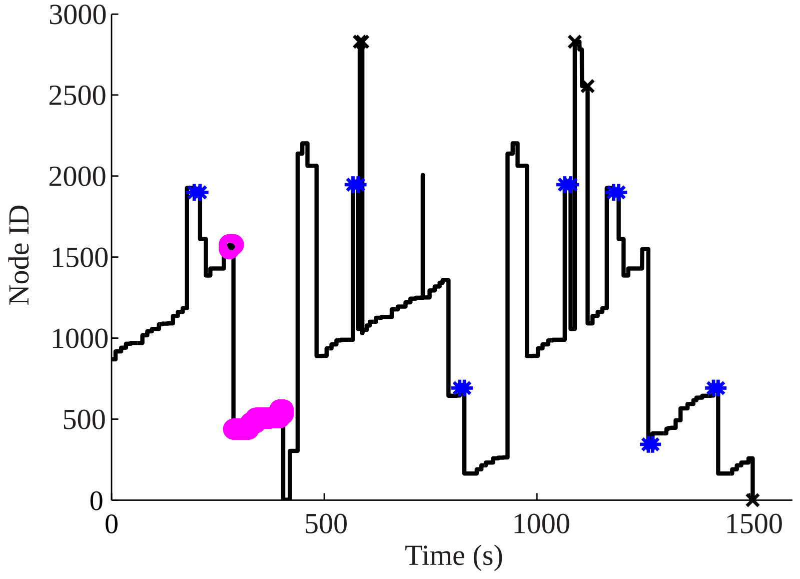


Figure 6.24: Node switching with invocations of augmented nodes shown by 'x'; the controller would have ceased execution were it not for these added nodes.

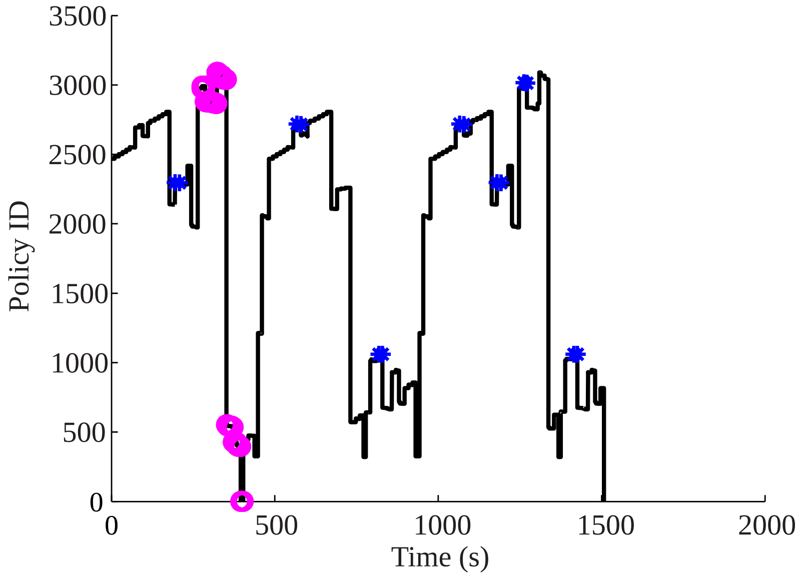


Figure 6.25: Policy switching during an experiment.

tasks than order-based approaches. That said, the automata should make use of all available policies, and provide a method of recovery, in order to maintain robustness to disturbance that is the hallmark of order-based approaches.

6.3.2 Ackermann Steered Car-like Parking Simulations

This section provides an example of policy-based planning with the more complex system model of an Ackermann steered car. Here, the scenario is one of searching for an available parking space, and then parking. The environment is known; what is unknown is whether a given parking space is available or occupied. The system has a local sensor for detecting open parking spaces; thus, the system must search for an available parking space by systematically driving past all the parking spaces. If an open parking space is found, the system changes behavior from searching to parking, and executes the parking maneuver as illustrated in Figure 6.26. The results demonstrate coupled planning and control for a complex system that exhibits complex behaviors that change based on reactions to the changing environment.

The environment, shown in Figure 6.27, consists of two city blocks accessible from ten entering roads. Each road consists of two lanes that follow the American standard of driving on the right side. One block is surrounded by 40 parking spaces; 20 for the clockwise direction and 20 for the counterclockwise direction. The entry/exit points are labeled 1-10 clockwise starting from the north/south lanes at the top left of the environment. The parking spaces are identified with a numeric identifier adjacent to each space. The roadway lanes and parking spaces are sized for an urban environment. The robot system uses an Ackermann steered kinematic model that controls the forward velocity and the rate of steering angle change; see Appendix F for details.

The parking demonstrations use a collection of 16 PF style policies, which are instantiated in the policy cache relative to the origin. The cache includes policies for traveling straight down a roadway lane, for parking and leaving a given space, and for turning at intersections. Figure 6.28 shows examples of the policies for parking and leaving, which treated as meta-policies for planning purposes. Associated with the inlet policy of the parking policy is a sensor that determines whether the parking space is available. If the parking space is unavailable, then the parking meta-policy prepares some other policy further down the roadway lane. Figure 6.29 shows an example intersection, the deployed policies, and the extent of the robot body into the workspace. Since this is a

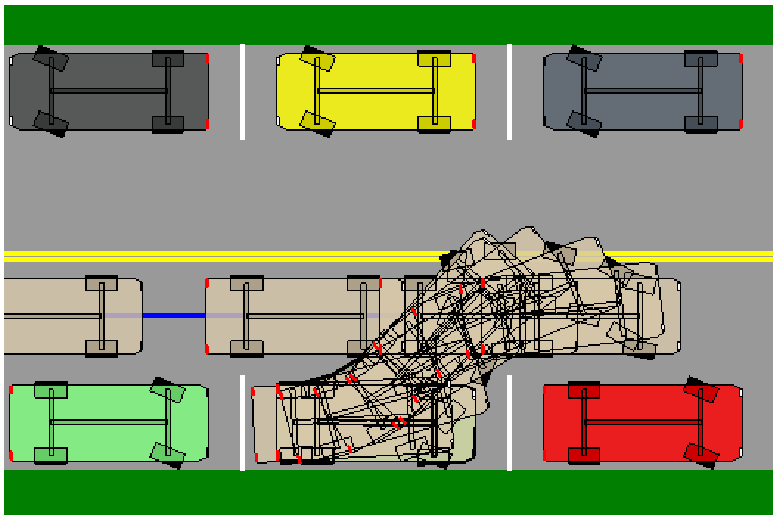


Figure 6.26: Parking behavior induced by the composition of local policies. The feedback control policies guarantee the safety of the maneuver.

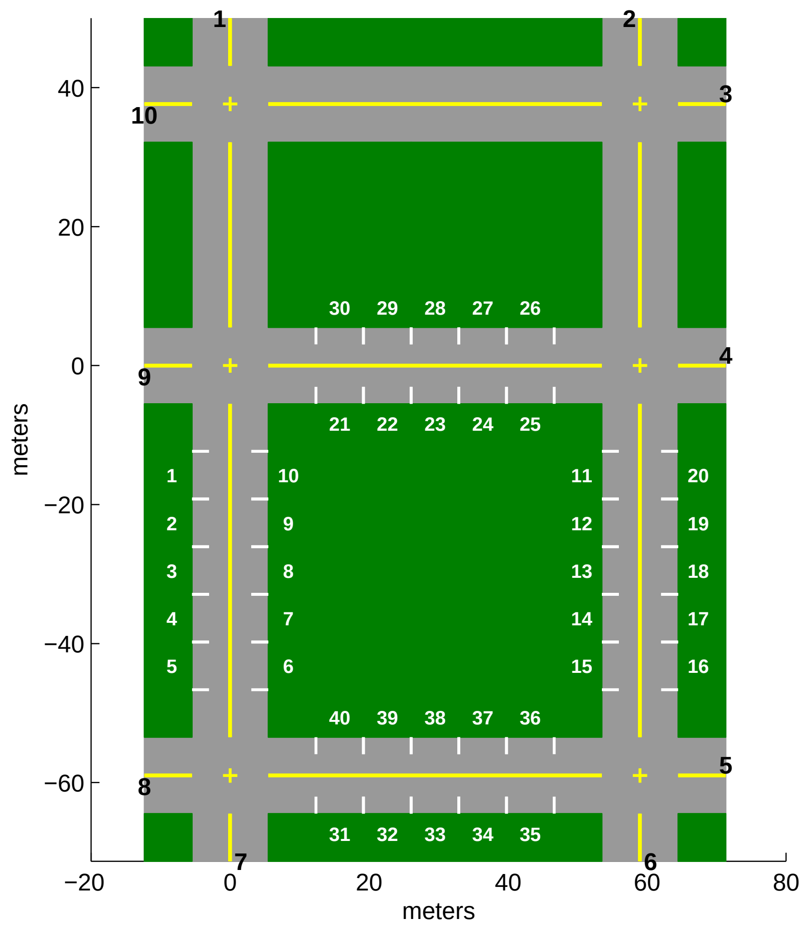
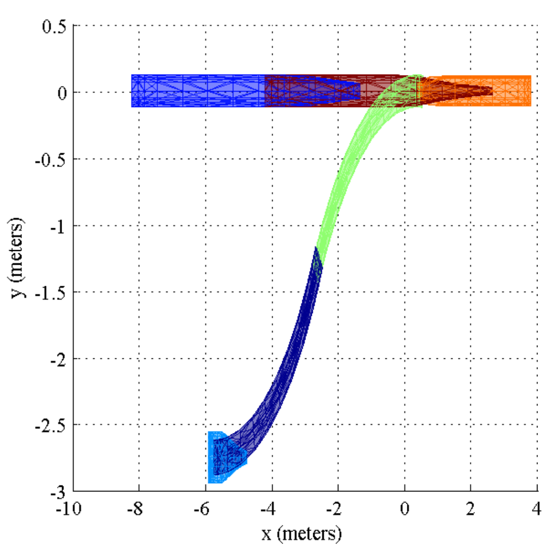


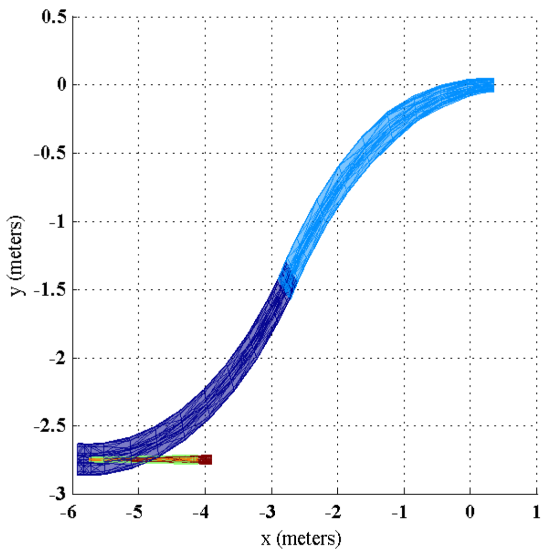
Figure 6.27: The environment has 40 parking spaces arranged around the middle city block. Initially, there are five empty parking spaces randomly chosen in the environment.

simulation, only those policies needed for basic traffic are deployed. No attempt is made to fill the free pose space in order to provide robustness.

For the simulations in this section, a total of 306 policies are deployed in the environment. The regularity of the environment allows an automated approach to policy instantiation based on a collection of reference points defined relative to the intersections and parking spaces. The policy total includes 40 parking meta-policies and 40 leaving meta-policies, as well as 24 each left, right and straight maneuvers at the six intersections. Policies to enter and leave the environment are added at the 10 roadways connecting the environment to the outside world. Given the suite of 306 policies, the prepares graph is automatically defined as described in Chapter 5.

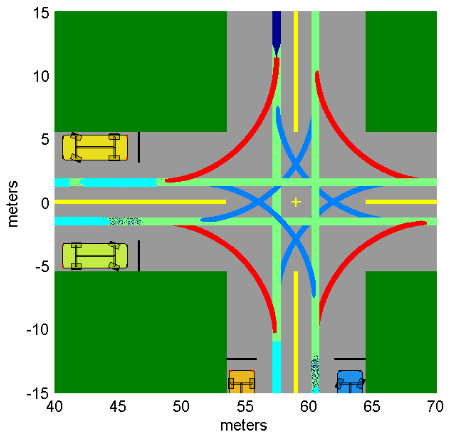


a) Policies for parking.

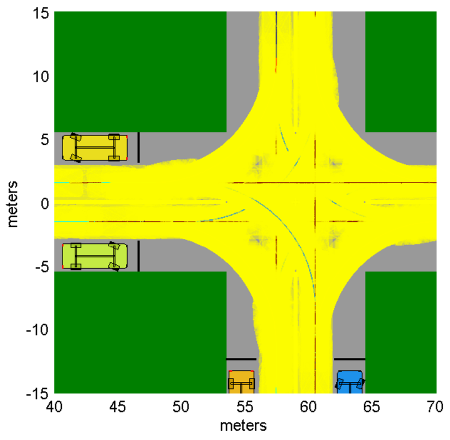


b) Policies for leaving

Figure 6.28: Details of policies used for parking and leaving. The policies, which are shown relative to the cache reference point, are shown wider than normal to show details. Six policies are associated with parking. Five policies are used to exit a parking space and prepare policy in the traffic lane.



a) Connected policy domains projected into workspace



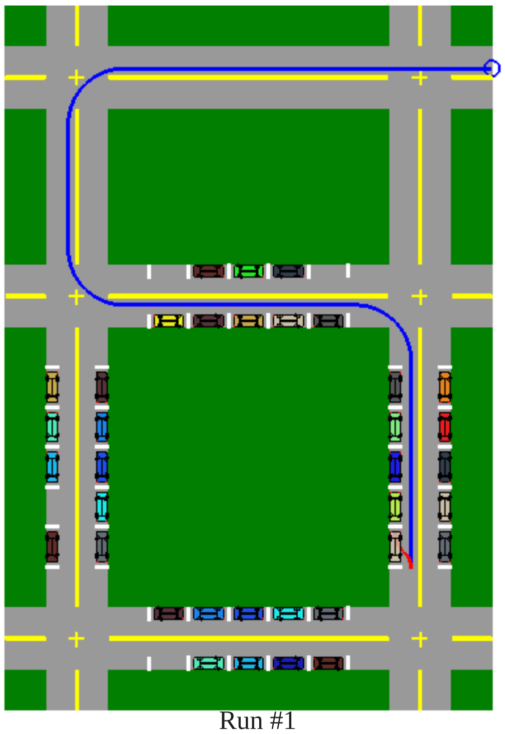
b) Body extent over the policy domains

Figure 6.29: Deployment of policies at an intersection. The polices include those that pass straight through the intersection, as well as left and right turns. Other policies are used to tie the straight sections to the turns. The policy domains, which are widened to increase visibility, appear as thick lines in (a).

Basic Parking Scenarios

The basic scenario considers a single car that must park in the environment. The environmental input is a sensor called ‘Park’ that tells the car if a parking space is available; the system output identifies which policy to activate. The car may enter from any of the ten roadways connecting to the two blocks. The car can only determine whether there is a free parking space if we are in a policy next to it. This means that ‘Park’ cannot become True if the vehicle is not next to a parking space or in one. Also, for implementation reasons, we assume that the input ‘Park’ remains True after parking. We have no assumptions on the goals of the environment, and make no assumptions about the availability of an empty parking spot. The allowable system transitions include the transitions of the prepares graph, the vehicle cannot park if there is no parking space available, as indicated by the ‘Park’ input, and if there is an empty parking space the car must park; removing the last restriction may allow the vehicle to pass an open spot before parking. The system goal encodes a list of policies the vehicle must visit infinitely often if it has not parked yet. The list of policies to visit defines the area in which the vehicle will look for an available parking space; in this case, the visit policies correspond to the eight lanes around the parking spaces (four going clockwise and four going counter clockwise). Note, this goal condition is true if either the vehicle visits these policies infinitely often (when there is no parking space available) or it has parked. Defining a different list of policies to visit would change the search strategy induced by the automaton. Additional specifications could be written to tie the search strategy to the point of entry, but this would increase the size and complexity of the automaton.

For simulations shown in Figures 6.30 and 6.31, a new vehicle is introduced at a random entrance. The parking spaces are filled according to the previous run. As the automaton executes, if a parking policy is a successor to the current state, the empty/occupied status is checked via the local ‘Park’ sensor. This work does not address the required sensor, but assumes a binary output. Transition to the parking policy is enabled if the associated space is empty. If the transition is enabled, ‘Park’ remains True so that other transitions are disabled until the vehicle pose enters the domain of the parking meta-policy, and the system parks. Six runs are simulated using the global parking automaton; The first five runs park. In Run #6, there are no parking spaces available; therefore, the vehicle continues to circle past every possible parking space, waiting on another vehicle to leave.



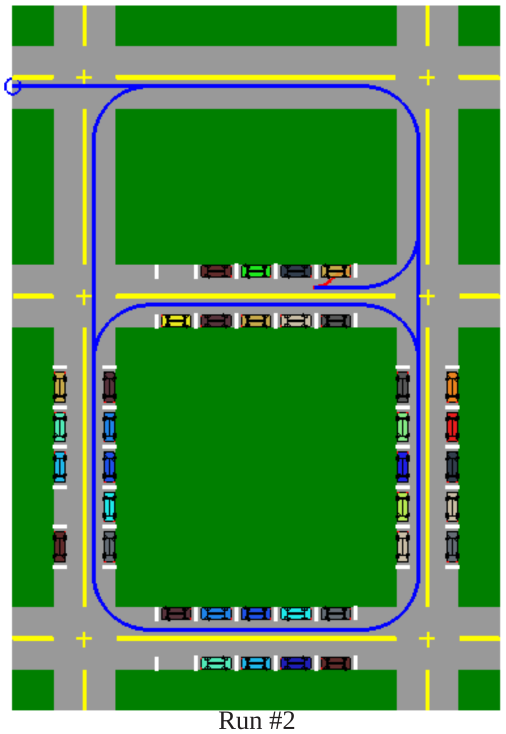


Figure 6.30: Two executions of the basic parking scenario. The initial conditions for each run are circled.

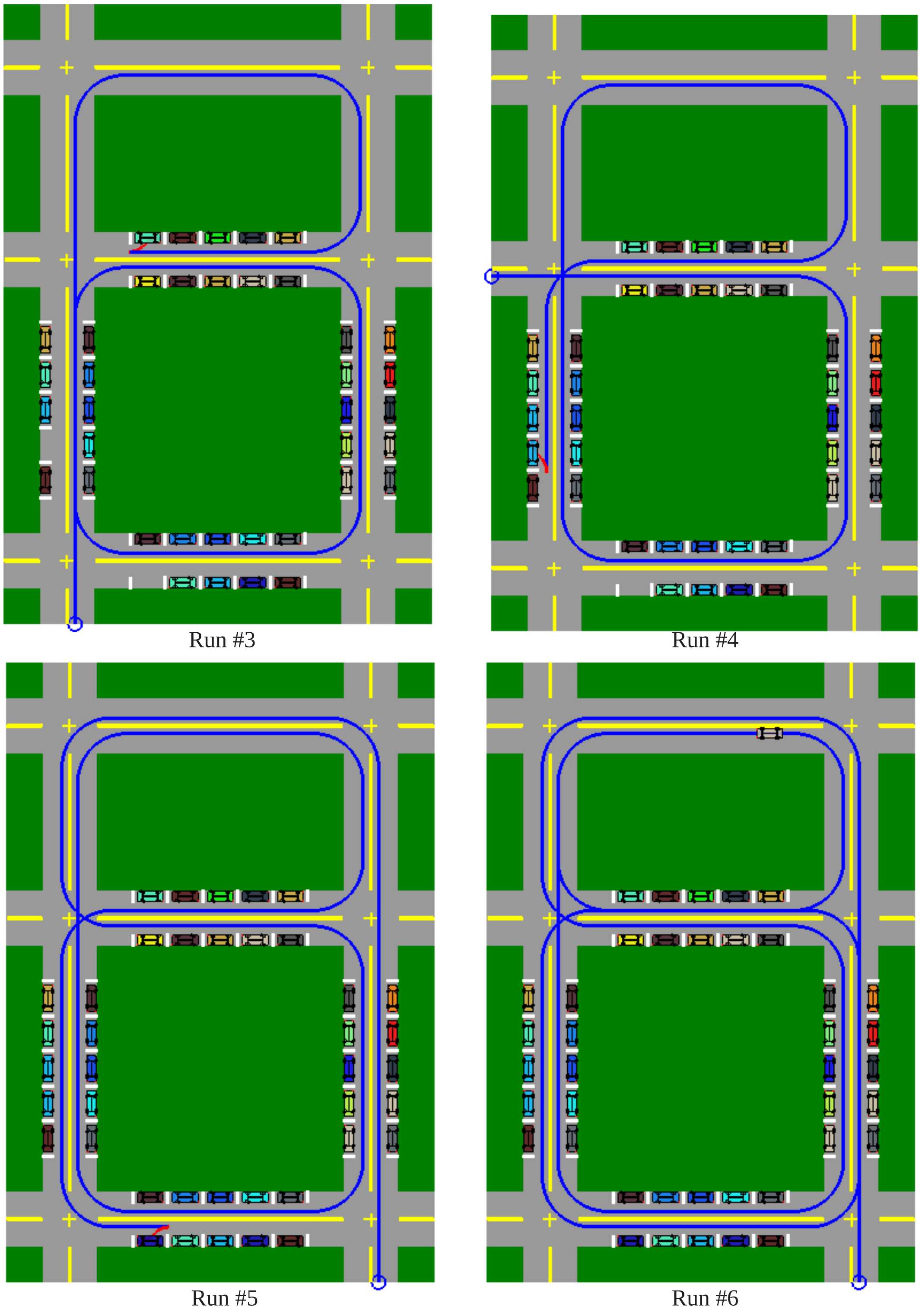


Figure 6.31: Four executions of the basic parking scenario. The initial conditions for each run are circled. The last run continues to loop as no parking spaces are available.

6.3.3 Multi-vehicle Scenarios

The automata-based approach to policy composition naturally extends to multi-agent systems [68]. The local policies guarantee predictable local behavior of a single agent; the automata governs the switching between local policies to coordinate the high-level behavior of an agent. Taking this approach further, Kress-Gazit *et al.* [68] use the environmental inputs to coordinate behavior between agents using automata. Each agent runs its own automata-based hybrid controller, which responds to other agents via environmental inputs; that is, the outputs of one agent become inputs to another agent. This section details a simulation using the policy composition approach advocated in this thesis with the automata-based multi-agent coordination scheme advocated in [68]. The simulation results illustrate several issues that arise with this approach.

In order to expand the basic parking approach to allow for multiple vehicle scenarios, the LTL formulas from above are modified. The approach uses an additional input and several outputs. The additional input is 'Hazard', which causes the vehicle to stop in place. The hazard can be triggered by proximity to another vehicle, or by an external device such as a stop-light. When the hazard clears, the robot should resume motion as before. In a real system, many hazards can be avoided by slowing down, and waiting for the other vehicle to clear. For simplicity, these simulations require the system to stop. When the vehicle stops in response to a 'Hazard', the system outputs 'Stop'. Additional outputs signal '$\mathrm{LeftTurn}$' and '$\mathrm{RightTurn}$' as appropriate. There are also outputs that signal the vehicles intentions for 'Parking' and 'Leaving'. The automaton outputs can be sensed by other vehicles in the environment.

The LTL specifications from above are modified to take the new inputs and outputs into consideration, and allow a new "leaving" behavior. Each vehicle in the simulation runs a local copy of one of two automata. The only coordination is via the individual 'Hazard' sensor. We now consider each automaton in turn.

Parking Automaton The parking automaton for the multi-vehicle scenario is similar to the individual case, but includes the stopping behavior and the additional outputs. The system transitions include all the conditions of the individual parking case, plus the conditions that activate the outputs for turning, stopping, or parking as needed. The system goal is includes the parking conditions, but also allows for a vehicle to remain stopped if a broken stop-light or accident ahead blocks the roadway. With these specifications, the parking automaton has 2142 nodes.

Leaving Automaton In this scenario, a vehicle is leaving its parking space and exiting the block via some specified exit. The leaving automaton for the multi-vehicle scenario has an extra input that specifies which of the ten possible exits the vehicle will exit. The initial environment specification is such that only one exit is specified. Two different vehicles leaving two different parking spots may use the same synthesized automaton with different inputs that designate the desired exit. We require the exit specification to be constant, meaning it cannot change once it is given. We make no assumptions on the infinite behavior of the environment, therefore the goal component remains set to True. Initially, the car is leaving a parking space, hence it must turn on the left turn signal.

The system transitions are include the policy prepares relations, which policies turn on the left/right signals, and always stop on hazard. The system goal specifies that the vehicle must go to the designated exit policy unless it stops. With these specifications the leaving automaton has 1908 nodes.

The key to using these automata in a decentralized multi-agent scenario is the coordination provided by the 'Hazard' sensor. Each vehicle executes its own hazard sensor with a single binary value 'Hazard.' The 'Hazard' input is based on either a timed stop-light or a proximity/precedence sensor.

The stop-light alternates between north/south and east/west travel along the roadways. Each intersection transitions at the same time; there is a slight overlap where all directions are stopped. Any vehicle entering the policies just before the left/right/straight policies at each intersection checks the current value of the stop-light. If the "red light" is visible, the 'Hazard' flag is set to True.

The 'Hazard' sensor is a discrete hybrid automaton in its own right, that attempts to determine precedence based on the robot's internal state and binary outputs, and the other robots relative pose, velocity, and binary outputs. Thus the "sensor" is a mixture of continuous measurements and discrete logic. The 'Hazard' checks proximity of other vehicles, and determines the precedence relationships between vehicles; that is, which vehicle must yield to the other. For this simulation, the 'Hazard' sensor is hand-coded and tuned to given the proper performance. The sensor automaton sets 'Hazard' to True whenever the car is too close to a car ahead of it (keeping safe distance), whenever a car ahead is backing up to park (being polite), whenever the car is leaving a parking space and another car passes by and whenever another car is leaving a parking space which the car will park in next. In this decentralized coordination scheme, each vehicle's 'Hazard' sensor must infer the intentions of the others based their outputs. There is no centralized communication of intentions.

Figure 6.32 shows the continuation of Run #6 with the hazard inputs added to the parking automaton, and the new leaving automaton controlling the second vehicle. In the first snapshot, vehicle #6 is just beginning to approach the intersection, while vehicle #7 stops for the light. The second snapshot shows vehicle #7 dutifully waiting for the signal, while vehicle #6 has passed through the intersection. Although not shown, after the stop-light changes, vehicle #7 exits the area

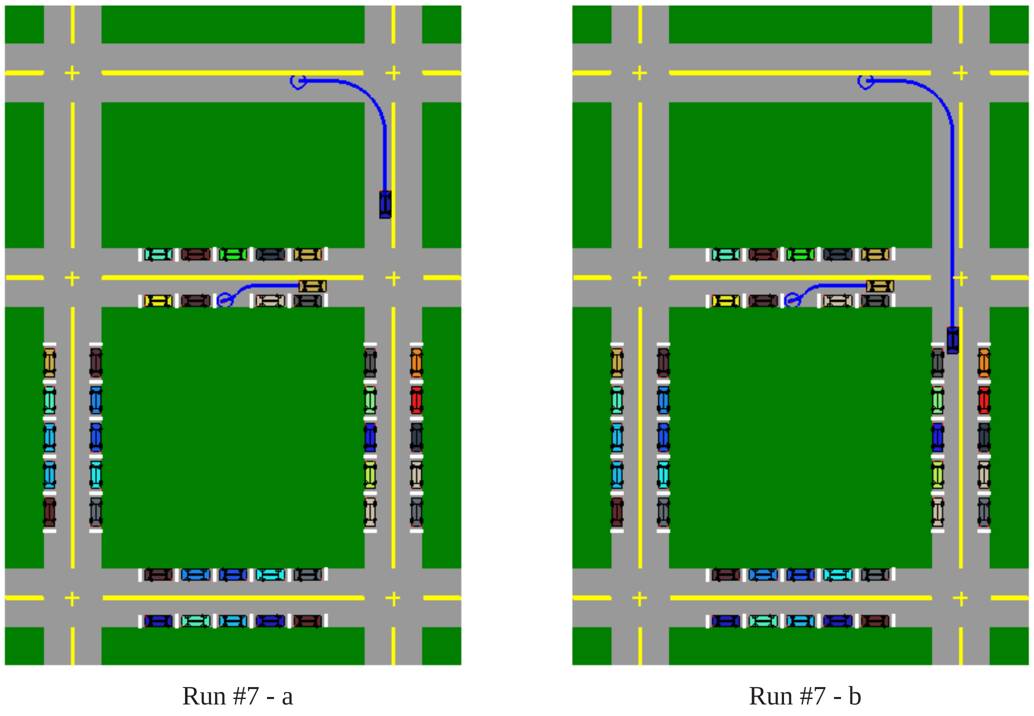


Figure 6.32: In this continuation of Run #6, the two snap shots show a simple multiple vehicle scenario. A timed stop-light triggers a 'hazard' input that causes the vehicle heading east to stop. This allows the vehicle from Run#6 to travel through the intersection, and eventually park in the newly available parking spot.

and vehicle #6 continues around under the control of the global parking automaton and parks in the newly open spot.

Figure 6.33 shows an example of a more complex multi-vehicle simulation. At this point in time, seven cars are moving in the workspace. Initially, 35 of the 40 parking spaces were randomly specified as occupied. In this simulation, eight cars enter the block at different times and from different entry points, looking for a parking space. The times and entry points are (t=0.06 seconds, Entry = 10), (1.0,2), (2.0,7), (5.0,8), (7.0, 5), (10.0,6), (15.0, 8), (22.0,5). During the execution, three cars leave their parking spaces and exit the workspace. The times, parking spaces, and exit point are (t=13.0, Parking=23,exit=1), (15.0, 6, 7), and (30.0, 32, 5). The simulation runs until 76.33 seconds of elapsed time when the last car exits or is parked. Figure 6.34 shows a general snapshot of the simulation at a later time. Cars whose 'Stop' output is True are marked with red ellipses; that

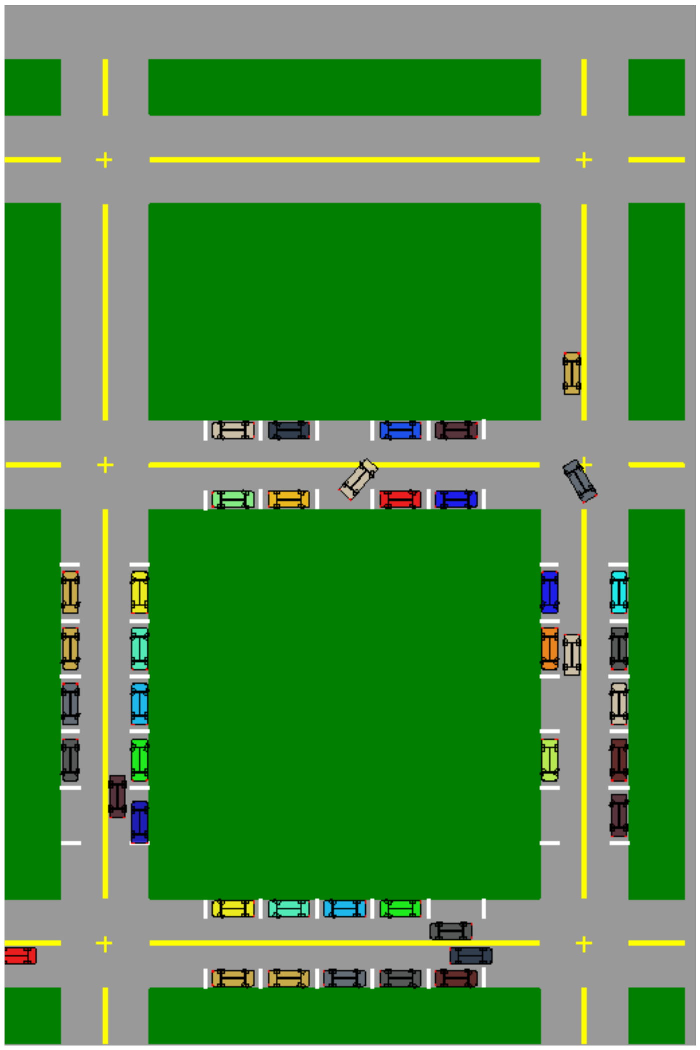


Figure 6.33: A snapshot of a more complex multi-vehicle simulation. Each vehicle executes an automaton that encodes the high-level specification "stop on hazard" and either "drive around until you find a free parking space and then park" or "leave your parking space and exit the block". Coordination between robots is done via an individual 'Hazard' sensor in a decentralized approach. This snapshot is taken at 15.91 seconds.

is, those cars who stop because the 'Hazard' input is True. The three stopped cars in Figure 6.34 are obeying stop-lights.

Figure 6.35 shows several close up looks at different traffic behaviors encountered during the simulation. In (a), the blue car which is leaving the parking space has stopped, indicated by a red ellipse, to let the brown car drive by. This 'Hazard' was invoked based on a "proximity sensor." In (b), red car is parking while the blue car waits for it to finish before passing. In (c), the orange car is stopping to allow the gray car to complete a left turn, according to the precedence established by the individual car's 'Hazard' sensors. The white car on the left is leaving the parking space that later will be occupied by the brown car. Figures 6.35-d and (e) are two snapshots of two cars parking simultaneously in opposite lanes. The car that started the parking maneuver later (bottom lane) pauses to allow the other car to park safely. Figure 6.35-f shows two cars stopping before a stop-light. While the white car stopped based on the stop-light, the black car behind stopped based on the proximity to the car ahead of it.

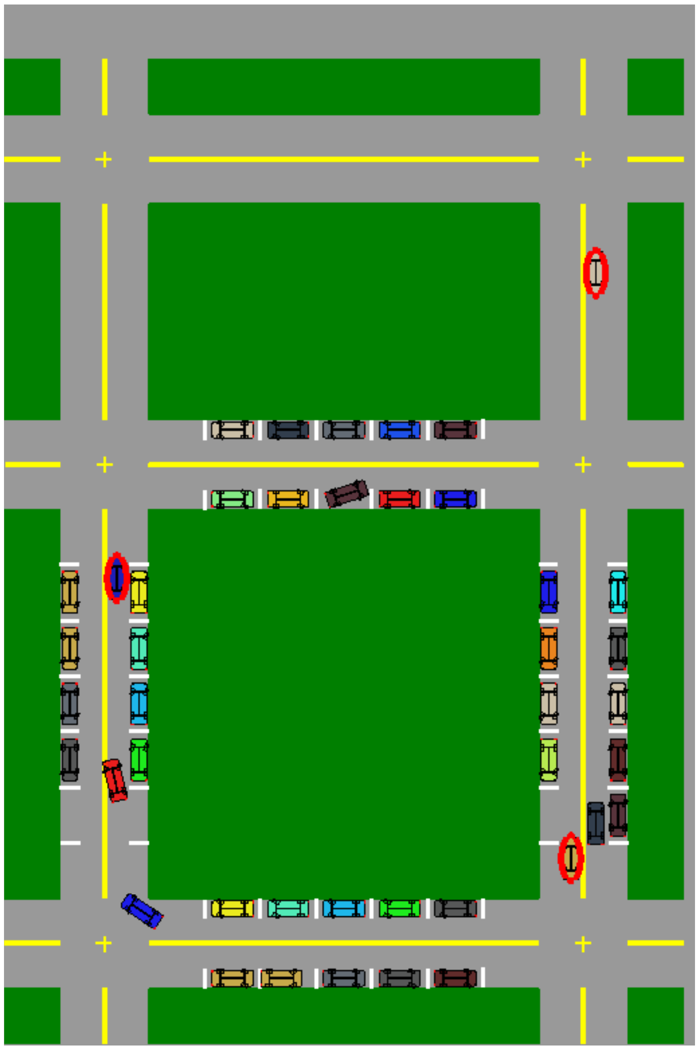
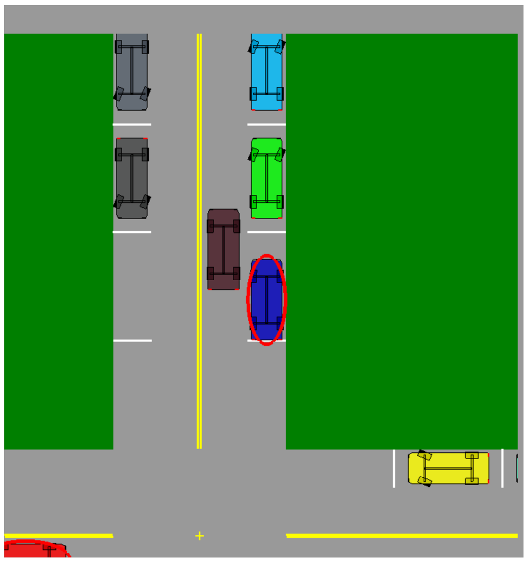
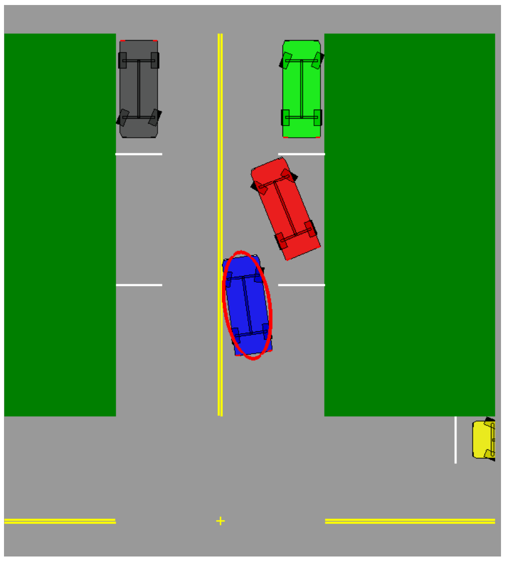
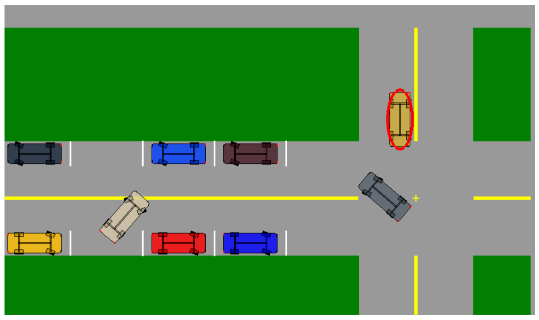
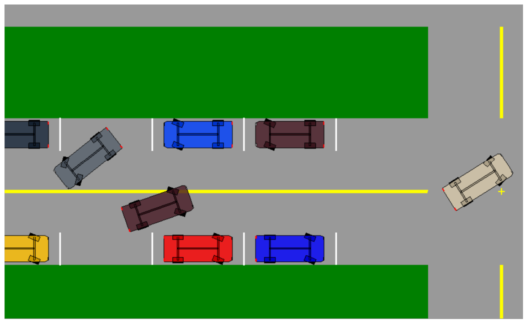


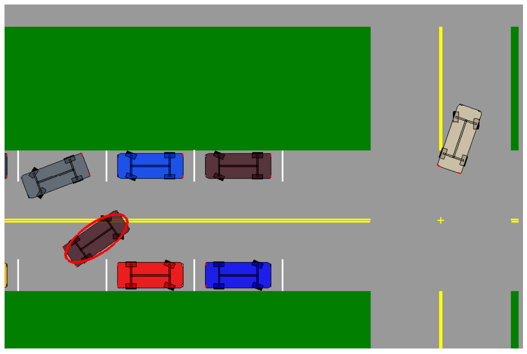
Figure 6.34: A later snapshot taken at 31.33 seconds during the simulation. In this figure, cars surrounded by red ellipses are cars that are stopping due to the 'Hazard' input signaled by the timed stop-light.


(a) Blue car leaving (t=15.91 s)


(b) Red car parking (t=34.69 s)


(c) Yielding to turn in progress (t=16.29s)


(d) Two cars parking (t=26.41s)


(e) Two cars parking (t=27.18)

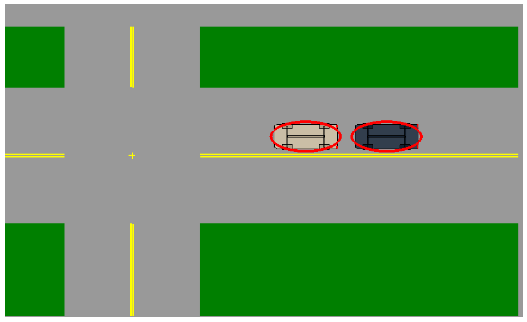

(f) Two cars at stop-light (t=46.39s)

Figure 6.35: Close up looks at different behaviors seen throughout the simulation.

Sensors, or more specifically the binary inputs used by the automaton, are fundamental to the success of this decentralized approach. First, as mentioned above, the sensors must satisfy the assumptions made about them in the LTL formulas for the environment; otherwise the automaton will not be correct. Failing to trigger 'Hazard' may allow collision as the local policies do not consider obstacle avoidance. Second, even if the sensors do satisfy these assumptions, they may still cause correct, yet unintended behavior. For example, if the proximity sensor set the 'Hazard' input to True whenever another vehicle was in a certain radius, even if the other vehicle was behind in a forward driving lane, both vehicles may get deadlocked; that is, both would stop forever. While this behavior satisfies the original specification, it does not follow the spirit of finding a parking space. On the other hand, both cars stopping might be a desired behavior when an accident occurred, therefore we would not want to forbid it in the specifications.

Currently, there are no guarantees that the implemented 'Hazard' sensor automaton is correct in all cases, and will not introduce deadlock. Such unintended behavior would not be present in a centralized approach where the controller has full knowledge and not just local information as is the case here; however, the centralized approach does not scale well. The decentralized approach, which does scale well for additional robots, may deadlock for a poorly designed hazard sensor; thus, much work remains to develop automatic ways of specifying the 'Hazard' sensor automaton and prove that the composition of these multiple automata is free of deadlock.

6.4 Summary

This chapter has presented several experiments which validate the approach advocated in this thesis. The approaches to planning in the space of control policies, and composing local policies to induce the desired behavior, is demonstrated with experiments on real robots and simulations on realistic systems. A range of planning approaches and scenarios are demonstrated. To our knowledge, this is the first experimental verification of these techniques on real wheeled mobile robots with non-circular body shapes; that is, body shapes where orientation is fundamental to the safety of the approach.

Several broad conclusions can be drawn from these results. In general, order-based approaches are preferred over sequence based approaches due to the enlarged domain; this is in keeping with the aim of designing "global" policies. Automata-based approaches are useful for generating complex reactive tasks; the policy composition approach advocated in this thesis extends these techniques to real world, complex systems.

Overall, the results validate the approach; however, several issues have been identified. First, the policies can only induce behaviors that the system execute. If the mechanical system is incapable responding to the controls, the properties of composable policies will be violated. Thus, either the system dynamics must be modified, the policies redesigned, or additional policies added to provide more robustness. Since disturbances are a fact of life, automata-based approaches should make use of all available policies in keeping with the global policy theme, and provide a method of recovery in the face of large disturbances. The hybrid control policy should also have a method of identifying undesirable limit cycles, and have a recovery strategy. Finally, while the automata-based approach to decentralized multi-agent control is promising, the design of a hybrid sensor automata, which can provide provably correct performance with the composition of individual automata, remains an open problem.

Chapter 7

Conclusion and Future Work

This thesis extends sequential composition of local feedback control policies to wheeled mobile robots in a way that enables the automatic synthesis of hybrid control policies. The resulting hybrid control policy inherits the safety and convergence guarantees from local feedback control policies, and provably satisfies the high-level behavior by construction. This thesis demonstrates this approach on real mobile robots with multiple interacting constraints. The robots, which have non-circular body shapes in addition to nonholonomic constraints and input bounds, operate in confined and cluttered environments. This thesis treats these constraints in a holistic manner, and enables existing methods of formal symbolic planning to be applied to these highly constrained systems. By leveraging symbolic planning techniques, complex tasks can be specified at a high-level, and then executed in a manner that guarantees the correct behavior on real systems. We define the basic requirements for local policies to be composable in a hybrid control framework, which guides our policy designs. While wheeled mobile robot navigation is the chosen domain, the ideas in this thesis are extensible to many constrained dynamical systems provided one can define composable policies.

The approach uses the composition of memoryless state feedback control policies to address high-level task specifications in a provably correct manner. This thesis develops several generic feedback policies that encode the low-level behaviors in a way that enables their formal composition, and demonstrates several symbolic planning approaches on real mobile robots. The symbolic planning methods automatically define switching strategies among the local policies that realize high-level behavioral specifications, or indicate that the desired behavior cannot be realized with the current suite of instantiated policies. This approach enables a formal method of constructing near-global hybrid feedback control policies that respect local constraints.

This chapter provides a summary of the thesis contributions and discusses the approach's strengths and weaknesses. The discussion points to future research, which will improve the approach and extend its applicability. The goal is to allow even more complex systems to benefit from policy composition in a way that guarantees formal correctness, and provides a natural method of specifying complex behaviors.

7.1 Contributions

This thesis enumerates several composability requirements that must be satisfied before policies can be composed in the hybrid control framework: *i)* domains lie completely in the free state space of the system, *ii)* the system must reach the designated goal set in finite time, *iii)* under influence of a given policy the system trajectory must not depart the domain except via a specified goal set, and *iv)* the policies must have efficient tests for domain inclusion given a known state. While not surprising, these requirements guide the evaluation of specific policy designs, and suggest tests to validate a specific policy instantiation.

New tests are developed in order to verify that the local feedback policies satisfy these composability requirements. The thesis develops an approach to verify that the policy domains are collision free without constructing the free configuration space. Our approach is based on expanding the policy domain to account for the body extent, and then testing the projection into workspace for collision. Using a discrete approximation of the cell surface, the approach uses an exact mapping to points on the expanded cell. These expanded points are projected to workspace, where the resulting tests are trivial. The thesis presents proof of correctness for the expanded cell approach to collision testing. For the remaining composability requirements, the thesis presents validation techniques based on the specific policy designs.

This thesis introduces composable flow-through policies to the sequential composition paradigm. Flow-through policies allow the designer to put off the implications of Brockett's theorem, and design smooth time-invariant polices for nonholonomic systems over a local region. The constraints of Brockett's theorem are realized through the switching behavior of the hybrid control policy. Flow-through policies naturally encode many desired navigation behaviors, but introduce added complexity in the prepares test. The policies must now consider the full system state when evaluating the prepares relationship; that is, second-order systems must also include a velocity test.

The standard prepares relationship between policy domains is extended to allow a policy to prepare a set of policies, without preparing any one policy in the set. This extension adds flexibility during policy instantiation to define larger goal sets, which tend to enable larger policy domains. The extended set-based prepares definition introduces non-determinism into the prepares graph used for planning; thus this added flexibility comes with a cost that must be born by the discrete planner. The thesis used D*-lite to address the non-deterministic transitions [81].

We have developed two families of generic feedback policies that are applicable to several nonholonomic systems. These policies, which are detailed in the appendices, form the foundation for the experimental results presented in the thesis. It is important to note, that these policies are only examples of composable policies. Any policy that satisfies the composability requirements may be used in the policy composition framework.

To aid in the deployment of the policies, the thesis demonstrated an approach to semi-automated policy instantiation based on a limited cache of policies that induce basic behaviors. Although specifically applied to the policies introduced in Chapter 5, the approach is general and can be applied to any policy that meets the composability requirements of Chapter 3. The thesis developed a technique for evaluating the relative completeness of a given suite of policies; this gives a qualitative method of evaluating one suite of policies against another.

Finally, the thesis provides several demonstrations of the coupled planning and control framework using policy composition. Both simulations and experiments are presented using a variety of system models in constrained environments. The policy composition approach demonstrates emergent behaviors, such as K-turns, during simple navigation, as well as complex multi-task behaviors governed by automata. The automata are automatically synthesized based on the suite of local feedback control policies instantiated using our techniques. This thesis represents the first known experiments with these approaches on constrained systems operating in cluttered or confined environments.

Benefits of Policy Composition Since the hybrid control policy is based on local feedback control policies, the overall controller inherits the properties of the individual policies. This allows the individual policies to be tuned to local conditions, whether to provide safety or enhanced performance. The local policies, in order to be composable, have provable convergence guarantees, and retain the robustness to disturbances that is the hallmark of feedback control. The local feedback control policies are designed to be memoryless, and allow for real time control. Because each local policy has an explicit domain, the hybrid control approach is inherently safe. If a disturbance takes the system outside the domains of all policies, the robot is halted and execution of the hybrid control policy terminates.

By planning in the space of control policies using the prepares graph, the planning becomes very flexible with regard to task. This approach opens the door to formal synthesis of hybrid feedback controllers for complex systems; for example the parking controller demonstrated in Chapter 6. The approach allows analysis of the reachability of a goal, or realizability of a specification, with a given policy suite during the discrete planning phase, prior to execution.

Using automata to execute the local feedback policies allow the systems to exhibit complex, multi-task behaviors. The approach enables tasks to be specified at a high-level, and then executed in a continuous manner, using a hybrid control policy synthesized from the suite of local control policies and associated prepares graph. Repetitive tasks are naturally encoded in the framework, which allows the approach to induce limit cycle type behaviors.

Drawbacks of Policy Composition Unfortunately, the power and flexibility of policy space planning does not come for free, and it not applicable in all situations. The design of suitable policies is not trivial. The policies must have explicit domain representations in order to quickly evaluate the suitability of a given policy, which precludes the use of many simple discrete representations. Designing suitable domain representations requires insight into the system and its constraints, and the environment at hand.

Given a set of generic composable policies, there is significant upfront cost to instantiating and validating the policies. This upfront cost is mitigated by two factors. First, is the flexibility of planning in the space of policies, as demonstrated in this thesis. The second factor is the ability to reuse existing deployments within a known environment.

The demonstrations in this thesis assume a static known environment. The policies in this thesis do not adapt moving obstacles, or unknown obstacles, except in the limited case of invalidating whole policies within the suite and re-planning using D*-lite. This thesis does not explore adding policies as an environment is explored; thus, the current approach is not well suited for initial exploration of an unknown environment.

The approach is limited to those behaviors that are instantiated. If there are not enough policies deployed, or they fail to cover a large enough fraction of the free configuration space, the approach may not be robust to significant disturbances. The discrete planner can only take advantage of policies that are previously instantiated. Thus, there is a implicit demand that the designer consider the needs of the system during definition of the policy suite. This points to the need for more automated methods for policy instantiation that can be applied on-line if significant disturbances are encountered.

The policies demonstrated in this thesis rely on vehicle pose estimates. The safety of these policies is dependent on accurate localization. Repeatable disturbances due to the localization can induced unwanted limit cycles if they violate the monotonic switching of the orderings. The hybrid control approach needs a supervisor to recognize and address this problem.

7.2 Future Work

This thesis has demonstrated the usefulness and flexibility of the policy composition approach on real constrained systems. There remain several fruitful avenues of exploration that build upon this thesis; these include work to overcome the drawbacks mentioned above, as well as extension to more complex systems. Several directions offer opportunities for multi-disciplinary collaboration between computer scientists, engineers, and control theorists.

7.2.1 Extension of the Basic Approach

Building on the foundation provided by this thesis, there are two areas that need further study.

Disturbance Quantification As described in this thesis, disturbances are a fact of life that must be dealt with for any real system; local feedback policies coupled with the order-based approaches to hybrid policy design offer some inherent robustness to such disturbances. Problems remain where large disturbances take the system outside the domain of all policies, or repeatable disturbances induce undesirable limit cycles. At present, we do not have a quantifiable description of when these disturbances are "too large", or likely to induce undesired limit cycles. Robustness analysis of the local policies, and more importantly the overall hybrid control policy, is an open area of research. We would like to provide guarantees such that disturbances within a certain bound and rate of occurrence will not induce undesirable limit cycles, and will remain within the domain of the overall hybrid control policy.

Hierarchical Design The component policies and meta-policies represent one type of hierarchy described in this thesis. The synthesized hybrid control policies, both order-based and automata-based, are another level in the overall hierarchy. In this thesis, the definition of meta-policies was strictly an engineering choice based on intuition gained by working with the component policies. Grouping component policies in meta-policies reduces the burden on the planning algorithm by reducing the size of the prepares graph, but also limits the planning flexibility. To date, we do not have a formal method of evaluating the choices of individual component policies versus various groupings in meta-policies, other than the basic computation complexity of determining the prepares graph for larger collections of policies.

Another level of hierarchical design would treat a synthesized hybrid control policy as a meta-policy within some higher-level framework. For example, the LAGR experiments were conducted on one floor of a building. Each separate floor would have its own deployment of local policies, with connections provided by the elevators. One option is to combine the policies from every floor into one large suite of policies, with its associated prepares graph. Another, more scalable option, is to treat each floor as a meta-policy, and then plan at both the floor level, and then between floors at the hybrid meta-policy level. Determining the appropriate level of abstraction and number of layers within this hierarchical framework is currently an *ad hoc* decision based on engineering intuition.

Going one step further, and considering an automata-based hybrid control policy as a meta-policy within a hierarchical framework leads to a notion of *hybrid prepares*, where the prepares test depends on both the continuous goal set of the overall goal policy and the discrete outputs of the automaton. Thus, work remains for incorporating these tools within a large scale fully autonomous framework.

7.2.2 Extension of Policy Design Techniques

To fully realize the benefits of policy composition, additional design tools must be developed. The design of the composable policies for the nonholonomic systems described in this thesis only required three dimensions, yet the specification of the domains required much thought. As the system complexity increases along with the dimension of the configuration space, the ability of human designers to specify composable policies becomes even more challenging. In this section, we outline several directions for research that will expand the ability to design composable policies, both for the systems considered in this thesis and the more complex systems mentioned later.

Sensor-based Policies The policies in this thesis were based on knowledge of full state information, which required localization. One can design policies that use sensor based measurements to define policy domains and goals [55, 97]. For example, consider the flow-through policies of [97] that move a vehicle through a doorway using visual servoing. As long as the policies satisfy the composability requirements, sensor-based policies can be readily incorporated into our policy composition framework.

Value Function Approximation One of the challenges of policy design is to design a policy domain that acts as a funnel. That is, it captures a relatively large region of state space, but brings the system to a relatively small goal set, allowing for simple prepares tests. The geometric approach followed in this thesis is somewhat limited, but provides the ability to test for collision via the expanded cells and test for state inclusion during execution.

On the other hand, optimal control techniques can use dynamic programming to find a value function that corresponds to the maximal cell definition. This approach normally depends on discrete representations that lack simple inclusion tests. One natural approach uses a finite set of basis functions to approximate the value function [43]. The combination of basis functions can be used to quickly check for state inclusion and test for collision as demonstrated in this thesis. The flexibility of the basis function approach will likely allow for the definition of more expressive cells; however, the selection of the proper set of basis functions is something of an art. The composability properties must be verified for the approximate function, and not for the value function used in the initial optimal control problem.

Local Reactive Planning With the increases in computing power and memory, the line between control and planning is becoming more blurred. Many systems, most notably those participating in DARPA's "Grand Challenges", are using local planning in real time to determine control inputs that define certain trajectories [28]. The systems use local planning to react to obstacles, while using conventional grid based planning, or provided way points, to define the desired path. Assuming the system is capable of doing this planning fast relative to the system dynamics, the local planner acts as a feedback control policy.

This opens the door to combining local reactive planning with the policy composition approach at the high level. By defining cells that provide boundaries on the planning, and then planning within the cells, the system has the freedom to react to unexpected obstacles within the cells, while maintaining a predictable transition between regions. One can imagine defining cells based on road lanes, or other geographic data. Thus, the local planning can be directly incorporated into the policy composition/automata synthesis approach advocated in this thesis. This opens the door to formal methods of guaranteeing high-level behaviors, while preserving the low-level ability to react to unexpected changes on the local level.

Another possibility is to use local planning as an exploration strategy, but use cells and policy composition as a compact representation of free space. This allows one robot to explore the environment using some technique, and then share data with other robots in the form of a suite of cells and prepares graph. This approach would benefit from machine learning/function approximation approaches to define and instantiate the cells on-line during exploration. The robot that is exploring the unknown space instantiates cells and defines the prepares graph for the other robots during execution. The suite of policies and prepares graph is a compact representation of the available free space. The suite and graph representation would also be useful for recovering from deep dead ends where maintaining a full cost map is impractical.

Model-based Local Control As the policy composition and hybrid control approach is extended to systems with second-order dynamics, or complex high-dimensional systems, defining closed form controllers will become difficult. One alternative is to use model-based control methods, such as Model Predictive Control (MPC), to specify control actions [42]. Here a finite set of control actions is evaluated at each step, and the best performing series of actions is chosen. With MPC, the evaluations are based on the outcomes of discrete simulation steps. This approach is related to dynamic programming and optimal control methods; however, here the approach is based on a greedy finite horizon simulation. If a conservative domain representation can be found, over which the MPC approach is guaranteed to find a solution, MPC can be incorporated into the design of local control policies.

7.2.3 Extension to More Complex Systems

The real payoff for policy composition is with more complex systems, whose dynamics are fundamental to the control. Planning methods must take these dynamics into consideration during planning, otherwise the plans will not be feasible. This represents an obvious path for continued research, but one that requires advanced policy design techniques.

Purely-Kinematic Systems with Second-order Shape Dynamics Directly building on this thesis, the policies should be modified to account for second-order dynamics on the shape space. This thesis only considers nonholonomic systems with first-order dynamics; the natural extension is to consider direct control of torques, and account for second-order motor and inertial dynamics. This could allow for more aggressive control techniques that account for system limitations. The control of the shape variables is fully actuated; therefore, a variety of control techniques are available on the bounded shape space.

Mixed-Mechanical Systems Another natural extension is to apply the approach to so called mixed-mechanical systems, such as the snake-board [112]. Due to the interesting mathematics of such systems, recent work has focused on developing gaits for these systems; however, the work has generally focused on open-loop control in obstacle free environments [112, 96]. Research should use the intuition gained from the open-loop gaits to design feedback policies with explicit domains and goal sets. Then, the policy composition approach advocated in this thesis opens up these systems to address real navigation and control tasks.

General Dynamical Systems Researchers often come up with systems with complex dynamics, whose performance is limited by available control methods. As an example, consider systems that are not statically stable such as the balancing "ball-bot" and systems than are capable of true bipedal running via energy storage [49, 74]. Control policies for these systems must respect the system dynamics during any transition between behaviors. While some results have been obtained designing controllers for very specific behaviors, it is our hypothesis that the systems will require formal analysis of policy composition to generate useful behaviors. To switch between steady-state behaviors, the system must respect the dynamics and the domain of attraction of each local policy. Thus, the design of composable policies becomes a fundamental challenge for moving these systems into the real world in a way that allows for robust behaviors and complex tasks. Provided composable policies can be found for these systems, the various planning tools demonstrated in this thesis allow for planning of real world task for these systems.

As another example, consider today's humanoid robots. The zero-moment point control (ZMP) approach is based on keeping the robot in a stable configuration as the system moves [54]. This limits the behaviors of the robot, as the system cannot pass through a transient unstable configurations. The formal policy composition approach offers a chance to extend the capabilities of such systems by designing composable control policies for different regions of the robot's state space, and formally composing them using synthesized automata. Hybrid control policies can be used to induce cyclic behaviors such as walking or running that are more expressive, while enhancing the performance and safety of the system. The automata will be used to switch between behaviors, such as balancing, walking, running, kicking, and climbing, based on the system's instantaneous state, while reacting to the systems hybrid dynamics induced by intermittent contact. The key is to develop approaches to synthesize the hybrid control policy automatically, in a way that provides formal guarantees across the state space of the system. Analyzing the system for "composability" may also lead to insight into designs that simplify the control design by the addition of passive elements that remove or add energy at certain points in the state space.

7.2.4 Extension of Planning Tools

The demonstrations provided in this thesis highlight the power and flexibility of the policy composition approach. They also point to several shortcomings that should be addressed in the discrete planning domain. Addressing these issues will increase the power of the policy composition approach for the policies and systems described above.

Sensor Automata and Composition The automata synthesis approach described in Chapter 6 depends on sensors that provide binary signals to the automata during run time. As with the 'Hazard' sensor, these "sensors" are often hybrid automata in their own right. That is, the binary sensor values depend on a mixture of continuous variables and discrete logic. Defining these sensors is currently done on an *ad hoc* basis. Techniques are needed to synthesize these sensor automata, and prove that the composition of multiple sensor and control automata are valid, and preserve the desired specifications and liveness conditions. An example, taken from the simulations in this thesis and the DARPA Urban Grand Challenge (UGC), is the need to resolve precedence at intersections and four-way stops. Formal synthesis methods coupled with local policy composition offers a way to automate what is currently a labor intensive, error prone process; as evidenced by the failures of most of the teams that entered the DARPA UGC.

Formal Recovery Methods and Global Synthesis Disturbances are a fact of life. Thus, the automata synthesis should include all available policies to maximize the domain over which the automata is valid, and have a formal method of recovery. For the repetitive behaviors, such as patrolling, demonstrated in this thesis, it is sufficient to augment the automata with unused policies, and then search for a node that had the correct discrete sensor values and whose associated policy contained the current pose. A more complex scenario, such as the mail delivery robot, will require a more formal recovery approach to allow the system to recover gracefully. Instead of an *ad hoc* approach, a formal and automated approach to defining a recovery strategy is desired. The recovery approach should also include all policies to maximize the domain.

Automata Synthesis with Heuristic Costs A major shortcoming of the automata synthesis approaches demonstrated in this thesis is that they do not consider heuristic costs. Within a given policy suite, there may be many policy combinations that will address a given scenario. In fact, this is desirable for maximum planning flexibility. The synthesis approach needs to be able rank different policy choices based on their relative cost. Current techniques only consider the number of transitions made, that is the number of edges traversed in the graph walk, when choosing policies. The focus of current techniques is on dealing with the state explosion problem using efficient data structures such as Binary Decision Diagrams (BDD) [20, 23]. There has been some work in combining BDDs with heuristic search; for example, consider the Set{A*} approach [52]. That work may prove fruitful for automata synthesis research.

Automata Synthesis with Non-deterministic Outcomes The automata synthesis used in this thesis only considered deterministic prepares graphs. This eliminated the use of the extended prepares relationship. There has been some work on defining sequence based approaches which allow non-deterministic prepares graphs [62]; however, the synthesis techniques for reactive automata do not allow non-deterministic transitions at this time.

Automata Synthesis with Hybrid Stability Analysis The stability of the order-based hybrid policies is based on the assumption of monotonic policy switching. With automata, limit cycles are allowed. In the hybrid systems community, it is well known that switching between stable vector fields can induce instability [13, 30, 79]. While this is not an issue for the kinematic systems addressed in this thesis, stability analysis will be fundamental to more complex systems. There are several approaches to analyzing the stability of existing hybrid systems [13, 30, 79]; however, going in reverse, the synthesis problem must address this issue during construction.

To conclude, this thesis advocates an approach to specifying robot controllers that respects low-level constraints by design, and provides a natural interface for specifying user intentions. Low-level feedback control policies induce local behaviors in a guaranteed manner. The user interacts at a high-level to specify intended behaviors. The robot then automatically synthesizes a hybrid control policy that can realize the intention, or reports that the goal is not realizable for the current collection of policies and initial condition. The hybrid control policy inherits the desirable properties of the local feedback policies, while guaranteeing the high-level behaviors. This approach, which is demonstrated on a class of nonholonomically constrained systems in this thesis, is widely applicable, and likely necessary for the dynamically capable robots of the future.

Appendix A

Modeling Framework

This appendix provides a detailed presentation of the modeling framework used in this thesis, and sets the notation used throughout the document. In addition to definitions of the relevant terms, we provide detailed derivations of the models used in the examples. First, this appendix provides a generic description of the environment and notation for the generic navigation problem. Within this context, the distinction between workspace, configuration space, and state space is highlighted. Next, the section presents the model used for nonholonomic Pfaffian constraints. The appendix continues with a discussion of the geometric relationships among configuration variables and the nonholonomic constraints. Finally, the section concludes with the presentation of the specific system models used in this thesis.

A.1 Work space, Configuration space, and State Space

The robotic 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 $\mathcal{W} \subset \mathbb{R}^2$. To address the navigation problem, the robot body must move along a path that reaches the overall goal, while avoiding obstacles along the way. For this thesis, the obstacles in the workspace are represented as the union over a finite set of convex regions $\{O_k\} \subset \mathcal{W}$, where O_k denotes the k^{th} convex obstacle. The obstacles are assumed to be in a known fixed location.

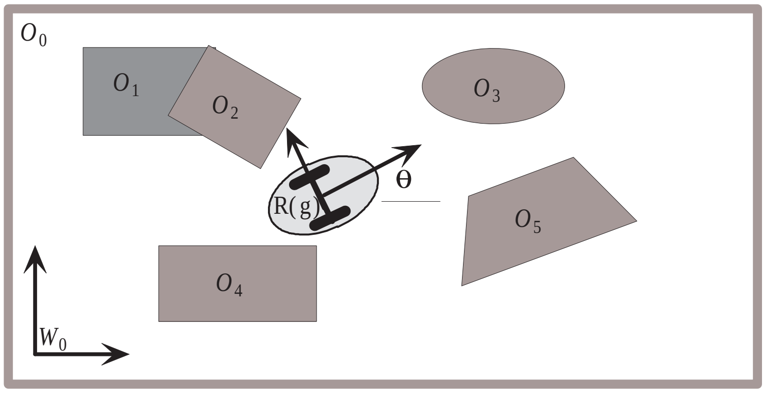


Figure A.1: Representation of planar workspace with five obstacles and robot. The workspace frame is denoted W_0; the body pose relative to W_0 is denoted $g = \{x, y, \theta\}$. The body occupies $R(g) \subset \mathcal{W}$. The workspace boundary is denoted as O_0.

Let $g = \{x, y, \theta\} \in \mathcal{G} = \mathbb{R}^2 \times \mathbf{S}^1$ denote the body *pose*, which is the position and orientation of a body fixed reference frame relative to the world frame. This relationship is shown in Figure A.1. Let $R\left(g\right) \subset \mathcal{W}$ denote the two-dimensional set of workspace points occupied by the body at pose g. Thus, for all body poses, g, along a collision free path,

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

For this thesis, $R\left(g\right)$ is assumed to be a convex set that is fixed relative to the body reference frame of the robot.

To fully specify the robot, certain internal variables must be specified in addition to the body pose. These internal variables are referred to as *shape* variables, as they are typically internal variables such as wheel rotations, steering angles, or joint angles [1, 96]. 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 space and $\mathcal{Q}$ denotes the configuration space of the system. The shape space $\mathcal{R}$ may or may not be bounded. The free configuration space, denoted $\mathcal{Q}_{\text{free}}$, is the set of all collision free configurations; that is

$$\mathcal{Q}_{\text{free}} = \left\{ q = \{g, r\} \in \mathcal{Q} \mid R\left(g\right) \bigcap \bigcup_k O_k = \emptyset \right\} \, .$$

The configuration evolution is governed via inputs to the system. The relation of input to configuration velocity is specified by the equations of motion for the system; these equations of motion must be derived for each system. There are two fundamental classes of systems. For *kinematic systems*, the control inputs $u \in \mathcal{U}$ directly control the configuration velocities; that is $\dot{q} = f\left(q, u\right)$. Thus, for kinematic systems there is a first-order relationship between input and velocity such that $f : \mathcal{Q} \times \mathcal{U} \to \mathcal{T}\mathcal{Q}$, where $\mathcal{T}\mathcal{Q}$ represents the *tangent bundle* of the configuration space. The state space of the system is simply $\mathcal{X} = \mathcal{Q}$.

In contrast, the inputs for *second-order dynamical* systems specify the system accelerations, which effect velocities via integration; that is $\ddot{q} = f(q, \dot{q}, u)$. The mapping, $f : \mathcal{T}\mathcal{Q} \times \mathcal{U} \to \mathcal{T}\mathcal{T}\mathcal{Q}$, between input and velocity/acceleration is a generally non-linear function of state. To fully specify the motion of the system, both the configuration and its associated velocities must be specified; thus, the *state* of the second-order system is $\mathcal{X} = \{q, \dot{q}\} \in \mathcal{T}\mathcal{Q}$.

For both kinematic and second-order systems, the equations of motion given by the nonlinear function f is derived from the system constraints. Given f, 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 while respecting any configuration space bounds. To be a valid control, the inputs must be chosen from the bounded input space $\mathcal{U}$. Thus, solving the navigation problem requires solving a constrained non-linear control problem. The nature of the constraints is the next topic.

A.2 System Constraints

This thesis considers several classes of constraints, including input bounds, velocity bounds, configuration bounds, and nonholonomic constraints.

The most basic constraint is an equality constraint on some configuration variable, $h(q) = \text{constant}$. For these constraints, the evolution of the system evolves on a sub-manifold of the configuration space defined by the constraint. In this case, the dimension of the configuration space is reduced, and one only needs to consider the remaining configuration parameters. In this thesis, all of the systems are reduced to the minimum number of configuration variables.

The second type of constraint is an inequality constraint of the form $h(q) \leq 0$ or $h(q) \geq 0$. Obstacles are in this class of constraint, as are steering limits. These constraints are typically hard mechanical limits; therefore, the control policy should avoid the constraint surfaces.

Inequality constraints on velocities are generally bounds imposed by actuator limitations or safety considerations. A common limitation is the maximum speed output by motors. It is permissible to approach this type of limiting surface. Other constraints may be imposed by safety considerations, whether due to externally imposed speed limits or internal limitations due to vehicle dynamics. As a latter example, the turning rates may be bounded at higher speeds to prevent rollover. Whether velocities or torques, the actuator input space is bounded by a collection of inequality constraints.

The final constraint we consider is an equality constraint on velocities; this thesis is only concerned with so called *Pfaffian* constraints, which are linear in the velocities [1, 87, 94]. Pfaffian constraints, which have the form $c\,(q) \cdot \dot{q} = 0$, are able to express the velocity constraints inherent in wheeled vehicles. The constraints dictate that any valid velocity must lie in the null space of the constraints; the constraint null space, which represents the set of all possible velocities at a given configuration, is called the *constraint distribution*, denoted D_q.

Pfaffian constraints are classed as either *holonomic* or *nonholonomic*. Holonomic constraints have an equivalent configuration constraint of the form $h(q) = 0$; holonomic Pfaffian constraints are said to be "integrable." Nonholonomic Pfaffian constraints are said to be non-integrable because they do not have an equivalent configuration constraint, and therefore, do not reduce the dimension of the configuration space[1]. The *Lie Algebra Rank Condition (LARC)* test provides a convenient test over the constraint distribution to see if a particular set of Pfaffian constraints is non-integrable, and hence, nonholonomic [1, 87, 94].

For a brief example, consider the kinematic unicycle model commonly used in robotics. The system configuration is given by the pose, hence $q = \{x, y, \theta\}$; there are no shape variables in this model. The system is constrained such that its sideways velocity is zero; in Pfaffian form, this constraint is

$$\begin{bmatrix}\sin\theta & -\cos\theta & 0\end{bmatrix} \cdot \begin{bmatrix}\dot{x}\\ \dot{y}\\ \dot{\theta}\end{bmatrix} = 0\,.$$

The commonly used basis for the null space, represented as matrix columns, is

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

Treating our inputs as $u = \{v, \omega\} \in \mathcal{U}$, the forward velocity and turning rate respectively, we have $\dot{q} = A\,(q) \cdot u$. Thus, the nonholonomic constraints are used to derive the equations of motion. Stated differently, the columns of $A\,(q)$ span the constraint distribution D_q. Given bounds on the input space $\mathcal{U}$, the set of reachable velocities in D_q is likewise bounded. If rate of turning ω is bounded, the unicycle model is often referred to as "car-like" in the literature [120].

Notice, that the kinematic unicycle is under actuated; only two inputs are used to control three configuration variables. The LARC is used to verify that these two inputs are sufficient to guaranteed

[1]Technically inequality constraints are also "nonholonomic" constraints, but we reserve the term for non-integrable velocity constraints [87].

controllability with respect to the full configuration space [1, 87, 94]. One of the fundamental difficulties in controlling nonholonomic systems [2] is dealing with the effects of this under-actuation.

To assist in the analysis and control design, we now take recourse to differential geometry and geometric mechanics, and use the language of *fiber bundles* and *connections*. This allows us do define specific relationships between the body pose and the shape variables. While the remainder of this appendix focuses on the nonholonomic constraints that determine the equations of motion, the reader should keep in mind that any real system is subject to other bounds, which act to limit the set of achievable velocities/accelerations.

A.3 Fiber Bundles and Connections

This section presents a brief overview of the fiber bundle concept as it relates to the nonholonomic systems encountered in this thesis. For a full treatment, the reader is referred to [1]. This subsection provides an abstract overview of the terms, the ideas are made concrete in Appendix A.4.

Consider a decomposition of the configuration space into two subspaces, as depicted in Figure A.2. These spaces are referred to as the base (B) and fiber spaces (F). Given a projection $\pi : \mathcal{Q} \to B$, the fiber at $b = \pi(q) \in B$ is defined as $F = \pi^{-1}(b)$. Locally, $\mathcal{Q} \cong F \times B$; if this is true everywhere, then $\mathcal{Q}(F, B, \pi)$ is a *trivial fiber bundle*. If the fiber is homeomorphic to a group G, then $\mathcal{Q}(G, B, \pi)$ is called a (trivial) *principal fiber bundle*. The configuration spaces of mechanical systems are trivial principal fiber bundles [112].

For systems considered in the proposed thesis, the configuration variables that are directly controlled define the base space and the group G is a Lie group corresponding to rigid body motion. For most of the systems in this thesis, $G = SE(2)$, where $SE(2)$ is the Lie group manifold corresponding to the body pose. This thesis abuses notation slightly by using g to represent either the local coordinates $(x, y, \theta) \in \mathbb{R}^3$ or the group element in $SE(2)$, and $\mathcal{G}$ to represent either the local $\mathbb{R}^3$ chart or $SE(2)$. The form being used will be clear from context. The base variables correspond to the *shape* variables described in Chapter 5 [96].

The real power of this geometric analysis comes with second-order systems. For certain second-order systems, the equations of motion may be reduced to second-order equations defined only on the base variables [1, 90]. Not only does this reduce the dimension of the second order equations, the resulting control on the base space is free of nonholonomic differential constraints. The fiber velocities are reconstructed from base velocities using the connection, which naturally satisfies the nonholonomic constraints. While these second-order effects are not explored in this thesis, the formalism provides useful insight into the connection between the inputs and the body pose velocities. The analysis will also form the basis of future extensions of this thesis's work to second-order systems.

The relationship between motions on the base space and motions along the fiber is governed by a *connection* [1, 90]. The concept of a connection is quite general, and can be used for systems with non-trivial momentum effects between the base and fiber spaces; a common example in the literature is the snake-board [96]. This thesis is restricted to simpler *purely kinematic* systems, so named because the connection encodes a linear first-order (kinematic) map that only depends on the configuration of the system, and not the velocity. We defer the formal definition until later in this section, after the necessary preliminaries are defined.

[2] It is the constraints that are "nonholonomic", but we will follow the literature and refer to systems subject to nonholonomic constraints as "nonholonomic systems."

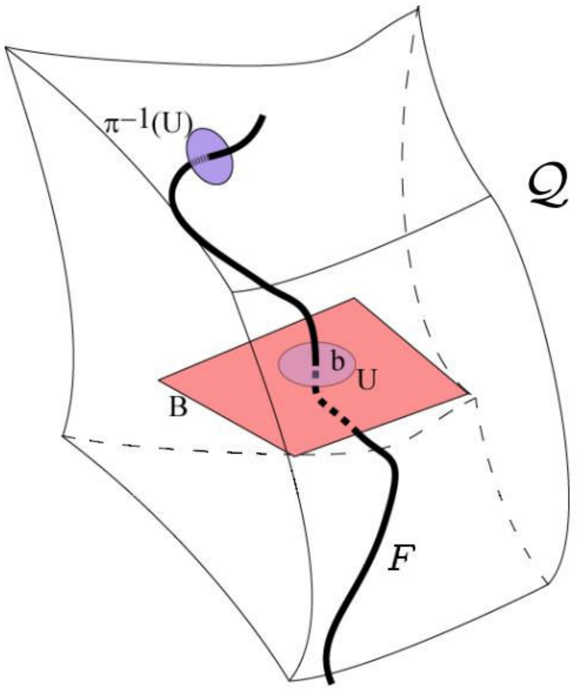


Figure A.2: Base-fiber decomposition of configuration space. (Courtesy Elie Shammas)

A connection, $A : \mathcal{T}_q\mathcal{Q} \to \mathcal{T}_qF$, projects arbitrary velocities onto the fiber tangent space[3]. Define the *horizontal space* $\text{hor}_q = \text{kernel}\,(A_q)$. For an arbitrary vector X_q in $\mathcal{T}_q\mathcal{Q}$, the horizontal part is $X_q^h = X_q - A\,(q) \cdot X_q$. The *vertical* component lies in the fiber tangent space[4] [1]. Given a local trivialization of the fiber bundle $\mathcal{Q} = G \times \mathcal{Q}/G$, let the coordinates (g, r) represent the fiber (group) and base components [1]. For an arbitrary tangent vector, let the components in the local trivialization be denoted by $(\dot{g}, \dot{r})$. The connection is represented in local coordinates as $A = \omega^a \frac{\partial}{\partial g^a}$ (using summation notation), where

$$\omega^a\,(q) = dg^a + A_\alpha^a\,(r, g)\,dr^\alpha\,. \tag{A.2}$$

The vertical projection is locally given by $(\dot{g}^a, \dot{r}^\alpha) \mapsto (\dot{g}^a + A_\alpha^a\,(r, g)\,\dot{r}^\alpha, 0)$. Locally, the horizontal projection is given by $(\dot{g}^a, \dot{r}^\alpha) \mapsto (-A_\alpha^a\,(r, g)\,\dot{r}^\alpha, \dot{r}^\alpha)$. The term $-A_\alpha^a\,(r, g)\,\dot{r}^\alpha$ represents the components of motion along the fiber group that are induced by motion in the base space.

For purely kinematic systems, the number of independent nonholonomic constraints is equal to the dimension of the fiber space. If the horizontal space is defined to be the constraint distribution D_q that is the set of admissible velocities $D_q = \text{hor}_q$, then the connection one-form $A(q)$ is uniquely determined by the nonholonomic constraints [1, 112]. The base variables are directly controlled such that $\dot{r} = f\,(u)$ or $\ddot{r} = f\,(u)$. Then, given the base velocities $\dot{r}$, the fiber velocities are uniquely determined by $\dot{g}^a = -A_\alpha^a\,(r, g)\,\dot{r}^\alpha$; in this thesis, we will use a more compact notation $\dot{g} = A\,(q)\,\dot{r}$. Note that even though the systems are called purely *kinematic*, the system can have second order dynamics on the base space.

[3]Here we follow the literature and abuse notation to use A as both the Pfaffian constraint distribution and the derived connection [1]. The reason for this abuse will become apparent.

[4]Note, horizontal and vertical are somewhat misleading terms, the vectors are not orthogonal for constrained systems.

The connection provides a convenient form for deriving the equations of motion; unfortunately, the constraints of some systems (e.g. the Ackermann steered car) do not have the simple form given in (A.2). This makes determining the connection more difficult. To simplify, the concept of a *principal connection* is introduced. The principal connection requires that the system have certain invariance properties, or *symmetries* [1, 90]. The formal definition requires some additional terminology related to the fiber Lie group.

Recall, that a *principal fiber bundle* is a fiber bundle such that the fibers $F = \pi^{-1}(b)$ are everywhere homeomorphic to a structure group G [1]. Given a projection $\pi : \mathcal{Q} \to \mathcal{Q}/G$, where $\mathcal{Q}/G$ corresponds to the base space B, and $\mathcal{Q} = \mathcal{Q}/G \times G$ everywhere, $\mathcal{Q}(\mathcal{Q}/G, G, \pi)$ is a trivial principal fiber bundle. For a point in the configuration space $q = (h, r)$, with $h \in G$ and $r \in B$, the left action of a group element $g \in G$ corresponds to motion along the fiber given as $L_g q = (gh, r)$. The right action is given by $R_g q = (hg, r)$. The Lie algebra, $\mathfrak{g}$, of the Lie group is the tangent space at the identity element, that is $\mathfrak{g} = T_e G$. The lifted action $T_q \Phi_g(v_q)$ on $v_q \in \mathcal{T}_q\mathcal{Q}$ gives the vector at $\Phi_g(q)$ obtained by parallel transport of v_q by the action Φ_g. The short-hand notation, $g^{-1}\dot{g}$, is used to represent the action $T_g L_{g^{-1}}(\dot{g})$, which returns the velocity in body coordinates, $\xi^b = g^{-1}\dot{g} \in \mathfrak{g}$. The infinitesimal generator of the action corresponding to $\xi \in \mathfrak{g}$, denoted ξ_G, generates a vector field over G according to $\xi_G(h) = \frac{d}{dt}(\exp(\xi t) \cdot h)\,|_{t=0}$ for all $h \in G$. For trivial principal bundles, $\xi_G(h) = T_e R_h \xi$ [90]. At a given point $q = (g, r) \in \mathcal{Q}$, $\xi_{\mathcal{Q}(q)} = (T_e R_g(\xi), 0)$. The adjoint operator, $\mathrm{Ad}_g : \mathfrak{g} \to \mathfrak{g}$, is given by $\mathrm{Ad}_g = T_{g^{-1}} L_g \circ T_e R_{g^{-1}}$. The group orbit is $\mathrm{Orb}(q) = \{gq \mid g \in \mathcal{G}\}$; the orbit is an immersed sub-manifold. The orbit tangent space is given by the generators at the point; that is $T_q \mathrm{Orb}(q) = \{\xi_G(q) \mid \xi \in \mathfrak{g}\}$.

Each of these abstract operators has a concrete representation for systems whose structure group G is $SE(2)$. An element of the group, $g \in SE(2)$, is represented as a 3×3 matrix

$$g = \begin{bmatrix} \cos\theta & -\sin\theta & x \\ \sin\theta & \cos\theta & y \\ 0 & 0 & 1 \end{bmatrix} \in SE(2) \ .$$

Elements of the group represent both body configurations and rigid body motions on another group element. The composition of two elements is simple matrix multiplication. Thus, for $g, h \in SE(2)$, $L_g h = gh$ and $R_g h = hg$. The body velocity $\xi^b = [\xi_x \ \xi_y \ \xi_\omega]^T$ is represented as a Lie algebra element by the matrix

$$\xi = \begin{bmatrix} 0 & -\xi_\omega & \xi_x \\ \xi_\omega & 0 & \xi_y \\ 0 & 0 & 0 \end{bmatrix} \in \mathfrak{se}(2) \ .$$

The velocity in world coordinates is given by $\dot{g} = g\xi$. As in the literature, we will intermingle the use of 3-tuples to represent elements of the matrix Lie group and algebra. Given $h = [h_x, h_y, h_\theta]^T \in$

$SE(2)$, the group action L_g , where $g = [x, y, \theta]$, yields

$$\begin{aligned} L_g h &= \begin{bmatrix} \cos\theta & -\sin\theta & x \\ \sin\theta & \cos\theta & y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos h_\theta & -\sin h_\theta & h_x \\ \sin h_\theta & \cos h_\theta & h_y \\ 0 & 0 & 1 \end{bmatrix} \\ &= \begin{bmatrix} \cos(\theta + h_\theta) & -\sin(\theta + h_\theta) & x + h_x\cos\theta - h_y\sin\theta \\ \sin(\theta + h_\theta) & \cos(\theta + h_\theta) & y + h_x\sin\theta + h_y\cos\theta \\ 0 & 0 & 1 \end{bmatrix} \\ &\cong \begin{bmatrix} x + h_x\cos\theta - h_y\sin\theta \\ y + h_x\sin\theta + h_y\cos\theta \\ \theta + h_\theta \end{bmatrix} \end{aligned} \tag{A.3}$$

To obtain the mapping for the lifted action $T_h L_g : T_h SE(2) \to T_{gh} SE(2)$, we differentiate the vector form of $L_g h$ with respect to the group coordinates to yield

$$T_h L_g = \left\{ \frac{\partial L_g}{\partial h_i} \right\} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} . \tag{A.4}$$

Similarly, the lifted right action $T_h R_g : T_h SE(2) \to T_{hg} SE(2)$ is given by

$$T_h R_g = \begin{bmatrix} 1 & 0 & -y\cos h_\theta - x\sin h_\theta \\ 0 & 1 & x\cos h_\theta - y\sin h_\theta \\ 0 & 0 & 1 \end{bmatrix} . \tag{A.5}$$

For matrix groups, $\mathrm{Ad}_g(\xi) = g\xi g^{-1}$. For $\mathfrak{se}(2)$, the matrix that encodes the adjoint operator is

$$\mathrm{Ad}_g = \begin{bmatrix} \cos\theta & -\sin\theta & y \\ \sin\theta & \cos\theta & -x \\ 0 & 0 & 1 \end{bmatrix} . \tag{A.6}$$

This matrix operates on the vector form of ξ.

Given these Lie group and Lie algebra actions, we now return to defining the principal connection for purely kinematic systems.

A *principal connection*, $\mathcal{A}$, is a map $\mathcal{A} : \mathcal{T}_q\mathcal{Q} \to \mathfrak{se}(2)$ such that

$$\begin{aligned} \mathcal{A}(\xi_Q(q)) &= \xi, \;\; \forall \xi \in \mathfrak{g}, q \in Q \\ \mathcal{A}(T_q\Phi_g(v_q)) &= \mathrm{Ad}_g \mathcal{A}(v_q) \, . \end{aligned}$$

[1]. This last property requires that the principal connection is invariant under the group action g.

The *horizontal space* of the connection $\mathcal{A}$ at $q \in \mathcal{Q}$ is $\mathrm{hor}_q = \{v_q \in \mathcal{T}_q\mathcal{Q} \mid \mathcal{A}(v_q) = 0\}$, and $\mathcal{T}_q\mathcal{Q} = \mathrm{hor}_q \oplus \mathrm{ver}_q$. Thus, for $v_q \in \mathcal{T}_q\mathcal{Q}$ we have the decomposition $v_q = \mathrm{hor}_q v_q + \mathrm{ver}_q v_q$, where

$$\mathrm{ver}_q v_q = (\mathcal{A}(v_q))_{\mathcal{Q}(q)} \, . \tag{A.7}$$

From the definition of an infinitesimal generator, $(\mathcal{A}(v_q))_{\mathcal{Q}(q)}$ is the vector at q generated by the Lie algebra element $\mathcal{A}(v_q)$. The horizontal part of v_q is given as $\mathrm{hor}_q v_q = v_q - (\mathcal{A}(v_q))_{\mathcal{Q}(q)}$. The connection A is related to the principal connection as $A(v_q) = (\mathcal{A}(v_q))_{\mathcal{Q}(q)}$.

The principal connection provides a simple method of determining the connection [1]. The connection in the local trivialization is

$$\mathcal{A}(\dot{g}, \dot{r}) = \mathrm{Ad}_g \left(T_g L_{g^{-1}} \dot{g} + \mathbb{A}(r) \dot{r} \right) , \tag{A.8}$$

where $\mathbb{A}$ represents the local connection determined by the constraints at the identity of the group. Appendix subsections A.4.2 and A.4.3 provide specific examples of these ideas.

Given these preliminaries, we may now define the class of systems used in this thesis. A *principally kinematic system* is a system where

- the tangent space $\mathcal{T}_q\mathcal{Q}$ is the direct sum of the constraint distribution D_q and the tangent to the group orbits; that is $T_qQ = D_q + T_q\mathrm{Orb}(q)$.
- $D_q \bigcap T_q\mathrm{Orb}(q) = \{0\}$.
- the Lagrangian L, the difference between kinetic energy and potential energy of the system, is invariant under the group action of G on $\mathcal{T}\mathcal{Q}$.
- the constraint distribution D is invariant, that is given $D_q \subset \mathcal{T}\mathcal{Q}$ then $T_gD_q = D_{gq} \subset T_{gq}\mathcal{Q}$.

The first two conditions apply to purely kinematic systems. The constraint distribution, D_q, defines the horizontal space for the connection; the group orbits define the vertical space. Together they span the configuration tangent space. As stated earlier, this choice uniquely specifies the connection.

The last two conditions specify the invariance properties that differentiate between *principally kinematic* and *purely kinematic* systems. In the literature, the terms are often used interchangeably [1]. In this thesis, we will make a distinction between the two terms, as in [90]. Any principally kinematic system is purely kinematic; but the reverse is not true. In Appendix A.4.4 we provide an example of a system that is purely kinematic; that is, it has a kinematic mapping between base and fiber velocities, but lacks the invariance properties and is therefore NOT principally kinematic.

The remainder of this appendix presents detailed derivations of the equations of motion for the systems used in this thesis.

A.4 Examples

A.4.1 Vertical Rolling Disk (Unicycle)

The kinematic unicycle model presented in Appendix A.2 did not have a base variable, and hence, does not fit into the fiber bundle framework. However, by considering the rotation of a drive wheel, the model does fit. This more complex model, called the *vertical rolling disk*, is suitable for modeling second order dynamics where the accelerations are the control inputs, and not $\{v, \omega\}$ as in Appendix A.2.

Consider the example vertical rolling disk shown in Figure A.3 [1, 90]. The disk moves over the plane, with a configuration space of $SE(2)$. We represent $SE(2)$ using a local chart $(x, y, \theta) \subset \mathbb{R}^3$, where (x, y) are the coordinates of the point of contact on the plane and θ is the orientation of the disk with respect to the x-axis of the plane. The angle of rotation about a horizontal axis through the disk center, with respect to the vertical, is ψ. Let the local configuration be given as the vector $q = \begin{bmatrix} x & y & \theta & \psi \end{bmatrix}^T$. The system is controlled by (imaginary) motors that provide torque about both the vertical and horizontal axes of rotation, which provides control of θ and ψ directly. Therefore,

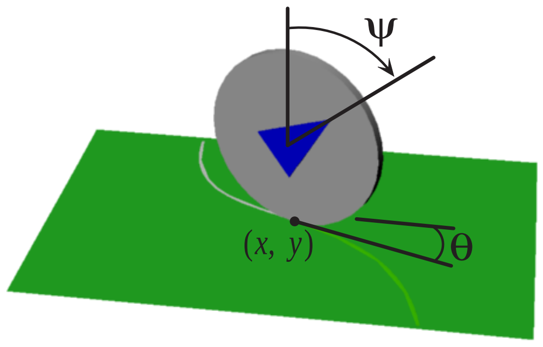


Figure A.3: Vertical rolling disk with (θ, ψ) as base variables.

the base variables are designated as (θ, ψ), with projection

$$\pi(q) = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot q \,.$$

The fiber variables, (x, y), are a group under the action of translation.

The disk rolls without slipping, which constrains the instantaneous velocity of the point of contact to be zero, and yields the following constraints

$$\begin{aligned} \dot{x} - R\dot{\psi}\cos\theta &= 0\,, && \text{(A.9)} \\ \dot{y} - R\dot{\psi}\sin\theta &= 0\,. && \text{(A.10)} \end{aligned}$$

In Pfaffian form, the constraints are

$$\begin{bmatrix} 1 & 0 & 0 & -R\cos\theta \\ 0 & 1 & 0 & -R\sin\theta \end{bmatrix} \cdot \dot{q} = 0\,. \tag{A.11}$$

For this simple system, we can rewrite (A.9) and (A.10) to obtain equations that specify the relationship between the base variable velocities and the fiber velocities, $\dot{x} = R\cos(\theta)\,\dot{\psi}$ and $\dot{y} = R\sin(\theta)\,\dot{\psi}$. Thus, given a specification of the base velocity, $\dot{\psi}$, and base variable, θ, the evolution of the fiber is strictly first order; that is the fiber variables are kinematic with respect to the base variables. This is the fundamental nature of purely kinematic systems.

While the equations of motion are easily derived for this simple case, we will derive the connection from first principles to illustrate the base/fiber paradigm. Let $\dot{q} = \left\{\dot{x}, \dot{y}, \dot{\theta}, \dot{\psi}\right\} \in \mathcal{T}_q\mathcal{Q}$ be an arbitrary velocity vector. The Pfaffian constraints for the vertical rolling disk, given in (A.11), have the simple form given in (A.2). The corresponding connection one-forms in coordinates are

$$\begin{aligned} \omega_1 &= dx - R\cos\theta\, d\psi \\ \omega_2 &= dy - R\sin\theta\, d\psi \end{aligned}$$

The components of the local connection are extracted as $A_2^1 = -R\cos\theta$ and $A_2^2 = -R\sin\theta$; the remaining terms are zero. The horizontal part of the arbitrary velocity vector is

$$\text{hor}_q\dot{q} = \begin{bmatrix} R\dot{\psi}\cos\theta & R\dot{\psi}\sin\theta & \dot{\theta} & \dot{\psi} \end{bmatrix}^T$$

, and the vertical part is

$$\mathrm{ver}_q\dot{q} = \dot{q} - \mathrm{hor}_q\dot{q} = \begin{bmatrix}\dot{x} - R\,\dot{\psi}\cos\theta & \dot{y} - R\,\dot{\psi}\sin\theta & 0 & 0\end{bmatrix}^T .$$

Note that the horizontal vector, as given, obeys the Pfaffian constraints; therefore, $\mathrm{hor}_q\dot{q} \in \mathcal{D}_q$, the constraint distribution. Thus, the horizontal part, which encodes the motion induced by the base motion, is inherently consistent with the constraints. The vertical part encodes any remaining portion of the arbitrary velocity that is inconsistent with the constraints. This remaining portion represents the group action independent of base motion that occurs along the fiber tangent vector; such independent motion may be due to a disturbance beyond the system control.

Formally, the connection $A : \mathcal{T}_q\mathcal{Q} \to \mathcal{T}_qF$ is used to define the horizontal and vertical spaces. In this thesis, we are mainly concerned with the mapping between base velocities and fiber velocities, where the base velocities are driven by the system inputs. Thus, we abuse notation an use $A\left(q\right) : \mathcal{T}_qB \to \mathcal{T}_qG$ to denote the more compact mapping where the terms are taken directly from the connection. For the vertical rolling disk, $\dot{g} = A\left(q\right)\dot{r}$ with

$$A\left(q\right) = \begin{bmatrix}0 & R\cos\theta\\ 0 & R\sin\theta\\ 1 & 0\end{bmatrix} . \tag{A.12}$$

Note the similarity to (A.1) if we let $v = R\dot{\psi}$ and $\omega = \dot{\theta}$.

A.4.2 Differential-drive System

A common mobile robot platform is a differential-drive system. The body of the robot is driven by two independently controlled wheels. By controlling the relative speeds between the two wheels, the system can control both its forward velocity and the rate of body rotation.

The body of the robot moves across the plane, with a configuration space of $SE(2)$. Locally, we represent $SE(2)$ as $(x, y, \theta) \subset \mathbb{R}^3$, where (x, y) are the coordinates of midpoint of the line connecting the drive wheel centers, and θ is the orientation of the body with respect to the x-axis of the plane. This arrangement is shown in Figure A.4. The angles of rotation of the drive wheels about a horizontal axis through the wheel centers is denoted ψ_L and ψ_R, where L and R denote the left and right wheels relative to the body heading. Positive rotation moves the vehicle forward. The local configuration is $q = \begin{bmatrix}x & y & \theta & \psi_L & \psi_R\end{bmatrix}^T \in SE(2) \times S^1 \times S^1$.

The vehicle is subject to four nonholonomic constraints, three of which are independent. Each drive wheel is assumed to roll without slipping and is prevented from sliding sideways relative to its instantaneous heading. The sliding sideways constraints are redundant. In Pfaffian form, these independent constraints are

$$\begin{bmatrix}\sin\theta & -\cos\theta & 0 & 0 & 0\\ \cos\theta & \sin\theta & -c & -R & 0\\ \cos\theta & \sin\theta & c & 0 & -R\end{bmatrix} \cdot \dot{q} = \mathbf{0}\,.$$

These constraints are not in the simple form given in (A.2). While it is possible to derive these equations of motion using algebra, we will take recourse to the principal connection. We first

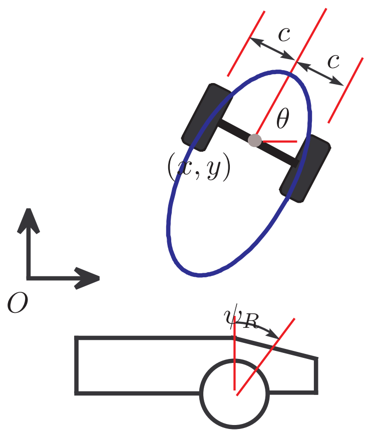


Figure A.4: Differential drive robot with two drive wheels $\{\psi_L, \psi_R\}$ as base variables.

rewrite the constraints at the fiber identity element as

$$\begin{bmatrix} 0 & -1 & 0 & 0 & 0 \\ 1 & 0 & -c & -R & 0 \\ 1 & 0 & c & 0 & -R \end{bmatrix} \cdot \begin{bmatrix} \xi \\ \dot{r} \end{bmatrix} = \mathbf{0}\,.$$

Using simple algebra, these constraints are equivalent to

$$\begin{bmatrix} 0 & -1 & 0 & 0 & 0 \\ 1 & 0 & 0 & -\frac{R}{2} & -\frac{R}{2} \\ 0 & 0 & 1 & \frac{R}{2c} & -\frac{R}{2c} \end{bmatrix} \cdot \begin{bmatrix} \xi \\ \dot{r} \end{bmatrix} = \mathbf{0}\,. \tag{A.13}$$

The constraints in (A.13) are in the simple form given in (A.2), which allows us to directly write the connection terms $\mathbb{A}_1^1 = -\frac{R}{2}$, $\mathbb{A}_2^1 = -\frac{R}{2}$, $\mathbb{A}_1^2 = 0$,$\mathbb{A}_2^2 = 0$, $\mathbb{A}_1^3 = \frac{R}{2c}$, and $\mathbb{A}_2^3 = \frac{-R}{2c}$, or more compactly as

$$\mathbb{A}\left(r\right) = \begin{bmatrix} -\frac{R}{2} & -\frac{R}{2} \\ 0 & 0 \\ \frac{R}{2c} & -\frac{R}{2c} \end{bmatrix} \tag{A.14}$$

Given this unique specification, (A.7) and (A.8) determine the projection in local coordinates onto the vertical space ver_q; this yields

$$\text{ver}_q v_q = \text{ver}_q\left(\dot{g}, \dot{r}\right) = \left(T_e R_g\left(\text{Ad}_g\left(T_g L_{g^{-1}}\dot{g} + \mathbb{A}\left(r\right)\dot{r}\right)\right), 0\right)\,.$$

The principal connection for the diff-drive system is

$$\begin{aligned} \mathcal{A}\left(\dot{g}, \dot{r}\right) &= \begin{bmatrix} \cos\theta & -\sin\theta & y \\ \sin\theta & \cos\theta & -x \\ 0 & 0 & 1 \end{bmatrix} \left(\begin{bmatrix} \dot{x}\cos\theta + \dot{y}\sin\theta \\ \dot{y}\cos\theta - \dot{x}\sin\theta \\ \dot{\theta} \end{bmatrix} + \begin{bmatrix} -\frac{R}{2} & -\frac{R}{2} \\ 0 & 0 \\ \frac{R}{2c} & -\frac{R}{2c} \end{bmatrix} \begin{bmatrix} \dot{\psi}_L \\ \dot{\psi}_R \end{bmatrix} \right) \\ &= \begin{bmatrix} \dot{x} + y\,\dot{\theta} \\ \dot{y} - x\,\dot{\theta} \\ \dot{\theta} \end{bmatrix} + \begin{bmatrix} \cos\theta & -\sin\theta & y \\ \sin\theta & \cos\theta & -x \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} -\frac{R}{2} & -\frac{R}{2} \\ 0 & 0 \\ \frac{R}{2c} & -\frac{R}{2c} \end{bmatrix} \begin{bmatrix} \dot{\psi}_L \\ \dot{\psi}_R \end{bmatrix}\,. \end{aligned} \tag{A.15}$$

In $\mathfrak{se}(2)$, the generator operator is

$$T_e R_g = \begin{bmatrix} 1 & 0 & -y \\ 0 & 1 & x \\ 0 & 0 & 1 \end{bmatrix} ;$$

therefore, the vertical component is

$$\begin{aligned} \mathrm{ver}_q v_q &= \left\{ T_e R_g \mathcal{A}\left(\dot{r}, \dot{g}\right), 0 \right\} \\ &= \begin{bmatrix} \begin{bmatrix} 1 & 0 & -y \\ 0 & 1 & x \\ 0 & 0 & 1 \end{bmatrix} \left(\begin{bmatrix} \dot{x} + y\,\dot{\theta} \\ \dot{y} - x\,\dot{\theta} \\ \dot{\theta} \end{bmatrix} + \begin{bmatrix} \cos\theta & -\sin\theta & y \\ \sin\theta & \cos\theta & -x \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} -\frac{R}{2} & -\frac{R}{2} \\ 0 & 0 \\ \frac{R}{2c} & -\frac{R}{2c} \end{bmatrix} \begin{bmatrix} \dot{\psi}_L \\ \dot{\psi}_R \end{bmatrix} \right) \\ 0 \\ 0 \end{bmatrix} \\ &= \begin{bmatrix} \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} + \begin{bmatrix} -\frac{R}{2}\cos\theta & -\frac{R}{2}\cos\theta \\ -\frac{R}{2}\sin\theta & -\frac{R}{2}\sin\theta \\ \frac{R}{2c} & -\frac{R}{2c} \end{bmatrix} \begin{bmatrix} \dot{\psi}_L \\ \dot{\psi}_R \end{bmatrix} \\ 0 \\ 0 \end{bmatrix} \end{aligned} \tag{A.16}$$

and the horizontal part is

$$\begin{aligned} \mathrm{hor}_q v_q &= v_q - \mathcal{A}\left(\dot{r}, \dot{g}\right)_{\mathcal{Q}}(q) \\ &= \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \\ \dot{\psi}_L \\ \dot{\psi}_R \end{bmatrix} - \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \\ 0 \\ 0 \end{bmatrix} - \begin{bmatrix} \begin{bmatrix} -\frac{R}{2}\cos\theta & -\frac{R}{2}\cos\theta \\ -\frac{R}{2}\sin\theta & -\frac{R}{2}\sin\theta \\ \frac{R}{2c} & -\frac{R}{2c} \end{bmatrix} \begin{bmatrix} \dot{\psi}_L \\ \dot{\psi}_R \end{bmatrix} \\ 0 \\ 0 \end{bmatrix} \\ &= \begin{bmatrix} \begin{bmatrix} \frac{R}{2}\cos\theta & \frac{R}{2}\cos\theta \\ \frac{R}{2}\sin\theta & \frac{R}{2}\sin\theta \\ -\frac{R}{2c} & \frac{R}{2c} \end{bmatrix} \begin{bmatrix} \dot{\psi}_L \\ \dot{\psi}_R \end{bmatrix} \\ \dot{\psi}_L \\ \dot{\psi}_R \end{bmatrix} \\ &= \begin{bmatrix} \frac{R}{2}\cos\theta & \frac{R}{2}\cos\theta \\ \frac{R}{2}\sin\theta & \frac{R}{2}\sin\theta \\ -\frac{R}{2c} & \frac{R}{2c} \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \dot{\psi}_L \\ \dot{\psi}_R \end{bmatrix} . \end{aligned} \tag{A.17}$$

As this horizontal vector encodes the admissible velocity induced by the base velocities, we will abuse notation and let $A\left(q\right)$ represent the unique *horizontal lift* from the base velocities to the configuration velocities derived from the Pfaffian constraints.

For diff-drive this thesis uses $\dot{g} = A\left(q\right)\dot{r}$, with

$$A\left(q\right) = \begin{bmatrix} \frac{R}{2}\cos\theta & \frac{R}{2}\cos\theta \\ \frac{R}{2}\sin\theta & \frac{R}{2}\sin\theta \\ -\frac{R}{2c} & \frac{R}{2c} \end{bmatrix} . \tag{A.18}$$

A.4.3 Ackermann Steered Car-like System

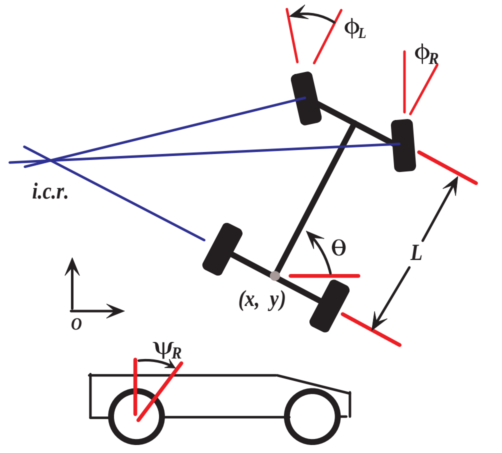
Consider the rear-wheel drive system shown schematically in Figure A.5-a. This car-like system has four wheels in contact with the ground. The model used in this thesis ignores body dynamics and the effects of tire/ground interaction, and assumes that each wheels rolls without slipping or sliding sideways. The two rear wheels provide the motive force via traction with the ground, while the two front wheels provide steering. The vehicle body moves in the plane as it rotates about the *instantaneous center of rotation* (i.c.r.), which lies at the intersection of the perpendicular bisector to each wheel. The rolling without slipping constraints force each wheel to rotate at slightly different speeds according to the distance from the i.c.r. The angle of each front wheel is slightly different so that the bisectors intersect the rear axle line at the i.c.r. This type of steering, called Ackermann steering, prevents slipping of the wheels, which increases drag and wearing of the tires.

Typically a single motor provides torque to both drive wheels, and the angular velocity of each drive wheel is related by a set of gears called the *differential*. To simplify the model, this motor/differential pair is modeled as a single drive wheel located along the medial axis of the vehicle, as shown in Figure A.5-b. This imaginary wheel rotates about its axle by an angle ψ. The steering angle of each wheel is controlled by a single steering wheel, with a mechanical linkage coupling each front wheel. In the simple model, a single front wheel, with a steering angle ϕ, is used to represent the steering input. Because the distance, L, between contact points has not changed and the i.c.r has not changed, the motion of the two-wheeled vehicle is kinematically equivalent to the motion of the four-wheeled car-like vehicle.

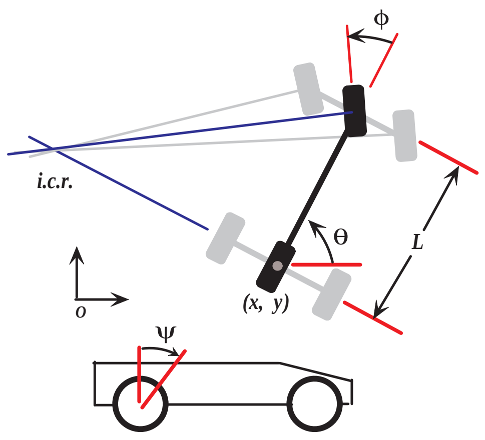
The configuration space has two parts. Locally the body pose is given by $g = \{x, y, \theta\}$, where (x, y) is the location of the midpoint of the rear axle with respect to a global coordinate, and θ is the orientation of the body with respect to the x-axis. The body pose evolves on the $SE(2)$ manifold. The configuration of the drive wheel is $\psi \in \mathbf{S}^1$; and that of the steering wheel is $\phi \in \mathbf{I} = (-\phi_{\max}, \phi_{\max})$, a bounded interval. Thus globally, the configuration space is $\mathcal{Q} = SE(2) \times \mathbf{S}^1 \times \mathbf{I}$.

The rolling without slipping or sliding constraints give three independent constraints. Both the front and rear wheels are prevented from sliding transverse to the rolling direction, which gives

$$\dot{x} \sin\theta - \dot{y} \cos\theta = 0 \tag{A.19}$$



a) Car-like system with Ackermann steering



b) Simplified model

Figure A.5: Car-like system with Ackermann Steering. The figure on the right represents an equivalent simplified model.

and

$$\dot{x}\sin(\theta+\phi) - \dot{y}\cos(\theta+\phi) - L\dot{\theta}\cos\phi = 0. \tag{A.20}$$

The rear drive wheel is assumed to not slip along its rolling direction, that is it does not spin freely. This gives a constraint of

$$\dot{x}\cos\theta + \dot{y}\sin\theta - R\,\dot{\psi} = 0, \tag{A.21}$$

where R is the drive wheel radius. These are Pfaffian constraints, and may be represented as

$$\begin{bmatrix} \sin\theta & -\cos\theta & 0 & 0 & 0 \\ \sin(\theta+\phi) & -\cos(\theta+\phi) & -L\cos\phi & 0 & 0 \\ \cos\theta & \sin\theta & 0 & -R & 0 \end{bmatrix} \cdot \dot{q} = 0\,,$$

where $\dot{q} = \begin{bmatrix}\dot{x} & \dot{y} & \dot{\theta} & \dot{\psi} & \dot{\phi}\end{bmatrix}^T$.

Car-like systems are often modeled using a drift-less kinematic model, where the drive wheel configuration is dropped. The rear forward velocity, v, is specified as one control input, and the steering rate, ω, is the second input. For $q = \begin{bmatrix}x & y & \theta & \phi\end{bmatrix}^T$, the model becomes

$$\dot{q} = \begin{bmatrix} \cos\theta \\ \sin\theta \\ \frac{\tan\phi}{L} \\ 0 \end{bmatrix} v + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \omega\,. \tag{A.22}$$

With this model, the control vector fields in (A.22) annihilate the constraints given in (A.19) and (A.20). With the kinematic model, the constraint in (A.21) is implicit. This is the model for car-like systems given most often in the robotics literature [72, 94]. Straightforward calculations show that this drift-less model is small-time locally controllable [94]. Although simpler, this model loses some modeling freedom, specifically that of considering dynamical effects. Thus, this thesis considers the more general purely kinematic model.

Returning to the full configuration space $\mathcal{Q} = SE(2) \times \mathbf{S}^1 \times \mathbf{I}$, we note that the drive and steering angles are controlled, so let the base space $B = \mathbf{S}^1 \times \mathbf{I}$ and the fiber space $G = SE(2)$. Unfortunately, the system constraints lack the simple form give in (A.2), which prevents us from directly defining the connection as in the first example.

It is possible, from inspection and some careful algebra, to derive the decomposition without recourse to the principle connection. We present that result here to show the form, and then derive the connection from first principles using the connection. For an arbitrary velocity vector, $v_q \in T_q\mathcal{Q}$, can be decomposed as

$$v_q = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \\ \dot{\psi} \\ \dot{\phi} \end{bmatrix} = \begin{bmatrix} R\,\dot{\psi}\cos\theta \\ R\,\dot{\psi}\sin\theta \\ \frac{R}{L}\dot{\psi}\tan\phi \\ \dot{\psi} \\ \dot{\phi} \end{bmatrix} + \begin{bmatrix} \dot{x} - R\,\dot{\psi}\cos\theta \\ \dot{y} - R\,\dot{\psi}\sin\theta \\ \dot{\theta} - \frac{R}{L}\dot{\psi}\tan\phi \\ 0 \\ 0 \end{bmatrix} = \operatorname{hor} v_q + \operatorname{ver} v_q\,, \tag{A.23}$$

the horizontal and vertical components [59]. As a purely kinematic system, the horizontal part obeys the constraints, with the fiber velocity portion given as a purely kinematic relation with the base variable velocities.

Using first principles, we now derive the connection formally. At the group identity element, the constraints given in (A.19)-(A.21) for the car-like system are

$$\dot{y} = 0\,, \tag{A.24}$$

$$\dot{x}\sin\phi - \dot{y}\cos\phi - L\dot{\theta}\cos\phi = 0\,, \tag{A.25}$$

and

$$\dot{x} - R\,\dot{\psi} = 0. \tag{A.26}$$

Substituting (A.24) and (A.26) into (A.25), and simplifying we obtain

$$\dot{\theta} - \frac{R}{L}\,\dot{\psi}\tan\phi = 0\,. \tag{A.27}$$

Taking (A.24), (A.26), and (A.27), which all have the simple form given in (A.2), the local connection $\mathbb{A}\left(r\right)$ is given by

$$\mathbb{A}\left(r\right) = \begin{bmatrix} -R & 0 \\ 0 & 0 \\ -\frac{R}{L}\tan\phi & 0 \end{bmatrix}.$$

Given this unique specification, (A.7) and (A.8) determine the projection in local coordinates onto the vertical space ver_q; this yields

$$\mathrm{ver}_q v_q = \mathrm{ver}_q\left(\dot{g}, \dot{r}\right) = \left(T_e R_g\left(\mathrm{Ad}_g\left(T_g L_{g^{-1}}\dot{g} + \mathbb{A}\left(r\right)\dot{r}\right)\right), 0\right)\,.$$

The principal connection for the car-like system is

$$\begin{aligned} \mathcal{A}\left(\dot{g}, \dot{r}\right) &= \begin{bmatrix} \cos\theta & -\sin\theta & y \\ \sin\theta & \cos\theta & -x \\ 0 & 0 & 1 \end{bmatrix}\left(\begin{bmatrix} \dot{x}\,\cos\theta + \dot{y}\,\sin\theta \\ \dot{y}\,\cos\theta - \dot{x}\,\sin\theta \\ \dot{\theta} \end{bmatrix} + \begin{bmatrix} -R\,\dot{\psi} \\ 0 \\ -\frac{R}{L}\,\dot{\psi}\,\tan\phi \end{bmatrix}\right) \\ &= \begin{bmatrix} \dot{x} - \left(R\,\cos\theta + \frac{R}{L}\,y\,\tan\phi\right)\dot{\psi} + y\,\dot{\theta} \\ \dot{y} + \left(\frac{R}{L}\,x\,\tan\phi - R\,\sin\theta\right)\dot{\psi} - x\,\dot{\theta} \\ \dot{\theta} - \frac{R}{L}\,\dot{\psi}\,\tan\phi \end{bmatrix}. \end{aligned} \tag{A.28}$$

The generator of the connection specifies the vertical vector. In $\mathfrak{se}(2)$, the generator operator is

$$T_e R_g = \begin{bmatrix} 1 & 0 & -y \\ 0 & 1 & x \\ 0 & 0 & 1 \end{bmatrix}.$$

Therefore, the vertical component is

$$\begin{aligned}
\mathrm{ver}_q v_q &= \left\{T_e R_g \mathcal{A}\left(\dot{r},\dot{g}\right),0\right\} \\
&= \begin{bmatrix} \begin{bmatrix} 1 & 0 & -y \\ 0 & 1 & x \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \dot{x} - \left(R\cos\theta + \frac{R}{L}\, y\, \tan\phi\right)\dot{\psi} + y\,\dot{\theta} \\ \dot{y} + \left(\frac{R}{L}\, x\, \tan\phi - R\,\sin\theta\right)\dot{\psi} - x\,\dot{\theta} \\ \dot{\theta} - \frac{R}{L}\,\dot{\psi}\,\tan\phi \end{bmatrix} \\ 0 \\ 0 \end{bmatrix} \\
&= \begin{bmatrix} \dot{x} - R\,\dot{\psi}\,\cos\theta \\ \dot{y} - R\,\dot{\psi}\,\sin\theta \\ \dot{\theta} - \frac{R}{L}\,\dot{\psi}\,\tan\phi \\ 0 \\ 0 \end{bmatrix} \qquad \text{(A.29)}
\end{aligned}$$

and the horizontal part is

$$\begin{aligned}
\mathrm{hor}_q v_q &= v_q - \mathcal{A}\left(\dot{r},\dot{g}\right)_{\mathcal{Q}}(q) \\
&= \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \\ \dot{\psi} \\ \dot{\phi} \end{bmatrix} - \begin{bmatrix} \dot{x} - R\,\dot{\psi}\,\cos\theta \\ \dot{y} - R\,\dot{\psi}\,\sin\theta \\ \dot{\theta} - \frac{R}{L}\,\dot{\psi}\,\tan\phi \\ 0 \\ 0 \end{bmatrix} \\
&= \begin{bmatrix} R\,\dot{\psi}\,\cos\theta \\ R\,\dot{\psi}\,\sin\theta \\ \frac{R}{L}\,\dot{\psi}\,\tan\phi \\ \dot{\psi} \\ \dot{\phi} \end{bmatrix}, \qquad \text{(A.30)}
\end{aligned}$$

as given in (A.23).

For the Ackermann steered car, this thesis uses $\dot{g} = A(q)\,\dot{r}$, with

$$A(q) = \begin{bmatrix} R\cos\theta & 0 \\ R\sin\theta & 0 \\ \frac{R}{L}\tan\phi & 0 \end{bmatrix}. \qquad \text{(A.31)}$$

The examples considered thus far – the vertical rolling disk, the differential-drive, and the Ackermann steered car – have been principally kinematic, and therefore, purely kinematic. We now consider a system that is purely kinematic, but is NOT principally kinematic. The key difference between the system is the lack of invariance properties.

A.4.4 Diff-drive towing a trailer

Consider the system shown in Figure A.6, which consists of a differential-drive robot towing a trailer. The trailer body is attached to a rotating joint whose axis intersects the center of the diff-drive axle. The trailer body extent contains that of the differential-drive body, so the trailer is treated as the robot body. For this example, the body reference frame is attached to the rotating joint. The

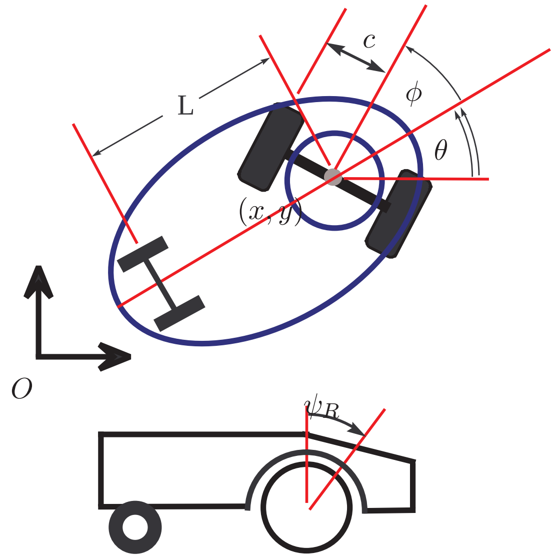


Figure A.6: Differential-drive robot towing a trailer. The system has two drive wheels $\{\psi_L, \psi_R\}$ as base variables.

system is driven by the two wheels attached to the differential-drive robot; the trailer has wheels located a distance L from the rotation point. Denoting the angle between the differential-drive robot and the trailer body as ϕ, the configuration is given by $q = (x, y, \theta, \phi, \psi_L, \psi_R)$.

The system is subject to four independent nonholonomic constraints. Three are the same as the differential-drive robot: the two drive wheels roll without slipping, and the drive wheels cannot slide sideways. The fourth is that the wheels on the trailer cannot slide sideways. In Pfaffian form, these constraints are

$$\begin{bmatrix} \sin(\theta+\phi) & -\cos(\theta+\phi) & 0 & 0 & 0 & 0 \\ \cos(\theta+\phi) & \sin(\theta+\phi) & 0 & -c & -R & 0 \\ \cos(\theta+\phi) & \sin(\theta+\phi) & 0 & c & 0 & -R \\ -\sin\theta & \cos\theta & -L & 0 & 0 & 0 \end{bmatrix} \cdot \dot{q} = \mathbf{0}\,.$$

The number of independent nonholonomic constraints is equal to the dimension of the fiber space; however, these constraints are not in the simple form given in (A.2).

To see the lack of invariance, we define the fiber $F = SE(2) \times \mathbf{S}^1 \approx (x, y, \theta, \phi)$, and base $B = (\psi_L, \psi_R)$. The fiber is the direct product of two groups; therefore, F is a group. The base space is defined by the two actuated drive wheels.

Define the matrix $A(q)$, which spans the null space of the Pfaffian constraints, as

$$A(q) = \begin{bmatrix} \frac{R\cos(\theta+\phi)}{2} & \frac{R\cos(\theta+\phi)}{2} \\ \frac{R\sin(\theta+\phi)}{2} & \frac{R\sin(\theta+\phi)}{2} \\ \frac{R\sin(\phi)}{2L} & \frac{R\sin(\phi)}{2L} \\ \frac{-R}{2c} & \frac{R}{2c} \\ 1 & 0 \\ 0 & 1 \end{bmatrix}.$$

The matrix $A\left(q\right)$ is homeomorphic to the constraint distribution $D_q \subset \mathcal{TQ}$. The lifted action of the $SE(2)$ fiber components only act on the $SE(2)$ velocity components; likewise, the lifted action of the $\mathbf{S}^1$ component only acts on $\dot{\phi}$. Therefore, $T_g D_q \neq D_{gq}$, so the constraint distribution is not invariant; therefore, the system is NOT principally kinematic. However, by inspection, $dq = A\left(q\right)\dot{r}$, where $\dot{r} = \begin{bmatrix}\dot{\psi}_L & \dot{\psi}_R\end{bmatrix}^T$. Therefore, by definition, the system is purely kinematic.

In this case, the constraints and equations of motion are determined by carefully analyzing the system, and applying algebraic manipulations. The systematic approach available to principally kinematic systems does not apply. The hybrid control approaches developed in this thesis depend only upon $A\left(q\right)$, and not on a principal connection; therefore, the policy design and planning techniques are directly applicable to a differential-drive towing a trailer. Provided that the trailer body covers the differential-drive system, and the system extent is treated as a single rigid body.

Appendix B

Details of Control Policies For Fully Actuated Systems

This appendix provides details of policy derivations, and proofs of lemmas referenced in Chapter 4.

B.1 Vector Field Design Details

The vector field design presented in Chapter 4 is based on the pull-back of a potential function defined over a unit n-ball. This section provides details for the general mapping used from arbitrary convex polytopes to n-balls, and the specific mapping for polygons to the unit disk in $\mathbb{R}^2$. The section concludes with the derivation of the harmonic potential function used on the disk.

B.1.1 Mapping of Convex Polytopes to n-ball

In this thesis, the basic cells used for fully actuated systems are convex polytopes, which can be specified by the intersection of a set of half space constraints. It takes a minimum of $n+1$ half space constraints to bound n-dimensional space. This makes the boundaries easy to specify, and involves trivial calculations to check a point for inclusion. In lower dimensions, polytopes are the familiar *polygons* in $\mathbb{R}^2$and *polyhedra* in $\mathbb{R}^3$.

Each half space constraint is represented by a point, $p \in \mathbb{R}^n$, and a unit normal, $\mathbf{n} \in \mathbb{R}^n$. Define the normal direction as the outward[1] pointing normal with respect to the polytope being defined. Therefore, the normal direction changes as the system goes from cell to cell across a common face. Let p be the center of the polytope face being specified by the half space constraint. Note, this is an over parameterization of the half-space constraint. All that is required is the distance from the origin, and not a point on the hyper plane. We choose to carry p for use in later transformations.

A necessary condition for the set $\{(p_i, \mathbf{n}_i) \mid i = 1, \ldots, m\}$ of half-space constraints to specify a valid polytope is that the face normals $\mathbf{n}_i$ positively span the free space. Let $\beta_i(q) = -\mathbf{n}_i \cdot (q - p_i)$, the distance from a point q to the hyperplane defining the i^{th} half space constraint. Another necessary condition is that the center point of each face, p_i, is contained in the intersection of the other half space constraints, that is

$$\forall\, i = 1 \ldots m\, ,\, \forall\, j = 1 \ldots m\, ,\, i \neq j\, ,\; \beta_j(p_i) > 0\, .$$

If q is an interior point of the cell, then $\beta_i(q) > 0$ for all $i \in 1 \ldots m$. This allows us to compactly specify a convex polytope $\mathcal{P}$ as

$$\mathcal{P} = \{q \in \mathbb{R}^n \mid \forall\, i = 1 \ldots m\, , \beta_i(q) > 0\}\ ,$$

[1]This is a change in notation from earlier papers, but is consistent with outward pointing normals used for arbitrary cells in Chapter 5.

assuming that the m half-space constraints form a valid polytope. If $\beta_i\left(q\right) = 0$ for some, but not all, of the half space constraints, q is on the boundary of the cell.

Define q_β such that

$$q_\beta = \arg\max_{q\in\mathcal{P}} \prod_{i=1}^{m} \beta_i\left(q\right) \ ,$$

and let

$$\beta_{\max} = \prod_{i=1}^{m} \beta_i\left(q_\beta\right) \ .$$

Thus $\beta_{\max}$ is the maximum value of the product of distances to each face on the interior of the cell. Define the scaled distance product function $\beta\left(q\right)$ as

$$\beta\left(q\right) = \beta_{\max}^{\frac{1-m}{m}} \prod_{i=1}^{m} \beta_i\left(q\right) \ . \tag{B.1}$$

Lemma B.1.1 *The set of local maxima of $\beta\left(q\right)$ on the interior of $\mathcal{P}$ is a singleton. Furthermore, $\beta\left(q\right)$ is free of local minima on the interior of $\mathcal{P}$.*

Proof: By construction, $\beta\left(q\right)$ is positive over the interior of $\mathcal{P}$ and zero along the boundary of $\mathcal{P}$. Therefore $\beta\left(q\right)$ has at least on point corresponding to a global maximum on the interior of $\mathcal{P}$. Let such a point, denoted q_β, be given. Without loss of generality, transform the polytope such that q_β is at the origin.

Assume there exists another local maxima on the interior, and denote such point as q_m.

Define the line segment between the origin and q_m as $\mathbf{l}\left(t\right) = q_m\, t$, and note that β restricted to the line segment is given by

$$\begin{aligned} \beta\left(t\right) = \beta\left(l\left(t\right)\right) &= \beta_{\max}^{\frac{1-m}{m}} \prod_{i=1}^{m} -\mathbf{n}_i \cdot \left(\mathbf{l}\left(t\right) - p_i\right) \\ &= \beta_{\max}^{\frac{1-m}{m}} \prod_{i=1}^{m} \left(-\mathbf{n}_i^T q_m\, t + \mathbf{n}_i^T p_i\right) \\ &= \beta_{\max}^{\frac{1-m}{m}} \prod_{i=1}^{m} \left(a_i\, t + d_i^0\right) \end{aligned} \tag{B.2}$$

where $a_i = -\mathbf{n}_i^T q_m$, $d_i^0 = \mathbf{n}_i^T p_i$, and $t \in [0,1]$.

The derivative, $D_t\beta\left(t\right)$, along the line segment parameterized by t is

$$D_t\beta\left(t\right) = \beta_{\max}^{\frac{1-m}{m}} \sum_{i=1}^{m} \left(a_i \prod_{\substack{j=1\\ j\neq i}}^{m} \left(a_j\, t + d_j^0\right) \right) \ . \tag{B.3}$$

The second derivative is

$$\begin{aligned} D_{tt}\beta\left(t\right) &= \beta_{\max}^{\frac{1-m}{m}} \sum_{i=1}^{m}\left(a_i \sum_{\substack{j=1\\ j\neq i}}^{m}\left(a_j \prod_{\substack{k=1\\ k\neq i\\ k\neq j}}^{m}\left(a_k\, t + d_k^0\right)\right)\right) \\ &= \beta_{\max}^{\frac{1-m}{m}} \sum_{i=1}^{m}\left(a_i \sum_{\substack{j=1\\ j\neq i}}^{m}\left(a_j \frac{1}{a_i\, t + d_i^0} \prod_{\substack{k=1\\ k\neq j}}^{m}\left(a_k\, t + d_k^0\right)\right)\right) \\ &= \beta_{\max}^{\frac{1-m}{m}} \sum_{i=1}^{m}\left(\frac{a_i}{a_i\, t + d_i^0} \sum_{\substack{j=1\\ j\neq i}}^{m}\left(a_j \prod_{\substack{k=1\\ k\neq j}}^{m}\left(a_k\, t + d_k^0\right)\right)\right) \\ &= \beta_{\max}^{\frac{1-m}{m}} \sum_{i=1}^{m}\left(\frac{a_i}{a_i\, t + d_i^0} A_i\right) , \qquad \text{(B.4)} \end{aligned}$$

where

$$A_i = \sum_{\substack{j=1\\ j\neq i}}^{m}\left(a_j \prod_{\substack{k=1\\ k\neq j}}^{m}\left(a_k\, t + d_k^0\right)\right) .$$

q_β and q_m are assumed to be local maximum points; therefore $\beta\left(t\right)$ must be constant or there must be a local minima in $t \in (0,1)$. In either case, $D_t\beta\left(t^*\right) = 0$ for some $t^* \in (0,1)$. Note, that

$$D_t\beta\left(t\right)\, |_{t^*} = A_i + a_i \prod_{\substack{k=1\\ k\neq i}}^{m}\left(a_k\, t^* + d_k^0\right) ,$$

which implies that

$$A_i = -a_i \prod_{\substack{k=1\\ k\neq i}}^{m}\left(a_k\, t^* + d_k^0\right) . \qquad \text{(B.5)}$$

Substitute (B.5) into (B.4) and simplify to obtain

$$D_{tt}\beta\left(t^*\right) = -\beta_{\max}^{\frac{1-m}{m}} \sum_{i=1}^{m}\left(\frac{a_i^2}{a_i\, t + d_i^0} \prod_{\substack{k=1\\ k\neq i}}^{m}\left(a_k\, t^* + d_k^0\right)\right) .$$

Each term in the summation is positive, therefore $D_{tt}\beta\left(t^*\right) < 0$, which implies $\mathbf{l}\left(t^*\right)$ is a local maximum. This contradicts the assumption that β is constant or has a local minimum along the line segment. Therefore, q_m must equal q_β, and there is a single local maximum on the interior of $\mathcal{P}$. Since β is positive over the interior of $\mathcal{P}$, and there is only one local maximum on the interior, there cannot be a local minimum in the interior.

□

From Lemma B.1.1, conclude that $\beta(q)$ monotonically decreases as q approaches the boundary of $\mathcal{P}$ along a ray from q_β in all directions.

Given the specification of a valid convex polytope, we construct a mapping to the unit ball using the scaled distance product, $\beta(q)$. First, note that the desirable properties of the constructed vector fields are invariant under rigid body transformation. Let T be the rigid body motion that transforms the cell so that the point q_β is at the origin, and the center point, p, of the designated outlet face lies on the negative x_1 axis. Such an operator maps face points to face points, and face normals to face normals. Unless otherwise noted, henceforth assume that the cell is transformed such that $p_i = T(p_i)$, $\mathbf{n}_i = T(\mathbf{n}_i)$, and $q = T(q) \in T(\mathcal{P})$.

Given a convex polytope $\mathcal{P}$ and transformation T, define $\varphi : T(\mathcal{P}) \to \mathcal{B}$ as

$$\varphi(q) = \frac{q}{\|q\| + \beta(q)}\,, \tag{B.6}$$

where $\beta(q) : T(\mathcal{P}) \to \mathbb{R}$ is given in (B.1). The scaling term $\beta_{\max}^{\frac{1-m}{m}}$ in $\beta(q)$ makes the mapping invariant to scale. The mapping in (B.6) maps the origin to the origin, and any point on the boundary to the $(n-1)$-sphere.

The Jacobian matrix, $D_q\varphi$, is given by $D_q\varphi = \left[D_{x_j}\varphi_i\right]$, where x_j is the j^{th} component of q, and φ_i is the i^{th} component of $\varphi(q)$. Each element of the Jacobian is

$$\begin{aligned} D_{x_j}\varphi_i &= \frac{\delta_{i,j}}{\|q\| + \beta(q)} - \frac{\varphi_i}{(\|q\| + \beta(q))^2}\left(D_{x_j}\|q\| + D_{x_j}\beta(q)\right) \\ &= \frac{\delta_{i,j}}{\|q\| + \beta(q)} - \frac{\varphi_i}{(\|q\| + \beta(q))^2}\left(\frac{\varphi_j}{\|q\|} + D_{x_j}\beta(q)\right). \end{aligned} \tag{B.7}$$

where

$$\delta_{i,j} = \begin{cases} 1 & i = j \\ 0 & \text{otherwise} \end{cases}\,.$$

Note, when $\|q\| = 0$, $\varphi_i = \varphi_j = 0$, and

$$\lim_{\|q\|\to 0} D_{x_j}\varphi_i = \frac{\delta_{i,j}}{\beta_{\max}^{\frac{1}{m}}}\,.$$

The distance function partial derivative is

$$D_{x_j}\beta(q) = \beta_{\max}^{\frac{1-m}{m}} \sum_{i=1}^{m}\left(-\mathbf{n}_{i,j}\prod_{\substack{k=1\\k\neq i}}^{m}\beta_k(q)\right)\,, \tag{B.8}$$

where $\mathbf{n}_{i,j}$ is the j^{th} component of the i^{th} normal vector.

Lemma B.1.2 *The mapping φ is full rank on the interior of the cell $\mathcal{P}$.*

A sketch the proof is given for arbitrary dimensions; a detailed proof is given later for 2D.

Proof: (Sketch) The mapping φ is designed to preserve angles relative to the origin, and scale the radial component. Consider the spherical coordinate representation of the mapping. The

Jacobian has diagonal 1's for the angle coordinates. In order for the mapping to be singular, the partial derivative of the radial scaling with respect to the radius must be zero.

Let q_I be the point of intersection with the boundary, along the ray from the origin through the arbitrary point q in the interior of $\mathcal{P}$. This relationship is shown in Figure B.1. The radial scaling is given by

$$r\left(t\right) = \frac{\rho\, t}{\rho\, t + \beta\left(q_I t\right)}\,,$$

where $\rho = \|q_I\|$, and $t \in [0, 1]$. Then,

$$\begin{aligned} D_t r\left(t\right) &= \frac{\rho}{\left(\rho\, t + \beta\right)} - \frac{\rho t\left(\rho + D_t\beta\right)}{\left(\rho\, t + \beta\right)^2} \\ &= \frac{\rho\left(\rho\, t + \beta\right)}{\left(\rho\, t + \beta\right)^2} - \frac{\rho t\left(\rho + D_t\beta\right)}{\left(\rho\, t + \beta\right)^2} \\ &= \frac{\rho\,\left(\beta - t\, D_t\beta\right)}{\left(\rho\, t + \beta\right)^2}\,, \end{aligned}$$

which only equals zero when

$$\beta - t\, D_t\beta = 0\,.$$

However, by Lemma B.1.1, β is monotonically decreasing as we move along the ray parameterized by t, so that $D_t\beta < 0$. Since $t \geq 0$ and $\beta > 0$ over the interior of $\mathcal{P}$,

$$\beta - t\, D_t\beta > 0$$

for all $q \in \mathcal{P}$. Therefore, the Jacobian in spherical coordinates is full rank.

The mapping from Cartesian to spherical coordinates is full rank, except at the origin, where a proper limit for this mapping exists. Therefore, as the Jacobian is full rank everywhere on the interior, the mapping φ is full rank over the interior of $\mathcal{P}$ [8].

□

The mapping fails to be C^∞ at the origin due to the use of the radial retraction. However, this isolated discontinuity can be accommodated by a local smoothing function.

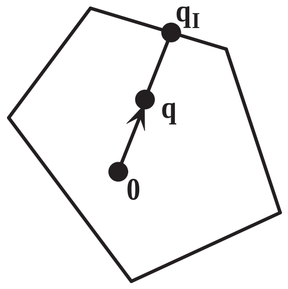


Figure B.1: The point q and origin determine a point of intersection q_I in the linear retraction mapping.

Mapping to a Unit Disk

The simulations presented in this thesis are for systems evolving on $\mathbb{R}^2$, therefore this section presents the specifics for the mapping of polygons to the unit disk.

From (B.6),

$$\varphi(q) = \left(\frac{x}{\sqrt{x^2+y^2}+\beta(q)}, \frac{y}{\sqrt{x^2+y^2}+\beta(q)} \right). \tag{B.9}$$

The Jacobian, $D_q\varphi$, is

$$D_q\varphi = \frac{1}{\left(\sqrt{x^2+y^2}+\beta(q)\right)^2} \begin{bmatrix} \frac{y^2}{\sqrt{x^2+y^2}} + \beta(q) - xD_x\beta(q) & -\frac{xy}{\sqrt{x^2+y^2}} - xD_y\beta(q) \\ -\frac{xy}{\sqrt{x^2+y^2}} - yD_x\beta(q) & \frac{x^2}{\sqrt{x^2+y^2}} + \beta(q) - yD_y\beta(q) \end{bmatrix} \tag{B.10}$$

Lemma B.1.3 *The mapping φ given in (B.9) from an arbitrary polygon to the unit disk is full rank everywhere on the interior of the polygon.*

Proof: If $q = 0$, then a simple limit operation yields

$$D_q\varphi \mid_{q=0} = \begin{bmatrix} \frac{1}{\beta_{\max}^{\frac{1}{m}}} & 0 \\ 0 & \frac{1}{\beta_{\max}^{\frac{1}{m}}} \end{bmatrix},$$

which is full rank. Assume, $\|q\| \neq 0$, and consider the determinant of $D_q\varphi$,

$$\text{Det}\left(D_q\varphi\right) = -\frac{\left(x^2+y^2+\sqrt{x^2+y^2}\,\beta(q)\right)\left(-\beta(q) + x\,D_x\beta(q) + y\,D_y\beta(q)\right)}{\sqrt{x^2+y^2}\left(\sqrt{x^2+y^2}+\beta(q)\right)^4}. \tag{B.11}$$

The denominator and the first term in parenthesis in the numerator are positive, non-zero numbers for $\|q\| > 0$. Therefore, to lose rank, the second term, $-\beta(q) + x\,D_x\beta(q) + y\,D_y\beta(q)$ must be zero. Assume this is true for some $q = (x, y) \in \mathcal{P}$. Then,

$$\beta(q) = D_q\beta(q)\;q\,. \tag{B.12}$$

However, by Lemma B.1.1, $\beta(q)$ is monotonically decreasing as we move along the ray through q originating at the origin. Therefore, $D_q\beta(q)\;q < 0$ for all $q \in \mathcal{P}$, while $\beta(q) > 0$, contradicting (B.12). Therefore, the determinant is non-zero.

□

The mapping is singular on the boundaries, and in fact, the Jacobian is the zero matrix at a vertex of the polygon. This necessitates an approximation of the cell near the polygon vertices; see [26] for an approach that uses fillet curves.

Mapping Unit Disk to Unit Disk

The convergent vector field design requires a diffeomorphism $\psi : \mathcal{B}\backslash q_g^b \to \mathcal{B}\backslash 0$ between a unit disk mapped from the polygon and a unit disk whose origin corresponds to the goal. The mapping ψ

maps boundary to boundary and goal to origin; that is $\psi(\partial\mathcal{B}) = \partial\mathcal{B}$ and $\lim_{q\to q_g} \psi\left(q_g^b\right) = 0$, where $q_g^b = \varphi\left(q\right)$.

For the unit disk, a mapping based on complex numbers serves the purpose. Let $z = q^b = \varphi(q)$ be an arbitrary point in the unit disk represented in complex plane. Let $z_g = q_g^b = \varphi(q_g)$ be the goal point in the complex plane. Then

$$z_\psi = \psi(z) = \frac{z - z_g}{1 - \bar{z}_g \cdot z}\,, \tag{B.13}$$

where $\bar{z}_g$ is the complex conjugate of z_g. Clearly, $z_\psi = 0$ if $z = z_g$. Simple algebraic calculations show that the boundary maps to the boundary.

B.1.2 Harmonic Functions on a Unit Disk

Given the mapping φ from above, and a potential function, γ_b, on the unit ball, define the potential over the polytope as the pullback given by

$$\gamma = \gamma_b \circ \varphi\,.$$

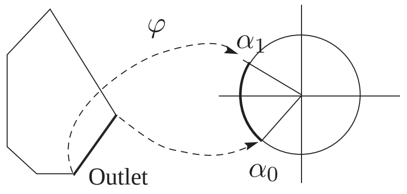
The potential function on the unit ball is based on a harmonic potential function, which is a solution to Laplace's equation

$$\nabla^2\gamma_b = \frac{\partial^2\gamma_b}{\partial {x_1}^2} + \cdots + \frac{\partial^2\gamma_b}{\partial {x_n}^2} = 0\,. \tag{B.14}$$

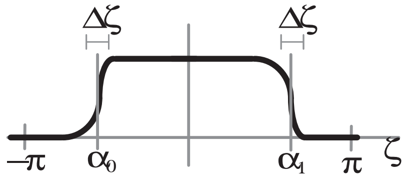
The harmonic potential function has several "nice properties" relevant to this work; there are no interior local minima and the solution is C^∞ smooth [35, 106].

On the unit n-ball, the solution is a computable integral function; on the unit disk with piecewise continuous boundary conditions the solution exists in closed form [35, 106]. Let $q_d = \varphi\left(q\right) = \left(x_d, y_d\right)$ be the Cartesian coordinates of a point in disk mapped from a point in the polygon. For the disk, the most natural representation is in polar coordinates, so define

$$\begin{aligned}\rho &= \sqrt{x_d^2 + y_d^2}\,,\\ \theta &= \operatorname{atan2}\left(y_d, x_d\right)\,.\end{aligned}$$



(a)



(b)

Figure B.2: Generating potential on polygon. a) Mapping from polygon to disk with outlet zone identified. b) Discontinuous boundary conditions approximated by continuous functions as $\Delta\zeta \to 0$.

If the boundary condition is 0 along the outlet zone, and 1 along the inlet zone, the solution to Laplace's equation on the unit disk in $\mathbb{R}^2$ is

$$\begin{aligned}\gamma_b\,(\rho,\theta) = \frac{\alpha_1-\alpha_0}{2\,\pi} \quad &+ \quad \frac{1}{\pi}\tan^{-1}\left(\frac{\rho\sin\left(\alpha_1-\theta\right)}{1-\rho\cos\left(\alpha_1-\theta\right)}\right) \\ &- \quad \frac{1}{\pi}\tan^{-1}\left(\frac{\rho\sin\left(\alpha_0-\theta\right)}{1-\rho\cos\left(\alpha_0-\theta\right)}\right)\,, \end{aligned} \tag{B.15}$$

where α_i denote the angle coordinates of the vertices of the outlet face. Figure B.2 shows the mapping and boundary condition used in solution. Except for the discontinuities at $(\rho,\theta)=(1,\alpha_0)$ and $(\rho,\theta)=(1,\alpha_1)$, (B.15) obeys Laplace's equation and satisfies the boundary conditions. To calculate the gradient, differentiate (B.15) in terms of x_d and y_d and simplify to yield

$$D_{q_d}\gamma_b = \begin{bmatrix} \frac{-\sin(\alpha)+\sin(\beta)+\rho\,(2\,\cos(\theta)\,\sin(\alpha-\beta)+\rho\,(-\sin(\alpha-2\,\theta)+\sin(\beta-2\,\theta)))}{\pi\,(1+\rho^2-2\,\rho\,\cos(\alpha-\theta))\,(1+\rho^2-2\,\rho\,\cos(\beta-\theta))} \\ \\ \frac{\cos(\alpha)-\cos(\beta)+\rho\,(-(\rho\,\cos(\alpha-2\,\theta))+\rho\,\cos(\beta-2\,\theta)+2\,\sin(\alpha-\beta)\,\sin(\theta))}{\pi\,(1+\rho^2-2\,\rho\,\cos(\alpha-\theta))\,(1+\rho^2-2\,\rho\,\cos(\beta-\theta))} \end{bmatrix}^T . \tag{B.16}$$

The gradient of the potential in the polygon is $D_q\gamma = D_{q_d}\gamma_b\,D_q\varphi$, which is all that is needed to calculate the negative normalized gradient vector field $\hat{X}$.

B.2 Component Policy Design Details

This section provides details for the policy design for second order systems.

B.2.1 Unconstrained Dynamics Control Policy

Recall from Section 4.1.2 the second order system of the form

$$\ddot{q} = u\,, \tag{B.17}$$

where u is unbounded. Using either the convergent or flow-through vector fields as a velocity reference, the velocity regulation control law is

$$u = K\,\left(X(q)-\dot{q}\right)+\dot{X}(q)\,\,, \tag{B.18}$$

where $K > 0$ is the "velocity regulation" gain, which acts to decrease the error and the feed-forward term, $\dot{X}(q) = D_q X\,\dot{q}$, accounts for the change in the vector field as the system moves in the $\dot{q}$ direction [105].

Lemma B.2.1 *In the absence of constraints, including those of the cell boundary, the integral curves of the vector field $X(q)$ are attractive to the trajectories of the closed loop system defined by (B.17) under the influence of (B.18).*

Proof: Define the velocity error, $\mathbf{e} = X(q)-\dot{q}$, and consider the set

$$\mathcal{V} := \left\{(q,\dot{q})\;\mid \|\mathbf{e}\| = 0\right\}.$$

Define a Lyapunov-like function of the form

$$\begin{aligned}\eta_v &= \frac{1}{2}\mathbf{e}^T\mathbf{e} \\ &= \frac{1}{2}\left(X(q)-\dot{q}\right)^T\ \left(X(q)-\dot{q}\right) \end{aligned} \tag{B.19}$$

Evaluating the time derivative of (B.19) along the trajectories of the closed loop system, and substituting (B.18) yields

$$\begin{aligned}\dot{\eta_v} &= \left(X(q)-\dot{q}\right)^T\ \left(\dot{X}(q)-\ddot{q}\right) \\ &= \left(X(q)-\dot{q}\right)^T \\ &\qquad \left(\dot{X}(q)-K\ \left(X(q)-\dot{q}\right)-\dot{X}(q)\right) \\ &= -K\ \left(X(q)-\dot{q}\right)^T\ \left(X(q)-\dot{q}\right)\ . \end{aligned} \tag{B.20}$$

For $K > 0$, $\dot{\eta_v} < 0$ for all non-zero velocity error; therefore, the set $\mathcal{V}$ is both attractive and invariant. This implies that the velocity error asymptotically approaches zero [105].

□

The orientation error, defined as the angle between the desired velocity, $X(q)$, and the current velocity, $\dot{q}$, is given by

$$\vartheta = \cos^{-1}\frac{\dot{q}^T X}{\sqrt{\dot{q}^T\dot{q}\,X^T X}}\ , \tag{B.21}$$

where $X = X(q)$ and $\|\dot{q}\| > 0$; if $\|\dot{q}\| = 0$, then define $\vartheta = 0$.

Lemma B.2.2 *In the absence of acceleration constraints, and for initial velocities such that $\dot{q}^T X > 0$, there exists a lower bound on K such that the orientation error, ϑ, monotonically decreases.*

Proof: First, consider the isolated case where $\dot{q} = 0$. Define $\vartheta\ |_{\dot{q}=0} = 0$, since differentially the acceleration will be in the direction of the desired velocity and the orientation error will instantaneously remain zero. As shown below, the orientation error will remain zero for all time.

Now, assume $\|\dot{q}\| > 0$, and consider the set

$$\mathcal{U} := \{(q,\dot{q})\ \mid \vartheta = 0\}\ .$$

Define a Lyapunov-like function of the form

$$\begin{aligned}\eta_u &= \sin^2\vartheta = 1-\cos^2\vartheta \\ &= 1-\frac{X^T\dot{q}\,X^T\dot{q}}{\dot{q}^T\dot{q}\,X^T X}\ . \end{aligned} \tag{B.22}$$

Evaluating the time derivative of (B.22) along the trajectories of the closed loop system, and simplifying yields

$$\begin{aligned}\dot{\eta_u} &= \frac{\left(X^T\dot{q}\right)^2}{\left(\dot{q}^T\dot{q}X^TX\right)^2}\left(\left(2\dot{q}^T\ddot{q}\right)X^TX+\dot{q}^T\dot{q}\left(2X^T\dot{X}\right)\right)-2\frac{X^T\dot{q}}{\dot{q}^T\dot{q}X^TX}\left(X^T\ddot{q}+\dot{q}^T\dot{X}\right)\\ &= \frac{X^T\dot{q}}{\left(\dot{q}^T\dot{q}X^TX\right)^2}\Big(2X^T\dot{q}\,X^TX\,\dot{q}^T\ddot{q}+2X^T\dot{q}\,\dot{q}^T\dot{q}\,X^T\dot{X}+\\ &\qquad\qquad -\,2\dot{q}^T\dot{q}\,X^TX\left(X^T\ddot{q}+\dot{q}^T\dot{X}\right)\Big)\end{aligned} \tag{B.23}$$

Substituting (B.18) into (B.23) and simplifying yields

$$\begin{aligned}\dot{\eta_u} &= \frac{2\left(X^T\dot{q}\right)}{\left(\dot{q}^T\dot{q}X^TX\right)^2}\Big[K\left(\left(X^T\dot{q}\right)^2X^TX-\dot{q}^T\dot{q}\left(X^TX\right)^2\right)\\ &\qquad +\Big(\dot{q}^T\dot{q}\left(X^T\dot{q}-X^TX\right)X^T\dot{X}\\ &\qquad\qquad +\,X^TX\left(X^T\dot{q}-\dot{q}^T\dot{q}\right)\dot{q}^T\dot{X}\Big)\Big]\,.\end{aligned} \tag{B.24}$$

Now consider the case where $\vartheta=0$, that is $\dot{q}$ is aligned with $X(q)$ so that $\dot{q}^TX=\|\dot{q}\|\,\|X\|$. The leading term is a finite positive number since $\|\dot{q}\|>0$, and $\|X\|>0$ by construction. All parenthetical terms inside the brackets of (B.24) are zero; to see this substitute kX for $\dot{q}$ with $0<k\le 1$ and simplify. Therefore, $\dot{\eta_u}=0$, which implies that the set $\mathcal{U}$ is invariant. In other words, if the orientation error is zero, it remains zero.

Away from $\mathcal{U}$, the leading term of (B.24) is positive and bounded, because initially $\dot{q}^TX>0$. Assuming the system has finite initial velocity and $\|X(q)\|$ is finite, it follows that the velocity error is finite; then by Lemma B.2.1, the error magnitude decreases; therefore, $\|\dot{q}\|$ remains finite for all time. Rewrite the parenthetical portion of the first parenthetical term in brackets as

$$\begin{aligned}&\left(\left(X^T\dot{q}\right)^2X^TX-\dot{q}^T\dot{q}\left(X^TX\right)^2\right)\\ =\;&X^TX\left(\left(X^T\dot{q}\right)^2-\dot{q}^T\dot{q}\left(X^TX\right)\right)\\ =\;&X^TX\left(\left(\|X\|\,\|\dot{q}\|\cos\vartheta\right)^2-\dot{q}^T\dot{q}\left(X^TX\right)\right)\\ =\;&\|X\|^4\,\|\dot{q}\|^2\left(\cos^2\vartheta-1\right)\,.\end{aligned} \tag{B.25}$$

This term is clearly negative provided $\vartheta\neq 0$, which means that for $K>0$ the effect is to decrease η_u. The second parenthetical term in brackets has an indeterminate sign, but is finite since all the terms are bounded.

Therefore, for *sufficiently large* K, $\dot{\eta_u}$ can be made negative definite if $0<|\,\vartheta\,|\le\frac{\pi}{2}$. This implies that $\dot{q}^TX$ remains positive, and therefore $\dot{\eta}_u$ is always negative for sufficiently large K. Since $\dot{\eta}_u<0$, we conclude that $\mathcal{U}$ is attractive and invariant, and that ϑ monotonically decreases under the influence of (B.18).

$\square$

To define a composable policy, the policy must also guarantee that the system speed is limited to $\|X(q)\|$. This allows prepares tests to be conducted on adjacent policies by comparing the reference speeds of the vector fields.

Lemma B.2.3 *In the absence of acceleration constraints, and for initial velocities such that $\dot{q}^T X > 0$ and $\|\dot{q}\| \leq \|X\|$, there exists a lower bound on K such that speed never exceeds the reference speed; that is $\|\dot{q}\|$ remains less than $\|X\|$.*

Proof: Given an initial condition where $\|\dot{q}\| \leq \|X\|$, in order to exceed the desired speed, there will exist a time at which $\|\dot{q}\| = \|X\|$.

Assume that $\|\dot{q}\| = \|X\|$, and let $\eta_s = \frac{1}{2}\dot{q}^T\dot{q}$. The change in speed is

$$\begin{aligned}
\dot{\eta}_s \big|_{\|\dot{q}\|=\|X\|} &= \dot{q}^T \ \left(K\left(X-\dot{q}\right)+D_qX\dot{q}\right) \\
&= K\left(\dot{q}^TX-\dot{q}^T\dot{q}\right)+\dot{q}D_qX\dot{q} \\
&= K\left(\|\dot{q}\|\,\|X\|\cos\vartheta-\|\dot{q}\|^2\right)+\dot{q}D_qX\dot{q} \\
&= K\,\|X\|^2\left(\cos\vartheta-1\right)+\dot{q}D_qX\dot{q}\,. \qquad \text{(B.26)}
\end{aligned}$$

The first term is clearly negative for $K > 0$. For the general vector field, the term $\dot{q}D_qX\dot{q}$ is of indeterminate sign, but is finite. Therefore, for *sufficiently large* K, $\dot{\eta}_s$ can be made negative if $\dot{q} \neq X$. If $\dot{q} = X$, the term $\dot{q}D_qX\dot{q}$ encodes the change in $\|X\|$, and the system follows the desired speed profile.

□

Intuitively, making K sufficiently large ensures that the control policy is correcting more quickly than the vector field is changing. Formally, the sufficiently large K is determined such that

$$K > \max\left[\max_{q,\dot{q}} \frac{\left(\dot{q}^T\dot{q}\left(X^T\dot{q}-X^TX\right)X^T\dot{X}+X^TX\left(X^T\dot{q}-\dot{q}^T\dot{q}\right)\dot{q}^T\dot{X}\right)}{\left(\dot{q}^T\dot{q}\ \left(X^TX\right)^2-X^TX\ \left(\dot{q}^TX\right)^2\right)},\ \max_{q,\dot{q}\,\|\dot{q}\|=\|X\|}\frac{\dot{q}D_qX\dot{q}}{\dot{q}^T\dot{q}-\dot{q}^TX}\right]. \qquad \text{(B.27)}$$

This is a worst case limit based on the vector field derivative. Note that both the numerator and denominator of the first term approach zero as $\vartheta \to 0$ or $\|\dot{q}\| \to 0$; therefore, determining a proper upper bound for K is difficult. In this work, K has been chosen by sampling the state space over the cell for points with $\|\dot{q}\| < \|X\|$; a reasonable limit exists.

Lemma B.2.4 [Lemma 4.1.2 in Section 4.1.2] *In the absence of acceleration constraints, with sufficiently large K and initial velocities such that $\|\dot{q}\| = 0$, or $\|\dot{q}\| \leq \|X\|$ and $\dot{q}^TX > 0$, the trajectories of the closed loop system defined by (B.17) under the influence of (B.18), converge to the integral curves of the vector field $X(q)$ in such a way that the trajectory never exits the cell except by the outlet zone and $\|\dot{q}\| \leq \|X\|$ while the system remains in the policy domain. For flow-through vector fields, the system trajectory exits the cell in finite time.*

Proof: If $\|\dot{q}\| \leq \|X\|$ initially, by Lemma B.2.3 one concludes the reference speed is never exceeded.

For initial velocities such that $\dot{q}^TX > 0$, we know the orientation error is initially less than $\frac{\pi}{2}$. By Lemma B.2.2, for sufficiently large K the orientation error is monotonically decreasing.

Assume the trajectory exits the cell in the *inlet* zone, thereby violating the conditional invariance requirement. At the point of departure, $\dot{q}^T X < 0$ given the inward pointing vector field orthogonal to the cell boundary. This implies that $\vartheta > \frac{\pi}{2}$, requiring that the orientation error increased along its trajectory. This contradicts Lemma B.2.2.

For flow-through policies, the vector field $X(q)$ is nowhere zero over the cell; the system cannot come to rest and remain stationary, because the system experiences an acceleration along the vector field. Therefore, the trajectory must leave the cell via the outlet zone under the influence of (B.18) for the given conditions. The speed is non-zero almost everywhere; therefore, the system exits the cell in finite time.

Likewise, for convergent policies the vector field $X(q)$ is non-zero everywhere except at the goal, and the system converges to a neighborhood of the goal in finite time.

□

The utility of lemmas B.2.2 and B.2.4 is limited by two factors. First, a large value for K can lead to an overly aggressive policy over the cell that may prove troublesome for implementation. Secondly, and most importantly, all real world systems have acceleration limits, which may very well be violated by the feed-forward term of (B.18), regardless of the value of K and the velocity error.

B.3 Details of Hybrid Control Policies for Constrained Idealized Dynamical Systems

This appendix provides details of the hybrid control policies introduced in Section 4.1.2. For the model

$$\ddot{q} = u \,, \tag{B.28}$$

consider the following dynamic constraints,

$$\begin{aligned} \|\dot{q}\|_2 &\leq \mathrm{V}_{\max} \,, & \text{(B.29)} \\ \|u\|_2 = \|\ddot{q}\|_2 &\leq \mathrm{A}_{\max} \,. & \text{(B.30)} \end{aligned}$$

The approach presented in Chapter 4 used a velocity reference scaling and hybrid control policies defined over individual cells to guarantee convergence without violating the constraints.

From Section 4.1.2, the reference vector field is $X(q) = s(q)\ \hat{X}(q)$, where

$$s(q) = \min\left(\frac{s^*}{\left\| D_q \hat{X} \right\| + \lambda} \,, \mathrm{V}_{\max} \right) \,. \tag{B.31}$$

with s^* and λ defined as constants. The spectral norm $\left\| D_q \hat{X} \right\|$ encodes "slow down while turning." The vector field derivative is $\dot{X} = D_q X \dot{q}$, with

$$\begin{aligned}
D_q X &= s(q)\ D_q \hat{X} + D_q s(q)\ \hat{X}(q) \\
&= \frac{s^*}{\left\| D_q \hat{X} \right\| + \lambda} D_q \hat{X} - \frac{s^* D_q \left\| D_q \hat{X} \right\|}{\left(\left\| D_q \hat{X} \right\| + \lambda \right)^2} \hat{X}(q) \\
&= \frac{s^*}{\left\| D_q \hat{X} \right\| + \lambda} \left(D_q \hat{X} - \frac{D_q \left\| D_q \hat{X} \right\|\ \hat{X}(q)}{\left(\left\| D_q \hat{X} \right\| + \lambda \right)} \right)
\end{aligned}$$

Consider the limiting case where $u = D_q X\ \dot{q}$ and $\|\dot{q}\| < \|X(q)\|$, then

$$\begin{aligned}
\|u\| &< \left\| \frac{s^*}{\left\| D_q \hat{X} \right\| + \lambda} \left(D_q \hat{X} - \frac{D_q \left\| D_q \hat{X} \right\|\ \hat{X}(q)}{\left(\left\| D_q \hat{X} \right\| + \lambda \right)} \right) \frac{s^*}{\left\| D_q \hat{X} \right\| + \lambda} \right\| \\
&< \frac{(s^*)^2}{\left(\left\| D_q \hat{X} \right\| + \lambda \right)^2} \left\| \left(D_q \hat{X} - \frac{D_q \left\| D_q \hat{X} \right\|\ \hat{X}(q)}{\left(\left\| D_q \hat{X} \right\| + \lambda \right)} \right) \right\|
\end{aligned}$$

Thus, being somewhat conservative, let

$$s^* \leq \min_q \frac{\sqrt{\mathrm{A}_{\max}}\ \left(\left\| D_q \hat{X} \right\| + \lambda \right)}{\sqrt{\left\| D_q \hat{X} - \frac{\hat{X}\ D_q \left\| D_q \hat{X} \right\|}{\left\| D_q \hat{X} \right\| + \lambda} \right\|}} . \tag{B.32}$$

The system can then follow the reference vector field without exceeding the acceleration bound so long as the speed at q does not exceed $\frac{s^*}{\left\| D_q \hat{X} \right\| + \lambda}$.

A hybrid control strategy is used to expand the policy domain; define Φ_S, Φ_A, and Φ_T, where the subscripts S, A, and T refer to "Save," "Align," and "Track" respectively. The control policies obey the prepares relationship

$$\Phi_S \succeq \Phi_A \succeq \Phi_T .$$

B.3.1 Save Control Policy

Consider the case where the best the system can do, using all available acceleration, is prevent collision with the cell boundary. The Save control policy, Φ_S, is used to apply all available acceleration in way that prevents collision with the cell boundaries *if it is at all possible*. The exact form of the Save control policy is dependent on the structure of the cell. The work to date has focused on the use of arbitrary convex polytopes. For a convex polytope, $\mathcal{P}$, the Save control policy developed by Rizzi [105] has maximal domain. This section presents this Save policy, and develops a new expression for the savable set defining the domain of the policy. First, the policy is presented in its basic form; then the switched dynamics induced by the policy are discussed.

The Save control policy, Φ_S, applies all acceleration normal to the boundary at the projected collision point, in order to slow the system and prevent collision if at all possible. The goal of the policy is to bring the system to rest within a given cell without violating the cell boundaries. Define

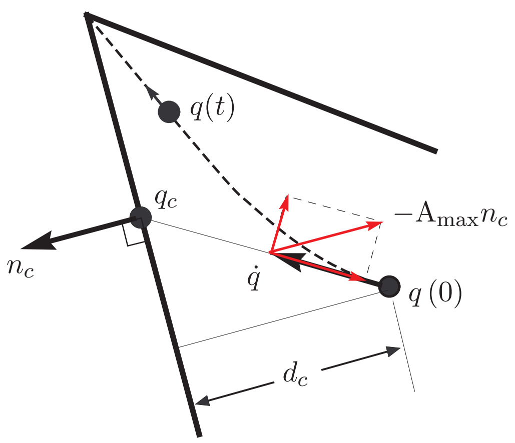


Figure B.3: Collision projection based on current velocity. The acceleration components of the Save control policy always act to slow the overall speed and push the trajectory away from the point of imminent collision to a local maximum of the distance to collision. At the local maximum, the collision normal is aligned with the current velocity.

q_c as the *collision point* on the cell boundary along the direction of the current velocity; that is q_c is the point of boundary intersection if no control input is applied (see Figure B.3). Let n_c denote the outward pointing boundary normal at the collision point; this is termed the *collision normal*. For now, under the general position assumption, assume the collision point is contained in one face of the polytope. That is, the collision does not occur at the intersection of two or more faces of the polytope. Define the Save control policy, Φ_S, as

$$u = -A_{\max} n_c \,. \tag{B.33}$$

The effect of the Save control policy is to accelerate maximally away from the projected collision point.

The effect of Φ_S can be decomposed into a component along the current velocity, and a component orthogonal to the current velocity, as shown in Figure B.3. In every case, the component along the current velocity acts to slow the system down, while the orthogonal component acts to steer the trajectory away from the closest boundary.

Note that the control policy always acts to maximally decrease the component of velocity towards the shortest collision distance. This pushes the trajectory away from the point of imminent collision, and locally increases the time to impact. The acceleration away from the point of first impact will continue until the velocity vector is oriented toward the intersection of two or more faces of the polytope, as shown in Figure B.3. Thus, as the system is accelerating away from the point of imminent collision, it is accelerating towards another face, until the system velocity is oriented toward the intersection of two polytope faces. In this case, accelerating in the direction of either face's surface normal would decrease the time to impact of at least one of the faces. This introduces a discrete change in the required acceleration direction.

When the collision point is on the intersection of two or more faces, redefine the collision normal to be in the positive linear space of the normals of intersecting faces. In the planar case, where the current velocity is directed toward a vertex, the collision normal is aligned along the negative direction of the current velocity. In the general case, the collision normal is oriented so that

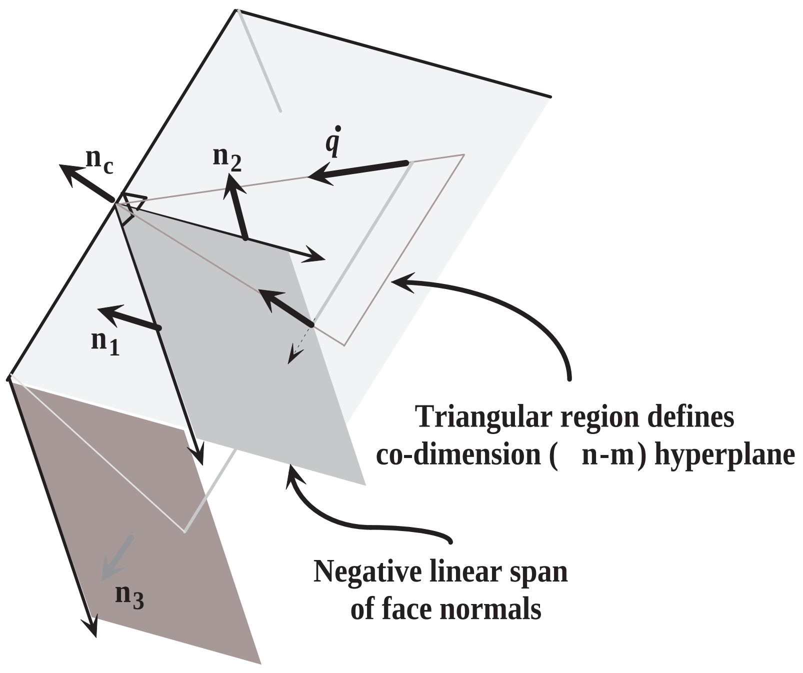

$\dot{q}$
$\mathbf{n_c}$
$\mathbf{n_2}$
$\mathbf{n_1}$
$\mathbf{n_3}$
Triangular region defines co-dimension (n-m) hyperplane
Negative linear span of face normals

Figure B.4: Collision with the intersection of two faces using the Save policy. Face 1 is transparent. With $m = 2$ and $n = 3$, the velocity is contained in a co-dimension 1 plane.

it and the current velocity vector form a co-dimension $(n - m)$ hyper-plane normal to the surface formed by the intersection of the polytope faces, where n is the dimension of the configuration space and m is the number of faces intersecting at the collision point. Figure B.4 shows an example of this for 3-dimensional configuration space. The acceleration component contained in this co-dimension $(n - m)$ hyper-plane is by definition normal to the surface formed by the intersection if the polytope faces. The acceleration pushes the trajectory along the intersection surface towards a “corner”, formed by intersection with third face. The process continues until the intersection surface is a point, and the collision normal is oriented directly opposite the current velocity, which drives the system to rest.

Lemma B.3.1 [Rizzi [105]] *The Save control policy, Φ_S, is capable of bringing to rest any condition in $\mathcal{TP}$ that can be brought to rest without violating the given constraints.*

Proof: Based on [105].

Assume Φ_S cannot prevent collision with the boundary for some initial state, and further assume the existence of another control policy Φ'_S that can prevent collision.

Any boundary violation under the influence of Φ_S involves collision with the “nearest” boundary component, that is the boundary component with the shortest time to impact. However, Φ_S acts to maximally increase the time to impact of the nearest boundary component. If $\Phi'_S \neq \Phi_S$, then Φ'_S must not act to maximally increase the shortest time to impact. But if Φ_S cannot prevent the collision, then neither can Φ'_S.

$\square$

Given the proof of correctness, we seek an expression for defining the savable set for convex polytopes. The distance, d_c, to the *collision* plane defined by the collision point and the collision normal is given by

$$d_c = -n_c^T \left(q - p_c \right) \, ,$$

where n_c is the collision normal, and p_c is a point on the face. Define the collision speed as

$$s_c = n_c^T \dot{q} \, ,$$

where s_c is the velocity component along the normal to the collision point. The collision speed encodes how fast the system is approaching the boundary.

The time required to bring the collision speed to zero, using maximum acceleration in the constant direction of the collision normal, is

$$t_b = \frac{s_c}{\mathrm{A}_{\max}} \, .$$

The distance covered during the braking maneuver is

$$d_b = s_c \, t_b - \frac{1}{2} \mathrm{A}_{\max} t_b^2 = \frac{s_c^2}{2 \mathrm{A}_{\max}} \, .$$

Using these definitions, define the collision avoidance ratio with the initial collision face, ζ_1, as

$$\zeta_1 = \frac{d_b}{d_c} \, .$$

Note, that if $\zeta_1 < 1$ then collision with the first face can be avoided, while $\zeta_1 > 1$ implies that collision is inevitable.

Now, consider the change in ζ_1 as time evolves. From the initial point, both the braking distance d_b and the collision distance d_c decrease by the distance traveled over some differential time period. Write ζ_1 as a function of time, obtaining

$$\begin{aligned} \zeta_1 \left(t \right) &= \frac{d_b - \int_0^t \left(s_c - \mathrm{A}_{\max} \, \tau \right) \, d\tau}{d_c - \int_0^t \left(s_c - \mathrm{A}_{\max} \, \tau \right) \, d\tau} \\ &= \frac{d_b - s_c \, t + \frac{1}{2} \mathrm{A}_{\max} \, t^2}{d_c - s_c \, t + \frac{1}{2} \mathrm{A}_{\max} \, t^2} \, . \end{aligned} \tag{B.34}$$

Here we assume that the cell is a convex polytope, with the collision normal constant over some finite range, Taking the time derivative of (B.34),

$$\begin{aligned} \dot{\zeta}_1 \left(t \right) &= \frac{-s_c + \mathrm{A}_{\max} \, t}{d_c - s_c \, t + \frac{1}{2} \mathrm{A}_{\max} \, t} - \frac{d_b - s_c \, t + \frac{1}{2} \mathrm{A}_{\max} \, t^2}{\left(d_c - s_c \, t + \frac{1}{2} \mathrm{A}_{\max} \, t^2 \right)^2} \left(-s_c + \mathrm{A}_{\max} \, t \right) \\ &= \frac{2 \, \left(d_b - d_c \right) \left(s_c - \mathrm{A}_{\max} \, t \right)}{\left(d_c - s_c \, t + \frac{1}{2} \mathrm{A}_{\max} \, t^2 \right)^2} \, . \end{aligned} \tag{B.35}$$

In the time period before the collision, $s_c - \mathrm{A}_{\max} \, t > 0$ and $d_c - s_c \, t + \frac{1}{2} \mathrm{A}_{\max} \, t^2 > 0$, therefore, the sign of $\dot{\zeta}_1$ depends on the relative values of d_b and d_c. If $d_c > d_b$, then the derivative of the collision ratio is negative, and the collision ratio never increases beyond the unity value signifying imminent collision. Intuitively, the remaining braking distance goes to zero before the collision distance, and $\zeta_1 \to 0$. On the other hand, if $d_b > d_c$, $\dot{\zeta}_1$ is positive, signifying no recovery. In this case, the collision distance goes to zero before the braking distance, and the collision ratio "blows up." While this proves that the system will not collide with the initial collision face, it fails to prove that there will not be a collision with any face on the polytope.

A discrete change in collision normal occurs when the velocity is oriented towards the intersection of two polytope faces. This discrete change requires a modification to the collision ratio calculation, as the acceleration is no longer orthogonal to either face. Therefore, although the Save policy may be able to avoid a first collision face in isolation, collision with the second face may be unavoidable. As the closed-loop dynamics in response to the constant acceleration are easy to determine, it is possible to determine the collision ratio when the system velocity is aligned with the intersection of two or more faces based on the new collision normal.

Let $\mathbf{n}_1$ equal the original collision normal defined by projected collision with a single face, and let p_1 denote a location in the associated face. Using the Save policy defined in (B.33), the closed-loop dynamics are

$$q(t) = q(0) + \dot{q}(0)t - \frac{1}{2}\mathrm{A}_{\max}\mathbf{n}_1 t^2\,, \tag{B.36}$$

$$\dot{q}(t) = \dot{q}(0) - \mathrm{A}_{\max}\mathbf{n}_1 t\,. \tag{B.37}$$

The instantaneous time to collision for the closed loop system is

$$t_{c_1} = -\frac{n_1^T\left(q(t) - p_1\right)}{n_1^T \dot{q}(t)}\,.$$

Likewise, the time to collision with the second face is

$$t_{c_2} = -\frac{n_2^T\left(q(t) - p_2\right)}{n_2^T \dot{q}(t)}\,,$$

where the second face is determined by checking the component of velocity orthogonal to the first face with respect to collision with other faces. Equating t_{c_1} and t_{c_2}, solve for the time at which the velocity is oriented toward the intersection of two faces, which we denote t_2. Let d_{c_2} denote the orthogonal distance to intersection of faces 1 and 2. With the redefined collision normal n_c, define the secondary collision ratio as

$$\zeta_2 = \frac{\left(n_c^T \dot{q}(t_2)\right)^2}{2\mathrm{A}_{\max} d_{c_2}},.$$

For higher dimension systems, continue these calculations beginning at t_2, and solving for the closed-loop response given the new collision normal. The iterations continue until the intersections of additional faces results in a single vertex point. If at iteration i, the calculated value for ζ_i is greater than one, collision is inevitable and the iteration halts. Given the iterations, define the overall collision ratio ζ_c as

$$\zeta_c = \max_i \zeta_i\,.$$

Section 4.1.2 introduced the notion of the *savable set* as the domain of the Save control policy for a given cell. Given the definition of the overall collision ratio ζ_c, formally define the savable set, $\mathcal{S}$, as

$$\mathcal{S} = \mathscr{D}(\Phi_{S_\mathcal{P}}) := \{(q\,,\dot{q}) \mid q \in \mathcal{P},\, \zeta_c < 1\}\,.$$

The goal set of the Save control policy, $\mathscr{G}(\Phi_{S_\mathcal{P}})$, is any rest condition within the cell, or more formally

$$\mathscr{G}(\Phi_{S_\mathcal{P}}) = \{(q\,,\dot{q}) \mid q \in \mathcal{P}\,,\, \|\dot{q}\| = 0\}\ .$$

The savable set is positive invariant under the Save control policy because Save does not increase the collision ratio, ζ_c, for any state in the savable set. Because the system is always applying negative acceleration relative to the current velocity, the system comes to rest in finite time.

Lemma B.3.2 [Lemma 4.1.3 in Section 4.1.2] *For a given convex polytope and initial velocity such that $\zeta_c < 1$, the Save control policy never increases ζ_c. Therefore, ζ_c remains less than one, collision is avoided, and the system remains in the savable set $\mathcal{S} = \mathscr{D}(\Phi_{S_\mathcal{P}}) := \{(q\,,\dot{q}) \mid q \in \mathcal{P},\, \zeta_c < 1\}$ and eventually comes to rest.*

Proof: The collision avoidance ratio ζ_c is defined by the worst case. If $\zeta_c = \zeta_1 < 1$, the calculations associated with (B.35) show that the Save policy will decrease ζ_i so that ζ_c remains less than 1. If $\zeta_c = \zeta_i < 1$ for some $i > 1$, the calculations are repeated for the defined collision normal beginning at time t_i. Again, ζ_c is shown to decrease; therefore, the system remains in the savable set. The system will eventually come to rest, thereby avoiding collision.

□

B.3.2 Align Control Policy

The Align control policy applies maximum acceleration to the system in order to quickly bring the velocity into the domain of the Track control policy whenever collision with the cell boundaries is not imminent.

The Align control policy continuously transitions from the Save control policy to a condition where maximum acceleration is applied along the velocity error vector, which decelerates the system and turns the velocity toward the desired velocity vector, $X(q)$. The domain of the Align control policy, given the collision ratio defined above, is

$$\mathscr{D}(\Phi_A) = \{(q\,,\,\dot{q}) \mid q \in \mathcal{P}\,,\, \zeta_c < 1\}\ .$$

Let

$$\upsilon = \max\left(0, \frac{\mu - \zeta_c}{\mu}\right)\ ,$$

where $\mu \in (0,1)$ is a user defined parameter defining the collision avoidance margin, and define the Align control policy as

$$\Phi_A\,:\,u = \begin{cases} \mathrm{A_{max}}\,\frac{(1-\sigma(\upsilon))\,\Phi_S + \sigma(\upsilon)\,\hat{e}}{\|(1-\sigma(\upsilon))\,\Phi_S + \sigma(\upsilon)\,\hat{e}\|} & \dot{q}^T X \le \dot{q}^T\dot{q} \\ \mathrm{A_{max}}\,\frac{(1-\sigma(\upsilon))\,\Phi_S - \sigma(\upsilon)\,\hat{\dot{q}}}{\left\|(1-\sigma(\upsilon))\,\Phi_S - \sigma(\upsilon)\,\hat{\dot{q}}\right\|} & \text{otherwise} \end{cases}\ , \tag{B.38}$$

where $\hat{e} = \frac{X(q)-\dot{q}}{\|X(q)-\dot{q}\|}$, $\hat{\dot{q}} = \frac{\dot{q}}{\|\dot{q}\|}$, and $\sigma : \upsilon \to [0,1]$ is a transition function with $\sigma(0) = 0$ and $\sigma(1) = 1$. Demonstrations in Chapter 4 use $\sigma(\upsilon) = \sqrt{\upsilon}$.

The Align control policy guarantees that the system remains in its domain, as the policy transitions to the Save control policy action when $\zeta_c \ge \mu$. Recall that under the Save control policy, the collision ratio ζ_c is guaranteed to not increase. Therefore, the system will not exit the $\zeta_c \le \mu$ domain, once the threat of imminent collision is over. When $\zeta_c \ge \mu$, the action of the Align policy is "saving", when $\zeta_c < \mu$ the action is "aligning".

Because the domain, $\mathscr{D}(\Phi_A) \subset \mathcal{S}$, the worst the Align control policy will do is bring the system to rest. That is, the goal set of the Align control policy, $\mathscr{G}(\Phi_A)$, is

$$\mathscr{G}(\Phi_A) = \{(q\,, \dot{q}) \mid q \in \mathcal{P}\,,\ \|\dot{q}\| = 0\}\ ,$$

which prepares the Track control policy. In the normal case, the Align control policy brings the system velocity orientation towards the desired velocity orientation, while at the same time reducing the speed of the system. If acceleration along the unit error vector would tend to increase the velocity, that is when $\dot{q}^T X > \dot{q}^T \dot{q}$, the system switches to accelerate against the current velocity. In all regions, the Align control policy decreases the system speed, and brings the system to rest in finite time.

B.3.3 Track Control Policy

The Track control policy brings the system velocity into alignment with the vector field $X(q)$ by using maximum available acceleration and transitioning continuously to the velocity reference control law. The domain of the Track control policy is

$$\mathscr{D}(\Phi_T) = \left\{(q\,,\,\dot{q}) \mid \dot{q}^T X > 0\,,\ \|\dot{q}\| \le \|X(q)\|\right\}\ .$$

Although the Track control policy works for convergent policies, this description will focus on flow-through style policies. For flow-through policies, the Track control policy must guarantee that the system trajectory does not exit the cell other than by the outlet zone. The goal of the Track control policy is

$$\mathscr{G}(\Phi_T) = \{(q\,,\,\dot{q}) \mid q \in \partial\mathcal{P}_{\text{outlet}}\,, \|\dot{q}\| \le \|X(q)\|\}\ ,$$

i.e. the system exits via the outlet zone with speed no faster than the desired speed.

To accomplish this goal, the Track control policy monotonically decreases the orientation error between the current velocity and the desired velocity. The approach uses some of the available acceleration to keep the orientation error constant as the trajectory evolves, and uses the remainder of the available acceleration to decrease the error.

The vector field derivative, $\dot{X} = D_q X\,\dot{q}$, defines the amount the desired velocity, $X(q)$, changes as the system moves by $\dot{q}$. Let dQ be the acceleration vector applied to the system such that the change in orientation error is zero. Essentially, dQ, shown in Figure B.5, is a scaled version of $\dot{X}$ that has been rotated by the orientation error.

Consider the plane defined by the current velocity, $\dot{q}$, and the desired velocity, $X(q)$, which we term the *velocity plane*. Decompose the vector field derivative vector into three components: the component along the desired velocity, the amount orthogonal to the desired velocity in the velocity plane, and the remainder. The component along the desired velocity is the differential speed change. The second component encodes how the desired velocity vector differentially rotates in the velocity plane. The remainder encodes how the velocity plane differentially rotates in space. If the system is accelerated such that the current velocity differentially rotates in the velocity plane the same as the desired velocity, and rotates with the velocity plane, then the change in the orientation error will be zero. Define the following unit vectors: $\hat{\dot{q}}$, $\hat{M}$, $\hat{N}$, and $\hat{P}$, where $\hat{\dot{q}}$ is the unit vector along the current velocity, $\hat{M}$ is the orthogonal to the desired velocity in the direction given by the error vector $\mathbf{e} = X(q) - \dot{q}$, $\hat{N}$ is the unit vector orthogonal to the current velocity in the direction of $\hat{M}$, and $\hat{P}$ is the unit vector orthogonal to the velocity plane. These vectors are shown in Figure B.5. Note, that $\hat{M}$ and $\hat{N}$ are both in the velocity plane. If the current and desired velocities are aligned, define $\hat{M} = \hat{N} = \mathbf{0}$.

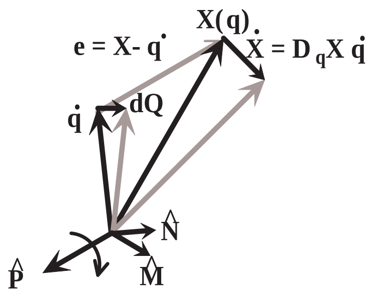


Figure B.5: Velocity vector relationships for the Track control policy.

Let $x = \hat{X}^T \dot{X}$ and $m = \hat{M}^T \dot{X}$, and define

$$P = \dot{X} - x\,\hat{X} - m\,\hat{M}\,,$$

where $\hat{X}$ is the unit vector along the desired velocity (recall $X(q) = s(q)\,\hat{X}(q)$). The vector P, orthogonal to both $\hat{X}$ and $\hat{M}$, defines how the desired velocity vector rotates out of the velocity plane. The scalar x defines the differential speed change, and the scalar m defines how the desired velocity vector rotates in the velocity plane. Considering the different magnitudes of $\dot{q}$ and $X(q)$, define

$$dQ = \frac{\|\dot{q}\|}{\|X(q)\|}\left(x\,\dot{\hat{q}} + m\,\hat{N} + P\right)\,.$$

This is equivalent to scaling $\dot{X}$, and rotating in the velocity plane by the orientation error. Given the desired velocity scaling, $s(q)$, from (B.31), $\|dQ\| \leq \mathrm{A}_{\max}$ because $\|\dot{q}\| \leq \|X(q)\|$ in $\mathscr{D}(\Phi_T)$, and $\left\|\dot{X}\right\| \leq \mathrm{A}_{\max}$. Letting $u = dQ$ will hold the orientation error constant, while allowing the speed to change proportionally. In general, because $\|dQ\| \leq \mathrm{A}_{\max}$, there will be some acceleration capacity left over to decrease the orientation error. The remainder of this section presents a strategy for efficiently using the remaining capacity.

Consider the control law $u = dQ + K^*\,(X(q) - \dot{q})$, where K^* is calculated to use the remaining acceleration capacity. The speed will never exceed the desired speed under this control, because the only component along the current velocity vector is directly proportional to the speed change of the reference vector field when $\|\dot{q}\| = \|X\|$ and the component along the error will tend to decrease speed. Thus, this control will decrease the orientation error, or at worst keep the error constant.

The available control can, however, be used more efficiently. Consider, if $m < 0$ the vector field is changing in a way that is already decreasing the orientation error. Also, if the speed change given by x is positive, then this component can safely be ignored; assuming that alignment is preferred over speed matching. Redefine dQ such that

$$dQ = \frac{\|\dot{q}\|}{\|X(q)\|}\left(\min(0,x)\;\dot{\hat{q}} + \max(0,m)\;\hat{N} + P\right)\,,. \tag{B.39}$$

and preferentially use the available acceleration for steering, then use any remaining acceleration for speed regulation. The Track control policy is defined as

$$\Phi_T \;:\; u = dQ + s\,\hat{N} + a\,\dot{\hat{q}}\,, \tag{B.40}$$

where

$$
\begin{aligned}
s &= \min\left(K\,\hat{N}^T\mathbf{e}\,,\ \sqrt{\mathrm{A_{max}}^2 - \frac{\dot{q}^T\dot{q}}{X^TX}\left(\mathrm{P}^T\mathrm{P} + \min\left(0,x\right)^2\right)} - \frac{\|\dot{q}\|}{\|X\|}\max\left(0,m\right)\right) \text{ and} \\
a &= \min\left(K\dot{\hat{q}}^T\mathbf{e},\ \sqrt{\mathrm{A_{max}}^2 - \frac{\dot{q}^T\dot{q}}{X^TX}\mathrm{P}^T\mathrm{P} - \left(\frac{\|\dot{q}\|}{\|X\|}\max\left(0,m\right) + s\right)^2} - \frac{\|\dot{q}\|}{\|X\|}\min\left(0,x\right)\right) .
\end{aligned}
$$

The s term is used to decrease the orientation error proportional to error, but limited by the available acceleration; the a term is used to increase the speed using a portion of the remaining acceleration.

In the limit, as the velocity error approaches zero, the Track control policy is identical to the velocity reference control policy given in (B.18). The vector field $X(q)$ is defined as in (B.31); thus, Φ_T is able to follow the integral curves of $X(q)$ without violating the constraints. Given this definition of the track control policy, the value of K only needs to be greater than zero for proper convergence, and not "sufficiently large".

Lemma B.3.3 *Under the influence of the Track control policy, the system (B.28), with constraints given in (B.29) and (B.30), and initial condition $\{q,\dot{q}\} \in \mathscr{D}(\Phi_{T_\mathcal{P}})$, converges to the integral curves of $X(q)$, defined in (B.31), in a way such that $\|\dot{q}\|$ remains less than $\|X\|$ and the trajectory never exits the cell except by the outlet zone. For flow-through vector fields, the system trajectory exits the cell in finite time. For convergent vector fields, the system converges to an arbitrarily small neighborhood of the goal in finite time.*

Proof: The policy is designed so that the orientation error monotonically decreases, and the speed never exceeds the desired speed.

Therefore, the proof directly follows that of Lemma B.2.4.

□

Appendix C

Test for Collision Free Cells

The policy design approach taken in this thesis defines conditionally invariant policies over cells in pose space. The cell defines the policy domain with respect to pose space. Given a conditionally invariant policy, the policy is *safe* **if** and only if all poses within its associated cell are collision free. In other words, since the system cannot leave the domain except via a goal set by definition of conditional invariance, the system can only collide with an obstacle if the cell intersects the boundary of the free pose space [1]. If the cell is completely contained in the free pose space, then the system must depart the cell in order to collide with an obstacle, and thus violate the assumption of conditional invariance. Therefore, we can guarantee that the hybrid control policy is safe if each cell used to define a policy within the hybrid control framework is safe. This appendix presents a novel approach to determining whether a given cell is fully contained within the free pose space, without explicitly constructing the free pose space.

Before presenting our approach, this section provides a brief overview of two alternate approaches that have been used by the motion planning community. This provides an introduction to the concept, and The second section provides the mathematical basis for our technique. A tractable testing procedure is developed in the third section. The chapter concludes with a discussion of the testing process.

C.1 Alternate Approaches

As with the rest of this thesis, this discussion assumes a single bodied robot moving in a bounded planar workspace populated with a finite number of obstacles. Many of the early path planning techniques assume a point robot moving through its environment[22, 75]. If a non-point robot has fixed orientation or is bounded by a circle, then the process of constructing the free pose space can be viewed as "expanding" the workspace obstacles to account for the finite robot size, which allows the robot to be treated as a point for planning purposes [22, 87]. This approach is shown in Figure C.1.

The process of "expanding" the obstacles is based on the planar Minkowski difference [22, 87]. Let $R \subset \mathbb{R}^2$ denote the set of points occupied by the robot relative to its reference point, and let $O \subset \mathcal{W}$ denote the set of points occupied by a particular obstacle in workspace. Define the expanded obstacle as

$$O_i \ominus R = \{p \in \mathcal{W} \mid \exists_{a \in O_i, b \in R}\, p = a - b\} \ ,$$

[1]For flow-through policies, the implicit requirement is that the goal set be contained in the domain of another safe policy

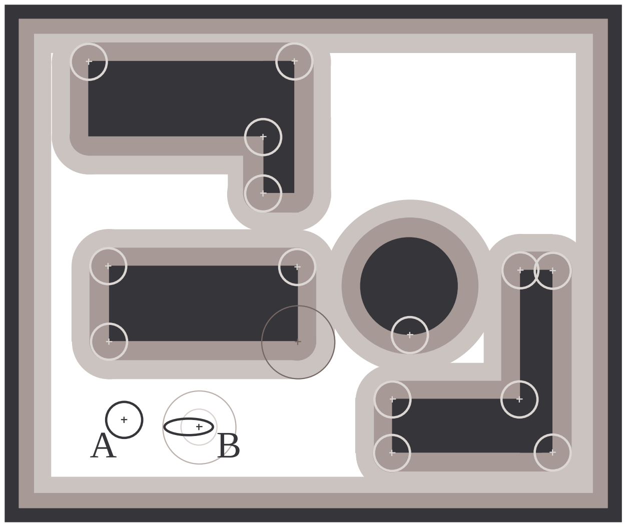


Figure C.1: The figure shows a bounded workspace with five black obstacles, including the workspace boundary. Two robots, labeled 'A' and 'B' are shown in the lower left; robot 'A' is circular, robot 'B' is shown by the black ellipse. The reference points are marked with small pluses. The dark gray regions shows how the obstacles are expanded to account for the size of robot 'A'. Robot 'A' can plan a path through the remaining space as if it was a point. A conservative approach would bound robot 'B' with the large circle centered at the reference point as shown, and then expand the obstacles as shown by the light gray regions. This results in a disconnected workspace in this example. As robot 'B' is actually narrower than robot 'A', and can pass between the obstacles at specific orientations, this conservative approach is not complete.

where $O_i \ominus R$ denotes the Minkowski difference between sets O_i and R. For circular robots, or objects with fixed orientation, this transformation to an expanded obstacle representation is exact, and point-based path planning approaches are complete. There are algorithms for constructing the boundary of the expanded planar obstacles for circles and polygons. For robots with non-circular shape and variable orientation, one approach is to expand the obstacles based on the minimum bounding circle whose center is the reference point attached to the robot. While this approach guarantees safety, it is overly conservative as illustrated in Figure C.1.

Another approach is to map the workspace obstacles to obstacles in the pose space. Conceptually, the planar workspace obstacles are expanded based on the robot body at a given orientation [22, 75]. These planar sets can be "stacked up" by considering each orientation as a slice of the body pose space; Figure C.2 shows an example of this approach. Note that this "stacking process" results in a mapping from the planar workspace to the $\mathbb{R}^3$ representation of the pose space. Even for simple shapes like polygonal robots and obstacles, the representation of the pose space obstacle boundary becomes a collection of curved surface patches in $\mathbb{R}^3$. Although there exist algorithms to construct these representations for obstacles and robots defined as semi-algebraic sets, the resulting representation of $\mathcal{G}_{\text{free}}$ is quite complex. Many modern planning techniques, such as probabilistic roadmaps and RRTs, use probabilistic techniques and collision testing to explicitly avoid constructing the free configurations space [78, 75].

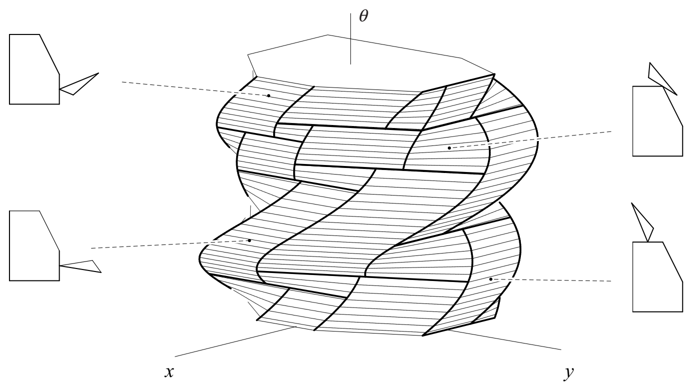


Figure C.2: The pose space obstacle for a triangular robot and five-sided obstacle in workspace. Notice how each vertex/facet combination becomes a curved surface in pose space. (Figure courtesy Howie Choset [22].)

C.2 Calculation of Expanded Cell

For the hybrid control technique used in this thesis, we must verify the safety of a given policy over its entire domain. We prefer guarantees over probabilistic approaches, but initially it appears difficult to test that a given cell, of arbitrary shape, is fully contained in the free pose space. This is because the test would apparently require constructing the pose space obstacles, and then testing for intersection between the surface patches that define the cell boundary and the curved pose obstacle boundary surfaces in three-dimensions. Our approach avoids these difficulties by inverting the problem.

Our approach expands the cell, which is used to define control policy domains, and not the obstacle; this approach allows simple intersection tests to be performed in the workspace. This section provides an overview of the approach, and then derives an exact analytic mapping from a point on the cell boundary to the corresponding point on the expanded cell boundary. The discussion begins with some mathematical preliminaries, then the general mapping is defined. A specific instance of this mapping is demonstrated for an elliptical robot body. Later sections use the general mapping to define a test that lends itself to simple calculations.

To motivate our approach, consider the two-dimensional iconic example shown in Figure C.3. In this example, the robot has fixed orientation, and the cell is represented by the two-dimensional funnel. As the robot body is placed at different positions within the cell, the robot occupies different regions of the workspace. Given a cell, robot body, and collection of obstacles, our approach verifies that all possible poses within the cell are collision free.

Recall from Chapter 5 that our control policies are defined over cells in the robot's pose space. The cells define the policy domain in the pose space; that is, the set of all poses $g \in \Xi_i$ define the poses for which a given policy is valid. The mapping $R(g)$ defines the set of points in workspace that a robot body occupies at a given pose; $R(g)$ is a function of both position and orientation of the robot body. For a given cell Ξ_i, let

$$R(\Xi_i) = \bigcup_{g \in \Xi_i} R(g) \; . \tag{C.1}$$

$R(\Xi_i)$ is the swept volume of $R(g)$ over all $g \in \Xi_i$; that is, $R(\Xi_i)$ is the set of all possible points in workspace occupied by the robot for any possible pose within the cell.

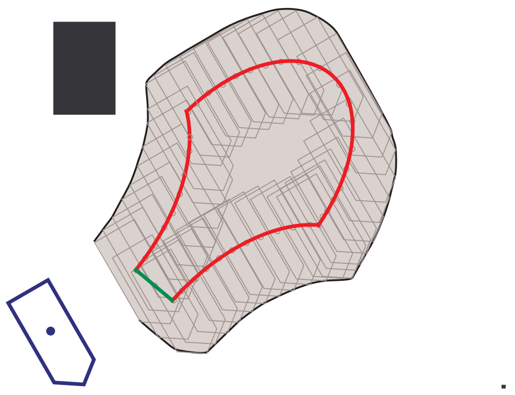


Figure C.3: Consider the iconic funnel used to represent the cell that defines the policy domain, and the dark polygonal robot body shown in the lower left. The body reference point is indicated by the dot in the center. In this example the robot body is at fixed orientation. If the robot body is placed at any position within the cell, it occupies a certain region of the planar workspace. If the robot body is convolved with all positions within the cell, the set of points occupied by the robot extends beyond the boundary of the cell, as indicated by the light gray region.

Definition: A cell is *collision free*, that is contained in the free pose space, if and only if

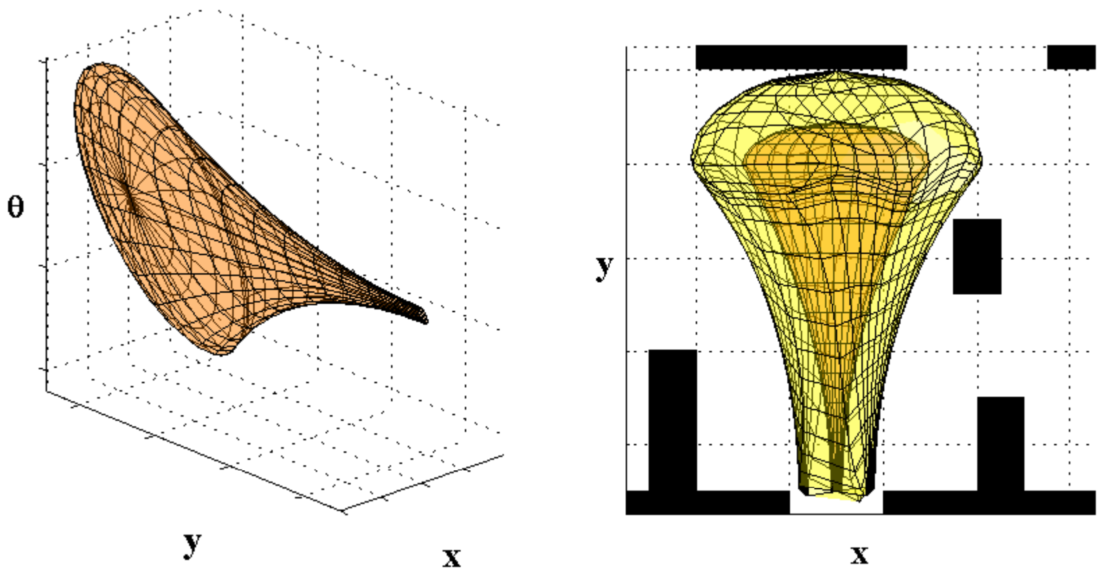
$$R\left(\Xi_i\right) \bigcap \left(\bigcup_k O_k \right) = \emptyset \,.$$

For collision to occur, an obstacle must intersect the boundary of $R\left(\Xi_i\right)$, denoted $\partial R\left(\Xi_i\right)$, or an obstacle must be completely contained in the interior of $R\left(\Xi_i\right)$. Thus, to test for collision, a mapping $R : \Xi_i \subset \mathcal{G} \to R\left(\Xi_i\right) \subset \mathcal{W}$ is needed; Figure C.4 illustrates the approach.

The expanded cell approach developed in this chapter finds tractable methods of approximating $R\left(\Xi_i\right)$, and then tests this approximate set for collision in the workspace. The chapter describes an approach that guarantees the approximation is safe, without being overly conservative.

For this approach, we first identify the local chart in $\mathbb{R}^3$ of the body pose space $\mathcal{G}$ that is used to define the cell Ξ_i, with $\mathbb{R}^3 = \mathcal{W} \times \mathbb{R}$, the planar workspace crossed with the real line. To be formally correct, we map the cell Ξ_i into this second copy of $\mathbb{R}^3$; in an abuse of notation, let Ξ_i denote the closure of the cell in $\mathbb{R}^3$ and let g denote the pose in this space[2]. We assume Ξ_i is a compact, connected, closed set, without holes; that is, it is homeomorphic to a ball in $\mathbb{R}^3$. The cell boundary is composed of piecewise differentiable surface patches, and has a well defined outward pointing normal almost everywhere. These conditions are true for the cells defined in this thesis. The obstacles are mapped from the workspace to $\mathcal{W} \times \{0\}$ in this same copy of $\mathbb{R}^3$.

[2]This "abuse" is recognition that the pose space $\mathcal{G}$ and the $\mathbb{R}^3$ representation of $\mathcal{W} \times \mathbb{R}$ are not the same spaces, even though they are both embedded in $\mathbb{R}^3$.



a) Cell in robot body pose space

b) $R(\Xi_i)$ - Extent of robot body in workspace

Figure C.4: Given a cell defined in pose space (a), we seek a mapping to $R(\Xi_i)$ in workspace (b). In this example, the lighter surface mesh shown in (b) represents the set $R(\Xi_i)$ for the cell in (a).

The approach developed in this chapter expands the cell $\Xi_i \subset \mathrm{I\!R}^3$ to account for the extent of the robot body, and then projects the expanded cell to the xy-plane given by $\mathcal{W} \times \{0\}$. The projection yields $R(\Xi_i)$, or more precisely its equivalent representation in this copy of $\mathrm{I\!R}^3$. Figure C.5 shows the expanded cell representation for the mapping in Figure C.4. The expansion, projection, 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*. of Ξ_i and $R(g)$. Where the Minkowski difference is used expand obstacles in order to indicate how close the body can approach an obstacle, and the Minkowski sum is used to indicate how far past the boundary of one set another extends. In this case, however, $R(g)$ is a two-dimensional set in $\mathrm{I\!R}^2$ and both Ξ_i and the expanded cell live in $\mathrm{I\!R}^3$. We extend the basic Minkowski sum definition based on calculations performed on a planar slice of the cell. The slice is taken at a given orientation θ (Figure C.6-a); the Minkowski sum is calculated for the cell restricted to the slice and the robot body at the same orientation. We will abuse notation and used the Minkowski sum symbol to denote our expanded cell as $\Xi_i \oplus R$. Formally, the expanded cell is given by

$$\Xi_i \oplus R = \left\{ \begin{bmatrix} x \\ y \\ \theta \end{bmatrix} + \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix} p \mid \{x, y, \theta\} \in \Xi_i \text{ and } p \in R(\{0,0,\theta\}) \subset \mathrm{I\!R}^2 \right\}, \qquad \text{(C.2)}$$

where $R(\{0,0,\theta\})$ represents the set of robot body points at the origin rotated by θ. This extended definition maps the two-dimensional point p into the three-dimensional space. This continuous mapping is equivalent to taking the standard Minkowski sum of a planar slice of Ξ_i at constant orientation with the robot at the same orientation, and then "stacking the slices" (see Figure C.6).

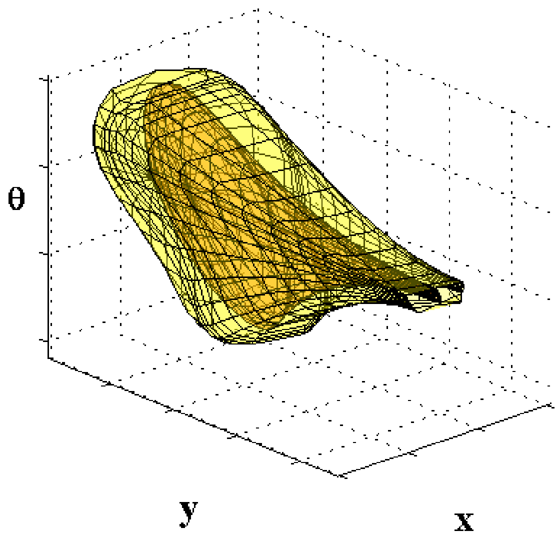


Figure C.5: The cell boundary (dark inner surface) is expanded to account for the robot body shape, and projected to yield $R(\Xi_i)$. Note, the goal set is not shown, and appears as an open face. We assume that the goal set will be contained within the domain of another policy, so expanding this set is unnecessary.

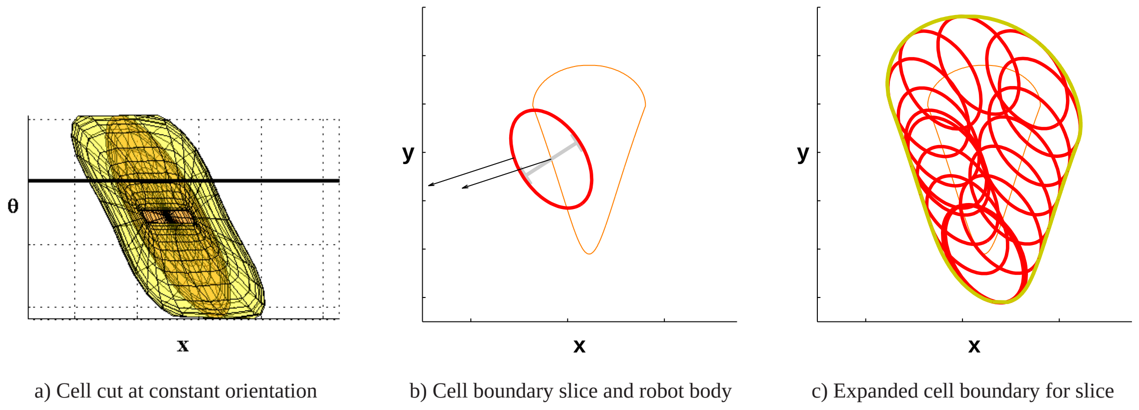

θ
x
y
x
y
x
a) Cell cut at constant orientation
b) Cell boundary slice and robot body
c) Expanded cell boundary for slice

Figure C.6: The calculation of the extended surface is based on a planar Minkowski sum. a) Consider a slice of 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 expanded surface is found by matching normals for the planar cell boundary and the robot body. c) The boundary of the extended cell is calculated.

In another abuse of notation, let $R(\Xi_i) = \pi_{xy}(\Xi_i \oplus R) \subset \mathcal{W} \times \{0\} \subset \mathbb{R}^3$, where π_{xy} is the trivial projection $\pi_{xy}(x, y, \theta) = (x, y, 0)$. This definition is identified with the definition given in $R(\Xi_i)$.

While this set-based calculation of $R(\Xi_i)$ is correct, it is impractical for actual testing because it depends on an infinite number of points in both $R(g)$ and Ξ_i. As a step toward reducing the complexity, consider the boundary of the expanded cell. For planar 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 [100, 101, 109]. That is $\partial(A \oplus B) \subset \partial A * \partial B$, where

$$\partial A * \partial B = \{a + b \mid a \in \partial A, b \in \partial B, \text{ s.t. } n_{\partial A}(a) \parallel n_{\partial B}(b)\} \tag{C.3}$$

with $n_{\partial A}(a)$ the boundary normal to A at a and $n_{\partial B}(b)$ the boundary normal to B at b, and $\parallel$ denotes the vectors are parallel[3]. This result says that any boundary point of the Minkowski sum of two sets will be the sum of two points taken from boundaries of the two sets. Furthermore, for a given point $a \in \partial A$, the choice of point $b \in \partial B$ is constrained to those points in ∂B such that the boundary normal at b is parallel the boundary normal at a. Note that the converse is not true; the sum of some boundary points of the two sets will be on the interior of $A \oplus B$ and not on the boundary.

We use this result to define a mapping from the boundary of the cell Ξ_i to a surface that covers the boundary of the expanded cell $\Xi_i \oplus R$. Recall how the planar Minkowski sum was used to define the mapping in (C.2) between a two-dimensional set and three-dimensional set; we define an analogous mapping for the convolution-like surface between the two-dimensional robot set and the three-dimensional cell. Given a robot pose on the cell boundary, $g = \{x, y, \theta\} \in \partial\Xi_i \subset \mathbb{R}^3$, 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 a plane parallel 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. Given $g \in \partial\Xi_i$ which defines a position (x, y) and orientation θ of the robot body, we find the points $p \in \partial R(0, 0, \theta)$ such that their normal $n_R(p)$ is parallel to the projection of the cell boundary normal $\pi_{xy} n(g)$ (see Figure C.6-b). Define the convolution-like surface $\partial\Xi_i * \partial R$, which contains the boundary of $\Xi_i \oplus R \subset \mathbb{R}^3$, as

$$\partial\Xi_i * \partial R = \left\{ \begin{bmatrix} x \\ y \\ \theta \end{bmatrix} + \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix} p \,\middle|\, \{x, y, \theta\} \in \partial\Xi_i \,, p \in \partial R(\{0, 0, \theta\}) \,, \text{s.t.}\, \pi_{xy} n(g) \parallel n_R(p) \right\} \tag{C.4}$$

The mapping is continuous except where the cell surface normal projection is not well defined. Piecewise differentiable cell surface patches map to piecewise differentiable expanded cell patches almost everywhere.

Example: Elliptical Robot Body

Consider the elliptical robot body shown in Figure C.6-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 a 2×2 positive definite matrix that encodes the elliptical shape.

We use the extended convolution operator (C.4) to find the point on the expanded cell boundary given a point on the cell boundary. Let $g = \{x, y, \theta\} \in \partial\Xi_i \subset \mathbb{R}^3$ represent the pose of the body reference point on the cell boundary, and let the cell surface normal $n(g)$ be given. Define the projected unit vector $\hat{n}_\pi(g) = \frac{\pi_{xy} n(g)}{\|\pi_{xy} n(g)\|}$. The as yet unknown point on the expanded cell that corresponds to g is $p = (x_p, y_p, \theta) \in \partial X i_i * \partial R \subset \mathbb{R}^3$. Let $v_p = \pi_{xy}(p - g)$ be the vector in the workspace frame from g to p, where p is a point on the robot body at g. The corresponding point p_{rb} on the

[3]If both sets, A and B, are convex, then $\partial(A \oplus B) = \partial A * \partial B$. Unfortunately, the planar slices of the cells are not convex in general.

robot body, in the body reference frame, is given by $p_{rb} = \mathrm{Rot}(\theta)^T\, v_p$ with $\mathrm{Rot}(\theta)$ the 2×2 rotation matrix. For points p on the boundary of the robot at g, rewrite the body function as

$$f\,(p) = v_p^T \mathrm{Rot}(\theta)\ M\, \mathrm{Rot}(\theta)^T\, v_p - 1\,, \tag{C.5}$$

where $v_p = \pi_{xy}\,(p - g)$ as before. 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 (C.5), 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.

For elliptical body representations, (C.4) gives a one-to-one mapping from the cell boundary to a point on the expanded cell boundary.

The mapping $(\partial\Xi_i * \partial R)\,(g)$ determines a point-by-point mapping for any point on the cell boundary to a set of points either on the boundary of $\Xi_i \oplus R$ or, on its interior. That is, $\partial\Xi_i * \partial R : \partial\Xi_i \rightarrow \partial\Xi_i * \partial R$. For more general body representations, such as polygons, (C.4) may result in a one-to-many mapping at certain points. For example, if the projected cell normal matches a body polygon edge normal, the mapping would give the line segment defining the expanded surface, and not a point. For the moment, we will assume the surfaces provide a one-to-one continuous mapping; that is only one point on $\partial R\,(0, 0, \theta)$ matches, and the surface normal is continuous over the boundary of Ξ_i. Analysis of the surface continuity and one-to-many point ideas are explored later in this chapter. While the convolution surface may define some points on the interior of the expanded cell, the convolution surface will contain the expanded cell boundary surface given a continuous surface normal over $\partial\Xi_i$.

The set of workspace points, $R(\Xi_i)$, occupied by the robot over the cell is found by projecting the convolution surface to $\mathcal{W} \times \{0\}$, which is identified with $\mathcal{W}$; that is $R(\Xi_i) = \pi_{xy}(\partial\Xi_i * \partial R)$.

Lemma C.2.1

$$\begin{aligned} R(\Xi_i) &= \pi_{xy}(\Xi_i \oplus R) \\ &= \pi_{xy}(\partial(\Xi_i \oplus R)) \\ &= \pi_{xy}(\partial\Xi_i * \partial R) \end{aligned} \tag{C.6}$$

Proof: The first line, $R(\Xi_i) = \pi_{xy}(\Xi_i \oplus R)$, is true by definition when we identify $\mathcal{W} \times \{0\}$ and $\mathcal{W}$. Functionally, $R(\Xi_i)$ is the union of the projections of each slice in $\Xi_i \oplus R$, where each slice is the Minkowski sum of the robot body an all points in the cell at that orientation.

Since Ξ_i is a closed set, $\pi_{xy}(\partial(\Xi_i \oplus R)) \subseteq R(\Xi_i)$.

To see that $R(\Xi_i) \subseteq \pi_{xy}(\partial(\Xi_i \oplus R))$, and hence $R(\Xi_i) = \pi_{xy}(\partial(\Xi_i \oplus R))$, consider passing a line from $\theta = \pm\infty$ through and orthogonal to $\mathcal{W} \times \{0\}$. For any point on the interior of $(\Xi_i \oplus R)$, the line passes through two boundary points; both boundary points and the associated interior point project to the same point in $\mathcal{W} \times \{0\}$. Thus, we conclude that

$$\pi_{xy}(\partial(\Xi_i \oplus R)) = \pi_{xy}(\Xi_i \oplus R) = R(\Xi_i) \ .$$

The convolution surface contains the expanded cell boundary, that is $\partial(\Xi_i \oplus R) \subseteq \partial\Xi_i * \partial R$, thus $\pi_{xy}(\partial(\Xi_i \oplus R)) \subseteq \pi_{xy}(\partial\Xi_i * \partial R)$ [100, 101, 109].

Any points in $\partial\Xi_i * \partial R$ not contained in $\partial(\Xi_i \oplus R)$ are in the interior of $\Xi_i \oplus R$ by definition of the Minkowski sum; interior points project to $\pi_{xy}(\partial(\Xi_i \oplus R))$ from the proof of the second line. Thus, $\pi_{xy}(\partial\Xi_i * \partial R) \subseteq \pi_{xy}(\partial(\Xi_i \oplus R))$ and we conclude

$$\pi_{xy}(\partial\Xi_i * \partial R) = \pi_{xy}(\partial(\Xi_i \oplus R)) \ .$$

Thus, we conclude that

$$\pi_{xy}(\partial\Xi_i * \partial R) = \pi_{xy}(\partial(\Xi_i \oplus R)) = \pi_{xy}(\Xi_i \oplus R) = R(\Xi_i) \ .$$

$\square$

By projecting $(\partial\Xi_i * \partial R)$ into $\mathcal{W} \times \{0\}$, we determine $R(\Xi_i) = \pi_{xy}(\partial\Xi_i * \partial R)$, and hence, the maximal extent of the robot body for any pose in the cell. There are several problems with this definition and is application to cells with discontinuous surface normals. First, the mapping in (C.4) does not guarantee a continuous cover of $\partial(\Xi_i \oplus R)$. For each parameterized differentiable surface patch on the cell boundary, $\partial\Xi_i * \partial R$ defines an exact parametric representation of a patch on the cell/robot convolution surface. The surface patches are not necessarily continuous if the normals along the patch boundaries are not continuous. Second, this test still requires an infinity of points to be tested. The next section addresses these issues, and uses (C.4) to generate a tractable collision test.

C.3 Collision Testing Using Expanded Cells

This section presents a tractable approach to testing for collision based on a discrete sampling of the cell surface and the mapping defined in (C.4). This section discusses the calculations, and methods to address the continuity issues introduced above.

C.3.1 Mesh Definition

The collision tests are a combination of exact and approximate tests. We are given a mesh representation of the parameterized surface patches that define the cell boundary. That is, we are given a finite collection of sample points spread around the surface, and a collection of edges that connect adjacent points to form a surface mesh representation. Triangulated surface mesh representations are commonly used. This thesis does not address the "best" way to construct a mesh representation for a given cell, or techniques for adaptively refining the mesh. We assume that each facet in the initial mesh is contained within a single differentiable surface patch.

Given the collection of mesh vertices on the cell boundary surface, the vertices are mapped to a point on the convolution surface of the expanded cell. This mapping is exact and analytic. Using the same mesh connections as the cell surface representation gives a mesh representation of the convolution surface patches of the expanded cell.

There are two problems with this approach: disconnected facets and facet expansion. Disconnected facets require that the mesh be "stitched together". Facet expansion requires that the mesh be refined to meet resolution requirements. Our simple techniques for refining the mesh as necessary, and stitching the disconnected patches together, are discussed below. For the moment, assume that these issues are addressed, and the expanded cell has a "continuous" surface mesh representation of sufficient resolution. The next sub-section presents the approach to testing for collision; afterwards, the following sub-section returns to discuss how these two issues are addressed.

There are guidelines for defining the initial cell mesh. Since the expanded cell is defined for slices of constant orientation, and the mapping $\partial\Xi_i \oplus \partial R$ does not change the orientation, the cell mesh should initially be sampled at a sufficiently fine resolution in orientation. In other words, there should not be relatively large gaps in orientation between any two pairs of vertices in the mesh. As a rule of thumb, consider the minimum bounding radius of the robot body boundary, and the desired sampling resolution in workspace based on obstacle size; the orientation sampling should be less than desired sampling resolution divided by bounding radius. Respecting this at the outset can reduce the burden on the refinement process. Second, discontinuous normals will lead to discontinuous maps; thus, it is prudent to finely sample along the boundaries of cell surface patches.

C.3.2 Collision Testing

The collision test is based on mapping the surface mesh of the convolution surface to the workspace to give an approximation of $R\left(\Xi_i\right)$. The expanded cell mesh vertices are projected to the workspace using π_{xy}; using the same mesh connections between vertices gives a collection of overlapping facets in workspace. These projected facets approximate $R\left(\Xi_i\right)$ since each vertex is contained in $R\left(\Xi_i\right)$ by Lemma C.2.1. Figure C.5-b shows the result of this projection.

Given the planar mesh approximation of $R\left(\Xi_i\right)$, each obstacle is tested for collision. First, the projected vertices of the expanded cell mesh are tested for inclusion within the collection of obstacles. This test is exact based on the point-wise analytic mapping; if a vertex is contained within an obstacle, then the cell is unsafe and must be modified. Assuming polygonal obstacles, or simple elliptical obstacles, these inclusion tests are trivial.

To guard against a small obstacle being contained within $R(\Xi_i)$, the obstacles are tested against the collection of facets in the projected expanded cell mesh. Assuming a polygonal representation of the obstacles, the vertices of each obstacle are tested for inclusion in any of the projected facets. This is an approximate test based on the projected facets. If any obstacle vertex is contained in the interior of any projected facet, the cell is assumed to be unsafe. This approximation may be conservative if the boundary of $R(\Xi_i)$ is highly curved, and the facets are relatively large. Obstacle 'A' in Figure C.7 shows an example where this test is overly conservative. Reducing the maxim facet diameter by increasing the mesh resolution will reduce these false positives.

If these two tests fail to find an intersection, then the cell is assumed to be safe. There are, however, at least two false negatives that must be guarded against. The test can fail in the presence of long thin obstacles and relatively large facets as shown by obstacle 'B' in Figure C.7; thus, the sampling resolution should be less than the minimum dimension over the set of obstacles. Another approach, is to add test points scattered over the obstacle interior in addition to the vertices; this approach also works for non-polygonal obstacles. A third approach to dealing with skinny obstacles is to test for line intersections between the obstacle boundaries and the projected facet edges; however, this increases the computational cost significantly.

Another failure, shown by obstacle 'C' in Figure C.7, occurs when the boundary of $R(\Xi_i)$ extends past the approximation. To avoid this failure, the obstacles should be padded by a distance based on the maximum error between the actual boundary of $R(\Xi_i)$ and the projected mesh approximation. Given the analytic mapping, and the piecewise differentiable surface patches, it is possible to bound the error, either through sample-based estimation or analytically for some mesh strategies [99]. These calculations depend on the mesh generation technique, and are beyond the scope of this thesis.

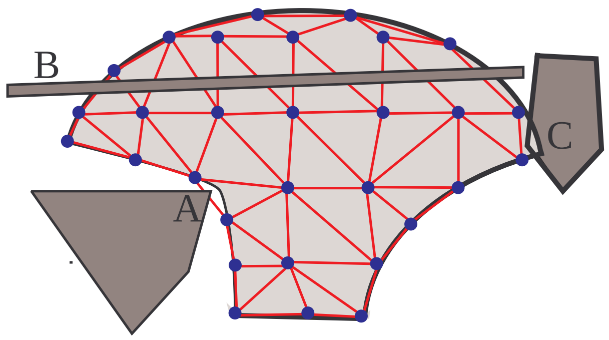


Figure C.7: Collision tests on the approximation of $R(\Xi_i)$ using overlapping facets. The light gray region denotes a representation of $R(\Xi_i)$. The grid and facets shows the approximation based on the projection of expanded cell surface mesh. The dark gray obstacle on the left labeled 'A' does not intersect $R(\Xi_i)$; however, the approach classifies the cell as unsafe based on the facet intersection near point 'A'. Obstacle 'B' intersects $R(\Xi_i)$, but the approximate tests misses the fact as no vertices in 'B' are contained in a facet, and no facet vertices are contained in 'B'. The obstacle labeled 'C' on the right does intersect $R(\Xi_i)$, but the approximation misses this collision.

C.3.3 Patch Stitching and Mesh Refinement

This mesh-based approach to testing cell safety depends on the projected expanded cell mesh accurately approximating $R(\Xi_i)$. Regardless of the accuracy of mesh approximation of the cell boundary surface, the true measure is how well the mesh mapping approximates $\partial\Xi_i * \partial R$. Thus, patch stitching and mesh refinement must be addressed.

While the differentiable surface patches on the cell are connected, they are not necessarily connected when mapped to the expanded cell. The most likely cause is discontinuous normals along the patch boundaries; in this case a single pose does not have a well defined normal. For a mesh vertex along patch boundaries, there may be multiple surface normals that apply the particular pose; in this case, the pose is duplicated with multiple vertices each associated with a particular surface patch, and therefore a particular surface normal.

One strategy for stitching the surface patches together is to define facets that tie these distinct vertices together. Along a common patch boundary, two adjacent poses from one patch are joined with a vertex that matches one of the poses but is assigned to the adjoining surface patch. Doing this for all common edges results in a line of degenerate triangles along the cell boundary, that likely map to non-degenerate triangular facets in the expanded cell surface. Figure C.8 shows a schematic example of this stitching process. One notes the obvious effect of facet expansion in this example.

To control the accuracy of the expanded cell approximation, and hence the accuracy of the $R(\Xi_i)$ approximation, the mesh representation of the expanded cell is post-processed and refined as necessary. As stated earlier, the vertices in the expanded cell mesh are projected to the workspace using π_{xy} to give a collection of overlapping facets that approximate $R(\Xi_i)$. Larger facets will tend to have larger error, so the length of facet edges is limited based on the desired workspace sampling resolution. Although there are approaches for adaptively adjusting a fixed number of vertices to create similar sized facets [99], this thesis implements a simple approach. Given the initial mesh, the refinement process iteratively adds vertices and splits large facets into multiple facets based on workspace measurements.

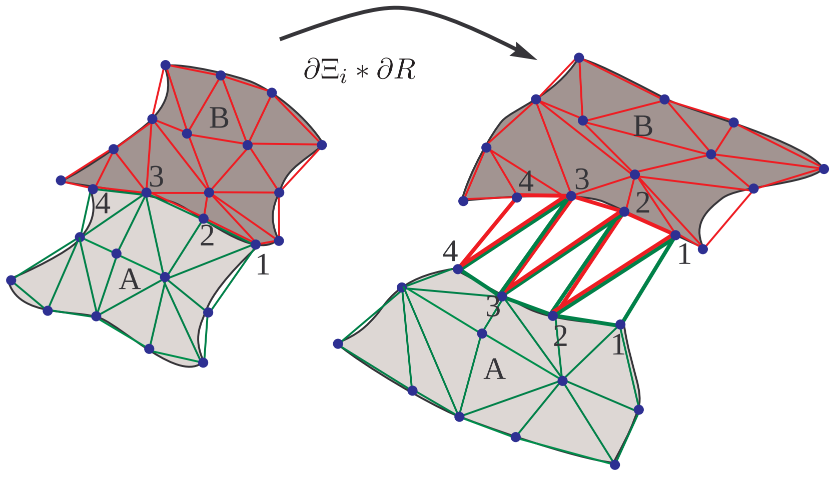


Figure C.8: Two continuous cell surface patches ‘A’ and ‘B’ are mapped to discontinuous patches on the expanded cell. By adding facets that contain vertices from both patches along the common boundary, the discontinuous patches can be stitched together.

The refinement process described here is based on systematically subdividing facets that are too large. The Euclidean distance between projected facet vertices is calculated; if the distance exceeds the defined sample resolution, then the facet is split. In the basic case, a new vertex is added along the long edge connecting the two vertices, and a vertex is added at the centroid of the two facets that share the edge. Each triangular facet is split into four triangles; the two facets that share the split edge are split into eight triangles. Normally, the sample points are added in the parameter space that defines the cell boundary. The cell surface patch associated with the new vertex is identified, and the appropriate normal is calculated. The new vertices are mapped to the expanded cell, and projected to the workspace. The basic refinement process continues until each edge in the projected mesh is less than the workspace sample resolution, or the parameter space separation is less than some minimal threshold. There are a few special cases to consider.

One set of special cases to consider is where a vertex is to be added to a facet that connects two different surface patches along the shared boundary between two cell surface patches. If the edge to be split has two vertices from one patch, then two new vertices are added, one for each patch and surface normal. The proper "stitching" facets are added to the list of facets. If the split is along an edge connecting two different surface patches, then a single vertex along the edge and a vertex at the centroid is added with interpolated normals. The normals are interpolated as illustrated by Figure C.9. In this case, the pose is the same; the only difference is in the interpolated normal. The decision to add a vertex is based on the distance between projected expanded cell vertices and the difference between interpolated normals, as the parameters space difference is zero. If the new vertex is added to an edge containing an existing interpolated vertex, the new vertex should interpolate its normal based on the interpolated normal of its neighbor. Thus, new vertices at the same pose are introduced to address the discontinuous normals, and stitch the surface patches together with an approximately continuous mapping.

Another special case relates to robot bodies defined by piecewise differentiable functions such as polygons. For elliptical robot bodies, the convolution surface mapping is one-to-one; there are only two points where the normals are parallel, and one is eliminated because it is always in the interior. For more general body shapes, the mapping may be many-to-one. Here we consider the case of polygonal body shapes; consider the case illustrated in Figure C.10. For most cases, assuming general position, a one-to-one mapping is preserved as the requirement of matching normals will choose a polygon vertex such that the cell normal is in the positive span of the adjacent edge normals. If the cell normal matches an edge normal on a polygon, the point on the cell boundary will map to

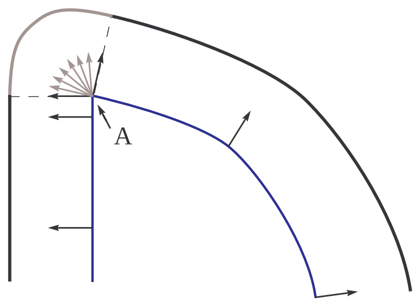


Figure C.9: A the junction of two cell boundary patches with discontinuous normal vectors, blend the normal vectors while holding pose constant to provide a continuous expanded cell boundary surface. Similar techniques are used with robot bodies defined by piecewise differentiable curves.

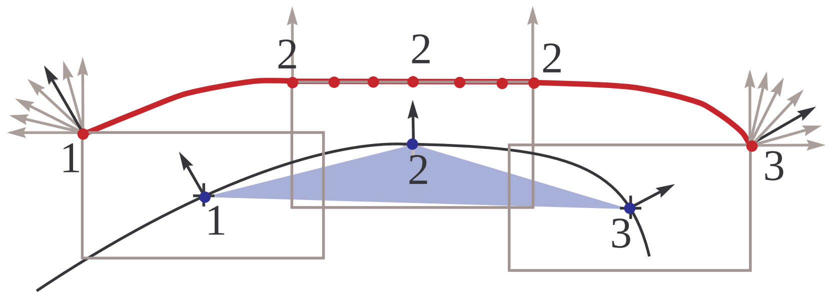


Figure C.10: With polygonal robot bodies, the expanded cell mapping experiences discontinuous jumps between body vertices. This figure shows an arc of the cell boundary and its mapping to the expanded cell for a rectangular robot body. In this case, the cell mesh vertex 1 and 3 map to single points on the expanded cell, but the points correspond to different vertices on the body rectangle. Cell vertex 2 is mapped to a line segment corresponding to the edge that matches the cell normal. To preserve a one-to-one mapping, a particular point along the edge must be chosen.

a line segment. To preserve a one-to-one mapping, one point in the line segment must be chosen. There are several "reasonable" choices.

If all the normals of the other facet vertices in the facets connected to this cell vertex would lead to a particular body vertex being chosen, it is reasonable to chose that body vertex for calculation of $p(g)$. If the other facets disagree on which body vertex to choose, then it is reasonable to interpolate between the body edge vertices. A more common situation is refining a facet edge where one facet vertex is associate with one body vertex, and the other facet vertex is associated with the other body vertex. In this case, it is reasonable to add an interpolated value in calculating $p(g)$. The vertex should track the interpolation value to use in later refinements. Thus, the many-to-one mapping is accommodated by iteratively adding vertices and facets during refinement.

The final special case, that is checked after the refinement process is complete, is when the three projected normals of a facet span $\mathbb{R}^2$, or two projections are equal and the other spans $\mathbb{R}$; in these cases, at some point in the region of the approximated surface patch the projected normal is null. That is, the cell surface normal is orthogonal to the projection plane. In this case, the a collection of vertices and facets are added to the mesh that represent a mesh over the robot body at the pose of the facet centroid. As the cell is compact with a well defined surface normal, these added vertices always project to the interior of $R(\Xi_i)$.

C.4 Testing Process

These collision tests outlined above represent a brute force approach that is not appropriate for real-time calculations; however, it is easy to implement and is suitable for cell validation during policy instantiation. The resulting mesh has a finite number of vertices and finite number of facets based on the sampling resolutions used to guide refinement and enforce termination. The accuracy of the technique is determined by the accuracy of the expanded cell surface mesh; smaller facets on the expanded cell surface mesh implies better accuracy. By carefully choosing the mesh resolution, and slightly padding the obstacles based on the maximum error bound, the approach can be guarantee the safety of a cell without being overly conservative.

During manual specification of policies, a coarse sampling of the expanded cell is used to check for the inclusion of the projected vertices in the obstacles. In the early stages, the refinement steps are skipped, and only the exact collision tests of projected vertex in obstacle is used to rapidly eliminate invalid cells. A visual inspection as in Figure C.5-b can also guide the selection of policy parameters during manual instantiation. Once reasonable choices for policy parameters are determined, the refined mesh is used for final validation.

For automated instantiation using the policy cache and reference points described in Chapter 5, these collision tests are automated and incorporated into the instantiation process. A fine resolution mesh is calculated and refined once for each policy in the cache. The vertices used in the approximation of $R\left(\Xi_i\right)$ are stored with the policy cache. During instantiation, the vertices are transformed relative to the reference point and then tested for collision using the collision tests described above. If collision occurs, that policy from the cache is not instantiated at the reference point. The rigid body transformation is fast relative to the mesh generation and refinement process, which only needs to run once for each policy in the cache.

The complexity of the collision testing has two distinct components. The first component is directly linked to the complexity of the mesh generation and refinement process used, and is not explored here. Given a expanded cell mesh with N_v vertices and N_f facets, and N_O obstacles with N_{OP} vertices and other test points, the complexity of the collision test is O($N_v * N_O$) inclusion tests for vertices in obstacles and O($N_f * N_{OP}$) tests for obstacle vertices in a facet.

Appendix D

‘PF’ Style Control Policies

This appendix provides a detailed derivation of the ‘PF’ style control policies used in this thesis. The policies are based on a variable structure control approach to *path following*, hence the name ‘PF’ [4]. While the approach is inspired by [4], the control technique is different. Furthermore, in keeping with the sequential composition approach advocated in this paper, the policies have explicit definitions for the domains of attraction allowing reasoning about their safety in free pose space.

This section begins by presenting the structure of the defined policies, and then presents an overview of the basic control approach for kinematic systems. The formal derivations for specific models are delayed until the end of the appendix. The appendix continues with a discussion of the specific curves – line segments and circular arcs – used in this thesis. The section presents verification that the policies satisfy the composability requirements given in Section 3.3. Given the general overview of the cell and control approach, the formal control laws for each system model are derived. Finally, the section concludes with a discussion of the limitations of lines and circular arcs, and the challenges of extending the technique to arbitrary curves.

D.1 Policy Structure

Throughout this thesis, the control policies are defined over local regions of body pose space, which are termed cells. The cells are defined in a local $\mathbb{R}^3$ representation of the $\{x, y, \theta\}$ pose space[1]. The cells are defined to be compact connected regions without holes. The cells must have continuous boundaries, so the $\pm\pi$ θ-dimension in body pose space is NOT identified for the cell definition.

For PF style policies, the cells are defined relative to a two-dimensional planar reference curve that is lifted to body pose space. Let $\hat{p}(s) = (\hat{x}(s), \hat{y}(s)) \in \mathbb{R}^2$ define a planar curve in workspace, where $s \in [0,1]$. We designate $\hat{p}(0)$ as the (x, y) position that corresponds to the goal set center; thus, the control policy is designed to move the system along the curve from $s = 1$ to $s = 0$. Let $\hat{\theta}(s) = \operatorname{atan2}(-\hat{y}'(s), -\hat{x}'(s)) + 2k\pi$, where $(-\hat{x}'(s), -\hat{y}'(s))$ is the tangent vector in the direction of travel; this maps the two-dimensional workspace curve into a three-dimensional body pose space curve $\hat{g}(s) = \left(\hat{x}(s), \hat{y}(s), \hat{\theta}(s)\right)$. By choice of the appropriate integer value for k, the solution to $\hat{\theta}(s)$ is restricted to be continuous along the curve parameterized by $s \in [0,1]$ with $\hat{\theta}(0)$ as the anchor point. The cell goal center reference point is $g_{\text{goal}} = \left\{\hat{x}(0), \hat{y}(0), \hat{\theta}(0)\right\}$.

To enable policy definition, each reference curve is further restricted as follows:

(i) $\hat{g}(s) \in C^k$, the space of functions with k continuous derivatives for $k > 1$,

[1]See Appendix A for a discussion of body pose space.

(ii) the reference path's minimum radius of curvature is larger than minimum allowable vehicle turning radius,

(iii) the path has a unique closest point for all planar points within a specified open neighborhood.

Condition *(i)* ensures that the proper derivatives are available for our control approach. Condition *(ii)* guarantees that the curve will have a full dimensional region of attraction. Condition *(iii)* guarantees that for a sufficiently small neighborhood of the path, the policy is uniquely defined; this constrains the definition of the cell boundaries.

Given a robot pose $g = (x, y, \theta)$ in a neighborhood of the $\hat{g}(s)$ curve, let $\bar{s}(g) \in [0, 1]$ specify the unique nearest point on the planar curve $\hat{p}$. That is, $\|(x, y) - \hat{p}(s)\| > \|(x, y) - \hat{p}(\bar{s})\|$ for all $s \neq \bar{s}$ [4]. At $\hat{g}(\bar{s})$, define a local frame $\mathscr{F}$ as shown in Figure D.1. Let $\mathscr{F}_{\tilde{x}}$ denote the three dimensional tangent vector to the $\hat{g}$ curve in the direction of travel and parallel to the xy-plane. Let $\mathscr{F}_{\tilde{\theta}}$ be parallel to the θ-axis of the body pose space and in the same direction. Let $\mathscr{F}_{\tilde{y}}$ form a right handed coordinate system with $\mathscr{F}_{\tilde{x}}$ and $\mathscr{F}_{\tilde{\theta}}$. Given a continuous curve $\hat{g}(s)$, the frame continuously varies along the path. Note, this convention differs from a conventional Frenet frame as it does not flip orientation based on the path curvature and the $\{\mathscr{F}_{\tilde{x}}, \mathscr{F}_{\tilde{y}}\}$-axes are parallel to the xy-plane [4]. As described in Section 5.3, the goal set orientation defined by the local x'-axis at the goal set corresponds to the local $\mathscr{F}_{\tilde{x}}$-axis at $\hat{g}(0)$.

The robot body pose is expressed in the local $\mathscr{F}$ frame along the curve as $\tilde{g}(g) = \left(\tilde{x}(g), \tilde{y}(g), \tilde{\theta}(g)\right)$. Since $\bar{s}(g)$ is the closest point on the curve, and the $\mathscr{F}_{\tilde{x}}$-axis is tangent to the curve, $\tilde{x}(g) = 0$ by definition. The control problem is to drive the lateral offset $\tilde{y}(g)$ and the heading error $\tilde{\theta}(g)$ to zero as the vehicle moves along the path while monotonically decreasing $\bar{s}(g)$. Hence, the natural error coordinates are $e(g) = \left(\bar{s}(g), \tilde{y}(g), \tilde{\theta}(g)\right)$ [4].

D.2 General Control Approach

The control approach for PF policies is based on a form of variable structure control called sliding mode, control [4, 31]. Sliding mode control works by using a surface to define a switching control policy. If the vehicle pose is "above" the sliding surface, then the system attempts to steer the vehicle pose onto the sliding surface by maximally decreasing $\tilde{\theta}(g)$ as the vehicle moves; for forward

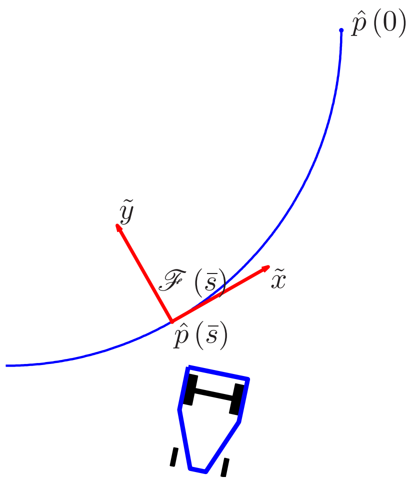


Figure D.1: Reference path with defined coordinates.

motion, this corresponds to turning right. If the vehicle pose is "below" the sliding surface, then the system attempts to steer the vehicle configuration onto the sliding surface by maximally increasing $\tilde{\theta}(g)$ as the vehicle moves; this corresponds to turning left while moving forward. In theory, the control instantaneously switches as it crosses the sliding surface; if the sliding surface is well designed, the "average" *Filipov* equivalent control causes the system to "slide along the surface" to the goal [31]. In practice, we define a "blending zone" on either side of the sliding surface.

To derive a sliding surface in the body pose space, consider the motion of the vehicle as it moves along a circular arc of radius $\tilde{\rho}$, as shown in Figure D.2. The lateral offset, $\tilde{y}$, is a function of the heading change, $\tilde{\theta}$, for a given radius. From Figure D.2, $\cos\tilde{\theta} = \frac{\tilde{\rho}-|\tilde{y}|}{\tilde{\rho}}$, or $\tilde{\theta} = \cos^{-1}\left(1 - \frac{|\tilde{y}|}{\tilde{\rho}}\right)$. Clearly, $\tilde{\rho}$ must be larger than the minimum turning radius of the vehicle.

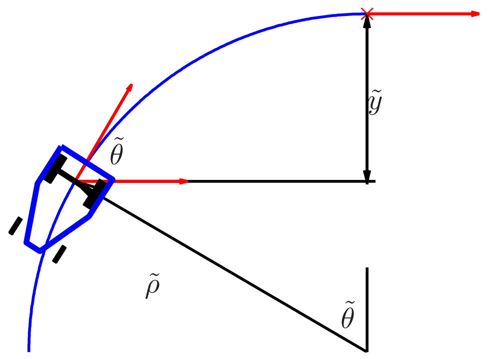


Figure D.2: Moving around a circular arc generates a lateral displacement; the relationship between lateral displacement and orientation change defines the sliding surface.

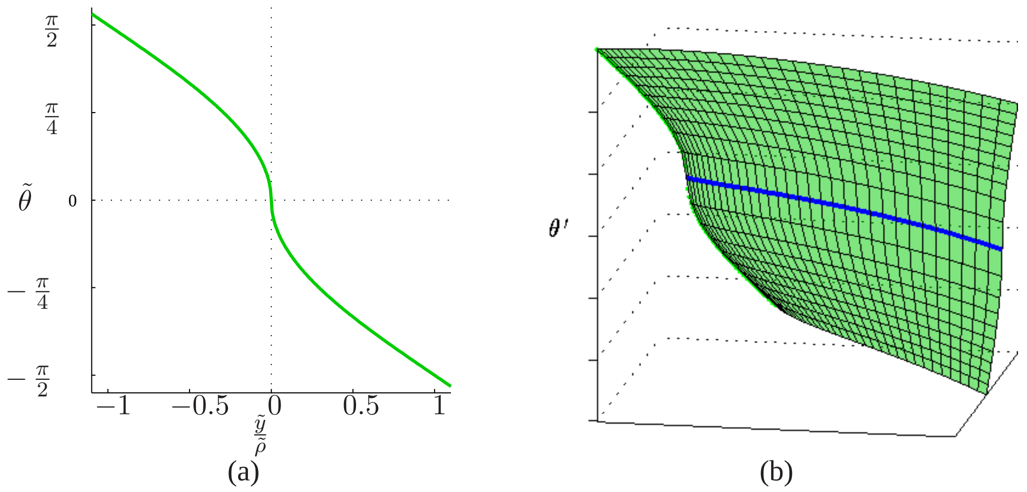


Figure D.3: PF control surface: a) Sliding control surface in the local $\mathscr{F}_{\tilde{y}}$-$\mathscr{F}_{\tilde{\theta}}$ plane defined by (D.1). b) Sliding control surface in body pose space formed by extruding the local curve along the $\hat{g}(s)$ curve.

Using the above intuition, we define $\sigma : \mathbb{R}^2 \to \mathbb{R}$ as scalar function over the local $\mathscr{F}_{\tilde{y}}$-$\mathscr{F}_{\tilde{\theta}}$ plane. For the initial discussion, let

$$\sigma\left(\tilde{y},\tilde{\theta}\right) = \begin{cases} \tilde{\theta} - \operatorname{sign}(\tilde{y})\cos^{-1}\left(1 - \frac{|\tilde{y}|}{\tilde{\rho}}\right) & |\tilde{y}| \le \tilde{\rho} \\ \tilde{\theta} - \operatorname{sign}(|\tilde{y}|)\left(\frac{\pi}{2} + \frac{\tilde{y}}{\tilde{\rho}} - 1\right) & \text{otherwise} \end{cases}, \tag{D.1}$$

where $\tilde{\rho}$ is a free parameter. The $\sigma^{-1}(0)$ curve, that is $\sigma^{-1}(0) = \left\{\left(\tilde{y},\tilde{\theta}\right) \mid \sigma\left(\tilde{y},\tilde{\theta}\right) = 0\right\}$, is shown in Figure D.3-a. The sliding surface in body pose space is formed by extruding the local $\sigma^{-1}(0)$-curve along the $\hat{g}(s)$ curve, as shown in Figure D.5-b; denote the extruded surface as $\Sigma^{-1}(0)$ where $\Sigma(g) = \sigma\left(\tilde{y}(g), \tilde{\theta}(g)\right)$. Stated more precisely, sliding mode control attempts to drive the error coordinates $e(g) = \left(\bar{s}(g), \tilde{y}(g), \tilde{\theta}(g)\right)$ to the $\Sigma^{-1}(0)$ sliding surface, and then along the $\Sigma^{-1}(0)$ surface toward $e(g) = (0,0,0) = \mathbf{0}$.

Pure sliding mode control works, but suffers from several drawbacks in practice. Sliding mode control requires an instantaneous change in velocity, which is unrealizable on real systems. Finite control update times make a sliding mode control system prone to chatter. To mitigate these effects, we define a blending region on either side of the sliding surface. If the system is outside the blending region, then the control performs maximal steering towards the sliding surface. Inside the blending region, the control interpolates between maximal steering and a neutral steering policy that moves the system along the sliding surface.

The neutral steering policy should be a continuous policy over the sliding surface. Unfortunately, the simple curve given by (D.1) has an undefined derivative at $\tilde{y} = 0$. Thus, we redefine the local frame sliding surface as

$$\sigma\left(\tilde{y},\tilde{\theta}\right) = \begin{cases} \tilde{\theta} - \operatorname{sign}(\tilde{y})\cos^{-1}\left(1 - \frac{|\tilde{y}|-\tilde{y}_o}{\tilde{\rho}}\right) & \tilde{y}_b \le |\tilde{y}| \le \tilde{\rho} + \tilde{y}_o \\ \tilde{\theta} - f_{\tilde{\theta}}(\tilde{y}) & |\tilde{y}| \le \tilde{y}_b \\ \tilde{\theta} - \operatorname{sign}(\tilde{y})\left(\frac{\pi}{2} + \frac{|\tilde{y}|-\tilde{y}_o}{\tilde{\rho}} - 1\right) & \text{otherwise} \end{cases}, \tag{D.2}$$

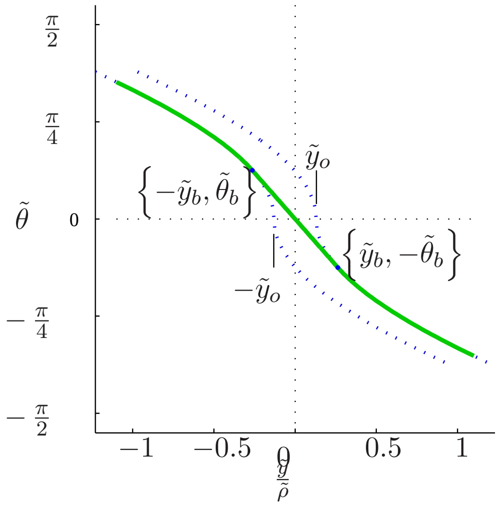


Figure D.4: The smoothed sliding surface defined in (D.2).

where $\tilde{y}_o$ is the symmetric offset between two surfaces of the type defined in (D.1), $f_{\tilde{\theta}}(\tilde{y})$ is function that provides C^2 continuity on (D.2) at the points $\left\{\pm\tilde{y}_b, \mp\tilde{\theta}_b\right\}$. For this thesis, $f_{\tilde{\theta}}$ is a cubic polynomial whose coefficients and $\tilde{y}_b$ are determined such that the sliding surface derivative is continuous at $\tilde{y} = \tilde{y}_b$. Figure D.4 shows the new sliding surface in the $\mathscr{F}_{\tilde{y}} - \mathscr{F}_{\tilde{\theta}}$ plane.

Figure D.5-a shows the blending region to either side of the sliding surface. The blending region is delineated by offsetting the sliding surface in the $\tilde{\theta}$ direction $\pm\Delta\tilde{\theta}_B$. Figure D.5-b shows the interpolating function used to provide C^2 continuity of the steering policy. The blending function is

$$B(g) = B\left(\tilde{y}(g), \tilde{\theta}(g)\right) = \begin{cases} \sqrt{3\left(\frac{\sigma(\tilde{y},\tilde{\theta})}{\Delta\tilde{\theta}_B}\right)^4 - 2\left(\frac{\sigma(\tilde{y},\tilde{\theta})}{\Delta\tilde{\theta}_B}\right)^6} & \left|\sigma\left(\tilde{y},\tilde{\theta}\right)\right| \leq \Delta\tilde{\theta}_B \\ 1 & \left|\sigma\left(\tilde{y},\tilde{\theta}\right)\right| > \Delta\tilde{\theta}_B \end{cases}, \quad \text{(D.3)}$$

where $\Delta\tilde{\theta}_B$ is the height of the blending zone relative to the sliding surface. The control effort can be smoothed by increasing either $\Delta\tilde{\theta}_B$ or the width of the linear offset,$\tilde{y}_o$, at the cost of slowing convergence to the reference path.

D.3 Cell Definitions

The cell boundary, which defines the neighborhood of $\hat{g}(s)$ where it is safe to invoke the steering policy, is based on the curves defined above. Although the control approach is general, this thesis is restricted to two curve types – straight line segments and circular arcs.

D.3.1 Line-segment Based Cell

For the moment, assume that $\hat{p}(s)$ is line segment so that $\tilde{\theta}(\hat{g}(s)) = \theta_{\text{goal}}$ and $\mathscr{F}(\bar{s}(g))$ only varies in position along the curve. Given the local frame $\mathscr{F}(0)$ at the goal set center $\hat{g}(0)$, and given a robot pose $g = (x, y, \theta)$, the local pose error $e(g) = \left(\bar{s}(g), \tilde{y}(g), \tilde{\theta}(g)\right)$ is specified. The value

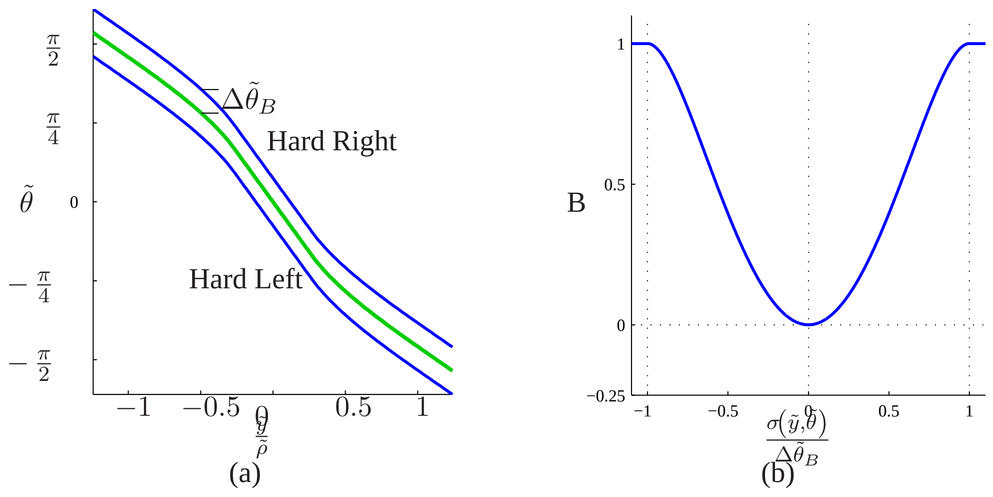


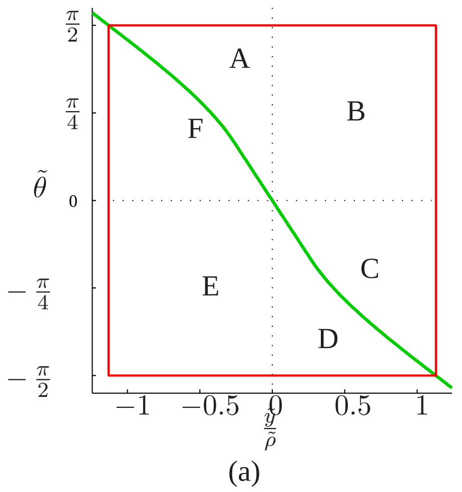
Figure D.5: a) Sliding control surface with blending ranges shown. b) The blending function used to interpolate between maximal and sliding control.

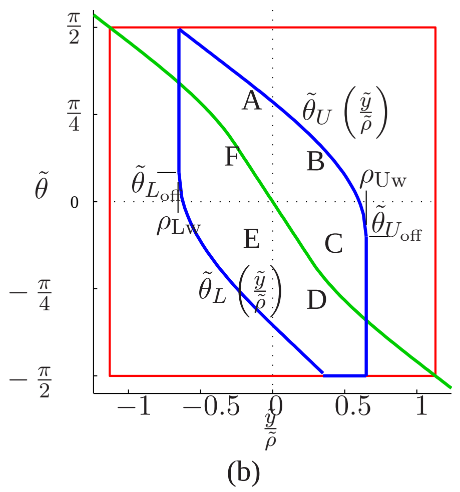
$\bar{s}(g) = \frac{\mathscr{F}_{\tilde{x}}(0)\cdot(g-\hat{g}(0))}{L}$, where L is the length of the cell and $\mathscr{F}_{\tilde{x}}(0)$ is the x-axis pointing out of the cell. The lateral offset is $\tilde{y}(g) = \mathscr{F}_{\tilde{y}}(0)\cdot(g-\hat{g}(0))$. The orientation relative to the local frame is $\tilde{\theta}(g) = \theta - \hat{\theta}(0)$.

If the direction of travel is oriented with a positive projection along the positive $\tilde{x}$-axis, the instantaneous motion induced by the sliding-mode based policy will decrease $\bar{s}$. By careful definition of the cell, that is the local neighborhood around $\hat{g}(s)$, a control policy that satisfies the objective of moving along the path while converging to $\left(\tilde{y}(g), \tilde{\theta}(g)\right) = \mathbf{0}$ may be defined for a variety of systems. For the systems of concern in this thesis, the instantaneous velocity is always along the body axis in the xy-plane. Therefore, for forward motion, we cap the range of $\left|\tilde{\theta}(g)\right| < \frac{\pi}{2}$, which guarantees that invoking the steering policy will make monotonic progress along the planar curve. The sliding surface should split the cell to prevent the control from driving the system out of the bounded $\tilde{\theta}$ range. That is the sliding surface should not exceed $\pm\frac{\pi}{2}$ within the cell boundary. For a given $\tilde{\rho}$, this constrains the cell width to the box shown in Figure D.6-a; for a given width, this constrains $\tilde{\rho}$.

To further define the cell boundaries, consider the six labeled regions shown in Figure D.6-a. On the sliding surface separating the regions, the neutral steering control is defined to drive the system along the sliding surface toward the $\hat{g}(s)$ curve; therefore the induced velocity is not transverse to the sliding surface. As the actions are symmetric about the origin, we will restrict our discussion to the regions (A,B,C) above the curve. Further, assume the vehicle is traveling forward; similar arguments apply equally to reverse motion with minor changes to region labels. If the system's $\left(\tilde{y}(g), \tilde{\theta}(g)\right)$ coordinates are in bounded region A, the control action will induce motion whose instantaneous motion moves the system towards the sliding surface or across the common face with region B. The velocity will not exit the upper bound so long as the sliding surface is less than the upper bound. In region C, the system is naturally oriented toward the sliding surface and away from the other bounding regions. Thus, once inside, the system will remain in region C until it converges to the sliding surface. Region B is the problematic region, and must be further constrained.

For poses in region B, the system is oriented away from the sliding surface. While the system will not cross the given upper bound in region B due to the control that drives the system toward





$\frac{\pi}{2}$
$\frac{\pi}{4}$
$\tilde{\theta}$
0
$-\frac{\pi}{4}$
$-\frac{\pi}{2}$
A
$\tilde{\theta}_U\left(\frac{\tilde{y}}{\tilde{\rho}}\right)$
F
B
ρ_{Uw}
$\tilde{\theta}_{L_{\mathrm{off}}}$
ρ_{Lw}
$\tilde{\theta}_{U_{\mathrm{off}}}$
E
C
$\tilde{\theta}_L\left(\frac{\tilde{y}}{\tilde{\rho}}\right)$
D
-1
-0.5
0
0.5
1
$\frac{\tilde{y}}{\tilde{\rho}}$
(b)

Figure D.6: PF cell: a) Bounding box and control regions defined relative to the sliding surface. b) Limiting surfaces defined in the local $\mathscr{F}$ frame.

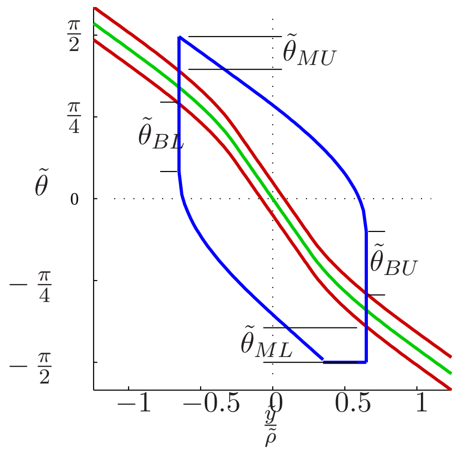
the sliding surface, the control may cross the right side bounding width in region B. To circumvent this problem, the $\tilde{\theta}$ orientation at the extreme width must be less than zero, so that the vehicle is oriented into the cell. Furthermore, the region B bound must be such that the system can turn fast enough to stay in region B until it enters region C. For this upper bound, we return to (D.1) and define $\tilde{\theta}_U(\tilde{y}) = \cos^{-1}\left(1 - \frac{\tilde{y}-\rho_{Uw}}{\tilde{\rho}_U}\right) - \tilde{\theta}_{U_{\text{off}}}$, where turning radius parameter $\tilde{\rho}_U > \rho_{\min}$ where $\rho_{\min}$ is the minimum turning radius induced by the maximal steering policy, $\rho_{Uw} < \tilde{\rho}$ is the cell width, and $\tilde{\theta}_{U_{\text{off}}}$ is a offset buffer that guarantees the system is oriented inwards at the far boundary. The lower bounding surface is defined analogously; note that the upper and lower bounding surfaces and widths are not necessarily symmetric. The values are constrained, but offer some freedom in defining the cell provided the upper bounding and sliding surfaces do not cross or touch in the $\tilde{y} \in [\rho_{Lw}, \rho_{Uw}]$ range. Figure D.6-b shows the resulting limiting surfaces for the cell. Note, the definition of $\tilde{\theta}_U$ also constrains region A and C.

The policy must guarantee that the steering along the upper (lower) boundaries will keep the velocity inward pointing with respect to the cell boundary. The simplest test is to force $\tilde{\rho}_U > \rho_{\min}$, and guarantee that the blending zone does not intersect the bounding surfaces. That is $\tilde{\theta}_{MU} = \tilde{\theta}_U(-\rho_{\text{Lw}}) - \sigma^{-1}_{|-\rho_{\text{Lw}}}(0) - \Delta\tilde{\theta}_B > 0$ and $\tilde{\theta}_{ML} = \tilde{\theta}_L(-\rho_{\text{Lw}}) + \sigma^{-1}_{|\rho_{\text{Uw}}}(0) + \Delta\tilde{\theta}_B > 0$. Likewise, $\tilde{\theta}_{BL} = \sigma^{-1}_{|-\rho_{\text{Lw}}}(0) - \Delta\tilde{\theta}_B - \tilde{\theta}_{L_{\text{off}}} > 0$ and $\tilde{\theta}_{BU} = \tilde{\theta}_{U_{\text{off}}} + \sigma{-1}_{|\rho_{\text{Uw}}}(0) - \Delta\tilde{\theta}_B > 0$.

Figure D.7-a shows the cell boundary and blending surfaces. The complete cell, formed by extruding the cell boundaries along the $\hat{g}(s)$ curve, is shown in Figure D.7-b. For any configuration within this cell, the smoothed sliding mode based policy will keep the vehicle body configuration within the cell and moving toward the $\hat{g}(0)$ point.

The cell pictured in Figure D.7-b is functional, but is not conducive to satisfying a prepares relationship. While a small cell, that is a narrow "tube" about the $\hat{g}(s)$ curve, can prepare a larger cell, this does not generalize the funnel metaphor discussed earlier. To this end, we add a constraint based on a Lyapunov-like function defined in our error coordinates $e(g) = \left(\bar{s}(g), \tilde{y}(g), \tilde{\theta}(g)\right)$. Let

$$V(g) = V(e(g)) = \exp^{-\frac{\bar{s}(g)}{s_v}}\left(\begin{bmatrix}\tilde{y}(g) & \tilde{\theta}(g)\end{bmatrix} W \begin{bmatrix}\tilde{y}(g) \\ \tilde{\theta}(g)\end{bmatrix}\right), \tag{D.4}$$



(a)

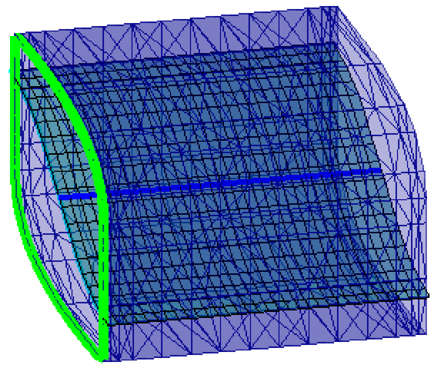

(b)

Figure D.7: PF cell: a) Limiting and blending surfaces defined in the local $\mathscr{F}$ frame. b) Cell formed by extruding the sliding and limit surfaces along a straight line.

where s_v is the length scaling and W is a positive definite weighting matrix. A level set of V at a given $\bar{s}$ is an ellipse in the local $\mathscr{F}_{\tilde{y}}$-$\mathscr{F}_{\tilde{\theta}}$ frame; the elliptical diameters expand along the length of the cell. Apply the restriction that $V(g) < V_{\text{cell}}$ to our cell, we restrict the cell definition as shown in Figure D.8. Care must be taken to verify that the induced velocity is inward pointing along the portion of the cell bounded by $V = V_{\text{cell}}$. As the level set is a smooth function of g, and the policy is piecewise smooth over the cell surface, the conditional invariance can be verified as described later.

For convenience, the free parameters for the line-segment based PF cell – including those of the example shown in Figure D.8 – are listed here:

Table D.1: Line-segment based PF cell parameters

Variable	Description	Range	Figure D.8
s	Arc length parameter	$[0,1]$	N/A
g_{goal}	Configuration of goal set center	$\in \mathcal{G}_{\text{free}}$	$(0,0,0)$
L	Length along negative $\mathscr{F}_{\tilde{x}}(0)$-axis	$(0,\infty)$	1.51 m
$\tilde{\rho}$	Radius of curvature for σ	limited by $\mathcal{U}$	2.45 m
$\tilde{y}_o$	Smoothing offset	$(0,\infty)$	0.025 m
$\tilde{y}_b$	Smoothing width	Calculated	0.0501 m
$\tilde{\theta}_b$	Smoothing height	Calculated	0.143 m
$\Delta\tilde{\theta}_B$	Blending offset	$\left(0,\frac{\pi}{2}\right)$	$\approx \frac{\pi}{42}$
$\{\rho_{U_w},\rho_{L_w}\}$	Cell width	$(0,\tilde{y}_o+\tilde{\rho})$	$\{0.125, 0.125\}$
$\{\rho_U,\rho_L\}$	Cell limit surface "curvature"	$(\rho_{\min},\infty)$	$\{2.45, 2.45\}$
$\left\{\tilde{\theta}_{U_{\text{off}}},\tilde{\theta}_{L_{\text{off}}}\right\}$	Cell limit surface offset	$(0,\pi/2)$	$\frac{\pi}{50}$
V_{cell}	Funnel level set value	$(0,\infty)$	0.5

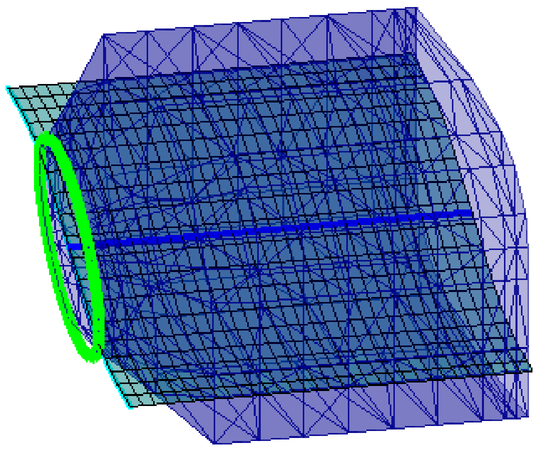


Figure D.8: PF cell: Funnel shaped cell formed by extruding the limit surfaces along the $\hat{g}(s)$ path and cut by the Lyapunov-like $V(e) = V_{\text{cell}}$ level-set. The goal set boundary is shown as a thick light colored ellipse.

D.3.2 Circular Arc Based Cell

The approach to defining the circular arc based cells is exactly the same as for line segment based cells. In this case, however, the frame derivatives along the path are non-zero; therefore, the simple invariance analysis based on $\tilde{\rho}_U > \rho_{\min}$ is insufficient to guarantee safety.

The cells are defined in the local pose error coordinates $e(g) = \left(\bar{s}(g), \tilde{y}(g), \tilde{\theta}(g)\right)$ given a robot pose $g = (x, y, \theta)$, goal location $\hat{g}(0)$, and goal frame $\mathscr{F}(0)$. Let $r(g)$ denote the vector from the center of curvature of $\hat{p}(s)$ in workspace to the robot (x, y) position, and φ the angle between $\hat{p}(0)$ and the robot position. This is shown in Figure D.9. The vector $r(g)$ intersects the defined path $\hat{p}(s)$ at $\hat{p}(\bar{s})$, where $\bar{s}(g) = \varphi(g)/\tilde{\varphi}$ and $\tilde{\varphi}$ defines the "length" of the cell; that is $\rho\tilde{\varphi}$ defines the arc length of $\hat{g}(s)$ for $s \in [0, 1]$ where ρ is the radius of the arc $\hat{p}(s)$. The curvature of the arc may be positive or negative, and is determined by the sign of $\tilde{\varphi}$. If $\tilde{\varphi} < 0$ the rotation is clockwise as shown in Figure D.9 and $\tilde{y}(g) = \rho - \|r(g)\|$; if $\tilde{\varphi} > 0$ the rotation is counter-clockwise from the goal and $\tilde{y}(g) = \|r(g)\| - \rho$. The orientation error is $\tilde{\theta}(g) = \theta - \hat{\theta}(0) - \bar{s}\tilde{\varphi} + 2k\pi$, where the integer k is chosen to yield a continuous function over the cell.

A necessary condition for a valid cell, that is one whose domain of attraction encompasses a non-zero volume of pose space, is that the radius of the circular arc, ρ, is greater than the minimum turning radius $\rho_{\min}$. Furthermore, the size of the bounded $\tilde{y}$-$\tilde{\theta}$ slice will be smaller than for the largest linear cell of the same system. This due to to impact of the frame derivatives; Section D.4 presents the formal calculations for the system models used in this thesis.

For tight turns, the ability to define significant funnel-like restrictions via (D.4) is limited due to the necessity of maintaining conditional invariance and the $\mathscr{F}$ derivatives. For this reason, the approach taken in this thesis is to use small tubes around arcs to prepare larger tubes around either line segment or arc based cells, and use the funnel-like line-segment based cells to prepare the smaller arc based cells.

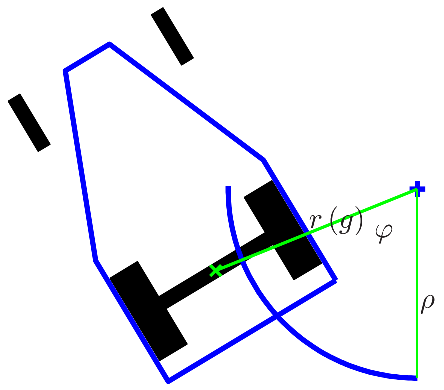


Figure D.9: Layout of the circular arc based cells.

Figure D.10 shows an example cell; the relevant parameters are shown in Table D.2. The only changes relative to Table D.1 are the arc angle $\tilde{\varphi}$ and the radius ρ.

Table D.2: Schedule

Variable	Description	Range	Figure D.10
s	arc length parameter	$[0, 1]$	N/A
g_{goal}	Configuration of goal set center	$\in \mathcal{G}_{\text{free}}$	$\{0, 0, 0\}$
ρ	Radius of curvature	$(\rho_{\min}, \infty)$	0.7 m
$\tilde{\varphi}$	Arc length of curve (radians)	$(-\pi, \pi)$	$-1.01\,\frac{\pi}{2}$
L	Length along path $\hat{p}\,(s)$	Calculated $\rho\,\tilde{\varphi}$	0.35π m
$\tilde{\rho}$	Radius of curvature for σ	$(0, \infty)$	2.45 m
$\tilde{y}_o$	Smoothing offset	$(0, \infty)$	0.025 m
$\tilde{y}_b$	Smoothing width	Calculated	0.0501 m
$\tilde{\theta}_b$	Smoothing height	Calculated	0.143 m
$\Delta\tilde{\theta}_B$	Blending offset	$\left(0, \frac{\pi}{2}\right)$	$\approx \frac{\pi}{45}$
$\{\rho_{U_w}, \rho_{L_w}\}$	Cell width	$(0, \tilde{y}_o + \tilde{\rho})$	0.11 m
$\{\rho_U, \rho_L\}$	Cell limit surface "curvature"	$(\rho_{\min}, \infty)$	2.45 m
$\left\{\tilde{\theta}_{U_{\text{off}}}, \tilde{\theta}_{L_{\text{off}}}\right\}$	Cell limit surface offset	$(0, \pi/2)$	$\frac{\pi}{50}$

As with the line segment based cell, the arc based cell can satisfy the requirements of Section 3.3. The test for inclusion is essentially the same; only the mapping into the local $\left(\mathscr{F}_{\tilde{y}}, \mathscr{F}_{\tilde{\theta}}\right)$

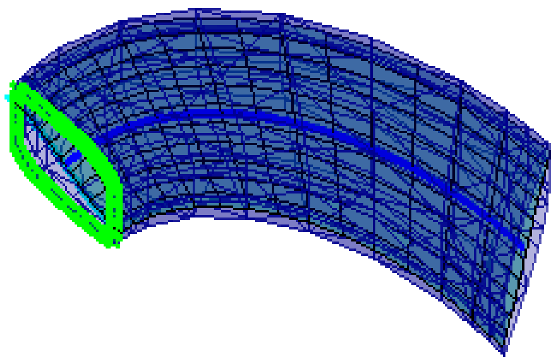


Figure D.10: PF arc-based cell formed by extruding the limit surfaces along the $\hat{g}\,(s)$ path.

frame varies. The expanded cell used to verify safety is likewise well defined. The cell is limited in a way that guarantees instantaneous progress along the curve, so finite time is guaranteed provided conditional invariance is verified. The test to verify conditional invariance is slightly more complicated due to the derivatives associated with the moving frame, but is numerically tractable. The remainder of this appendix presents the specific calculations for each system model required for a circular arc based cell.

D.4 Policy Designs

This subsection presents the equations used in the control policy calculations; the policies determine valid inputs from the bounded input set $\mathcal{U}$. Recall from Chapter 5 and Appendix A, that the global body pose velocity is given by $\dot{g} = A(q)\,\dot{r}$. In the remaining discussion, we assume a first-order kinematic system with $\dot{r} = u$, which implies $\dot{g} = A(q)\,u$. This section begins by providing an overview of the calculations that apply to all of the robot models considered in this thesis. The calculations and control strategies used for each model are then defined.

The variable structure control approach depends on a neutral steering policy that causes the system to follow the sliding surface. Since the mapping $A(q)$ is invariant under rigid body pose transformations, and the control surfaces are defined in the local frame $\mathscr{F}$, we derive the controls in the local frame. For $\tilde{g} = \left(\tilde{x}(g), \tilde{y}(g), \tilde{\theta}(g)\right)$, let $\dot{\tilde{g}} = \mathscr{F}(g) \cdot \dot{g}$; that is, $\dot{\tilde{g}}$ is the pose velocity expressed in the local frame. For the neutral steering policy to keep the system on the sliding surface, the general constraint is $D_{\tilde{g}}\Sigma(\tilde{g}) \cdot \dot{\tilde{g}} = 0$ for all $\tilde{g} = \left(0, \tilde{y}(g), \tilde{\theta}(g)\right) \in \Sigma^{-1}(0)$. Letting $\omega = D_{\tilde{g}}\Sigma(\tilde{g}(g)) \cdot A(\tilde{q})$ where $\tilde{q} = (\tilde{g}, r)$, ω defines an equality constraint on the base velocities. In other words, for a given pose on the sliding surface, to remain on the sliding surface the base velocities must lie on the line through the base tangent space origin defined by ω. By the chain rule, $D_{\tilde{g}}\Sigma(\tilde{g}(g)) = D_e\Sigma \cdot D_{\tilde{g}}e$. The first term, $D_e\Sigma(gt) = \begin{bmatrix} 0 & D_{\tilde{y}}\sigma(\tilde{y}) & 1 \end{bmatrix}$ is piecewise smooth. The latter term,

$$D_{\tilde{g}}e = \begin{bmatrix} D_{\tilde{x}}\bar{s} & D_{\tilde{y}}\bar{s} & D_{\tilde{\theta}}\bar{s} \\ D_{\tilde{x}}\tilde{y} & D_{\tilde{y}}\tilde{y} & D_{\tilde{\theta}}\tilde{y} \\ D_{\tilde{x}}\tilde{\theta} & D_{\tilde{y}}\tilde{\theta} & D_{\tilde{\theta}}\tilde{\theta} \end{bmatrix} ,$$

where $D_{\tilde{x}}$ denotes the derivative along $\mathscr{F}_{\tilde{x}}$ is calculated from the mappings given above. For the line-segment based policy,

$$D_{\tilde{g}}e = \begin{bmatrix} -\frac{1}{L} & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} ,$$

while for the arc-based policy

$$D_{\tilde{g}}e = \begin{bmatrix} \frac{1}{\rho|\bar{\varphi}|}\bar{s} & 0 & 0 \\ 0 & 1 & 0 \\ -\frac{1}{\rho} & 0 & 1 \end{bmatrix} ,$$

The change in $\tilde{\theta}$ as the robot moves along $\mathscr{F}_{\tilde{x}}$ tangent to the $\hat{g}(s)$ curve means that the robot must steer just to maintain its current error. While the line-segment based policy can reason about invariance based only on geometric considerations of the cell boundary and the minimum turning radius, the arc-based policy must consider the kinematic model over the cell. This thesis uses a variety of kinematic systems, each with differing control inputs, therefore each system model must be discussed separately.

D.4.1 Unicycle System / Vertical Rolling Disk / Differential-drive system

For the kinematic systems considered in this thesis, where $\dot{g} = A(q)\,u$, the mapping $A(q)$ is differs only by a constant wheel radius factor for the kinematic unicycle system and vertical rolling disk models defined in Appendix A. The mapping $A(q)$ for the differential-drive system differs from these two only by a constant change of coordinates. In each case, the mapping only depends on the body orientation; to make this point clear, we abuse notation and let $A(\theta) = A(q)$.

The neutral steering policy defines a line through the input space origin, $\omega(g) = D_{\tilde{g}}\Sigma(\tilde{g}(g)) \cdot A\left(\tilde{\theta}(g)\right)$. Any input chosen from $\mathcal{U}$ along $\omega(g) \cdot u$ induces motion that holds the value of $\Sigma(\tilde{g}(g))$ constant. Depending on the sign of $\Sigma(\tilde{g}(g))$, one half of the input space divided by $\omega(g) \cdot u = 0$ will decrease the magnitude of Σ by moving the system pose toward the sliding surface. We modify the sign of ω such that the negative half-space moves the system pose toward the sliding surface. Let $\omega_{\text{lim}}(g)$ define a constraint within the ω^- half-space that defines an aggressive steering set of inputs in the direction that decreases $|\Sigma(\tilde{g}(g))|$. That is, ω_{lim} specifies hard left or hard right as appropriate. Define the control constraint as

$$\omega_c(g) = \omega_{\text{lim}}(g) \cdot B(\tilde{g}(g)) + \omega(g) \cdot (1 - B(\tilde{g}(g))) \ .$$

These constraints are shown in Figure D.11. Let $d_\omega = d_\omega(\tilde{g}(g))$ define a distance along the negative of the ω_c vector, and define another constraint surface $-\omega_c$ parallel to ω_c at a distance d_ω along the $-\omega$ vector. As the pose approaches the sliding surface, $B \to 0$ and the control constraint $\omega_c \to \omega$ and $d_\omega \to \epsilon$, where $\epsilon > 0$ is a small number[2]. In this case, the control becomes less aggressive,

[2]As d_c approaches the limiting small number, the ω_c constraint moves $\epsilon/2$ in the positive ω_c direction so that the optimization lies on the original ω constraint surface

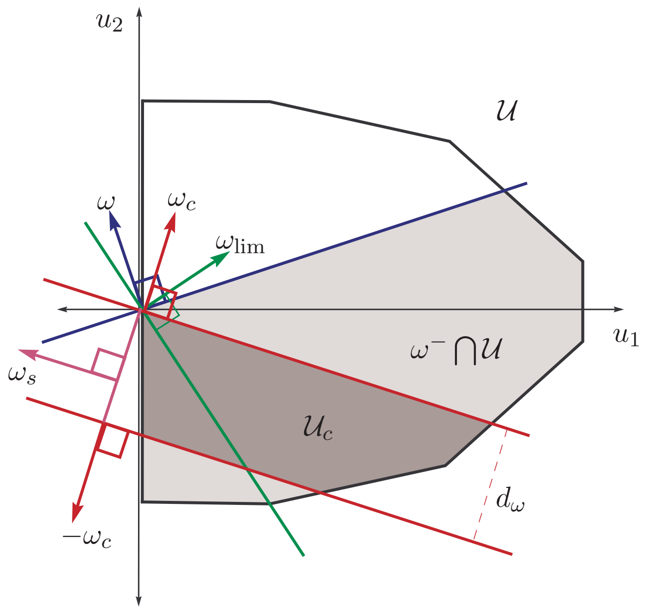


Figure D.11: For the bounded steering unicycle and differential-drive systems, the control policy constrains the input set based on the neutral steering policy and blending function B. Any input taken from the dark gray region label $\mathcal{U}_c$ will move the system toward the sliding surface.

and set of valid control inputs is constrained to lie along the line defined by ω_c such that the system follows the sliding surface. As the pose departs the blending zone away from the sliding surface, $B \to 1$, and $\omega_c \to \omega_{\lim}$ and d_ω increases to a distance equal to the diameter of $\mathcal{U}$.

The control input is chosen from $\mathcal{U}_c \subset \mathcal{U}$ between $\omega_c(g)$ and $-\omega_c(g)$, as shown in Figure D.11. For the case where $\mathcal{U}$ is a convex polygon, the region $\mathcal{U}_c$ is also a convex polygon. This lends itself to simple convex optimization techniques [10]. Each edge in the bounded set $\mathcal{U}$ is treated as a half-space constraint along with $\omega_c(g)$ and $-\omega_c(g)$. The costs associated with approaching each constraint are weighted. Most constraints have costs that increase rapidly as the input approaches a constraint, but are negligible away from the constraint. A constraint $\omega_s(g)$ that is orthogonal to $\omega_c(g)$ is added to the input constraints as shown in Figure D.11; this constraint is associated with a linear cost function that acts to push the input away from the origin. That is, $\omega_s(g)$ acts to prefer rapid motion. A quadratic term is added that weights changing the input from its previous value, which provides a measure of smoothing to the approach. Additional terms are easily added to the optimization cost function. As long as the input is taken from $\mathcal{U}_c$, this control policy induces the correct behavior over the cell.

D.4.2 Ackermann Steered Car-like system

The mapping $A(q)$ for the Ackermann Steered car-like system depends on the steering angle shape variable. This is a fundamental difference with the models for unicycle and differential-drive systems. In the case of the Ackermann steered car, the neutral steering policy is governed by the steering angle, and is not directly tied to the inputs.

For the Ackermann steered car, the sliding surface is used to define a reference steering angle over the cell. Define the neutral steering angle, that is the angle that holds $\Sigma(\tilde{g}(\gamma))$ constant as the system moves as, $\phi_{\text{ref}}(\tilde{g}) = \tan^{-1}\frac{Y'}{X'}$ where $\tilde{g}(g) = \left(0, \tilde{y}(g), \tilde{\theta}(g)\right)$, $X' = \frac{D_{\tilde{\theta}}\Sigma(\tilde{g})}{L}$ and $Y' = -D_{\tilde{x}}\Sigma(\tilde{g})\cos\left(\tilde{\theta}(g)\right) - D_{\tilde{y}}\Sigma(\tilde{g})\sin\left(\tilde{\theta}(g)\right)$. Let $\phi_{\lim}(\tilde{g})$ denote the absolute steering limit for the given vehicle, with the sign chosen based on whether the vehicle should steer right or left based on its direction of travel and relationship to the sliding surface. If $|\phi_{\text{ref}}(\tilde{g})| > |\phi_{\lim}|$, then $\phi_{\text{ref}}(\tilde{g}) = \text{sign}(\phi_{\text{ref}}(\tilde{g}))\,|\phi_{\lim}(\tilde{g})|$. This is necessary because the reference limit can be exceeded if the vehicle is on the inside of a sharp turn; limiting the reference in this case will allow the car to move to the arc, where the steering is within bounds. Define $\phi_{\text{des}}(\tilde{g}) = \phi_{\text{ref}}(\tilde{g})\cdot(1 - B(\tilde{g})) + \phi_{\lim}(\tilde{g})\cdot B(\tilde{g})$ to be desired steering angle over the cell. Steering according to the mapping $\phi_{\text{des}} : \Xi_i \to [-\phi_{\lim}, \phi_{\lim}]$ will cause the system to move toward the sliding surface and along $\hat{g}(s)$ toward the goal set.

In the case where the current steering angle does not match the desired steering angle, the control policy must steer the vehicle in such a way that the vehicle converges to the goal set and ϕ_{des} without exiting the cell. Over the boundary of the cell, define $\phi_{\text{exit}}(\tilde{g})$ as the steering angle at which the system would exit the cell; that is, the policy would violate conditional invariance. Define the steering margin,

$$\phi_{\text{margin}} = \min_{g \in \partial\Xi_i} |\phi_{\text{exit}}(\tilde{g}(g)) - \phi_{\text{des}}(\tilde{g}(g))| \ .$$

Define the steering error, $\phi_{\text{err}}(g, r) = \phi_{\text{des}}(\tilde{g}(g)) - \phi$ for the shape variables $r = (\psi, \phi)$. If $|\phi_{\text{err}}(g, r)| < \phi_{\text{margin}}$, then the system can safely move. Our control policy design is therefore a switched policy. If $|\phi_{\text{err}}(g, r)| > \phi_{\text{margin}}$, the system stops and steers until $|\phi_{\text{err}}(g, r)| < \phi_{\text{margin}}$. If $|\phi_{\text{err}}(g, r)| < \phi_{\text{margin}}$, then the control policy specifies inputs such that the vehicle steers fast enough that $|\phi_{\text{err}}(g, r)|$ monotonically deceases as the vehicle moves through the cell.

Following the constrained optimization approach, let $\omega(g)$ define a line through the input space origin such that any input along this line will hold the steering error constant. The error time derivative $\dot{\phi}_{\text{err}}(g,r) = \dot{\phi}_{\text{des}}(\tilde{g}(g)) - \dot{\phi} = 0$, with $\dot{\phi}_{\text{des}(\tilde{g}(g))} = D_{\tilde{g}}\phi_{\text{des}}(\tilde{g}(g)) \cdot \dot{\tilde{g}}(g) = D_{\tilde{g}}\phi_{\text{des}}(\tilde{g}(g)) \cdot A(\tilde{q})\,u$ with $\tilde{q} = (\tilde{g}, r)$ and $u = \left(\dot{\psi}, \dot{\phi}\right)$; thus, $\omega(g) = D_{\tilde{g}}\phi_{\text{des}}(\tilde{g}(g)) \cdot A(\tilde{q}) - \begin{bmatrix}0 & 1\end{bmatrix}$. One half-space defined by $\omega(g)$ will decrease the error magnitude; the other half-space will increase the error magnitude. If ω^- decreases the error magnitude, then let $\omega_c = \omega$, otherwise let $\omega_c(g) = -\omega(g)$, so that the negative half-space decreases the steering error. Given $\omega(g)$ defined appropriately, define $-\omega_c(g)$, and $\omega_s(g)$ as in Figure D.11. The distance $d_c(g)$ between the control constraints $\pm\omega_c$ is a function of the steering error; that is, the constraints approach one another as the steering error goes to zero.

Unlike the shape variables for the differential-drive and unicycle models, the steering angle is bounded. The bounds include absolute mechanical bounds as in ϕ_{lim}; we also allow safety bounds that require slower speeds for hard turns. To guarantee that the steering angle constraints are not violated, the steering constraints are mapped to velocity constraints given a nominal control time step, Δt. Assume the steering angle constraints are represented by a collection of half-space constraints each defined by a point $p_s = \left(\dot{\psi}_s, \phi_s\right)$ and normal $n_s = (n_\psi, n_\phi)$. The steering rate constraints are given by $p_r(\phi) = \left(\dot{\psi}_s, \frac{\phi_s - \phi}{\Delta t}\right)$ and normal $n_s(\phi) = \left(\frac{n_\psi}{\Delta t}, n_\phi\right)$. These constraints guarantee that the steering angle will not exceed the steering limit during the next time step. With these added constraints, the input optimization chooses an input from the valid set; the input will specify a forward speed and steering rate such that the steering error monotonically decreases and the system converges to the sliding surface while moving along $\hat{g}(s)$ towards the goal set.

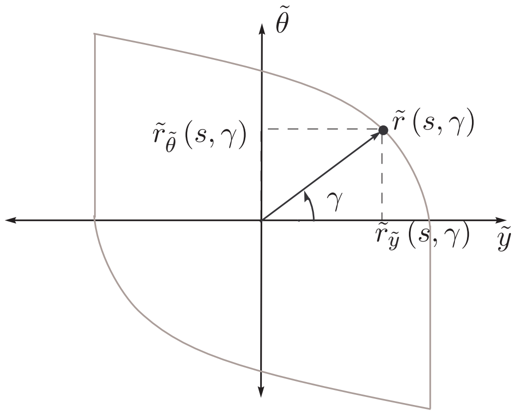
D.5 Policy Validation

Implicit in the constrained optimization approaches defined above is the existence of a valid input; that is, the set of valid inputs, $\mathcal{U}_c$, is not empty. During the policy instantiation, the policy must be validated to insure that such an input exists for all poses over the cell. To start, we assume that the ρ-terms used to bound the cells and define the sliding surface is larger than the minimum defined by the input constraints.

D.5.1 Collision Free

For a cell to be valid, it must be collision free; that is, it must be contained in the free pose space. This thesis uses the cell boundary normal to define a mapping from cell boundary to expanded cell boundary based on the body shape. As the cell boundary is piecewise smooth with well-defined surfaces, the expanded cell used to verify safety is well defined. This section presents the calculation of the boundary normal used for policy validation; for details on the collision tests see Appendix C.

As the cell is a tube around the curve $\hat{g}(s)$, this suggests a cylindrical parameterization of the cell boundary surface. Let $s \in [0,1]$ be the length along the cell defined as before, and γ be an angle around the local $\mathscr{F}_{\tilde{x}}$-axis measured relative to the local $\mathscr{F}_{\tilde{y}}$ axis. Let $\tilde{r}(s,\gamma)$ denote the three-vector specifying the cell boundary in the local frame; by definition, $\tilde{r}_{\tilde{x}}(s,\gamma) = 0$. See Figure D.12 for details. In the local frame, the boundary point $\tilde{r}(s,\gamma)$ is determined for the piecewise differentiable curves that define the cell boundary. The boundary point in the pose space is given by $g_c(s,\gamma) = \hat{g}(s) + \mathscr{F}(s) \cdot \tilde{r}(s,\gamma)$. For some of the limiting surfaces, for example the Lyapunov-like level set, the width boundary, and the upper bound at $\pm\frac{\pi}{2}$, $\tilde{r}$ is available in closed form given s and γ; for the $\tilde{\theta}_U$ and $\tilde{\theta}_L$, $\tilde{r}$ is determined numerically. Given $\tilde{r}$ the derivatives are determined in closed form.



$\tilde{\theta}$
$\tilde{r}_{\tilde{\theta}}(s,\gamma)$
$\tilde{r}(s,\gamma)$
γ
$\tilde{r}_{\tilde{y}}(s,\gamma)$
$\tilde{y}$

Figure D.12: PF cell boundary definitions for defining parameterized representation.

The normal of the parameterized surface representation is given by $n(g_c(s,\gamma)) = D_s g_c \times \mathcal{D}_\gamma g_c$. The derivatives in terms of s and γ are calculated according to the product rule; the derivatives of each component are well defined almost everywhere. Note that, $D_\gamma \mathscr{F} = \mathbf{0}$ and $D_\gamma \hat{g} = \mathbf{0}$ by definition, so that $D_\gamma g_c = \mathscr{F} \cdot D_\gamma \tilde{r}(s,\gamma)$. The terms $D_s \hat{g}$ and $D_s \mathscr{F}$ are defined by the planar curve used to specify the cell. At the interface between limiting surfaces, the derivative $D_\gamma \tilde{r}$, and hence the normal, is not well defined. The term $D_s \tilde{r}$ is zero along the cell boundary surface not limited by the Lyapunov-like function; along the surface limited by the Lyapunov-like function it is well defined. Thus, the normal can be calculated across the surface patches defined by individual curves. The parameterized representation yields a parameterized representation for the expanded cell used for collision testing.

D.5.2 Finite-time Convergence

The policies defined above guarantee convergence to the goal set in finite time. The input $u = \mathbf{0}$ is not allowed in the constrained input set $\mathcal{U}_c$. Thus, the kinematic systems are always moving. In the case of an Ackermann steered car with significant steering error, the finite steering rate will bring the system into the valid control zone $|\phi_{\text{err}}| < \phi_{\text{margin}}$ in finite time. For all the systems described above, the pose velocities are such that $\dot{\bar{s}} < 0$ for all valid inputs. This is due to the pose limitation of $\left|\tilde{\theta}\right| < \pm\frac{\pi}{2}$ and the non-zero control input. Thus, the system will reach $\bar{s} = 0$ in finite time. Note, the steering policy does not guarantee convergence to $\left\{\tilde{y}, \tilde{\theta}\right\} = \{0, 0\}$ at $s = 0$; only that the system is guaranteed to cross the $\mathscr{F}_{\tilde{y}}$-$\mathscr{F}_{\tilde{\theta}}$ plane that defines the cell goal set. Conditional invariance guarantees that the policy stays within the cell, and is therefore within the goal set boundary.

D.5.3 Conditional Invariance

Invoking the steering control policy over the cell induces motion that moves along the $\hat{g}(s)$ curve toward $s = 0$; to show that the system enters the goal set and safely remains in the cell, the conditional invariance requirement must be satisfied on the cell boundary.

This property is dependent on the cell parameters chosen. Generally, for the basic line-segment based cells shown in Figure D.7 can be verified geometrically provided $\tilde{\rho}, \tilde{\rho}_U, \tilde{\rho}_L$ are properly defined. The Lyapunov-like limiting surface, as do arc-based cells in general, use numerical validation using the steering policy.

Verifying the conditional invariance requires knowledge of the surface normal across the cell boundary; the normal is calculated as described in Section D.5.1. The parameterized representation allows the boundary poses and normals to be checked for conditional invariance using the policies defined below. Since the parameters are piecewise smooth, then surfaces are amenable to numeric validation.

For the cell to be conditionally positive invariant, the induced velocity along the cell boundary, excluding the goal set, must be inward pointing. That is, the $n(g) \cdot \dot{g} < 0$ where $n(g)$ is the outward pointing normal defined for the cell boundary. Thus, the conditional invariance requirement is restated as a input constraint, $n(g) \cdot A(q) \bigcap \mathcal{U} \neq \emptyset$. In other words, along the cell boundary, the control policy design must chose $u \in n(g) \cdot A(q) \bigcap \mathcal{U}$.

Given the cell boundary parameterized by ζ and γ, conditional invariance requires that

$$n\left(g\left(\zeta,\gamma\right)\right)^T A\left(g\left(\zeta,\gamma\right), r\right) u < 0 \tag{D.5}$$

for all ζ and γ over the cell boundary. Let $\Phi_{\Xi_i}(g, r)$ denote the action of the policies designed above such that $u = \Phi_{\Xi_i}(g, r)$. Define

$$L\left(\zeta,\gamma\right) = n\left(\zeta,\gamma\right)^T A\left(\left(g\left(\zeta,\gamma\right), r_{\text{des}}\right)\right) \cdot \Phi_{\Xi_i}\left(g\left(\zeta,\gamma\right), r_{\text{des}}\right) , \tag{D.6}$$

where r_{des} represents the shape variable specified by the control policies. For Ackermann-steered systems, $r_{\text{des}} = r_{\text{des}}(g) = (*, \phi_{\text{des}}(g))$ with $*$ meaning that ψ is arbitrary. For car-like/differential-drive systems $r_{\text{des}} = (*, *)$.

L is the result of the “best” input choice at a particular pose; the more negative the value of L, the more inward pointing the pose velocity is along the boundary. Although non-linear, the function $L(\zeta,\gamma)$ is piecewise smooth and generally “well-behaved” for the mapping $A(q)$. A valid cell satisfies the constraint

$$\max_{\zeta,\gamma} L\left(\zeta,\gamma\right) < 0\,. \tag{D.7}$$

Once the policy free parameters are chosen so that (D.7) is satisfied over the cell boundary, the policy satisfies composability requirements *(ii)* and *iii*.

For Ackermann steered cars, the steering margin $\phi_{\text{margin}}(g)$ is calculated during the conditional invariance tests.

D.5.4 Simple Inclusion Tests

The cells have simple inclusion tests, and thus satisfy composability requirement *(iv)* described in Section 3.3. A given robot pose g is mapped to the local pose error coordinates using simple calculations for the line-segment and arc-based cells. Recall, that the error coordinates are given by $e(g) = \left(\bar{s}(g), \tilde{y}(g), \tilde{\theta}(g)\right)$. The robot pose is in the cell if $0 \leq \bar{s} < 1$, $-\rho_{L_w} < \tilde{y} < \rho_{U_w}$, $\tilde{\theta}_L(\tilde{y}) < \tilde{\theta} < \tilde{\theta}_U(\tilde{y})$, and $V\left(\bar{s}, \tilde{y}, \tilde{\theta}\right) < V_{\text{cell}}$.

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 policies can use a more conservative inclusion test during policy switching by decreasing the policy width parameters ρ_{L_w} and ρ_{U_w}, which shrinks the ‘tube’ around the curve $\hat{g}(s)$.

D.6 Conclusion

PF policies satisfy the composability requirements, and are therefore composable in our hybrid control framework. The policies encode basic behaviors for robot navigation: turning in arcs and moving straight. The policies admit tractable validation tests for instantiating policies.

The path following approach that these policies are based on is more general than just line-segments and circular arcs defined here. A natural extension would define the policies for cycloids and continuous curvature arcs [95]. These would allow smoother transitions between policies; the difficulty comes in determining the $\bar{s}$ for a given pose. The line-segments and arcs can do this in closed form; continuous curvature arcs would require polynomial root finding techniques.

Appendix E

‘SQ’ Style Control Policies

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 chapter next discusses how the four composability requirements from Section 3.3 will be verified. These are discussed before the specific control design in order to define some terms used in the control design. The chapter concludes with a specific control design for these cells.

E.1 Cell Definition

This policy defines cells such that they fan out from the goal set using a generalization of a superquadric surface [51]; hence, the name ‘SQ’. The cell is defined relative to a frame attached to the goal set center as described in Section 5.3.

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

$$p\left(\zeta,\gamma\right) = \begin{bmatrix} -\zeta \\ \rho(\zeta,\gamma)\cdot\left(\cos\beta\,\cos\gamma - c\,\sin\beta\,\sin\gamma\right) \\ \rho(\zeta,\gamma)\cdot\left(\sin\beta\,\cos\gamma + c\,\cos\beta\,\sin\gamma\right)\end{bmatrix} \in \mathbb{R}^3\,. \tag{E.1}$$

The equations in parenthesis encode an ellipse with eccentricity $0 < c < 1$ rotated by β about the central (x') axis relative to the positive y'-axis, as shown in Figure E.1-a. With $0 < c < 1$, the conditional invariance velocity constraint $n\left(g\right)\cdot\dot{g} < 0$ requires that $-\frac{\pi}{2} < \beta < 0$. The function $\rho\left(\zeta,\gamma\right)$ governs the radius in the local cylindrical coordinate system as the system moves by ζ along the negative x' axis, as shown in Figure E.1-b. This representation is an extension to the standard superquadric representation because ρ may be a function of both ζ and γ [51]. The goal set is defined at $\zeta = 0$. A point $p\left(\zeta,\gamma\right)$ on the cell boundary in the local frame is mapped to the pose space as $g\left(\zeta,\gamma\right) = g\left(p\left(\zeta,\gamma\right)\right)$ using the generic cell transformation (5.2) from Section 5.3.

There is freedom in defining $\rho\left(\zeta,\gamma\right)$ provided the velocity constraint $n\left(g\right)\cdot\dot{g} < 0$ 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 E.1-b. In the portion corresponding to the funnel, let $\rho_f\left(\zeta,\gamma\right)$ be a monotonically increasing function in ζ that governs cell growth as ζ increases away

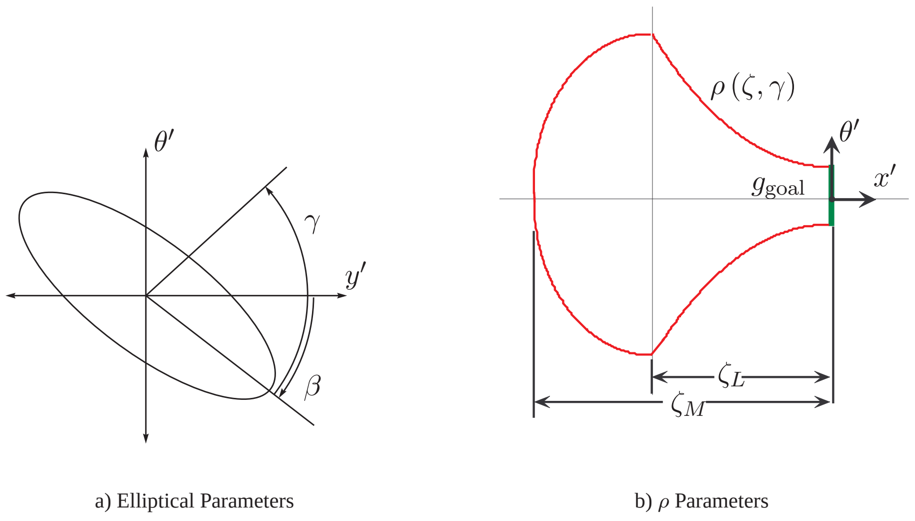


Figure E.1: Schematic representations of the generic SQ cell. Some of the important parameters are labeled.

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

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

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 E.2.

The first cell shape, as defined by $\rho_f = \rho_{f_1}$, results in the simple symmetric funnel shapes shown in Figure E.2-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 E.1-a. The composability requirements from Section 3.3 will dictate how much the cell can grow, and its shape by limiting the parameter values.

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 E.2-b. Let

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

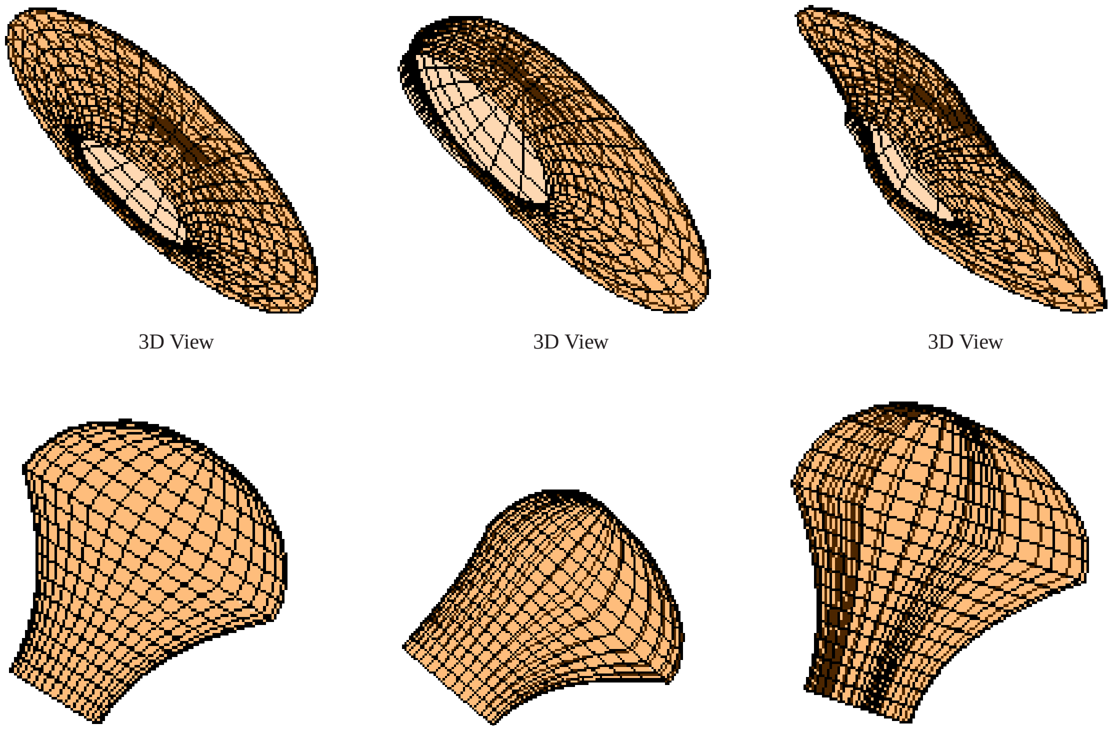

3D View
3D View
3D View
a) Cell using ρ_{f_1}
b) Cell using ρ_{f_2}
c) Cell using ρ_{f_3}

Figure E.2: Example cells for each ρ_f showing 3D and x-y views.

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 E.2-c. This cell is designed to generate a more aggressive turn than the first two cells allow for a given input set. Let

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

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.

E.2 Policy Validation

Given the classes of generic cells defined by the three ρ_f functions, it must be shown that particular instantiations of the cells satisfy the composability requirements given in Section 3.3.

E.2.1 Collision Free

We verify that the cell is contained in the free pose space, and is therefore safe, using the expanded cell approach described in Appendix C. The test is for a specific cell using a particular choice of cell parameter values, $\{R_o, R_e, R_r, c, \beta, \zeta_L, \zeta_M\}$ and optionally $\{\gamma_a\}$ or $\{K_{g_1}, \gamma_{g_1}, \sigma_{g_1}, K_{g_2}, \gamma_{g_2}, \sigma_{g2}\}$. Using the parameterized representation of the cell boundary given in (E.1), the approach described in Appendix C generates a representation of $R(\Xi_i)$. The set $R(\Xi_i)$ is tested for intersection with any obstacle. If intersection occurs, the cell parameter values must be modified.

The surface normal required for the expanded cell calculations is well defined over the parametric surface patches that define the cell boundary. The normal is calculated for the given set cell parameter values as $n(\zeta,\gamma) = \frac{D_\zeta g \times D_\gamma g}{\left\| D_\zeta g \times D_\gamma g \right\|}$. Both $g(\zeta,\gamma)$ and $n(\zeta,\gamma)$ are piecewise smooth functions. Given the implicit representation of the robot body boundary, the cell boundary point is analytically mapped to a point in $R(\Xi_i)$, which allows the collision tests outline in Appendix C.

E.2.2 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 3.3 and Section 5.4.

For the cell to be conditionally positive invariant, the induced velocity along the cell boundary, excluding the goal set, must be inward pointing. That is, the $n(g) \cdot \dot{g} < 0$ where $n(g)$ is the outward pointing normal defined for the cell boundary. Thus, the conditional invariance requirement is restated as a input constraint, $n(g) \cdot A(q) \bigcap \mathcal{U} \neq \emptyset$. In other words, along the cell boundary, the control policy design must chose $u \in n(g) \cdot A(q) \bigcap \mathcal{U}$.

Given the cell boundary parameterized by ζ and γ, conditional invariance requires that

$$n(g(\zeta,\gamma))^T A(g(\zeta,\gamma), r)\, u < 0 \tag{E.3}$$

for all ζ and γ over the cell boundary, and $q = (g, r)$. Let $\Phi_{\Xi_i}(g, r)$ denote the action of the as yet undefined control policy such that $u = \Phi_{\Xi_i}(g, r)$. Define

$$L(\zeta,\gamma) = n(\zeta,\gamma)^T A((g(\zeta,\gamma), r_{\text{des}})) \cdot \Phi_{\Xi_i}(g(\zeta,\gamma), r_{\text{des}})\ , \tag{E.4}$$

where r_{des} represents the shape variable specified by the control policies.

L is the result of the “best” input choice at a particular pose; the more negative the value of L, the more inward pointing the pose velocity is along the boundary. Although non-linear, the function $L(\zeta,\gamma)$ is piecewise smooth and generally “well-behaved” for the mapping $A(q)$. A valid cell satisfies the constraint

$$\max_{\zeta,\gamma} L(\zeta,\gamma) < 0\,. \tag{E.5}$$

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 (E.5) is satisfied for a given set of cell parameter values. Figure E.3 shows a typical constraint surface for the cell shown in Figure 5.9-a. 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.

Once the policy free parameters are chosen so that (E.5) is satisfied, the conditional invariance requirement is satisfied. The companion requirement of finite time convergence is discussed with the specific policy designs.

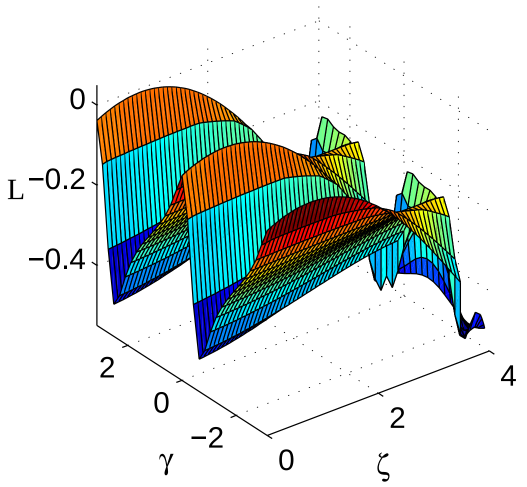


Figure E.3: Constraint surface for $L(\zeta, \gamma)$ from (E.4).

E.2.3 Simple Inclusion Tests

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 $\{\zeta_g, \gamma_g, \rho_g\}$, where ζ_g is the distance from g_{goal} along the central axis, γ_g is the angle relative to the ellipse major axis, and ρ_g 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 $0 < \zeta_g < \zeta_M$; if $\zeta_g \leq 0$ or $\zeta_g \geq \zeta_M$ then the point is outside the cell. If the point passes the first test, then find the cell boundary point,

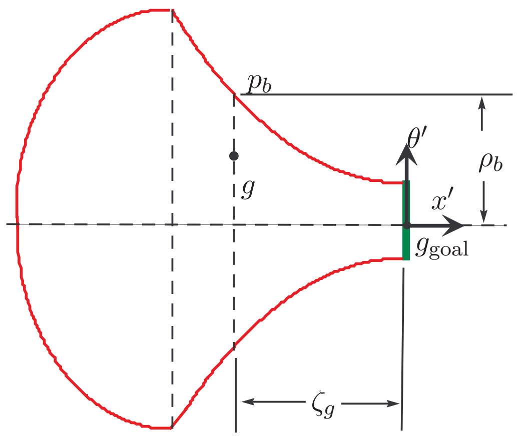


Figure E.4: Corresponding boundary points. Given a point g, determine its local cylindrical coordinates, $\{\zeta_g, \gamma_g, \rho_g\}$, and find the corresponding point $p_b = \{\zeta_g, \gamma_g, \rho_b\}$ on the cell boundary.

$p_b = \{\zeta_g, \gamma_g, \rho_b\}$ that corresponds to $g = \{\zeta_g, \gamma_g, \rho_g\}$, where ζ_g and γ_g are the same and ρ_b is the radius of the boundary point along the vector defined by ζ_g and γ_g. These points are shown in Figure E.4. From (E.1), the value of ρ_b is given as

$$\begin{aligned} \rho_b\left(\zeta_g, \gamma_g\right) &= \left\| \begin{bmatrix} \rho(\zeta_g, \gamma_g) \cdot (\cos\beta\, \cos\gamma_g - c\, \sin\beta\, \sin\gamma_g) \\ \rho(\zeta_g, \gamma_g) \cdot (\sin\beta\, \cos\gamma_g + c\, \cos\beta\, \sin\gamma_g) \end{bmatrix} \right\| \\ &= \rho\left(\zeta_q, \gamma_q\right) \frac{\sqrt{1 + c^2 - (c^2 - 1)\cos\left(2\gamma_g\right)}}{\sqrt{2}}\,. \end{aligned} \tag{E.6}$$

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

E.3 Policy Design

This chapter discusses two potential control designs for these SQ cells. Provided the conditional invariance properties are satisfied, it is possible to define a sliding surface, and use the variable structure control approach of Section D.4. In this section, we focus on a different technique based on level set control. At present this approach is limited to systems where the mapping $A\left(q\right)$ does not depend on the shape variables; to stress this point, this section abuses notation and writes $A(g) = A\left(q\right)$.

To define the control law, we use a family of level sets based on the cell boundary parameterization given in (E.1). 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 $\tilde{g}\left(g\right) = \{\zeta_g, \gamma_g, \rho_g\}$.

Recast the cell definition equations given in (E.1) and (E.2) 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 $0 \le \zeta'_L \le \zeta_g \le \zeta'_M \le \zeta_M$. The family of level sets used for control are defined by (E.1) with $\rho\left(\zeta_g, \gamma_g\right) = \rho_L\left(\zeta_g, \gamma_g\right)$ where

$$\rho_L\left(\zeta_g, \gamma_g\right) = \rho_f\left(\zeta'_L, \gamma_g\right) \frac{\sqrt{\left(\zeta'_M, -\zeta'_L,\right)^2 - \left(\zeta_g - \zeta'_L,\right)^2}}{\zeta'_M, -\zeta'_L,}\,. \tag{E.7}$$

That is, the control level set is governed the by (E.2), with $\rho_f\left(\zeta'_L, \gamma_g\right)$ 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 $\zeta_g = 0$ and $\gamma_g \in (-\pi, \pi]$ corresponds to the policy goal set. Increasing ζ'_M, while fixing $\zeta'_L = 0$, generates a family of level sets for $0 \le \zeta_g \le \zeta'_M$ that grow out from the goal; these are termed the *inner* level sets, as shown in Figure E.5. 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 E.5. Thus, given $\tilde{g}\left(g\right) = \{\zeta_g, \gamma_g, \rho_g\}$, 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, $\rho_L\left(\zeta_g, \gamma_g\right) = \rho_g$; thus values for ζ'_M and ζ'_L that satisfy

$$\rho_f\left(\zeta'_L, \gamma_g\right) \frac{\sqrt{\left(\zeta'_M - \zeta'_L\right)^2 - \left(\zeta_q - \zeta'_L\right)^2}}{\zeta'_M - \zeta'_L} - \rho_q = 0 \tag{E.8}$$

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 (E.8). If the configuration is within the outer family of level sets, as shown in

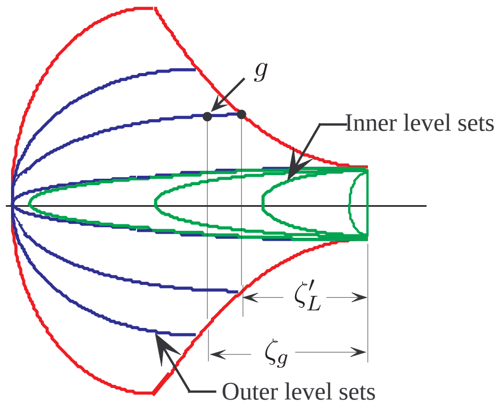


Figure E.5: Level set definition for control

Figure E.5, then $\zeta'_M = \zeta_M$ and we must determine the value of ζ'_L that satisfies (E.8). Unfortunately, ζ'_L cannot be found in closed form. Fortunately, (E.8) 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 $n(\zeta_g, \gamma_g)$ is defined as in Section E.2.1 using (E.1) and (E.7). The simplest constrained optimization is

$$u^* = \arg\min_{u \in \mathcal{U}} \left[n(\zeta_g, \gamma_g) \cdot A(g(\zeta_g, \gamma_g))\, u \right] \quad \text{s.t. } n(\zeta_g, \gamma_g) \cdot A(g(\zeta_g, \gamma_g))\, u < 0. \tag{E.9}$$

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, $[n(\zeta_g, \gamma_g) \cdot A(g(\zeta_g, \gamma_g))\, u]$, can be augmented by a simple quadratic term. Let $\Phi_{\Xi_i}(g)$ denote the control policy using this strategy; that is $u = \Phi_{\Xi_i}(g) = u^*$, where u^* is defined by (E.9).

Using $u = \Phi_{\Xi_i}(g)$ 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 the conditional invariance requirements for (E.5) at every point in the cell, which guarantees that a solution to (E.9) exists. The body velocity $\dot{g} = A(g) \cdot \Phi_{\Xi_i}(g)$ 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 conditional invariance under $\Phi_{\Xi_i}(g)$, all interior level sets will also satisfy conditional invariance. This can be checked for various values of ζ_M and ζ_L during deployment.

E.4 Conclusion

SQ policies satisfy the composability requirements, and are therefore composable in our hybrid control framework. The policies are naturally shaped like funnels; however, the conditional invariance constraints limit the size and shape of the cells. Thus, they can be limited, and not able to define as tight a turning radius when compared to the PF policies.

Appendix F

Robots Used in Demonstrations

The hybrid control approach advocated in this thesis, which is described in Chapter 3 and Chapter 5, has been validated on several robot systems in different environments. Although the techniques apply to any single-bodied purely-kinematic system, this thesis uses three particular robot models. Two are real differential-drive robots that vary in size and shape; the third is a simulated conventional Akermann-steered rear-wheel drive car-like system. This appendix provides details on the particular robots by discussing the body size and shape, the control limitations, and modeling assumptions.

F.1 'Deminer' Differential-drive Robot

The first tests use the standard differential-drive robot shown in Figure F.1, which has a convex, roughly elliptical body shape. This is called the 'Deminer' robot. To simplify calculations of $R(\Xi_i)$, the composite set of points occupied by the robot body over all positions and orientations in the cell, the robot body and wheels are tightly approximated by an analytic ellipse centered in the body coordinate frame; this is shown in Figure F.2. The length of the major and minor axes of the bounding ellipse are 1.12 and 0.68 meters respectively.

The robot is driven by the larger front wheels. The wheels are individually controlled using PID velocity loops that send commands to individual H-bridge amplifiers. The control loop runs at 100 Hz, with velocity feedback given by encoders attached to the motors. The system assumes that the velocity control is fast relative to the system dynamics.

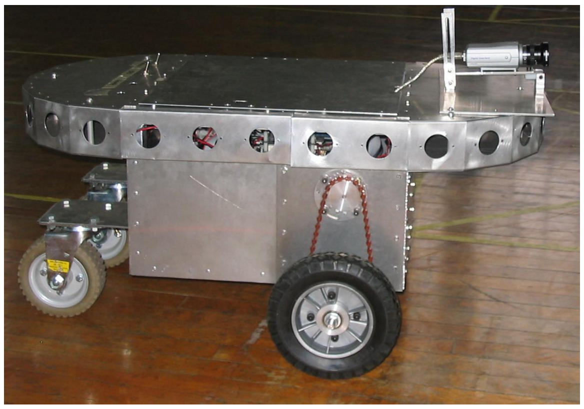


Figure F.1: 'Deminer' laboratory robot

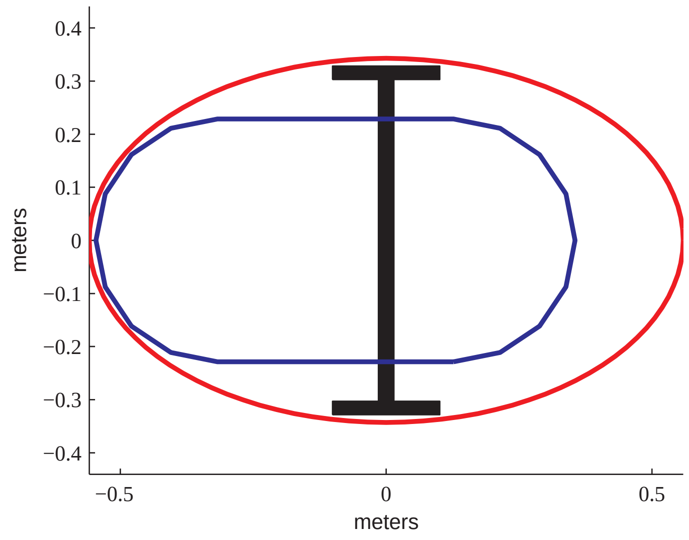


Figure F.2: ‘Deminer’ laboratory robot bounding ellipse

The hybrid control policies use the simplified kinematic unicycle model for control. The connection is given by

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

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. Given a specified input, the desired wheel velocities are determined. The policies could have used the differential-drive model as the models are interchangeable by a simple change of coordinates; the kinematic unicycle model was initially chosen because of the direct analogy to body velocities.

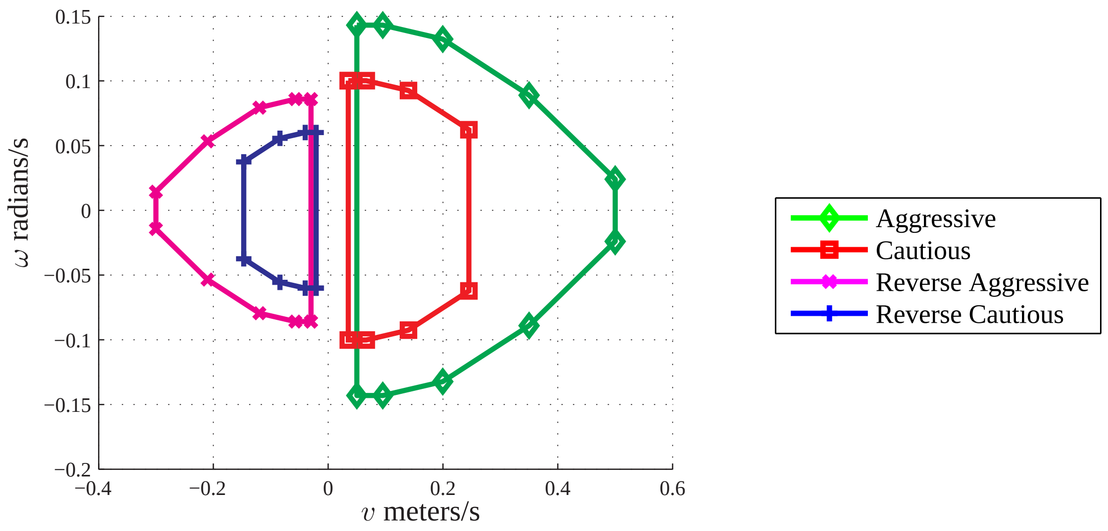


Figure F.3: Four sets of bounded steering inputs used in Deminer experiments. .

The inputs, which induce the body pose velocity $\dot{g} = A(q)\, u$, are chosen from one of four bounded input sets shown in Figure F.3. The system changes direction by switching between "Forward" and "Reverse" input sets; each having an "Aggressive" and "Cautious" set of values. The numerical values are based on the velocity limits of the motors, and scaled back for cautious modes and the reverse input sets. Each policy is associated with a specific choice of input set, which allows the system to account for local conditions. Although the robot is capable of zero-radius turns, the input bounds are chosen to model a conventional car-like system with bounded steering. The input bounds are convex polygons; this restriction is for computational convenience and allows the convex optimization technique discussed in Appendices D and E. The restriction to polygons is not fundamental.

The robot sensors consist of optical encoders attached to the drive motors and a single forward-facing camera. The encoders are used to provide velocity feedback, which in turn provides the inputs for a dead-reckoning pose estimation. An existing vision based localization scheme was tested on the system, but the accuracy was insufficient to allow for indoor navigation in cluttered environments. For this reason, the early experiments used pure dead-reckoning for localization.

The robot software executive is coded in C++ within the modular RHexLib framework [119]. The system includes modules for velocity calculations and localization, as well as the PID velocity control modules. The executive itself is coded as a RHexLib module. On initialization, the module reads a set of configuration files that specify the input bounds, the policy definitions, the initial pose of the robot, and the desired goal policy. The executive then uses an implementation of the minimax D*-lite algorithm to order the policies [81]. During execution, the executive chooses the active policy and calculates the desired inputs v and ω. These are converted to individual wheel velocity commands that are passed to the PID velocity control modules. The entire system runs at a 100 Hz update rate on a Pentium-based PC-104 computer operating under the QNX operating system.

F.2 'LAGR' Differential-drive Robot

The LAGR robot, shown in Figure F.4, is used in the majority of actual experiments shown in this thesis. An extended Kalman filter based localization system uses encoder-based velocity measurements to predict the system pose based on dead-reckoning, and updates the pose estimate based estimates of range and bearing to known landmarks [22]. The update step also uses a pose change estimate based on an inertial measurement unit. The positions of the landmarks relative to the robot are estimated by a custom set of four stereo camera pairs. The stereo cameras are mounted at 90 degrees from one another, which provides a near 360 degree field of view. The robot sensing package also includes the Global Positioning System (GPS) antenna shown in Figure F.4; GPS signals are not used for indoor localization.

This localization approach provides reasonably accurate pose estimates for the experiments. During execution, the pose estimate does experience jumps of several centimeters as new landmarks are detected and old ones disappear from view. These jumps are disturbances to the control system. The system does not have ground truth comparisons, so the effectiveness is qualitatively judged by the long runs shown in the experiments of Chapter 6. The overall performance is consistent across multiple loops around the hallways.

The robot mechanical system is a standard differential-drive system with two drive wheels and two smaller rear caster wheels. Figure F.5 shows the system with the bounding convex polygon used for estimating $R(\Xi_i)$. The system is approximately 1.23 meters by 0.79 meters with approximately 1 meter of extension behind the drive wheels. The area swept by the robot during turns is significant; the collision tests developed in this thesis allow policies to be deployed that guarantee safety.

The robot is driven by two 24 volt motors attached to the large pneumatic tires in the front. These tires provide motive force to the system. Two rear caster wheels provide stability, but also act as a significant disturbance as discussed later in this section. The 4096 count/revolution encoders, which are used to provide velocity estimates, are attached directly to the drive wheel axles. This

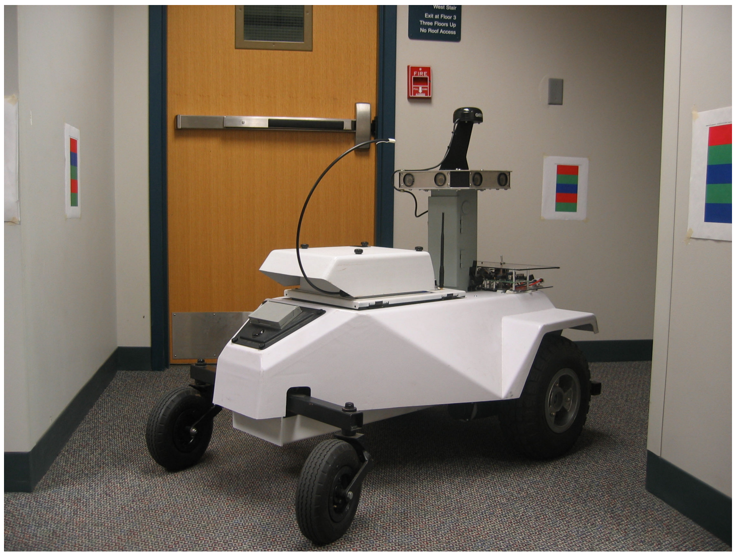


Figure F.4: The LAGR robot uses four stereo cameras to perform vision based localization while navigating. Three color-coded landmarks, which are used by the vision based localization system, are visible in the image.

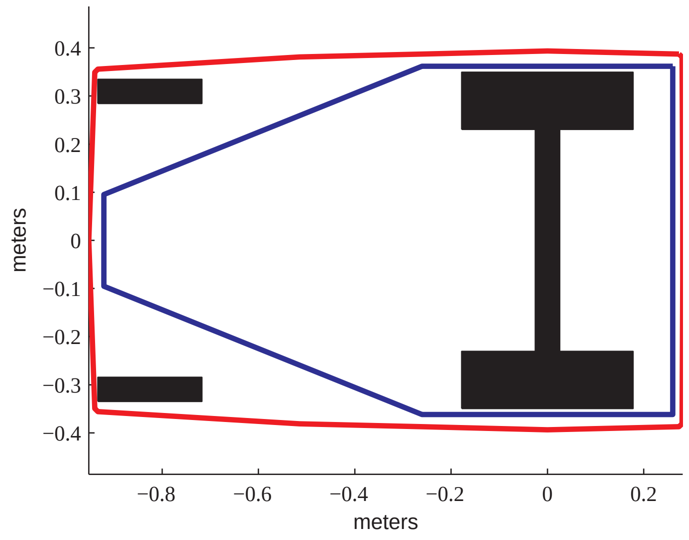


Figure F.5: The 'LAGR' robot with bounding polygon shown. The bounding polygon is used to guarantee the safety of a policy without being overly conservative.

provides significantly less velocity resolution than if the encoders were attached to the motor and had the advantage of the added gear ratio. Velocity estimates are noisy at lower speeds.

The velocity estimates are based on the difference between encoder counts divided by the elapsed time between encoder count samples. To smooth the estimates, the calculations use an adaptive windowing technique. The velocity calculations are made at 100 Hz; however, if the system is moving slowly the encoder count difference may be taken with respect to an encoder sample taken up to 0.05 seconds previously. The window size is based on the number of elapsed encoder counts; at least 50 counts uses a 0.01 second window, at least 33 counts uses a 0.02 second window, at least 20 counts/0.03 seconds, and 10 counts/0.04 seconds, to a maximum of 0.05 second window. This windowing technique smoothes the data somewhat, but introduces some delay with respect to the true velocity.

The motor control hardware is customized. The standard LAGR motor controller is limited in resolution to 7-bits and a maximum update rate of approximately 62 Hz. This control ability proved insufficient for two reasons. First, the noisy velocity signals limited the control gains that could be applied. Second, the caster wheel drag acts as a significant disturbance during turns. These factors resulted in significant overshoot when turning, and prevented the robot from operating according to the kinematic assumption used in the policy design. To improve the control response, the original motor control amplifier was replaced with a microprocessor and two H-bridge amplifiers. This allow the control signal resolution to be increased to 10-bits, and the update rate to be increased to 100 Hz.

With the customized motor controller, the motor velocities are governed via a low-level PID control loop for each wheel. The PID loop runs at 100 Hz, and specifies a PWM duty-cycle to the H-bridges connected to each motor. The velocity control used a PID approach with additional feed-forward terms. Due to the relatively noisy velocity signals, a gain scheduling approach was used based on the desired wheel velocities.

While this customized control has better response than the standard LAGR control hardware, the LAGR robot still did not provide accurate velocity tracking. Figure F.6 shows the wheel velocities and body velocities for the first 10 seconds of the experiment shown in Figure 6.15-d. The velocity

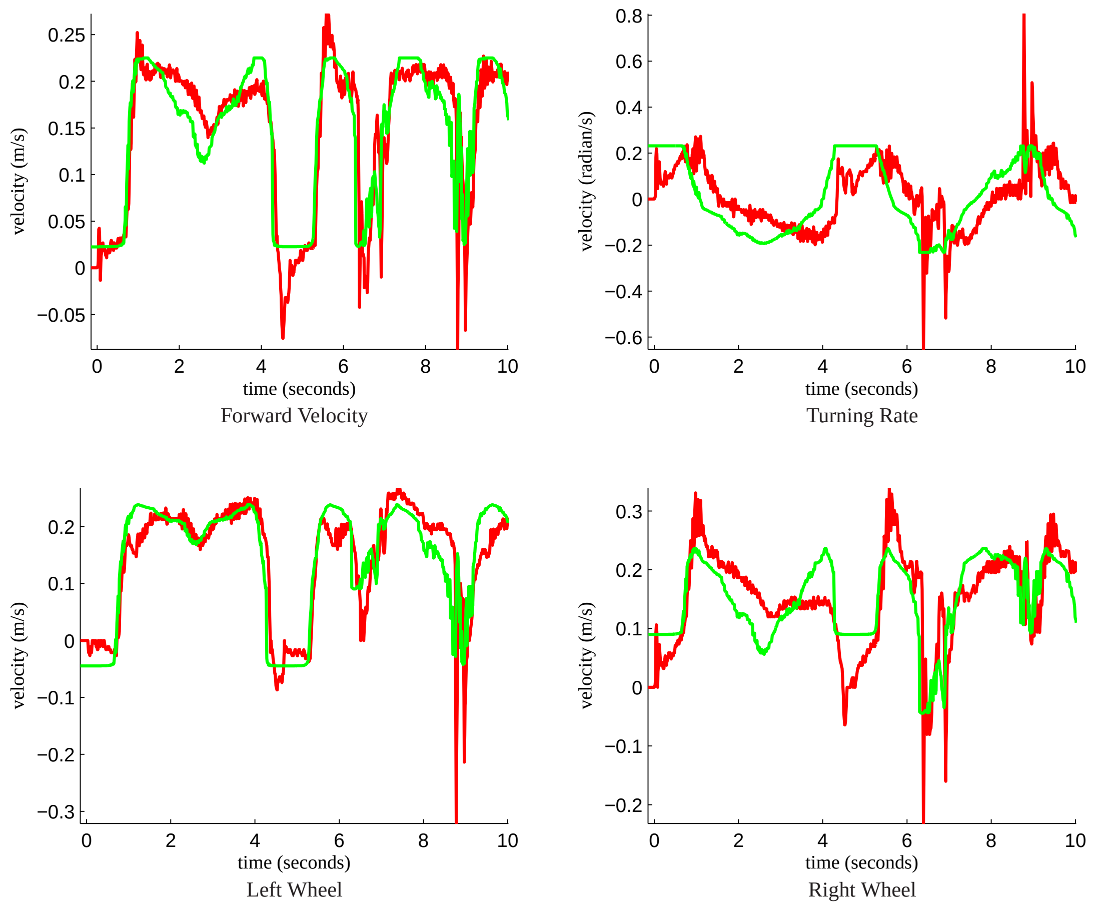


Figure F.6: The velocity response for the first 10 seconds of an actual run of our hybrid control policy as shown in Figure 6.15-d. The smoother light colored lines represent the commanded velocity, the noisier darker lines show the velocity estimates based on encoder feedback.

tracking errors can be traced to noisy velocity estimates, limited PID control gains, and significant disturbances during turns due to the caster wheels and wide pneumatic tires. While the robot is a robust mechanical platform for its intended purpose of outdoor navigation in rough terrain, the system is not well suited for precision high-speed navigation in confined environments. Although the system violates the assumptions of our kinematic control policies, the mostly successful experiments of Chapter 6 show the robustness of the hybrid control approach advocated in this thesis.

The control approach is the same as for the Deminer robot. The hybrid control policies use the kinematic unicycle model with forward velocity and turning rate as inputs, and then calculates the desired individual wheel speeds that are passed to the PID control loops. The hybrid control polices chose from one of twelve bounded input sets and one ‘Halt’ policy. Figure F.7 shows the collection of input sets $\mathcal{U}_i$ used for the simulations; for the experiments the forward velocities are further reduced by 50 percent due to the control difficulties highlighted above. The ‘Straight’ input sets are used for PF policies based on straight line segments; these input sets reduce the aggressiveness of steering to avoid oscillations caused by the caster wheel drag during aggressive turns. The ‘Turn’ input sets allow more aggressive turns and reduce forward speeds; this allows the system to encode “slow down while turning” for policies that do aggressive turns.

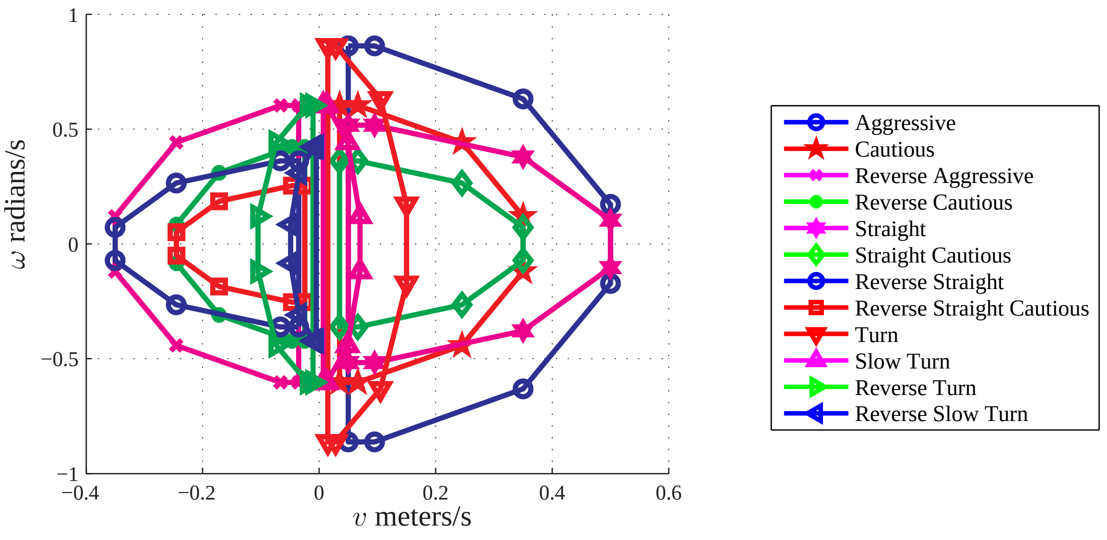


Figure F.7: Twelve sets of bounded steering inputs

The robot computing is divided between four separate computers connected via hardwired network connections. The low-level control is governed by a process running on one computer. This controller process handles velocity estimate calculations, accepts velocity commands, and calculates the voltage duty cycle using the local PID loops. The duty cycle is communicated to the microprocessor via a serial link; the microprocessor governs the pulse width modulation of the H-bridges. Two computers are responsible for running the four vision processes, one for each stereo pair, that detect landmarks and estimate range and bearing. The bearing, range, covariance estimates, and associated landmark ID are passed to the localization process running on the fourth computer. The localization process uses velocity updates from the controller process to predict the robot motion. The landmark information from the vision processes maps to a known location via the landmark ID; this information along with data from the IMU is used in a correction step of the Kalman filter based pose estimation. The estimated pose is passed to the robot software executive process running on the same computer. Upon receipt of a new pose estimate, the software executive determines the appropriate local policy, calculates a new control input command, and passes the desired wheel velocity to the control process.

The robot software executive process, which is coded in C++, runs within the standard LAGR process manager. The executive coordinates reading the configuration files, initializing the robot pose, and coordinating policy switching. The same executive is used for both order-based execution using D*-lite and automata-based execution using a synthesized automaton. The automata are synthesized before execution, and read in from a configuration file.

The simulations of the LAGR robot shown in Chapter 6 are run with the same executive code. Instead of using the vision based localization and PID control, the velocity commands are passed to a function that does numerical integration to provide localization. The function simulates delays between the desired velocity calculation and the actual velocity, and simulates delays between the actual pose and the estimate passed to the executive functions. This simulation allows the actual robot code to be tested prior to execution on the robot, and also allows simulations of the policies with ideal kinematics. The simulations assume a 50 Hz control update, with a numerical integration of configuration velocities at a 0.001 second time step. A delay between control calculation and velocity response of 0.02 seconds was modeled.

F.3 Ackermann Steered Car-like Robot

The parking and traffic simulations in Chapter 6 use a model of an Ackermann steered car. This is one of the more complex kinematic models for single bodied nonholonomic systems. This section provides an overview of the specific model used.

The vehicle simulations take place in an environment shown in Figure 6.27. The roadway lanes and parking spaces in these two urban blocks are sized using "green practice" standards [1], which result in narrower roadways than standard highways. The parking spaces are 6.86×2.44 meters. The roadway lanes are 5.49 meters from centerline to curb, leaving just under 3.05 meters for the driving lane. To make the parking problem more challenging, the robot system is modeled on a "mini-van", which is a relatively large vehicle as shown in Figure F.8. The size is approximately 5.1 meters by 1.85 meters.

The system model is that of a kinematic Ackermann-steered car as described in Appendix A.4.3. This rear-wheel drive model assumes the reference point is attached to the center of the rear axle. The mapping $A(q) : \mathcal{U} \to T_q\mathcal{G}$ is given by

$$A(q) = \begin{bmatrix} R\cos\theta & 0 \\ R\sin\theta & 0 \\ \frac{R}{L}\tan\phi & 0 \end{bmatrix}, \tag{F.1}$$

where $R = 0.406$ meters is the drive wheel radius and $L = 3.00$ meters is the wheelbase. The inputs are $u = \begin{bmatrix}\dot{\psi} & \dot{\phi}\end{bmatrix}^T$ where $\dot{\psi}$ is the rear drive wheel speed and $\dot{\phi}$ the rate of change of the steering angle. Note the dependence of $A(q)$ on the steering angle.

The steering angle is limited, which limits the turning rate of the vehicle. The vehicle turning circle, defined as the circle traced by the wheel farthest from the center of turning, is approximately 11.2 meters in diameter. This translates to a steering angle limit of 0.66 radians or 37.8 degrees. The

[1] http://www.nahbrc.org/greenguidelines/userguide_site_innovative.html

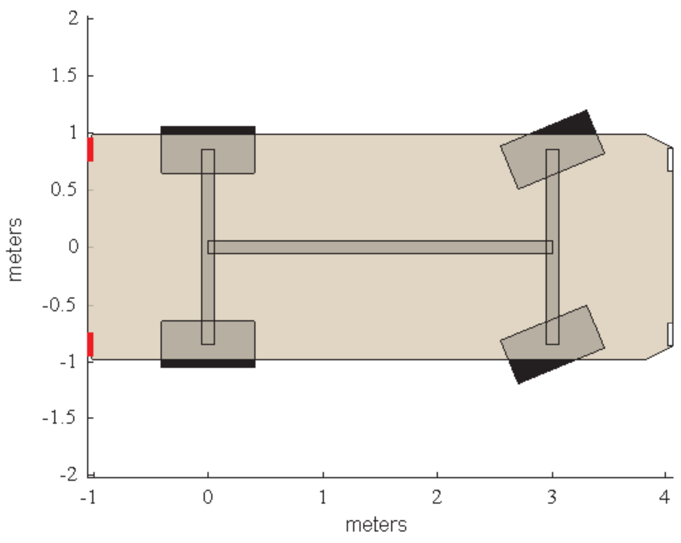


Figure F.8: The body plan of the Ackermann steered mini-van. The inputs are the drive speed of the rear wheels and the rate of steering angle change. The robot body is bounded by a polygon, which is used in the estimates of $R(\Xi_i)$. The extension of the tires beyond the robot body is ignored.

steering angle is further limited at higher speeds to enforce a safety factor that encodes "slow down while turning." Figure F.9 shows four different bounded sets that are associated with four different input sets.

The system uses four different input sets for the local control policies, as shown in Figure F.10. During execution, the hybrid control policies chose a drive speed and steering rate that is applied to the system. To enforce the steering limits shown in Figure F.9, the input set is further constrained during execution. The steering angle bounds are converted to rate limits using the formula $\dot{\phi}_i = \frac{\phi_i - \phi}{\Delta t}$, where $\dot{\phi}_i$ is a vertex of the new rate bounds, ϕ_i is a vertex on the steering angle bounds from Figure F.9, and Δt is the nominal control update rate. The resulting vertices are converted to

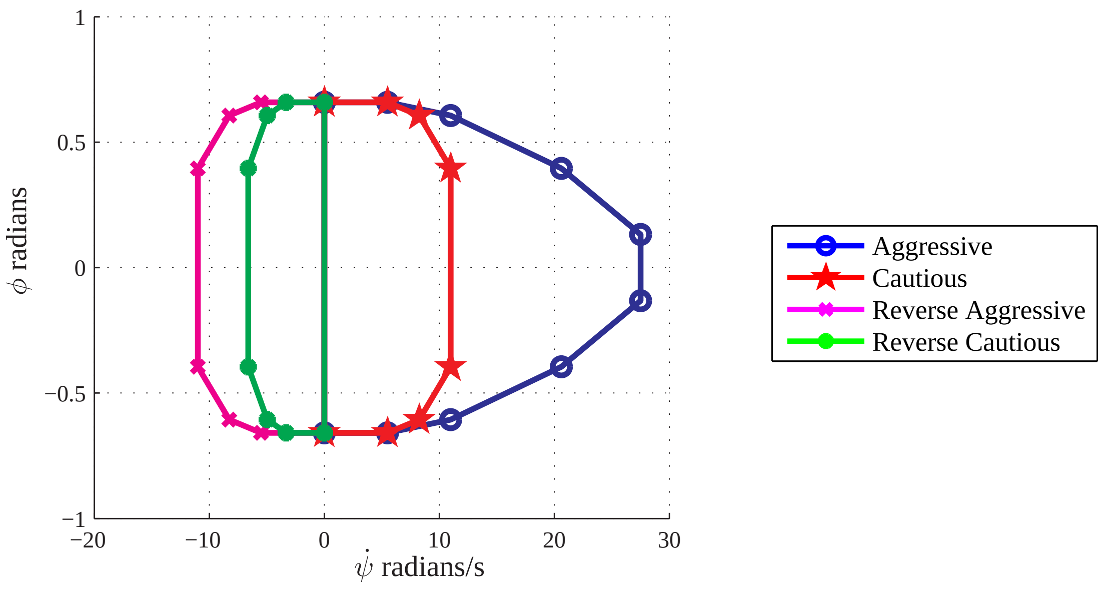


Figure F.9: The steering angle is limited as a function of forward velocity for safety.

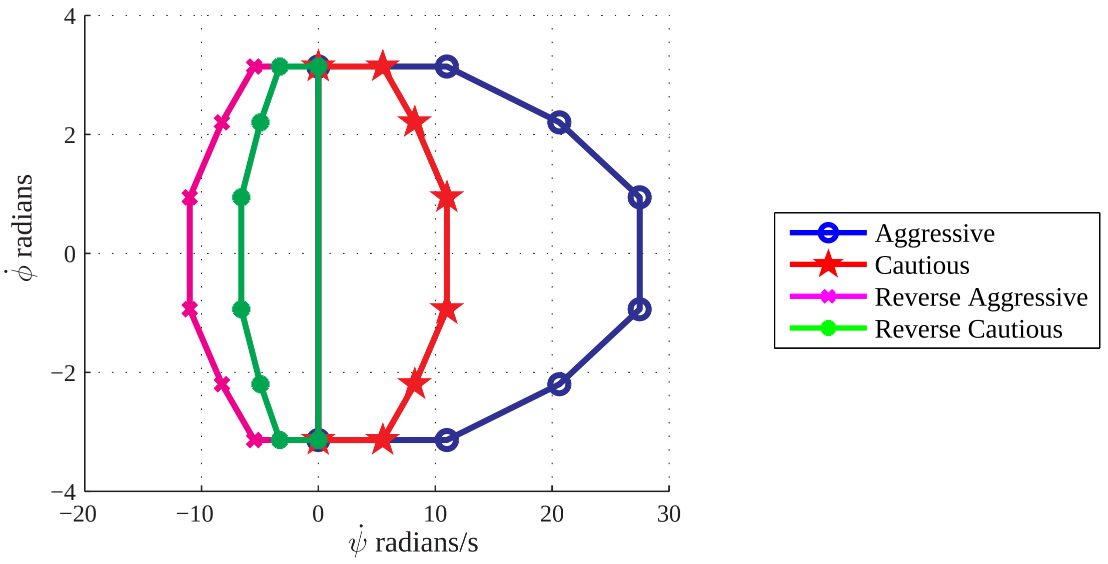


Figure F.10: The steering rate as a function of forward speed.

half-space constraints and added to the input constraints for the control optimization. These added velocity constraints guarantee that steering angle limit is not exceeded during the next time step.

The Chapter 6 simulations are executed using code written in Matlab$^{\text{TM}}$. The simulations assume the pose is fully known, and the control is exact without delay. The simulations assume a 100 Hz control update, with a numerical integration of configuration velocities at a 0.001 second time step.

Bibliography

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

[2] Fabio Ancona and Alberto Bressan. Patchy Vector Fields and Asymptotic Stabilization. *ESAIM: Control, Optimization, and Calculus of Variations*, pages 445–471, 1999.

[3] 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.

[4] A. Balluchi, A. Bicchi, A. Balestrino, and G. Casalino. Path Tracking Control for Dubin's Car. In *IEEE International Conference on Robotics and Automation*, pages 3123–3128, Minneapolis, MN, 1996.

[5] J. Barraquand and J.C. Latombe. Nonholonomic Multibody Robots: Controllability and Motion Planning in the Presence of Obstacles. *Algorithmica*, 10:121–155, 1993.

[6] Richard Bellman. *Dynamic Programming*. Princeton University Press, 1957.

[7] 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.

[8] William M. Boothby. *An Introduction to Differentiable Manifolds and Riemannian Geometry* . Academic Press, 1986.

[9] Johann Borenstein and Y. Koren. The Vector Field Histogram - Fast Obstacle Avoidance for Mobile Robots. *IEEE Transactions on Robotics and Automation*, 7(3):278–288, June 1991.

[10] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, March 2004.

[11] Michael S. Branicky. Stability of Switched Hybrid Systems. In *Proceedings of the 33rd Conference on Decision and Control*, pages 3498–3503, 1994.

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

[13] 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.

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

[15] 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.

[16] Karl Brauer. All Lined Up. http://www.edmunds.com/ownership/techcenter/articles/43858/article.html, November 2007.

[17] O. Brock and L. E. Kavraki. Decomposition-Based Motion Planning: A Framework for Real-Time Motion Planning in High-Dimensional Configuration Places. In *Proceedings of The 2001 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1469–1475. IEEE Press, May 2001.

[18] 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.

[19] Rodney A. Brooks. A Robust Layered Control System for a Mobile Robot. *IEEE Transactions on Robotics and Automation*, 2:14–23, 1986.

[20] Randall E. Bryant. Graph-Based Algorithms for Boolean Function Manipulation. *IEEE Transactions on Computers*, 35(8):677 – 691, August 1986.

[21] 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.

[22] 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.

[23] Edmund. M. Clarke and O. Grumberg D.A. Peled. *Model Checking*. MIT Press, Cambridge, Massachusetts, 1999.

[24] David C. Conner, Hadas Kress-Gazit, Howie Choset, Alfred A. Rizzi, and George J. Pappas. Valet Parking Without a Valet. In *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*, October 2007.

[25] 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.

[26] David C. Conner, Alfred A. Rizzi, and Howie Choset. Construction and Automated Deployment of Local Potential Functions for Global Robot Control and Navigation. Technical Report CMU-RI-TR-03-22, Carnegie Mellon University, Robotics Institute, Pittsburgh, Pennsylvania, USA, 2003.

[27] C. I. Connolly and R. A. Grupen. Nonholonomic Path Planning Using Harmonic Functions. Technical Report 94-50, UMass Computer Science, 1994.

[28] DARPA. Urban Grand Challenge, 2007. [Online; accessed 19-November-2007].

[29] 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.

[30] 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.

[31] R. DeCarlo, S.H. Zak, and G.P. Matthews. Variable Structure Control of Nonlinear Multivariable Systems: A Tutorial. *Proceedings of the IEEE*, 76(3), March 1988.

[32] 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.

[33] 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.

[34] E. Allen Emerson. Temporal and Modal Logic. In *Handbook of theoretical computer science (vol. B): formal models and semantics*, pages 995–1072. MIT Press, Cambridge, MA, USA, 1990.

[35] Lawrence C. Evans. *Partial Differential Equations*. American Mathematical Society, Providence, RI, 1998.

[36] 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.

[37] 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.

[38] Dieter Fox, Wolfram Burgard, and Sebastian Thrun. The Dynamic Window Approach to Collision Avoidance. *IEEE Robotics and Automation Magazine*, March 1997.

[39] Th. Fraichard. Motion Planning for Autonomous Car-Like Vehicles. Ercim News (42):26-28, July 2000.

[40] Th. Fraichard and H. Asama. Inevitable Collision States: A Step Towards Safer Robots? In *Proc. of the IEEE-RSJ Int. Conf. on Intelligent Robots and Systems*, Las Vegas, NV (US), October 2003.

[41] E. Frazzoli, M.A. Dahleh, and E. Feron. Maneuver-Based Motion Planning for Nonlinear Systems with Symmetries. *IEEE Transactions on Robotics*, 21(6):1077–1091, December 2005.

[42] C. E. Garcia, D. M. Prett, and M. Morari. Model Predictive Control: Theory and Practice – A survey. *Automatica*, 25(3):335–348, 1989.

[43] Geoffrey Gordon. Stable Function Approximation in Dynamic Programming. In *Proceedings of IMCL '95*, 1995.

[44] Luc C. G. J. M. Habets, Pieter J. Collins, and Jan H. van Schuppen. Reachability and Control Synthesis for Piecewise-affine Hybrid Systems on Simplices. *IEEE Trans. Automatic Control*, 51(6):938–948, June 2006.

[45] Luc C. G. J. M. Habets and Jan H. van Schuppen. Control of Piecewise-Linear Hybrid Systems on Simplices and Rectangles. *Hybrid Systems: Computation and Control, Lecture Notes in Computer Science*, 2034:261–274, 2001.

[46] Luc C. G. J. M. Habets and Jan H. van Schuppen. A Control Problem for Affine Dynamical Systems on a Full-dimensional Polytope. *Automatica*, 40(1):21–35, January 2004.

[47] Sven Hedlund. *Computational Methods for Optimal Control of Hybrid Systems*. PhD thesis, Lund Institute of Technology, May 2003.

[48] 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.

[49] Jonathan W. Hurst, Joel Chestnutt, and Alfred A. Rizzi. Design and Philosophy of the BiMASC, a Highly Dynamic Biped. In *IEEE Conference on Robotics and Automation*, April 2007.

[50] 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.

[51] Ale$\hat{v}$s Jakli$\hat{v}$c, Ale$\hat{v}$s Leonardis, and Franc Solina. *Segmentation and Recovery of Superquadrics*, volume 20 of *Computational Imaging and Vision*. Kluwer, Dordrecth, 2000. ISBN 0-7923-6601-8.

[52] Rune M. Jensen, Randy E. Bryant, and Manuela M. Veloso. SetA*: An Efficient BDD-Based Heuristic Search Algorithm. In *Proceedings of AAAI-2002*, Edmonton, Canada, August 2002.

[53] 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.

[54] S. Kajita, F. Kanehiro, K. Kaneko, K. Fujiwara, K. Harada, K. Yokoi, and H. Hirukawa. Biped Walking Pattern Generation by Using Preview Control of Zero-Moment Point. In *IEEE Conference on Robotics and Automation*, pages 1620– 1626, September 2003.

[55] 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.

[56] 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.

[57] L. H. Keel and S. P. Bhattacharyya. Robust, Fragile, or Optimal? *IEEE Transactions on Automatic Control*, 42(8):1098–1105, August 1997.

[58] J. M. Keil. Decomposing a Polygon into Simpler Components. *SIAM J. Comput.*, 14:799–817, 1985.

[59] Scott D. Kelly. *The Mechanics and Control of Robotic Locomotion with Applications to Aquatic Vehicles*. PhD thesis, California Institute of Technology, 1998.

[60] Oussama Khatib. Real-Time Obstacle Avoidance for Manipulators and Mobile Robots. *International Journal of Robotics Research*, 5(1):90–98, 1986.

[61] M. Kloetzer and C. Belta. A Fully Automated Framework for Control of Linear Systems from LTL Specifications. In *9th International Workshop on Hybrid Systems: Computation and Control*, Santa Barbara, California, 2006.

[62] M. Kloetzer and C. Belta. Managing Non-determinism in Symbolic Robot Motion Planning and Control. In *IEEE International Conference on Robotics and Automation (ICRA)*, Rome, Italy, 2007.

[63] Daniel E. Koditschek. Some Applications of Natural Motion Control. *ASME Journal of Dynamic Systems, Measurement, and Control*, 113(4):552–557, December 1991.

[64] Daniel E. Koditschek. The Control of Natural Motion in Mechanical Systems. *ASME Journal of Dynamic Systems, Measurement, and Control*, 113(4):548–551, December 1991.

[65] Ilya Kolmanovsky and N. Harris McClamroch. Developments in Nonholonomic Control Problems. *IEEE Control Systems*, 15:20–36, December 1995.

[66] A. N. Kolmogorov and S. V. Fomin. *Introduction to Real Analysis*. Dover Publications, Inc., 1975.

[67] Hadas Kress-Gazit. personal communication, 2007.

[68] Hadas Kress-Gazit, Georgios E. Fainekos, and George J. Pappas. Where's Waldo? Sensor-Based Temporal Logic Motion Planning. In *IEEE International Conference on Robotics and Automation*, Rome, Italy, 2007.

[69] B. H. Krogh. A Generalized Potential Field Approach to Obstacle Avoidance Control. In *SME Conf. Proc. Robotics Research: The Next Five Years and Beyond*, Bethlehem, Pennsylvania, August 1984.

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

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

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

[73] 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.

[74] Tom Lauwers, George A Kantor, and Ralph Hollis. A Dynamically Stable Single-Wheeled Mobile Robot with Inverse Mouse-Ball Drive. In *Proceedings of the 2006 IEEE International Conference on Robotics and Automation (ICRA '06)*, pages 2884 – 2889, May 2006.

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

[76] 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.

[77] Steven M. LaValle and James J. Kuffner. Randomized Kinodynamic Planning. In *IEEE International Conference on Robotics and Automation*, volume 1, pages 473–479, 1999.

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

[79] Daniel Liberzon and A. Stephen Morse. Basic Problems in Stability and Design of Switched Systems. *IEEE Control Systems*, pages 59–70, October 1999.

[80] Maxim Likhachev, David Ferguson, Geoffrey Gordon, Anthony (Tony) Stentz, and Sebastian Thrun. Anytime Dynamic A*: An Anytime, Replanning Algorithm. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, June 2005.

[81] Maxim Likhachev and Sven Koenig. Speeding Up the Parti-Game Alorithm. In *Advances in Neural Information Processing Systems 15*. MIT Press, 2003.

[82] Stephen. R. Lindemann, Islam I. Hussein, and Steven M. LaValle. Realtime Feedback Control for Nonholonomic Mobile Robots with Obstacles. In *IEEE Conference on Decision and Control*, San Diego, CA, 2006.

[83] 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.

[84] Stephen. R. Lindemann and Steven M. LaValle. Smooth Feedback for Car-Like Vehicles in Polygonal Environments. In *IEEE International Conference on Robotics and Automation (ICRA)*, Rome, Italy, 2007.

[85] 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*, Sendai, Japan, September 2004.

[86] T. Lozano-Peŕez, Matthew T. Mason, and R. H. Taylor. Automatic Synthesis of Fine-motion Strategies for Robots. *International Journal of Robotics Research*, 3(1):3–23, 1984.

[87] Matthew T. Mason. *Mechanics of Robotic Manipulation*. The MIT Press, 2001.

[88] G. Ayorkor Mills-Tettey, Anthony (Tony) Stentz, and M Bernardine Dias. DD* Lite: Efficient Incremental Search with State Dominance. In *Twenty-First National Conference on Artificial Intelligence (AAAI-06)*, pages 1032–1038, July 2006.

[89] 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.

[90] Jorge Cortés Monforte. *Geometric, Control and Numerical Aspects of Nonholonomic Systems*. Springer, 2002.

[91] Monte Carlo and Quasi-Monte Carlo Methods, November 2007.

[92] Rémi Munos. A Study of Reinforcement Learning in the Continuous Case by the Means of Viscosity Solutions. *Machine Learning Journal*, 40:265–299, 2000.

[93] Remi Munos and Andrew Moore. Variable Resolution Discretization in Optimal Control. *Machine Learning*, 49, Numbers 2/3:291–323, November/December 2002.

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

[95] W. L. Nelson. Continuous Curvature Paths for Autonomous Vehicles. In *IEEE International Conference on Robotics and Automation*, volume 3, pages 1260–1264, Scottsdale, AZ, 1989.

[96] J.P. Ostrowski and J.W. Burdick. The Geometric Mechanics of Undulatory Robotic Locomotion. *International Journal of Robotics Research*, 17(7):683–702, 1998.

[97] 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.

[98] Kaustubh Pathak and Sunil K. Agrawal. An Integrated Path-Planning and Control Approach for Nonholonomic Unicycles Using Switched Local Potentials. *IEEE Transactions on Robotics*, 21(6):1201–1208, December 2005.

[99] Per-Olof Persson and Gilbert Strang. A Simple Mesh Generator in MATLAB. *SIAM Review*, 46(2):329–345, June 2004. Available online at http://www-math.mit.edu/ persson/mesh/.

[100] 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.

[101] Martin Peternell and Friedrich Manhart. The Convolution of a Paraboloid and a Parametrized Surface. *Journal for Geometry and Graphics 7*, pages 157–171, 2003.

[102] A. Pnueli and E. Shahar. The TLV System and its Applications, 1996.

[103] 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.

[104] Elon Rimon and Daniel E. Kodischek. Exact Robot Navigation Using Artificial Potential Functions. *IEEE Transactions on Robotics and Automation*, 8(5):501–518, October 1992.

[105] 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.

[106] Robert C. McOwen. *Partial Differential Equations: Methods and Applications*. Pearson Education, Prentice Hall, 2nd edition, 2003.

[107] Bartek Roszak and Mireille E. Broucke. Necessary and Sufficient Conditions for Reachability on a Simplex. *Automatica*, 42(11):1913–1918, November 2006.

[108] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 1995.

[109] 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.

[110] 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.

[111] 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.

[112] Elie Shammas. *Generalized Motion Planning for Underactuated Mechanical Systems*. PhD thesis, Carnegie Mellon University, 2006.

[113] Reid Simmons. The Curvature-Velocity Method for Local Obstacle Avoidance. In *IEEE International Conference on Robotics and Automation*, April 1996.

[114] Anthony (Tony) Stentz. The Focussed D* Algorithm for Real-Time Replanning. In *Proceedings of the International Joint Conference on Artificial Intelligence*, August 1995.

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

[116] 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.

[117] John A. Thorpe. *Elementary Topics in Differential Geometry*. Springer, 1978.

[118] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic Robotics*. MIT Press, Boston, MA, September 2005.

[119] D.E. Koditschek U. Saranli, M. Buehler. RHex: A Simple and Highly Mobile Hexapod Robot. *The International Journal of Robotics Research*, 20(7):616–631, July 2001.

[120] 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.

[121] Douglas B. West. *Introduction to Graph Theory, 2nd ed.* Prentice-Hall, Englewood Cliffs, NJ, 2000.

[122] Wikipedia. Markov Decision Process — Wikipedia, The Free Encyclopedia, 2007. [Online; accessed 13-November-2007].

[123] 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.