SAFE CONTROL STRATEGIES FOR HOPPING OVER UNEVEN TERRAIN

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF MECHANICAL ENGINEERING
AND THE COMMITTEE ON GRADUATE STUDIES
OF STANFORD UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Brian Howley
September 2006

I certify that I have read this dissertation and that in my opinion it is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

Mark Cutkosky
(Principal Adviser)

I certify that I have read this dissertation and that in my opinion it is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

Chris Gerdes

I certify that I have read this dissertation and that in my opinion it is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

Ken Waldron

Approved for the University Committee on Graduate Studies:

Abstract

Legged robots have long been proposed as a means of locomotion in unstructured environments. In recent years, researchers have developed small, robust, biologically inspired robotic systems that demonstrate an impressive ability to traverse diverse terrain and climb over obstacles at high speeds. However, much of the analysis to support this work has assumed level or slowly varying terrain. This dissertation explores the concept of safety as applied to hybrid dynamical systems as an analysis tool for legged robots in highly variable terrain. The robot must negotiate bound but unknown changes in terrain height between steps without stumbling or violating joint constraint limits. The problem is cast as a non-cooperative game where the robot attempts to maintain safe operation despite adversarial changes in the environment. The magnitude of the environmental disturbance that the robot can safely withstand is a measure of the system's "ruggedness" which is proposed as a possible design criterion. This thesis first considers a single legged vertical hopping robot with feedback control. Mechanical properties (stiffness and damping) which maximize ruggedness over variable terrain are identified. The thesis then considers timer based or open loop control of the vertical hopping robot. The addition of the timer state variable increases the dimensionality of the hopper problem from two states to three and requires a numerical tool for determination of the maximal invariant safe subset and least restrictive control. The development of this tool is a major topic of the dissertation. The tool is applicable to a broad but restricted class of hybrid systems. Results are given for a robot hopping vertically open loop over level ground, stairs, and variable terrain. Extension to planar running and multiple leg systems is discussed.

Acknowledgements

I am extremely pleased and proud of my affiliation with Stanford University. The years here have been a wonderful experience. However, as a part time student under the Stanford Honors Cooperative Program, I've had a whole lot more of those years than I imagined prior to starting my degree program!

I would like to thank my reading committee, Chris Gerdes and Ken Waldron, for their time, consideration, and comments, and Gunter Niemeyer for his participation in my defense. I would also particularly like to thank Claire Tomlin for agreeing to chair my defense on a last minute basis. Your work has been a big inspiration and the basis for my study. I am pleased to apply your methods and approach to a new domain. Finally, I want to thank my thesis advisor, Mark Cutkosky, for all his help through the many years. Thanks for agreeing to accept me as a PhD candidate, for supporting me during my one year as a full time student (my time here on campus was a wonderful experience by the way), and for all the help and support with my dissertation and research.

The bulk of this research was done on my own time, but my tuition was paid for by my employer, Lockheed Martin Space Systems Company, through the Stanford Honors Cooperative program. I am very grateful for the support. I would like to thank the administrators of this program and particularly Jerry Ockerman of Lockheed Martin for all his help with enrollment and registration and all the administrative aspects I was spectacularly bad at. I would further like to thank my colleagues and management at Lockheed Martin for their support.

Of course, the really important people here are my family. Sadly, my father passed away a couple of years ago and I know he would have been proud to see me, finally,

complete this dissertation. I am glad I can share this success with my mother, though I think this may be the only section she reads. Thank you both for your love and support. But the people who really had to put up with the late nights, the early mornings, the “Dad’s too busy to ..”, are my wife, Sandy, and my children, Matthew and Steven. In this document I can only say thank you, but that only begins to express my feelings.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Context and Motivation

Legged robots can operate in environments that wheeled or tracked vehicles of similar size cannot. Generally, these machines follow one of two fundamental approaches to the problem of legged locomotion. One approach is to plan foot placement to maintain static stability throughout the step. Examples of statically stable machines include the *Autonomous Suspension Vehicle* (Waldron, 1984, [49]), *Titan VII* (Hirose, et. al., 1997, [23]), *Dante II* (Bares and Wettergreen, 1999, [2]), and a stair climbing bipedal robot (Shih, 1999, [42]). These machines rely on a high degree of motion planning or tele-operation and require detailed knowledge of the environment. Another approach, pioneered by Raibert (1986, [38]) relies on dynamic stability. Dynamically stable systems run or hop by exchanging kinetic and potential energy in different phases of motion. Recent examples of dynamically stable running machines include *RHex* (Saranli, et. al., 2000, [41]), the *Sprawl* family of robots (Cham, et. al., 2000, [11], and Clark, et. al., 2001, [16] 2001), and *Tekken*, (Fukuoka, et. al., 2003, [18]). These robots are relatively fast, from one to several body lengths per second, and rely on passive mechanical properties to stabilize against disturbances or variations in the environment.

Raibert (1986, [38]) distilled the problem of running into a simple set of actions and performed experiments on single legged hopping machines. The machines consist

of a body with an articulated hinge joint and a pneumatic piston. The piston acts as both spring and thrust actuator and forces a pogo-stick style of locomotion. The problem of control was decomposed into separate tasks that controlled hopping height, forward travel,and posture (hip angle). The hopper was capable of hopping in place at different heights, running forward and backward, and switching between hopping and running gaits. Hodgins and Raibert (1991, [24]) extended the control algorithms to control step length by adjusting forward speed and demonstrated the ability of a two legged version of the hopper to run up and down a short flight of stairs.

Raibert was inspired by biologists investigating animal and human motion. Cavagna, et. al. (1977, [10]) showed that for a wide range of animals energy is stored in tendons and muscles when running, trotting, or galloping. The exchange between kinetic and potential energy was a key insight to Raibert's notion of dynamic stability. Biologists, in turn, were likewise influenced by Raibert's work. Blickhan (1989, [4]) describes a spring–mass model for animal running and considers the influence of physiological constraints on hopping frequency, peak reaction force, and contact displacement. Using the spring–mass model, Blickhan and Full (1993, [5]) illustrate striking similarities in the dynamics of legged locomotion for a diverse range of animals including cockroaches, quail, kangaroos, and humans.

Biologists began to explore the role of both the natural dynamics and the central nervous system in legged locomotion. Full, et. al. (1998, [19]) found only minor changes in cockroach muscle activation patterns over smooth and uneven terrain. The results suggested that the central nervous system does not actively respond to sudden perturbations. Kubow and Full (1999, [29]) investigated a model of cockroach motion in the horizontal plane that assumes leg forces are generated purely open loop. They show the dynamics are stable against external perturbations without active feedback. Full and Koditschek (1999, [20]) proposed a model for legged motion wherein the central nervous system generates feed–forward signals to coordinate muscle groups and the leg mechanical impedances provide restoring forces, termed 'preflexes' (Brown and Loeb, 1999, [8]), to reject disturbances. The feed–forward signals are generated by neural circuits called Central Pattern Generators (CPG). Full and Koditschek speculate that the role of neural feedback is state-event dependent. For example, the

central nervous system may signal a transition from stance to swing phases of motion but doesn't try to control individual muscles to effect this transition.

Building on these insights, researchers from the University of Michigan and Stanford University developed the *RHex*, (Saranali, et. al., 2000, [41]) and *Sprawl* (Cham, et. al., 2000, [11]) family of robots, respectively. Both robots are dynamically stabilized six legged machines and both are designed to reduce mechanical complexity. *RHex* is battery powered and self contained. The legs rotate full circle about horizontal axes and are servo-controlled to follow a clock driven trajectory which functions as a CPG. In the original version of *RHex* the legs were shaped to provide compliance primarily in the radial direction. Subsequent versions of *RHex* used modified legs for stair climbing (Moore, et. al., 2002, [33]), swimming, and bi-pedal running (Neville and Buehler, 2003, [36]). In contrast, the *Sprawl* family of robots use telescoping rather than rotating legs. The hip joints are rotationally compliant with a high level of damping. The neutral position of the hips can be adjusted to adopt a sprawl or more upright posture. The *Sprawlita* (Cham, et. al., 2000, [11]) member of the family is pneumatically driven and control is completely open loop. *Sprawlita* is able to traverse hip height obstacles while traveling at several body lengths per second. Cham (2001, [13]) and Bailey (2004 [1]) investigate adaptive control strategies for climbing or descending sloped terrain. More recently, *iSprawl* (Kim, et. al., 2004, [26]) is a smaller, self contained, and battery powered member of the Sprawl family capable of running at speeds of over 15 body lengths per seconds.

Despite design differences, both *RHex* and *Sprawl* borrow heavily from biological insights and both demonstrate very impressive speed and maneuverability over diverse terrain. Yet, there is a need for further understanding over how the dynamic interaction between robot and environment translates into robot design principles (Komsuoglu and Koditscheck, 2000, [28], and Cham, et. al., 2004, [12]). For example, Saranali, et. al. (2001, [40]) describe several failures during experimental trials with the RHex robot including broken legs. While failure is a valuable experimental result, the example illustrates that other than through exhaustive experimentation or simulation it is difficult to answer questions like:

– *At what speeds and over what range of surface irregularities can a robot*

successfully traverse?

- *How will changes in control, mechanical properties, or leg morphology affect performance?*
- *Under what conditions and to what extent will additional sensors be advantageous?*

A comprehensive framework for the analysis of the interaction between legged robots and a variable environment is required to answer these questions. This thesis takes the position that robot performance is ultimately constrained by safe operating limits and investigates the notion of safety as a design criterion. Here, safety is specified by legal values for the state variables describing robot dynamics. Safe operation is maintained under worst case environmental disturbances. Worst case disturbances and the corresponding optimal control are determined by application of game theory to the safety problem. The approach has several advantages, including: 1) design limitations such as maximum torque and joint deflections are treated explicitly, 2) only the general nature of the control (e. g. feedback versus open loop) and not the specific implementation is prescribed so that safety over a wide range of possible control implementations is determined, and 3) the method does not assume small perturbations from a stable, steady state cycle and is appropriate for analysis of motion over highly irregular terrain.

A central problem to studying the effects of uneven terrain is determining a meaningful method of analysis. Large terrain variations preclude steady state running trajectories required for a traditional stability analysis. Furthermore, perfect knowledge of the environment is generally not practical, so the method of analysis must accommodate uncertainty. One approach to uncertainty is to treat the problem as a random process. This approach requires repeated convolutions of probability density functions and is computationally prohibitive. Even if the approach was feasible, sensitivity to assumptions about the statistics of the terrain variation and sensor and actuator errors would make interpretation of any results extremely difficult. An alternative is to treat the problem as a game between between the robot and its environment. Rather than exist passively, the environment is assumed to act perversely

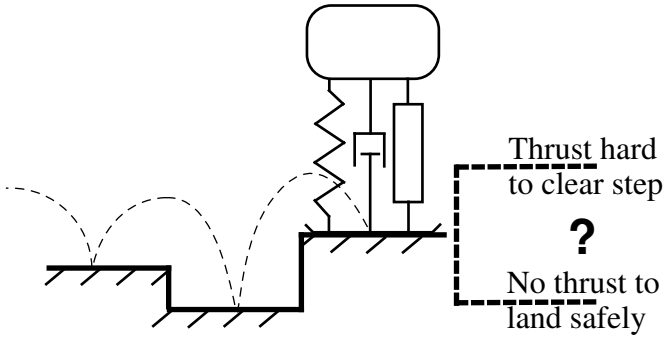


Figure 1.1: Hopping over uneven terrain cast as a game. The robot attempts to hop over uneven terrain without violating joint constraints or stumbling over obstacles. The environment is unknown to the robot but is assumed to vary in a way to force an unsafe condition.

but within some prescribed limits. A game theoretic approach transforms the uncertainty into a deterministic problem by considering worst case conditions (Bryson and Ho, 1975, [9]).

Consider the problem of a single legged robot hopping over an uneven surface as illustrated in Figure 1.1. The robot must jump high enough to clear or land on top of an obstacle during its ballistic phase. However, if the environment ahead is unknown, the change in elevation may be either higher or lower. If the robot applies maximum thrust and the elevation of the next step is lower, the robot may land with enough velocity to cause damage or lose control. On the other hand, if the robot applies less thrust and the next step in elevation is higher, it may fail to clear the step. In the context of a game, the robot attempts to stay within a safe operating regime while crossing over the terrain and the environment varies in a way that attempts to force the robot out of safe operation. The robot "wins" if it can negotiate the environment without violating constraints, and the environment "wins" otherwise. Because the robot's position and velocity during one step influence the position and velocity of succeeding steps, the game is played out over a sequence.

Game theoretic approaches to control and design problems have precedent. Isaacs (1965, [25]) developed differential games in the 1940's and 1950's to address military

problems such as the pursuit-evasion game. Church (1962, [15]) first proposed the controller synthesis problem as a solution to a game between a system and its environment. Bryson and Ho, [9], advocate a game theoretic approach as a conservative basis for controller design when little is known about the statistics of a disturbance. More recently, Lygeros, et. al. (1997, [32]), and Tomlin, et. al. (1998, [46], and 2000, [48]) use a game theoretic framework for the controller design of hybrid systems. A hybrid system has both discrete and continuous dynamics. For example, a hopping robot is a hybrid system because there are discrete changes in the robot's dynamics between contact and flight. The control problem is cast as a pursuit-evasion game where the controller wins if the systems remains within a safe subset of states for a specified period of time. The methodology searches for a feedback control law that guarantees that the system remains within the largest possible safe subset of the state space despite disturbances by the environment. The resulting control law is least restrictive in the sense that the control may take on any legal value except at the boundary of the subset where it must act to maintain safety. It can be used in combination with other control laws to achieve performance specifications under normal operating conditions and maintain safety when safety constraints would otherwise be violated. The approach is motivated by high confidence systems such as air traffic management and automated highway systems, but can be applied to other systems where safety can be expressed as a subset of the state space.

This thesis explores the notion of safety as a criterion for legged robot performance and examines the dependencies of hopping over uneven terrain on mechanical properties, control strategies, and environmental knowledge. The measure of safety can be quantified either by the size of the invariant safe subset or by the size of the environmental disturbances the system can safely tolerate. The maximum level of disturbance in which the system can safely operate is termed "ruggedness". Figure 1.2 shows the maximum safe terrain height variation for a single legged hopper as a function of normalized leg stiffness, 'k', and damping. The height variation is the maximum positive or negative change in ground level between successive hops, and the maximum safe variation in height is the ruggedness of the system. The results assume a maximum thrust capability equal to twice the robot's weight, a maximum

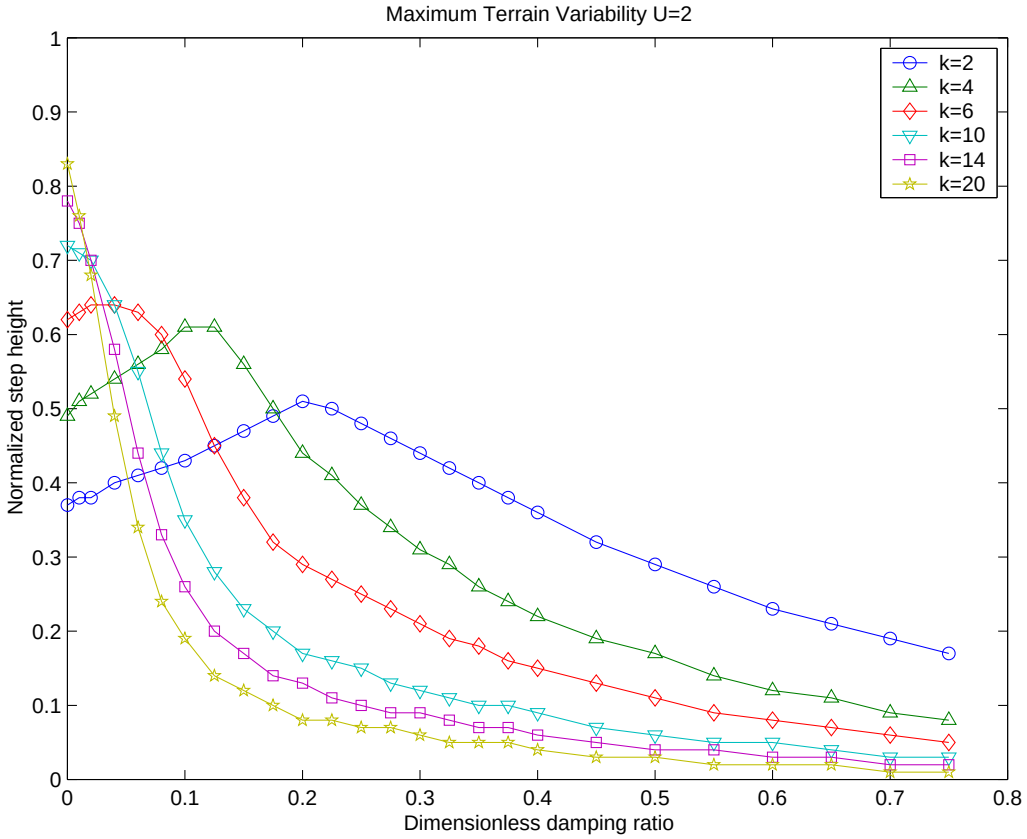


Figure 1.2: Hopper ruggedness as a function of leg stiffness and damping. The maximum safe variation in terrain height between steps is shown as a function of non-dimensional leg spring stiffness and damping. For zero damping, ruggedness is maximized at high spring stiffness but for even small levels of damping moderate to light spring stiffness is better.

compression of 50% of the unsprung leg length, and full state feedback control. Figure 1.2 shows that under these constraints the robot can negotiate terrain with a step to step variation in height of up to 80% of the unsprung leg length, but at very high stiffness and very low damping. At more realistic leg damping ratios, say between 0.1 and 0.3, maximum ruggedness is achieved with softer springs.

1.2 Thesis Overview

This thesis departs from previous work on legged robots by applying game theoretic principles to the problem of running over uneven terrain. The approach determines the largest possible subset of the state space in which the robot can continue to operate safely despite adverse environmental disturbances. The approach also determines the minimum level of control effort for safe operation. The emphasis is on problem formulation and analysis rather than experimentation.

Chapter 2 describes the dynamics of a single legged vertical hopper. The interaction between the robot and the environment is modeled as a hybrid system with discrete transitions between continuous phases of motion. A change in surface height between hops is allowed when the hopper is at the apex of flight, as if the robot were hopping in place on a treadmill with steps. The method for determining the maximal invariant safe subset developed by Tomlin, Lygeros, and Sastry (1998, [46], and 2000, [48]) is applied to the vertical hopper. Two previously unreported issues with the approach are addressed. One issue is that singular points, points in the state space where the time derivatives are zero, leave backward time propagation undefined and therefore must be avoided. The other issue is that the transition from one phase of motion to another may be influenced by the control or disturbance inputs. As control or disturbance values can change instantaneously, a mechanism must be introduced to prevent thrashing between modes.

The chapter continues by examining the effect of the robot's thrust capability and mechanical properties of stiffness and damping on terrain hopping ability. The metric adopted to describe terrain hopping ability is dubbed ruggedness: the maximum per step variation in terrain height the hopper can safely negotiate without stumbling or violating joint limits. Optimal values for stiffness and damping are a compromise between compression and thrust limited behaviors. With too much damping there is insufficient thrust to overcome obstacles. With too little damping the system is unable to protect against joint constraint violations. The best performance is generally achieved with moderate levels of stiffness and damping but optimal values vary with thrust.

Chapter 2 uses a graphical approach to determine the maximal invariant safe subset. The approach is intuitive and avoids the mathematics for determining optimal control and disturbance inputs. However this approach is insufficient for more complicated problems. Chapter 3 examines the mathematics underlying the optimization of the maximal invariant safe subset. The optimization formulated by Tomlin (1998, [47]) requires solution to a pair of inter-connected Hamilton–Jacobi equations. Numerical solution to these equations is complicated by the existence of 'shocks' or discontinuities that require sophisticated numerical techniques. Chapter 3 describes a restricted class of hybrid systems that allows the inter-connected optimizations to be separated and shows that the separate optimizations can be solved without numerical solution to the Hamilton–Jacobi equations. This result greatly simplifies the problem and motivates development of a numerical algorithm described in Chapter 4 and applied in Chapter 5.

Chapter 4 describes a numeric algorithm for the Maximal Invariant Safe Subset (MISS) based on the results in Chapter 3. The MISS algorithm uses a cell complex representation to define boundaries between safe and unsafe regions of the state space. The boundaries for a given phase of motion, for example the flight phase, are closed by subboundaries which define safe initial and exit conditions. After establishing an initial set of safe states and establishing predecessor-successor relationships between phases, the maximal invariant safe subset is found by iterating intersection and union operations on the initial and exit condition subboundaries between phases. Numerical integration with an optimal criterion for determining control and disturbance inputs is used to determine the continuous state space trajectories between the initial and exit condition subboundaries.

Chapter 5 applies the MISS algorithm to find the maximal invariant safe subset for a vertical hopper with timer based control. The problem is more complex than the closed loop problem described in Chapter 2 because the addition of a timer requires a third state variable. Three different environments are considered: level terrain, stair climbing, and variable terrain. On level terrain the algorithm finds stable and unstable orbits within the safe set. The results compare favorably with previous work reported by Koditschek and Bühler (1991, [27]), Ringrose (1997, [39]), and Cham (2000, [11]).

The results for stair climbing are similar to those for level ground but the size of the safe operating regime is smaller. However, for variable terrain variations as small as five percent of the unsprung leg length can always force the system to an unsafe state. The results over variable terrain are in contrast with the experimental success of clock driven multi–legged systems like *Sprawlita* (2001, [16]) and *Rhex* (2000, [41]). The discrepancy is due primarily to differences in the safety criteria for single versus multi–legged robots. Multi–legged systems are designed to operate when one or more legs stumble or slip, whereas single legged systems fail. The problem formulation for running and multi-legged systems is discussed in Chapter 6, but detailed analysis is beyond the scope of the present work. Chapter 5 concludes with a discussion of the numerical difficulties encountered with the MISS algorithm.

Chapter 6 describes extension of the MISS algorithm to more complex problems. Given the numerical and computational difficulties encountered in Chapter 5, solution to these problems is beyond the scope of the current work. However, the problem formulation demonstrates that the concept of safety can be applied to better understand the effects that sensors, control implementation, leg morphology, and robot posture have on the ability to negotiate highly variable terrain. First, two variations to the vertical hopper problem are considered. One variation is to allow the timer based control in Chapter 5 to adjust timer period and thruster duty cycle. This provides a mechanism for adaptive behaviors that may improve hopper ruggedness (see Cham, 2001, [13], and Bailey, 2004, [1]). A second variation is the addition of a measurement of the terrain. The measurement changes the game theoretic nature of the problem. The MISS algorithm can not provide a solution for worst case, but can show the efficacy of the measurement assuming the environment changes in an arbitrary rather than an adversarial manner.

Chapter 6 continues by considering the problem of planar running with single and multiple legs. The dynamics and possible phase transitions become more complex and additional constraints are added to the problem. An important new constraint is the onset of slipping at toe contact which is a function of the control thrust and hip torque.

Chapter 7 concludes this work with a brief summary of findings, conclusions, and

suggestions for future work.

1.3 Contributions

This dissertation draws heavily from the work of Tomlin, Lygeros, and Sastry (1997, [32], 1998 [46] and [47], and 2000, [48]) on the synthesis of controllers for hybrid systems. However, the application of their approach to the domain of legged robots has led to some refinements and the development of a numeric algorithm for the Maximal Invariant Safe Subset (MISS) that simplifies their approach. The MISS algorithm implicitly solves the Hamilton–Jacobi equations for optimal control avoiding the need for sophisticated numerical tools. The algorithm is restricted to a fairly broad class of hybrid systems and can be applied outside the domain of legged robots.

The main contributions of this thesis are described below:

- Formulation of the interaction between a legged robot and an uncertain environment as a game and application of optimal control theory, as described by Tomlin (1998, [47]) and Tomlin, et. al. (2000, [48]), to find the maximal subset of safe states and the corresponding control. The game theoretic approach avoids the assumption of small perturbations from a steady state solution required for stability analysis. The approach permits characterization of the dynamics and control in terms of the system's ability to reject environmental disturbances. The metric for this characterization is termed "ruggedness". For a legged robot, ruggedness is the maximum environmental variation or change from step to step that the robot can safely operate in and is posited as a useful metric for evaluating different robot designs.

- Determination of an optimal range of passive mechanical properties for vertical hopping under feedback control. Relevant mechanical properties are leg stiffness, damping, and maximum thrust. The leg is massless. For a given level of thrust and at zero damping, ruggedness is maximized at high stiffness; leg deflections can be quite small because thrust is not required to restore energy. However, at realistic damping levels ruggedness is maximized at moderate to

low leg stiffness (note, there is always some level of damping in a physical system, partly due to friction, and partly to cover energy loss at leg contact). For a given leg stiffness, the optimal damping separates obstacle limited and compression limited performance. Obstacle limited performance is the limiting case of hopping up stairs with insufficient thrust to overcome losses due to damping. Compression limited performance is the limiting case of hopping down stairs and landing hard enough to violate joint constraints. These results illustrate a tenet of biomimetic design: the need to tune mechanical compliances in conjunction with designing an appropriate feedback control (Cham, et. al., 2000 [11]). The results also demonstrate the existence of an optimal range for mechanical compliance even for highly unstructured terrain.

- Refinements to Tomlin's approach (Tomlin, et. al., 1998 [47] and Tomlin, et. al., 2000 [48]) including avoidance of singular points and avoidance of hybrid dynamical phase transitions that are explicit functions of the continuous control or disturbance input. Singular points are points in the continuous state space where the time derivative is zero and propagation in backwards time is undefined. Singular points can be avoided by constraining control inputs along surfaces within state space. Phase transitions that are explicit functions of the continuous control or disturbance inputs allow thrashing between modes since the input can change instantaneously. Thrashing can be avoided by creating transition phases that fix the value of the control over some finite region of the state space.

- Development of the MISS algorithm for determination of the maximal invariant safe set and least restrictive control for a restricted class of hybrid systems. The algorithm is an implementation of the game theoretic approach described by Tomlin, et. al. (2000, [48]), but uses a cell complex representation to define boundaries between safe and unsafe regions of the state space rather than a grid based representation. The boundary representation is made possible by exploiting properties of the Hamilton–Jacobi formulation of the optimal control problem for a slightly restricted class of hybrid systems. The approach

is significantly more computationally efficient than a grid based approach for high dimensional or complex problems that would otherwise require close grid spacing.

- Demonstration via the MISS algorithm that purely open loop (timer based) control of a single legged hopping robot is stable over level terrain and ascending stairs but that stable solutions do not exist for even small step-to-step variations in terrain height if the variations can be both positive and negative. The results over level terrain agree with previous work by Ringrose (1997, [39]) and help validate the algorithm. The results over variable terrain have not been reported previously but are consistent with observations by Kuo (2002, [30]) that pure feedforward control in simple oscillating systems is highly sensitive to unexpected disturbances. That pure feedforward systems such as *RHex* and *Sprawlita* have been so successful experimentally is likely due to the fact that stumbling is not a failure mode for a well designed multi–legged robot.

- Formulation of the MISS algorithm for more complex problems such as adaptive timer based control, additional measurements, and single and multi–legged planar running. Solution to these problems is beyond the scope of the current work, but the problem formulation shows the potential for this approach both within and beyond the domain of legged robots.

Chapter 2

The Single Legged Vertical Hopper

This chapter describes the dynamics of a single legged vertical hopper and formulates the interaction with an unknown environment as a hybrid dynamical system. The problem formulation assumes hopper position and velocity are continuously available for feedback and that thrusting can occur any time during ground contact. Variations in terrain height can occur between hops. The maximum and minimum change in terrain height is limited. To remain safe, the robot must clear step heights between hops and land without violating joint constraints. The feedback control which maximizes the set of position and velocity states over which the robot can safely operate through a sequence of steps is determined by application of game theory. The maximum environmental disturbance which admits a non-empty subset of safe states is determined over a range of leg stiffness and damping and leg thrust capability. Results show that for realistic ranges of the leg damping ratio, optimal performance is obtained with medium to low values of leg stiffness.

2.1 Previous Work

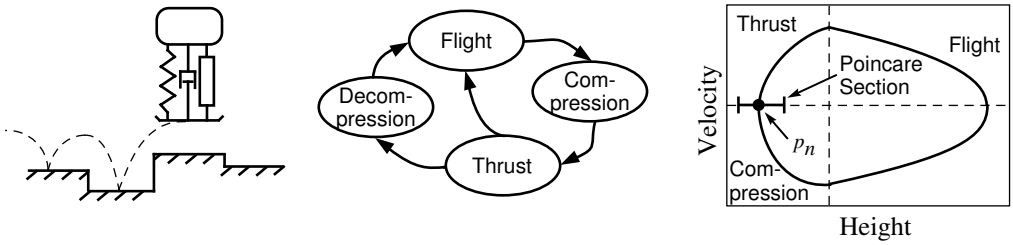
Although simple, the single legged vertical hopper is a good model of bouncing gaits in a variety of animals (Blickhan, 1989, [4]). The model also exhibits a surprising richness and complexity and has been studied by a number of researchers including Raibert (1986, [38]), Koditschek and Beuhler (1991, [27]), Ringrose (1997, [39]), Berkemeier

and Desai (1999, [3]), Komsuoglu and Koditschek (2000, [28]), Cham, et. al. (2001, [13] and 2004, [12]), and Bailey (2004, [1]).

Raibert, [38], was the first to develop machines to study dynamically stable legged locomotion. The machines consisted of a body with an articulated hinge joint and a pneumatic piston, and achieved a pogo-stick style of motion. The problem of control was decomposed into separate tasks that controlled hopping height, forward travel, and posture (hip angle). The hopper was capable of hopping in place at different heights and running forward and backward at different speeds. Hodgins and Raibert (1991 [24]) demonstrated the ability of a two legged version of the hopper to run up and down a short flight of stairs.

The success of Raibert's robots inspired a number of researchers to investigate the dynamics and stability of hopping motion. The robot is modeled as a spring–mass–damper system with a linear actuator as shown in Figure 2.1(a). Hopping behavior is divided into discrete phases: compression, thrust, decompression, and flight (Figure 2.1(b)). When the robot contacts the ground the compression phase begins and continues until the actuator begins thrusting. If thrusting stops before liftoff, a decompression phase exists while the robot remains in contact with the ground. At liftoff, the robot enters the flight phase which continues until the robot foot again contacts the ground and the sequence repeats. Hopping behavior can be illustrated on a phase plane diagram that traces robot height and vertical velocity through the different phases of motion (Figure 2.1(c)). Under steady state conditions the hopping height and velocities at takeoff and landing remain constant. That is, the robot follows the same phase plane position and velocity trajectory, or orbit, each cycle. The question of interest to researchers is under what conditions the phase plane orbit is stable.

Koditschek and Buehler (1991, [27]) investigated the stability of hopper dynamics by studying *Poincare* or *return maps* of the system. The return map is a discrete sampling of the system dynamics and is associated with a *Poincare* section: an N-1 dimensional surface in state space normal to the phase space trajectory. For the hopping robot, an appropriate Poincare section is at zero velocity at the bottom of the compression phase, as shown in Figure 2.1(c). The return map is a function,



(a) Hopper mechanism (b) Phases of motion (c) Phase diagram

Figure 2.1: A hopping robot and phases of motion. The robot modeled is as a spring–mass–damper system with a linear actuator (a). Hopping motion is decomposed into distinct phases (b). A phase plane diagram shows hopping height and velocity through each phase (c). In steady state the state trajectory forms a closed orbit that repeats between cycles and intersects at the same point on a Poincare section.

M, that relates intersections of the state space trajectory with the Poincare section between successive orbits:

$$p_{n+1} = M(p_n) \tag{2.1}$$

p_n and p_{n+1} are the intersection points between the nth and nth+1 orbit and the Poincare section. In steady state, intersections between successive orbits repeat and the return map defines a fixed point, $p*$:

$$p* : p* = M(p*) \tag{2.2}$$

The return map is a non-linear function so that the local stability of a fixed point does not guarantee global stability. However, Koditschek and Buehler show that Raibert's hoppers are globally, asymptotically stable for any "physically reasonable choice" of control gains (Koditschek and Buehler, 1991, [27]). The authors believe that the large domain of attraction for the fixed point is the key to Raibert's experimental success. They speculate that the mathematical characteristics they exploit for their analysis may be a design criterion for robotic interactions with periodic intermittent contact.

Ringrose (1997 [39]) introduced the idea of "self stabilizing" running in which

the robot's inherent motion can recover from minor perturbations without active feedback. Actuation is clock driven to a fixed pattern much like the Central Pattern Generators posited by Full and Koditschek (1999, [20]). In self stabilizing running, energy gained by thrusting is balanced by energy lost through friction. The energy loss function is quadratic so that if the robot lands too fast proportionally more energy is lost during contact and the robot lands with less velocity on the next hop. Conversely, if the robot lands too slowly the actuation increases robot energy and the robot lands with greater velocity on the next hop. Ringrose also demonstrated stability of the timing between initial ground contact and thruster actuation. If the robot take off velocity is higher than nominal it will spend a longer period of time in flight and actuation on the subsequent hop will occur earlier during contact adding less energy and resulting in lower take off velocity. Pitching motion is stabilized with a curved foot that imparts a corrective torque at contact if the robot lands off center. Ringrose experimented with a single legged hopper built to his design and simulated stable, open loop quadrupedal trotting and galloping with multiple legs.

Berkemeier and Desai (1999, [3]) compare the hopping height of robots controlled to thrust at maximum compression, and thus requiring high bandwidth feedback, against the hopping height for purely open loop control. They show that maximum hopping height for a given level of force is achieved when thrust is applied at maximum compression. They propose an "adaptive periodic forcing" approach that adjusts the period between thrusts based on the velocity measured at the time thrust is applied. The approach does not require high bandwidth control but has the same hopping height as the Raibert control scheme. They derive return maps for each of the different control mechanizations under the assumption that the hopper is lightly damped and show that their adaptive approach is stable over a range of gains.

Komsuoglu and Koditschek (2000, [28]) study the dynamic stability of the clock driven systems described by Ringrose and Berkemeier and Desai. They show that a stable hopping cycle exists assuming non-zero viscous damping in the leg and appropriate conditions on hopping period, thrust duration, and landing velocity.

Cham, et. al. (2001, [13]) study the hopping behavior of the *Sprawlita* robot. They report experimental results for ground speed versus stride frequency over level

and sloped terrain. Maximum speed running uphill is achieved at a lower stride frequency than the optimal frequency for level ground. Optimal stride frequencies also vary from robot to robot due to manufacturing differences. Based on the observation that actuator work is maximized when thrusting ends just prior to lift-off, Cham, et. al. propose an adaptive strategy that minimizes the time difference between thruster deactivation and loss of ground contact. The adaptive scheme requires only a simple contact switch for sensing and, like Berkemeier and Desai, does not require high bandwidth control.

Zhang, et. al. (2005, [50]) provide a stability analysis of the *Tekken* quadruped robot (Fukuoka, et. al., 2003, [18]). Following methods also described by Cham (2002, [14]) they find stable fixed point solutions for a bounding gait and propose a Delayed Feedback Control to compensate for energy loss due to friction and collision by adjusting leg angle at touch down. Simulation results show the ability to transition from a stance state to a bounding gait and the ability, with the control, to over come a disturbance step while running.

Other researchers have explored the use of non-linear oscillators as pattern generators for robot motion. The oscillator and robot form a coupled dynamic system. Fukuoka, et. al. (2003, [18]) use hip joint and body roll angle feedback as a means for adjusting phasing between pattern generators. They demonstrate walking over small obstacles and over moderately irregular terrain. Bailey (2004, [1]) explores the use of non-linear oscillators coupled with the robot dynamics in an adaptive scheme for the *Sprawlita* robot. His adaptive scheme improves the robot's hill climbing ability from the relatively modest five degree slope reported by Cham (2002, [14]) to a more robust $\pm$ 15 degrees.

In summary, much progress has been made in the field of legged robotic motion in recent years. Dynamic stability over level terrain is now well understood and researchers have been able to apply insights gained from studies in integrative biology to design and build robots capable of maneuvering over rough, uneven terrain. However, there are limitations to the current research. For example, none of the previous work explicitly considers robot mechanical or actuator limits, nor is there a quantitative understanding of how changes to the mechanical or sensory system will affect

performance under rugged, dynamic conditions. Thus, there is motivation to better understand robot dynamics in a variable environment. The following section begins with the dynamics of a single legged vertical hopper.

2.2 Vertical Hopper Dynamics

Figure 2.2 is a schematic of a single legged vertical hopper. The hopper has mass, m, and the location of the mass center is denoted by the $\oplus$ symbol. The hopper leg consists of a spring, damper, and linear actuator connected in parallel. The unsprung length of the leg from the hopper mass center is l. The spring and damper are linear time invariant elements. The spring has stiffness k in units of force per length, and the damping constant is b in units of force per velocity. The linear actuator can exert an upward force on the hopper body between 0 and f_{max}. The broken line at the bottom of the figure is a fixed reference. The height of the hopper mass center with respect to the fixed reference is y. The height of the ground directly beneath the hopper with respect to the fixed reference is h. Assuming a massless leg and a downward gravitational acceleration, g, the vertical dynamics for flight and contact phases of motion are given in equation 2.3.

$$\ddot{y} = \begin{cases} -g & \texttt{flight} \\ (1/m)[-k(y-h-l) - b\dot{y} + f] - g & \texttt{contact} \end{cases} \tag{2.3}$$

$$\text{where } 0 \leq f \leq f_{max}$$

In equation 2.3 the compression, thrust, and decompression phases of motion shown in Figure 2.1 are combined into a single contact phase because actuator thrust can assume any value between 0 and f_{max} while the hopper is in ground contact. The contact phase begins when $y - h - l = 0$ and $\dot{y} \leq 0$. The contact phase ends when $y - h - l = 0$ and $\dot{y} > 0$ or the ground reaction force, (GRF), goes to zero. The ground reaction force is given in equation 2.4 below. The $GRF \geq 0$ constraint means that the leg can only exert upward force on the body. In the flight phase of motion, the leg returns to its unsprung length, l, actuator force is zero, and the motion is

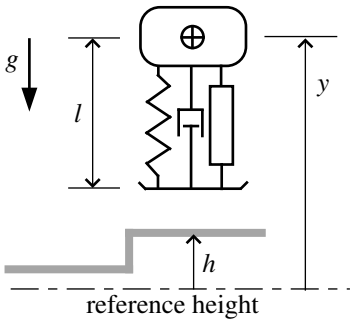


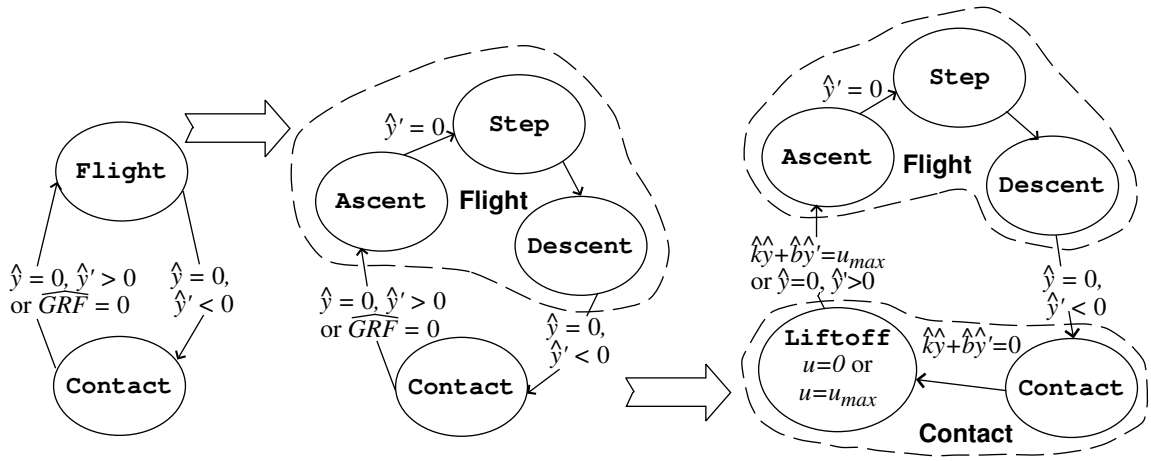
Figure 2.2: Schematic representation of a vertical hopper. The hopper dynamics are described the height, y, and velocity, $\dot{y}$, above a reference and the height, h, of the ground with respect to that reference. Parameters affecting hopper dynamics include unsprung leg length, l, spring and damping constants, and maximum thrust capability.

ballistic. The flight and contact phases of motion are illustrated in figure 2.3(a), but the transition between modes are given in terms of non-dimensional variables (denoted by the hat accent) described later.

$$GRF = -k(y - h - l) - b\dot{y} + f \geq 0 \tag{2.4}$$

In this analysis, the effects of uneven or variable terrain are admitted by allowing the height of the ground, h, to change when the hopper is at the apex of its flight phase. This construct treats the robot as if it were hopping in place while the ground moves underneath. To support this construct, the flight phase of motion is divided into three new phases as shown in Figure 2.3(b): `ascent` ($\dot{y} > 0$), `descent` ($\dot{y} \leq 0$) , and `step` ($\dot{y} = 0$).

The `step` phase is a discrete phase of motion in which the height of the ground, h, changes by an increment, Δh. The `step` occurs instantaneously at the apex of flight. If the robot has insufficient clearance at the time of the step then a collision occurs and the robot stumbles. Let h^- be the height of the ground immediately preceding the `step` and h^+ be the height of the ground immediately after the `step`. The relationship is given in equation 2.5 below. Since time does not transpire in `step`, the hopper state variables, $[y,\ \dot{y}]^T$, remain unchanged.



(a) Hopper Phases (b) Flight Phase Expanded (c) Contact Phase Expanded

Figure 2.3: Hopper flight and contact phases of motion. To accommodate a sudden change in height above ground the `Flight`phase is divided into `Ascent`, `Step`, and `Descent`. Furthermore, the contact phase is divided into normal `Contact` and `Liftoff`. The `Liftoff` phase addresses the problem of zero ground reaction force by fixing the value of the leg thrust. If leg thrust is zero hopper motion is ballistic. If leg thrust is u_{max} the motion remains in contact.

$$h^+ = h^- + \Delta h \quad \texttt{step} \tag{2.5}$$

$$\text{where } \Delta h_{min} \leq \Delta h \leq \Delta h_{max}$$

During the `step` phase, if ground height rises above the height of the hopper foot, $y - l$, then the robot fails to clear the ground and collides catastrophically. Alternatively, the ground could drop below the hopper to the extent that damage occurs after landing. The hopper is assumed to have failed if it lands with such force that the leg compresses to a joint limit, l_{min}. The leg compression constraint is $y - h \geq l_{min}$.

For parametric studies it is convenient to reduce the number of variables and convert the equations of motion to non-dimensional form. Since ground height is constant while the hopper is in contact, we redefine the state variable, y, to be the height above ground plus the unsprung leg length to eliminate h and l from the equations of motion in 2.3. Further, we convert y to a non-dimensional quantity, $\hat{y}$, by normalizing by the unsprung leg length, l. Expressions for the non-dimensional variable, $\hat{y}$, and the non-dimensional leg compression constraint, $\hat{y}_{min}$, are given in 2.6.

$$\hat{y} = (y - h - l)/l \tag{2.6}$$

$$\hat{y}_{min} = (l_{min} - l)/l \tag{2.7}$$

$$\Delta\hat{h} = \Delta h/l \tag{2.8}$$

Incorporating equation 2.5 into the expression for $\hat{y}$ above produces a discrete change in $\hat{y}$ at the `step` phase.

$$\hat{y}^+ = \hat{y}^- + \Delta\hat{h} \quad \texttt{step} \tag{2.9}$$

$$\text{where } \Delta\hat{h}_{min} \leq \Delta\hat{h} \leq \Delta\hat{h}_{max}$$
$$\text{and } \Delta\hat{h}_{max} = (\Delta h_{max})/l$$
$$\text{and } \Delta\hat{h}_{min} = (\Delta h_{min})/l$$

The continuous equations of motion given in 2.3 can be converted to non-dimensional form by substituting in $\hat{y}$ and by differentiating with respect to non-dimensional time, $\tau = \sqrt{(l/g)}\, t$. The non-dimensional equations of motion and ground reaction force are given in equations 2.10 and 2.11. In these equations the prime accents denote differentiation with respect to τ.

$$\hat{y}'' = \begin{cases} -1 & \texttt{ascent, descent} \\ -\hat{k}\hat{y} - \hat{b}\hat{y}' + u - 1 & \texttt{contact} \end{cases} \tag{2.10}$$

$$\text{where } 0 \leq u \leq (f_{max}/mg) = u_{max}$$

$$\widehat{GRF} = -\hat{k}\hat{y} - \hat{b}\hat{y}' + u \tag{2.11}$$

2.3 Hybrid Model Representation

The hopper dynamics includes both continuous phases and discrete changes in motion and therefore constitutes a hybrid system. Hybrid systems combine both continuous, or "real–time" behavior and discrete event, or "logical–time" behavior in a single framework. Examples of hybrid behavior include hysteresis and collisions (Branicky, et. al., 1994, [6]). Other examples of hybrid systems include manual and automatic transmissions and stepper motors (Brockett, 1993, [7]), computer disk drives, constrained robotic systems, intelligent vehicle/highway systems (Tomlin, et. al., 2000, [48], air traffic control (Tomlin, 1998, [47]), and water tank and steam boiler systems (Labinaz, et. al., 1996, [31], and Heymann, et. al., 1997, [22]).

The study of hybrid systems has received much attention lately as researchers begin to grapple with limitations of purely continuous and purely discrete event views of dynamic phenomena (Labinaz, 1996, [31]). In particular, Brockett (1993, [7]) describes hybrid systems as systems where the dynamics of the symbol manipulation process interact with physical dynamics and draws a distinction between hybrid model representation and the classical sampled data approach to computer control based on difference equations. Interestingly, Brockett goes on to compare his models against

motion control in higher animals foreshadowing work by Full and Koditschek (1999, [20]) by several years.

Researchers in computer science have tended to emphasize verification problems in hybrid systems, that is formal proof that a given system satisfies certain specifications (Lygeros, et. al., 1997, [32]). Researchers have used both model checking and deductive techniques which provide formal semantics and support proof techniques for verification (see, for example, Mosterman, et. al., 1998, [34]). On the other hand, researchers in the dynamics and control community have tended to emphasize modeling and simulation techniques and control synthesis. The diverse approaches have led to a proliferation of hybrid system models or representations, some of which are surveyed by Branicky, et. al. (1994 [6]), and Labinaz, et. al., (1996, [31]).

Branicky, et. al. observe that different hybrid model formalisms serve different purposes and that there is a trade-off between the generality of a model and what one can prove about such a model. Branicky, et. al., then provide their own general framework for a hybrid model and prove the existence of an optimal control. For our purposes we require a hybrid modeling formalism that permits what Branicky refers to as autonomous switching and autonomous jumps, but not controlled switching or controlled jumps. Autonomous switching refers to a sudden change in the continuous dynamics when the continuous state hits a threshold value (such as transition from `Flight` to `Contact` phase in Figure 2.3). Autonomous jumps are discrete changes in the state vector upon hitting a prescribed region in state space (such as the change in height during the `Step` phase). Controlled switching and jumps are switching and jumps that are triggered by external control commands.

This paper adopts the modeling formalism of a hybrid automaton described by Tomlin, et. al. (2000, [48]).

Definition 1 (Hybrid Automaton)**.** *A hybrid automaton, H, is a collection:*

$$H = (Q, X, \Sigma, V, I, f, Inv, R)$$

where

- *Q is a finite set of discrete states, $q_i \in Q$;*

- *X is a compact subset of Euclidian space, $X \subset \mathbb{R}^n$;*
- *Σ is a finite set of discrete inputs, $\Sigma_1 \cup \Sigma_2$, where Σ_1 is the set of discrete control inputs and Σ_2 is the discrete set of disturbance inputs;*
- *V is a set of continuous input variables, $U \cup D$, where U is the set of continuous control inputs and D is the set of continuous disturbance inputs and $U_q \subset U$ is the subset of permissible continuous control inputs for the discrete mode $q \in Q$, and $D_q \subset D$ is the subset of permissible continuous control inputs for the discrete mode $q \in Q$;*
- *$I \subseteq Q \times X$ is a set of initial states;*
- *$f : Q \times X \times V \mapsto X$ is a vector field describing the continuous time evolution of the state vector, x, for each $q \in Q$; f is assumed to be globally Lipschitz in X and continuous in V;*
- *$Inv \subseteq Q \times X \times \Sigma \times V$ is the invariant and defines combinations of states and inputs for which continuous evolution is allowed;*
- *$R : Q \times X \times \Sigma \times V \mapsto 2^{Q \times X}$ is a reset relation which encodes discrete transitions of the automaton.*

The discrete states in Q define modes or phases of motion. For example, q_1 could represent the hopper flight phase of motion and q_2 could represent the hopper contact phase of motion. The continuous state space, X, is unchanged between modes but the state variables themselves can change discontinuously. In an inelastic collision between particles, for example, X preserves the position and velocity of the individual particles (i.e., the state space does not "contract" to a single particle) and the velocity of the particles changes instantaneously.

The constraint that the continuous state space, X, is Euclidian can be relaxed somewhat. The qualitative theory of differential equations (see Nemytskii and Stepanov, 1989, [35]) shows that existence and continuity applies to systems in *locally* Euclidian spaces. This relaxation permits cyclic or periodic variables such as the angle around

a circle or a periodic clock. The joint angles for a double pendulum, for example, occupy a two dimensional toroidal surface rather than Euclidian space (Stillwell, 1993, [44]).

In Definition 1, discrete and continuous inputs can be divided into control and disturbance type inputs. Thus $\Sigma = \Sigma_1 \cup \Sigma_2$ where Σ_1 is a set of controlled discrete inputs and Σ_2 is a set of disturbance discrete inputs, and $V = U \cup D$ where U is the set of continuous control inputs and D is the set of continuous disturbance inputs. The allowable inputs are mode dependent and are stipulated by the invariant, *Inv*. Continuous control inputs U and D can change instantaneously, but within ranges specified by the invariant, during continuous evolution. Discrete control inputs can change only during discrete transitions prescribed by the reset relation, R.

The reset relation, R, is defined as a mapping onto the discrete and continuous state space. The mapping is shown as a function of the current discrete and continuous states and the discrete and continuous inputs. The relation is often illustrated using a "bubble" diagram as in Figure 2.3. The diagram shows the legal transitions from mode to mode and the trigger for each transition. The dependence on continuous inputs can be problematic. Since these inputs can change instantaneously, a mode transition which is sensitive to continuous control or disturbance inputs can thrash back and forth between different modes. This violates the principle of *divergence of time* stated by Mosterman, et. al. (1998, [34]). In this work we restrict the reset relation so that it is not a function of continuous inputs: $R : Q \times X \times \Sigma \mapsto 2^{Q \times X}$ Note, however, that discrete inputs, Σ, can be applied during the reset relation to impulsively change the continuous state vector such as during a collision or the `step` phase transition.

For our purposes, it is convenient to classify modes as either continuous or discrete. In a continuous mode, the continuous state vector, x, must evolve over a finite period of time before transition to another mode. In a discrete mode there are no continuous dynamics. The transition into and out of a discrete mode is instantaneous. However, the continuous state vector may be affected by the transition as in a collision, for example. Clearly, a properly constructed hybrid system model must avoid transitions back and forth between instantaneous modes. One way to prevent such behavior is

to ensure that legal transitions from discrete modes are only to continuous modes.

Definition 1 does not include any explicit output variables or relationships. We assume all continuous and discrete variables are available for output.

The hybrid automaton for the vertical hopper is specified in Figure 2.4. The figure defines the discrete and continuous states and inputs. Initial conditions for each discrete mode are specified by legal values for the control vector elements. The symbol $\vee$ is an inclusive 'or' indicating that the initial conditions for the `ascent` phase occur either at $\hat{y} = 0$ and $\hat{y}' > 0$ or when the ground reaction forces (equation 2.11) equal zero. The ground reaction force depends not only on the state, but also on the continuous control input, u. As discussed previously, this dependency can be problematic and is addressed in section 2.6.1.

Figure 2.4 shows the field relationship for the flight and contact phases of motion given in equation 2.10. The `step` mode is a discrete mode so the field relationship, $f_{\texttt{step}}$, is empty. Instead, the change in height during the `step` phase is captured in the reset relationship, R. The invariant, Inv, shows legal values for continuous states and the discrete and continuous inputs for each discrete mode.

Finally, the reset relationship, R, in Figure 2.4 defines the transitions between modes. The `descent` phase must transition to the `contact` phase. The `contact` phase must transition to the `ascent` phase. The `ascent` phase must transition to the `step` phase, and the `step` phase must transition back to the `descent` phase. The reset relationship shows the effect that discrete mode transitions have on the state vector. For example, in the transition from `descent` to `contact` mode, the initial contact state vector is set to $\hat{y} = 0$ and velocity is unchanged between modes. Enforcing the reset relation between modes can reduce the accumulation of numerical error when the continuous dynamics are integrated numerically. Additionally, the reset relation can include impulsive environmental disturbances. The transition from `step` to `descent` phase includes a discrete change in height above ground, $\hat{y}^+ = \hat{y}^- - \Delta\hat{h}$.

$$
\begin{aligned}
Q &= \{\texttt{descent}, \texttt{contact}, \texttt{ascent}, \texttt{step}\} \\
X &= \mathbb{R}^2 \text{ where } \{x \in X \mid x = [\hat{y}\ \hat{y}']^T\} \\
\Sigma &= \Delta\hat{h} \text{ where } -\Delta\hat{h}_{max} \leq \Delta\hat{h} \leq \Delta\hat{h}_{max} \\
V &= u \text{ where } 0 \leq u \leq u_{max} \\
I &= (\texttt{descent}, (\hat{y} \geq 0, \hat{y}' = 0)) \cup \ldots \\
&\quad (\texttt{contact}, (\hat{y} = 0, \hat{y}' \leq 0)) \cup \ldots \\
&\quad (\texttt{ascent}, ((\widehat{GRF} = 0) \vee (\hat{y} = 0, \hat{y}' \geq 0))) \cup \ldots \\
&\quad (\texttt{step}, (\hat{y}' = 0)) \\
f_{\texttt{ascent},\texttt{descent}} &= [\hat{y}'\ (-1)]^T \\
f_{\texttt{contact}} &= [\hat{y}'\ (-\hat{k}\hat{y} - \hat{b}\hat{y}' + u - 1)]^T \\
f_{\texttt{step}} &= \emptyset \\
\text{Inv} &= (\texttt{descent}, (\hat{y} \geq 0, \hat{y}' \leq 0), 0, 0) \cup \ldots \\
&\quad (\texttt{contact}, (\hat{y} \leq 0 \wedge \widehat{GRF} \geq 0), 0, u) \cup \ldots \\
&\quad (\texttt{ascent}, (\hat{y}' \geq 0), 0, 0) \cup \ldots \\
&\quad (\texttt{step}, (\hat{y}' = 0), \Delta\hat{h}, 0) \\
R &: R(\texttt{descent}, (\hat{y} \leq 1, \hat{y}' \leq 0), 0, 0) \mapsto \\
&\quad (\texttt{contact}, (\hat{y}^+ = 1, \hat{y}'^+ = \hat{y}'^-), 0, u) \vee \ldots \\
&\quad R(\texttt{contact}, (\hat{y} \geq 1, \hat{y}' \geq 0), 0, u) \mapsto \\
&\quad (\texttt{ascent}, (\hat{y}^+ = 1, \hat{y}'^+ = \hat{y}'^-), 0, 0) \vee \ldots \\
&\quad R(\texttt{contact}, (\widehat{GRF} = 0), 0, u) \mapsto \\
&\quad (\texttt{ascent}, (\hat{y}^+ = \hat{y}^-, \hat{y}'^+ = \hat{y}'^-), 0, 0) \vee \ldots \\
&\quad R(\texttt{ascent}, (\hat{y} > 1, \hat{y}' \leq 0), 0, 0) \mapsto \\
&\quad (\texttt{step}, (\hat{y}^+ = \hat{y}^-, \hat{y}'^+ = 0), \Delta\hat{h}, 0) \vee \ldots \\
&\quad R(\texttt{step}, (\hat{y} > 1, \hat{y}' = 0), \Delta\hat{h}, 0) \mapsto \\
&\quad (\texttt{descent}, (\hat{y}^+ = \hat{y}^- - \Delta\hat{h}, \hat{y}'^+ = 0), 0, 0)
\end{aligned}
$$

Figure 2.4: Hybrid automaton description of a vertical hopper. The model details the discrete and continuous states, discrete and continuous inputs, initial conditions, and the continuous and discrete dynamics. Note that the change in terrain height during the `step` phase is captured in the reset relation, R.

2.4 The Safety Property

This dissertation is concerned with determining whether a hybrid system, described by the automaton model in the previous section, can remain safe for some finite period of time. The safe set, F, is a subset of the discrete and continuous state space: $F \subseteq Q \times X$. The unsafe set of states, G, is the complement of the safe states: $G = F^c$. The safe set is assumed to be closed and the unsafe set of states is open, (Tomlin, et. al., 2000, [48]).

The single legged hopper is safe if joint limits are not exceeded while in contact, and if the robot does not hit the ground or stumble during the `step` phase of motion. The initial safe set for the hopper is given in equation 2.12 below. There are no safety constraints on $\hat{y}$ and $\hat{y}'$ in the `ascent` and `descent` phases so all values from the invariance relationship in figure 2.4 are included. Minimum leg length in `contact` is an additional constraint on $\hat{y}$. Collision avoidance ($\hat{y} \geq 0$) is a constraint on $\hat{y}$ during the `step` phase. Note that per the reset relation the value for $\hat{y}$ during `step` is a function of the discrete disturbance input, $\Delta\hat{h}$.

$$\begin{aligned} F \;=\; & (\texttt{descent}, (\hat{y} \geq 0, \hat{y}' \leq 0)) \cup \ldots \\ & (\texttt{contact}, (\hat{y}_{min} \leq \hat{y} \leq 0 \wedge \widehat{GRF} \geq 0)) \cup \ldots \\ & (\texttt{ascent}, (\hat{y}' > 0)) \cup \ldots \\ & (\texttt{step}, (\hat{y} \geq 0, \hat{y}' = 0)) \end{aligned} \tag{2.12}$$

To determine whether the hybrid system remains in F over some finite period of time, Tomlin, et. al. (2000, [48]) first define an *execution* of a hybrid automaton.

Definition 2 (Execution of a Hybrid Automaton)**.** *an* execution *of a hybrid automaton, H, is a hybrid trajectory,*

$$\chi = (s, q, x, \sigma, \upsilon)$$

with

- initial condition: $(q(s_0), x(s_0)) \in I$;
- continuous evolution: *for all i with* $s_i < s_i'$, $q(\cdot), \sigma(\cdot)$ *are constant,* $v(\cdot)$ *is piecewise continuous,* $x(\cdot)$ *is a solution to the differential equation* $\dot{x} = f(q, x, v)$ *over* $[s_i, s_i']$, *and for all* $t \in [s_i, s_i'), (q(t), x(t), \sigma(t), v(t)) \in Inv$;
- discrete evolution: *for all i,* $(q(s_{i+1}), x(s_{i+1}) \in R(q(s_i'), x(s_i'), \sigma(s_i'), v(s_i'))$.

The *safety property* of a hybrid automaton, denoted by $\Box F$, is a property of a set of executions over a subset of the state space.

Definition 3 (Safety Property)**.** *The* safety property, $\Box F$, *is a map from the set of executions of a hybrid automaton over a* safe *subset of the state space,* $F \subseteq Q \times X$, *so that:*

$$\Box F(\chi) = \begin{cases} \texttt{True} & \textit{if } \forall t \in s, (q(t), x(t)) \in F \\ \texttt{False} & \textit{otherwise} \end{cases}$$

2.5 Controller Synthesis and the Maximal Safe Subset

The safety property of a safe subset F is a function of the control and disturbance inputs. We assume a static (non-autonomous) feedback control based on the full continuous and discrete state vectors. The control and disturbance inputs are bound within prescribed limits. However, the internal structure of the control law is unspecified. Tomlin, et. al. (2000, [48]) pose the following controller synthesis problem:

> *Given a safe set F, determine (i) the maximal invariant safe set contained in F, and (ii) the controller which renders this set invariant*

A subset of the state space is controlled invariant if there exists a controller which guarantees that any execution that begins in the subset remains in that subset.

The algorithm proposed by Tomlin, et. al. (2000, [48]) starts with an initial safe subset, $W^0 = F$, and goes backwards in time determining the region of state space

that will remain within F despite antagonistic disturbance inputs. The algorithm is a form of dynamic programming (Bryson and Ho, 1975, [9]) and iterates until a controlled invariant subset of F is found, if one exists.

The backward time propagation is both continuous and discrete. For discrete event propagation Tomlin, et. al. (2000, [48]) define the *controllable predecessor* operator, $Pre_1(\cdot)$, on a subset of the state space, $K \subseteq Q \times X$:

$$\begin{aligned} Pre_1(K) &= \{(q,x) \in K \mid \exists(\sigma_1,u) \in \Sigma_1 \times U \ \forall(\sigma_2,d) \in \Sigma_2 \times D \\ &\quad (((q,x,\sigma_1,\sigma_2,u,d) \notin Inv) \wedge (R(q,x,\sigma_1,\sigma_2,u,d) \subseteq K))\}. \end{aligned}$$

$Pre_1(K)$ requires that the controllable actions, (σ_1, u) force a discrete transition, hence $(q,x,\sigma_1,\sigma_2,u,d) \notin Inv$. The set contains all states in K for which controllable actions can force the state to remain in K after the transition, that is $R(q,x,\sigma_1,\sigma_2,u,d) \subseteq K$.

The *uncontrollable predecessor* operator, $Pre_2(K^c)$ (Tomlin, et. al., 2000, [48]) contains all states in K^c, the complement of K, as well as all states from which disturbances (σ_2, d) can possibly force the state outside of K:

$$\begin{aligned} Pre_2(K^c) &= \{(q,x) \in K \mid \forall(\sigma_1,u) \in \Sigma_1 \times U \ \exists(\sigma_2,d) \in \Sigma_2 \times D \\ &\quad (R(q,x,\sigma_1,\sigma_2,u,d) \cap K^c \neq \emptyset)\} \cup K^c. \end{aligned}$$

$Pre_2(K^c)$ does not require a discrete transition, but if one does occur the new states are unsafe. $Pre_1(K)$ and $Pre_2(K^c)$ are disjoint sets and $Pre_1(K) \cap Pre_2(K^c) = \emptyset$.

The continuous portion of the algorithm requires an operator that can be used to eliminate those states which can be forced from safe to unsafe regions of the state space during the continuous evolution. Tomlin, et. al. (2000, [48]) define the *Reach* operator:

Definition 4 (Reach operator)**.** *Consider two subsets $G \subseteq Q \times X$ and $E \subseteq Q \times X$*

such that $G \cap E = \emptyset$.

$$\begin{aligned} Reach(G,E) \;=\; & \{(q,x) \in Q \times X \mid \forall u \in U \; \exists d \in D \\ & (((q(t),x(t)) \in G \text{ for } t > 0) \wedge ((q(s),x(s)) \notin E) \wedge \ldots \\ & ((q(s),x(s)) \in \Pi(Inv)) \text{ for } s \in [0,t])\} \end{aligned}$$

where $(q(s),x(s))$ *is the continuous state trajectory of* $\dot{x} = f(q(s),x(s),u(s),d(s))$ *starting at* $(q(0),x(0))$.

The set $Reach(G,E)$ describes those states from which for all $u \in U$ there exists a $d \in D$ such that the trajectory $(q(s),x(s))$ can be driven to G without reaching the "escape" set E. $\Pi(Inv)$ represents the state components of the invariant, *Inv*, so that the *Reach* does not include a discrete transition. In the definition above, G represents the set of states which are unsafe or can be made unsafe during a discrete transition subsequent to the *Reach*. E represents the set of states which can remain safe through a discrete transition.

Tomlin, et. al. (2000, [48]) describe the algorithm for constructing the maximal controlled invariant set for a hybrid system in terms of the *Reach* operator and the controllable and uncontrollable predecessor states:

Maximal controlled safe invariant for hybrid systems

initialization: $W^0 = F, W^{-1} = \emptyset, i = 0.$
while $W^i \neq W^{i-1}$ and $W^i \neq \emptyset$ **do**
$W^{i-1} = W^i \setminus Reach(Pre_2((W^i)^c), Pre_1(W^i))$
$i = i - 1$
end while

The first iteration through the **while** loop removes all states for which a disturbance can force the system outside of F without first causing a transition between modes that remain within F. Subsequent steps iterate to remove states that can be forced outside of F after transitions between modes. The index, i, decreases with each step to indicate propagation in backwards time. At each iteration $W^{i-1} \subseteq W^i$

so that the set W^i decreases monotonically. The algorithm continues until either a fixed point, $W^{i-1} = W^i \triangleq W^*$, or a null solution, $W^{i-1} = \emptyset$, is reached.

Implementation of the maximal controlled invariant algorithm requires determination of the $Pre_1(K)$ and $Pre_2(K^c)$ set of states and calculation of the *Reach* operator. Pre_1 and Pre_2 require inversion of the reset relation, R. In general the inverse relation exists when the transition is not an explicit function of continuous inputs. The computation of the *Reach* operation requires determination of a set of initial states from which trajectories can reach the desired escape set, W^i, avoiding the capture set, $(W^i)^c$, along the way. Tomlin, et. al. (2000, [48]) show that the *Reach* operation can be determined from the solution to an inter-connected pair of Hamilton–Jacobi equations over each continuous phase separately. The resulting feedback control is least restrictive in the sense that control is only applied to avoid the capture set, $(W^i)^c$. However, the solution to the Hamilton–Jacobi equations is in general quite difficult.

In the following section we demonstrate implementation of the maximal controlled safe invariant algorithm by application to the vertical hopper. The implementation will address issues concerning the reset relation mentioned earlier. Since the hopper example is only second order, the algorithm can readily be shown graphically. Insights gained lead to the development of geometric implementation of the algorithm suitable for higher order systems, but which does not require explicit solution of the Hamilton–Jacobi equations.

2.6 The Hopper Maximal Controlled Safe Invariant

Application of algorithm for the maximal controlled safe invariant to the hopper model specified in Figure 2.4 quickly results in problems that require structural change. One problem is the fact that the transition between `contact` and `ascent` modes occurs when the ground reaction force drops to zero and therefore depends directly on the control thrust, u. Another problem has to do with the hopper dynamics at the

singular point in `contact` where $\dot{x} = f(x, u) = 0$. At the singularity, backwards time propagation and the *Reach* operation are undefined. These problems and their resolution are discussed in more detail in the following two subsections.

2.6.1 Control Thrust and Transition to `Ascent`

First consider the transition from `contact`(q_2) to `ascent` (q_3).

$$q_2 \succ \begin{cases} q_3 & \text{if } \hat{y} = 0 \text{ or} \\ q_3 & \text{if } -\hat{k}\hat{y} - \hat{b}\hat{y}' + u = 0 \end{cases} \tag{2.13}$$

Where '$\succ$' denotes a forward time transition between hybrid modes. The transition from contact to ascent occurs when the hopper leg returns to its uncompressed length, ($\hat{y} = 0$), or when the normalized ground reaction force, given in equation 2.11 and repeated above, is zero.

The state space transitions from `contact` to `ascent` are illustrated in the phase plane diagram in Figure 2.5. Since $0 \leq u \leq u_{max}$ the transition is bound by the lines at $\hat{k}\hat{y} + \hat{b}\hat{y}' = 0$, $\hat{k}\hat{y} + \hat{b}\hat{y}' = u_{max}$, and $\hat{y} = 0$. The width of this transition region is determined by the strength of the normalized damping coefficient, $\hat{b}$, with respect to the leg's stiffness, $\hat{k}$. For lightly damped systems ($\hat{b} \ll \hat{k}$) the transition from contact to ascent can be approximated by $\hat{y} = 0$, however, for heavier damping the transition from contact to ascent must be carefully considered.

If the transition from `contact` to `ascent` occurs at $u = 0$ a change in the control can cause a transition back to `contact`. This opens the possibility of needless switching between `contact` and `ascent` modes. The situation going forward in time is illustrated in the upper portion of Figure 2.5 at point A1. If $u = 0$ the system switches from `contact` to `ascent` and follows the lower bound trajectory to point A2. If, however, $u = u_{max}$ the system remains in `contact` and follows the upper bound trajectory to point A2'. Intermediate trajectories, depicted by the cross–hatched area, can be achieved by modulating the control. The situation going backward in time is shown starting in `ascent` at point B1. If $u = 0$ the system remains in `ascent` and follows the upper bound trajectory to point B2. If $u = u_{max}$ the system switches

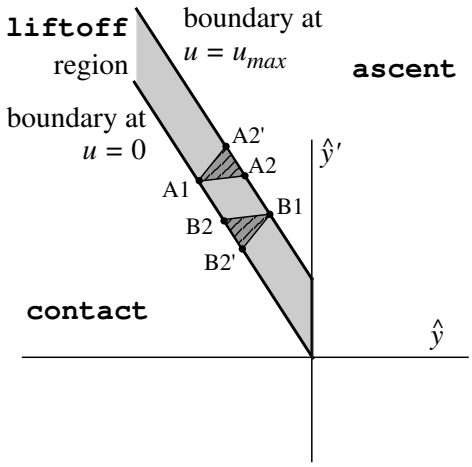


Figure 2.5: The transition between contact and ascent phases is influenced by the control, u. Going forward in time from point A1 the system will enter ascent at a point between A2 and A2'. Going backwards in time from point B1 the system will enter contact between point B2 and B2'. The bounding behavior for the contact/ascent transition is obtained at either $u = 0$ or $u = u_{max}$.

to `contact` and follows the lower bound trajectory to B2'. It is clear from the figure that the bounding behavior is at either $u = 0$ or $u = u_{max}$ so that intermediate or varying values for for u need not be considered.

It is useful to define a transitional mode, $q_5 =$ `liftoff`, so that the exit manifold for the contact phase and the entrance manifold for the ascent phase is independent of the control variable, u. The transitional mode "locks" the value of the control to $u \triangleq 0$ or $u \triangleq u_{max}$. The `liftoff` mode transitions are illustrated in Figure 2.3(c). The transition from `contact` to `liftoff` occurs at the $\hat{k}\hat{y} + \hat{b}\hat{y}' = 0$ manifold. The transition from `liftoff` to `ascent` occurs at the $\hat{k}\hat{y} + \hat{b}\hat{y}' = u_{max}$ for $\hat{y} < 0$ and at $\hat{y} = 0$ otherwise. For $u \triangleq 0$ in `liftoff`, the state trajectory follows a ballistic path to the exit manifold. For $u \triangleq u_{max}$ in `liftoff`, the state trajectory follows the logarithmic spiral of a damped second order system.

The appropriate value for u during `liftoff` depends on context. Referring again to Figure 2.5, suppose point B1 is the initial velocity required to achieve a minimum

height during `ascent`. Then, $u = u_{max}$ maximizes the safe range of exit conditions from `contact` since along the boundary at $u = 0$ all points above B2' will meet this constraint. On the other hand, if B1 is the maximum safe velocity upon entering ascent, then $u = 0$ maximizes the safe range of contact exit conditions since along the boundary at $u = 0$ all points below B2 will meet the maximum height constraint. A formal method for determining the optimal value for u is described in Chapter 3.

2.6.2 Singularity Avoidance

The second difficulty in applying the maximal safe invariant algorithm to the hopper problem is associated with the stable point in the contact region. At $u = 0$ the hopper equilibrium in contact is $\hat{y} = -1/\hat{k}$ and $\hat{y}' = 0$. If the initial descent velocity is low enough and the thrust remains at zero, the system will settle to equilibrium without ever leaving the contact phase. Technically, the system is safe, but it is also inert.

A more fundamental problem is that at the stable point backwards time propagation is undefined. All points in the neighborhood of the stable point eventually reach the stable point so starting from the stable point and propagating backwards in time yields infinite solutions. For this reason, the algorithm for the maximal invariant safe set must take steps to avoid the singularity at $f_q(x, u, v) = 0$.

Figure 2.6 shows a position and velocity trajectory for a hopper in contact. In this example the hopper has a normalized spring constant, $\hat{k} = 4$ and a damping coefficient $\hat{b} = 1$, which yields a non-dimensional damping ratio of 0.25. Starting from rest at $\hat{y} = 0$ and $\hat{y}' = 0$, the hopper falls under the influence of gravity with zero thrust and follows the solid line trajectory marked with circles. Without any thrust, the state trajectory would spiral in to the equilibrium position at $u = 0$ marked on the figure. In fact, without any thrust the system will eventually settle to this equilibrium from any initial condition. If, however, a thrust is applied at maximum deflection the equilibrium changes and the hopper follows the solid line trajectory marked with diamonds. The minimum level thrust that will return the hopper back to the starting position, u_{rtn}, is a function of the non-dimensional damping ratio, ζ, and is given in equation 2.14 below.

$$u_{rtn} = \frac{1 - e^{-2\pi\zeta/\sqrt{1-\zeta^2}}}{1 + e^{-\pi\zeta/\sqrt{1-\zeta^2}}}, \text{ where } \zeta = 0.5\hat{b}/\sqrt{\hat{k}} \tag{2.14}$$

Thus, singularities can be avoided by providing sufficient thrust to avoid remaining inside the singularity avoidance zone. Since thrust can vary with time, there are infinite ways of doing this. For example, going forward in time, if the state trajectory intersects the boundary $u = 0$ (marked by squares in 2.6) then one could apply maximum thrust to force the trajectory to exit the region somewhere along the $u = u_{rtn}$ boundary (marked by diamonds in the figure). Alternatively, upon intersecting the $u = 0$ boundary, one could coast along at $u = 0$ and then set $u = u_{rtn}$ at zero velocity and follow the u_{rtn} boundary until reaching the previous exit point. The net effect of both strategies is the same. Since one can reach any desired point along the u_{rtn} boundary by following the second strategy, this approach is interpreted as least restrictive (we temporarily ignore the `Liftoff` phase which also intersects the u_{rtn} boundary). Therefore, the $u = 0$ and $u = u_{rtn}$ boundaries of the singularity avoidance region define internal boundaries of the contact region.

2.6.3 Algorithm Execution

The algorithm for determining the maximal invariant safe set has an initialization phase and an iterative phase. The iterative phase progresses backwards in time applying the *Reach* operator to determine the continuous states and the least restrictive control between phase transitions. The *Reach* operation requires time propagation of the continuous equations of motion and determination of optimal control and disturbance inputs. A detailed treatment of the optimization problem is deferred until Chapter 3. Rather, for this problem optimal control is determined by examination of the state space trajectories.

Figure 2.7 is a plot of the maximal controlled invariant set for a vertical hopper. The hopper has a maximum thrust capability equal to twice its weight, and a leg with normalized spring stiffness of 4 and a damping coefficient of 1.0. The non-dimensional parameters are summarized in Table 2.1. The damping coefficient is equivalent to a damping ratio, ζ, of 0.25. The joint constraint limit is at -0.5 and the maximum and

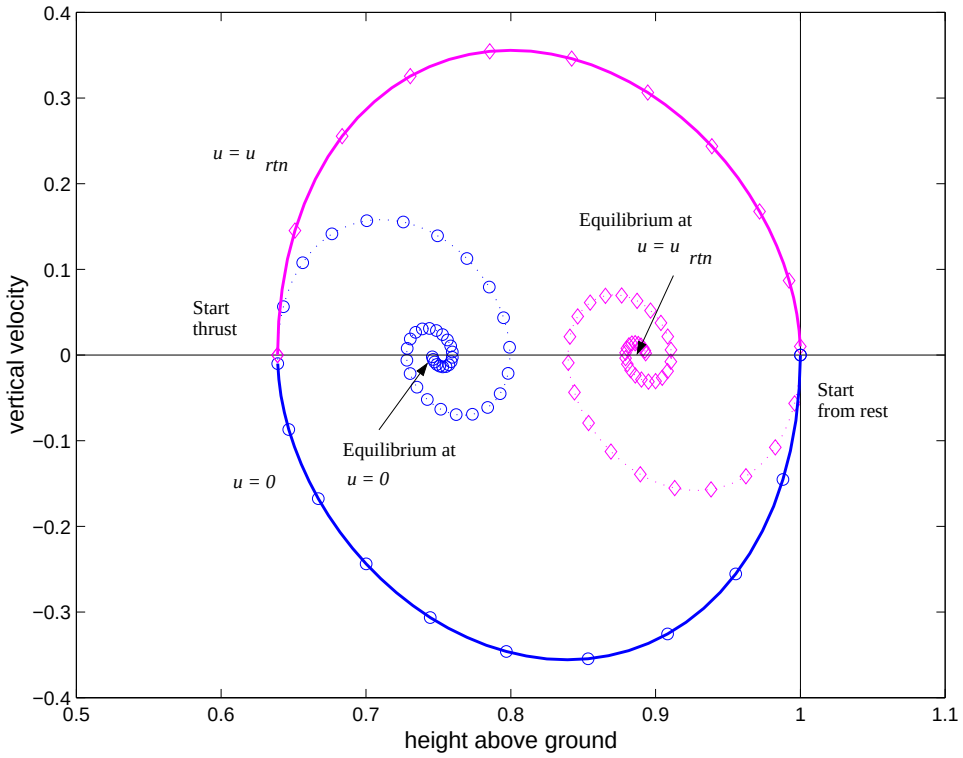


Figure 2.6: Singularity avoidance for the hopper in `contact`. Singularities, or equilibria, in the contact region are a function of thruster force. A singularity avoidance zone is defined by the position and velocity trajectory starting at initial contact, falling with zero thrust, and then applying applying a minimum return thrust, $u = u_{rtn}$, at maximum deflection.

Parameter	*Value*
stiffness, $\hat{k}$	4.0
damping coefficient, $\hat{b}$	1.0
damping ratio, ζ	0.25
max thrust, $\hat{u}_{max}$	2.0
joint constraint, $\hat{y}_{min}$	-0.5
max step, $\Delta\hat{h}_{max}$	0.3
min step, $\Delta\hat{h}_{min}$	-0.3
max initial height, $\hat{y}_{init-max}$	2.0
min initial velocity, $\hat{y}'_{init-min}$	-3.0
max initial velocity, $\hat{y}'_{init-max}$	3.0

Table 2.1: Vertical hopper non-dimensional parameter values for Figure 2.7.

minimum initial height and velocity constraints form a closed boundary for the initial safe set.

The horizontal and vertical axes of the plot in Figure 2.7 are the normalized height, $\hat{y}$, and the normalized vertical velocity, $\hat{y}'$, respectively. The state space is divided into regions corresponding to the hopper phases of motion. `Liftoff` is the region between the diagonal broken lines in the upper left and bound on the right hand side by the line at $\hat{y} = 0$. The boundaries of the `liftoff` phase are determined from the equations for zero ground reaction force as described in section 2.6.1. The `ascent` phase of motion covers the upper right portion of the state space from the `liftoff` boundary and bound below by the line at $\hat{y}' = 0$. The `descent` phase of motion covers the lower right quadrant of the state space bound by $\hat{y} \geq 0$ and $\hat{y}' < 0$. Finally, the `contact` phase of motion covers the lower and center left portion of the state space bound by $\hat{y} \leq 0$ and the broken line at `liftoff`. In this example, the subsets of the continuous state space for the different phases of motion are disjoint (i.e. do not overlap), but that is not a general requirement.

Initialization

The first step to the algorithm is to determine the initial safe set, W^0. The safety constraints are that the hopper clear any obstacle during the `step` phase and that the hopper avoid the $\hat{y} < \hat{y}_{min}$ joint constraint limit. The initial safe set of states is

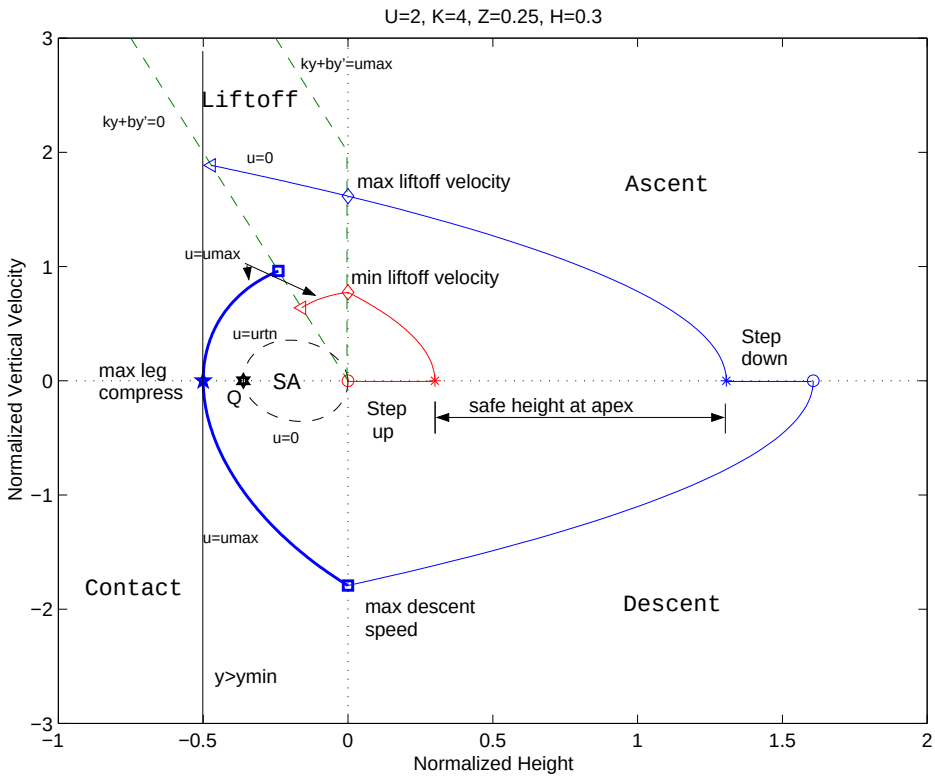


Figure 2.7: Maximal safe invariant set for a hopper with $\hat{k} = 4$, $\zeta = 0.25$, and $\Delta\hat{h}_{max} = 0.3$. The diagram shows the `contact`, `descent`, `ascent`, and `liftoff` phases of motion as well as the singularity avoidance boundary of the `contact` phase. In this example the algorithm for finding the maximum invariant safe set terminates after a single iteration through the phases of motion.

given in equation 2.15 below. The equation omits the upper and lower bounds given in Table 2.1 for brevity.

$$\begin{aligned} W^0 \;=\; & \{(\texttt{descent}, (0 \leq \hat{y} \wedge \hat{y}' \leq 0)) \cup \ldots \\ & (\texttt{contact}, (\hat{y}_{min} \leq \hat{y} \leq 0 \wedge \hat{y}' \leq 0) \cup (\hat{k}\hat{y} + \hat{b}\hat{y}' \leq 0 \wedge \hat{y}' \geq 0)) \cup \ldots \\ & (\texttt{liftoff}, (0 \leq \hat{k}\hat{y} + \hat{b}\hat{y}' \leq u_{max} \wedge \hat{y}_{min} \leq \hat{y} \leq 0)) \ldots \\ & (\texttt{ascent}, (\hat{k}\hat{y} + \hat{b}\hat{y}' \geq u_{max} \wedge \hat{y} \leq 0) \cup (\hat{y} \geq 0 \wedge \hat{y}' \geq 0)) \cup \ldots \\ & (\texttt{step}, (\hat{y} > \Delta\hat{h} \wedge \hat{y}' = 0))\} \end{aligned} \tag{2.15}$$

The controllable predecessor of W^0 is the set states from which a control input can force a discrete transition between modes but which remain within W^0. Due to our inclusion of `liftoff`, modal transitions occur only at state space boundaries and are not directly affected by control input. The controllable predecessor, $Pre_1(W^0)$, is the set of safe states at these transition boundaries and is given explicitly in equation 2.16 below. $Pre_1(W^0)$ is a subset of W^0.

$$\begin{aligned} Pre_1(W^0) \;=\; & \{(\texttt{descent}, (\hat{y} = 0 \wedge \hat{y}' \leq 0)) \cup \ldots \\ & (\texttt{contact}, (\hat{k}\hat{y} + \hat{b}\hat{y}' = 0) \wedge \hat{y}_{min} \leq \hat{y} \leq 0) \cup \ldots \\ & (\texttt{liftoff}, (\hat{k}\hat{y} + \hat{b}\hat{y}' = u_{max} \wedge \hat{y}_{min} \leq \hat{y} \leq 0) \cup \ldots \\ & (\texttt{liftoff}, (\hat{k}\hat{y} + \hat{b}\hat{y}' < u_{max} \wedge \hat{y} = 1) \cup \ldots \\ & (\texttt{ascent}, (\hat{y} \geq 0 \wedge \hat{y}' = 0)) \cup \ldots \\ & (\texttt{step}, (\hat{y} \geq \Delta\hat{h}_{max} \wedge \hat{y}' = 0))\} \end{aligned} \tag{2.16}$$

The uncontrollable predecessor of W^0 is the complement of W^0 and the set of states from which disturbance inputs can force the system outside of W^0. Considering first the `step` phase, it is obvious that for $\hat{y} < \Delta\hat{h}_{max}$ an environmental disturbance of $\Delta\hat{h} = \Delta\hat{h}_{max}$ will render the system unsafe. In `contact` the system is safe at $\hat{y} = \hat{y}_{min}$ but if $\hat{y}' < 0$ the system will become unsafe in the next instant. The uncontrollable

predecessor for W^0 is given in equation 2.17 below.

$$\begin{aligned} Pre_2((W^0)^c) &= (W^0)^c \cup \ldots \\ &\quad (\texttt{contact}, (\hat{y} = \hat{y}_{min} \wedge \hat{y}' < 0)) \cup \ldots \\ &\quad (\texttt{step}, (\hat{y} < \Delta\hat{h}_{max} \wedge \hat{y}' = 0)) \end{aligned} \tag{2.17}$$

Contact Phase

We apply the *Reach* operator to find the set of continuous states that can be forced to $Pre_2((W^0)^c)$. In `contact` this is the set $(\hat{y} = \hat{y}_{min} \wedge \hat{y}' < 0)$, as shown in equation 2.17 above. .

Figure 2.7 marks the point of maximum leg compression $(\hat{y} = \hat{y}_{min}, \hat{y}' = 0)$ with a star. Propagating the contact phase equations of motion backwards in time generates a curve that divides the `contact` phase into two regions. The backwards propagation is shown as a heavy line in figure 2.7 starting from "max leg compression" and terminating at "max descent speed". The contact states below this curve will propagate to violate the maximum leg compression constraint and therefore reach $Pre_2((W^0)^c)$. The contact states on and above this curve remain safe. The set of safe states is maximized if the propagation is at $u = u_{max}$.

Safe initial conditions for `contact` are those states along the `contact`–`descent` boundary above the"maximum descent speed" point shown in figure 2.7. The flip side of the problem is to determine the subset of exit conditions along the `contact`–`liftoff` boundary that can be reached safely. This calculation is used to determine $Pre_1(W^{-1})$ for the next iteration of the algorithm. Propagating the contact phase equations of motion forwards in time from the point of "max leg compression" (the star in figure 2.7) to the `liftoff` border generates a curve that closes the set of safe `contact` phase continuous states. Propagating in forward time with $u = u_{max}$ maximizes this set.

Figure 2.7 also shows the singularity avoidance region, marked "`SA`" and delimited by dashed lines, within the `contact` phase. Singularity avoidance forms an internal boundary for the safe set of `contact` phase states. The boundaries are determined as

described in section 2.6.2. The point Q, marked by a hexagram in figure 2.7, is the maximum leg compression starting at rest and falling under the influence of gravity ($u = 0$).

Control is not restricted except at the boundaries of the `contact` region. For example, if the initial contact velocity is somewhere between 0 and maximum descent speed, the control, u, can assume any legal value unless and until the state trajectory intersects the safety or singularity avoidance boundary. At the safety boundary the control is set to $u = u_{max}$ at least until hopper velocity is positive. If the state trajectory intersects the `SA` boundary then control is set to $u = 0$ or $u \geq u_{rtn}$ depending on where the intersect occurs. Note that the `SA` region does not limit landing or liftoff speeds. In this sense the `SA` control does not reduce the safe subset.

Descent Phase

Since the `descent` phase immediately precedes the `contact` phase, the maximum and minimum safe initial velocities for `contact` are the maximum and minimum exit velocities for `descent`. Here we propagate backwards in time from "max descent speed" to the `descent`–`step` phase boundary. Since `descent` is purely ballistic, there is no control and the backwards propagation is straightforward. Figure 2.7, only shows the propagation from "max descent speed" because the minimum safe exit speed is zero which immediately transitions to `step`. Safe initial conditions for `descent` are bound by the circle markers on the $\hat{y}' = 0$ line in Figure 2.7.

Step Phase

The safe exit conditions for `step` are the safe initial conditions for `descent`. `Step` is a discrete mode with no continuous time propagation. However, the environment acts to effect an instantaneous change, $\hat{y}^+ = \hat{y}^- + \Delta\hat{h}$. If $\Delta\hat{h}$ is negative, the environment causes a "step up" change in terrain height shown in figure 2.7. Therefore, prior to the `step` phase the hopper must be at least above the minimum descent height to avoid a collision, $\hat{y}^- \geq \max(0, -Dh_{min})$. Conversely, if $\Delta\hat{h}$ is positive the environments causes a step down change in terrain height. A step down increases landing speed

potentially over stressing the leg. The maximum height just prior to the `step` is the height from which the hopper can safely fall assuming a step down change of $\Delta\hat{h}_{max}$. Thus, the `step` phase reduces the range of safe values for $\hat{y}$ at the apex of flight by $\Delta\hat{h}_{max} - \Delta\hat{h}_{min}$. In Figure 2.7, the safe range for $\hat{y}$ prior to the `step` phase is between the asterisks on the $\hat{y}'$ line.

Ascent Phase

The `ascent` exit conditions are equal to the `step` phase initial conditions. The `ascent` phase initial conditions are the are found by propagating a ballistic trajectory backwards in time. In Figure 2.7, both upper and lower bound `ascent` phase trajectories terminate at $\hat{y} = 0$.

Liftoff Phase

The `liftoff` exit conditions are between the maximum and minimum liftoff velocities at $\hat{y} = 0$ and marked by diamond symbols in Figure 2.7. From the minimum liftoff velocity, we propagate backwards in time assuming $u = u_{max}$. The intersection at the far boundary of the `liftoff` region marks the minimum velocity in `contact` that could safely clear a step up of $\Delta\hat{h}_{max}$. From the maximum liftoff velocity, we propagate backwards in time assuming $u = 0$. The intersection at the opposite boundary of the `liftoff` region marks the maximum safe velocity in `contact` that will avoid the leg compression constraint upon returning to `contact` with a worse case drop in the terrain. The maximum achievable velocity in contact, marked by the square at the `contact`–`liftoff` border, is between the maximum and minimum `contact`–`liftoff` velocities so the system has a feasible solution.

Algorithm Termination

We have now taken the algorithm for determining the maximal controlled invariant set through one complete iteration to W^{-1} which is shown in Figure 2.7. The algorithm continues with a second iteration through the discrete modes again starting with the `contact` phase.

Starting at the `contact–liftoff` border, the maximum contact velocity is the point marked by a square in figure 2.7. Propagating backwards in time from this point at $u = u_{max}$, we follow the trajectory marked by the bold curve in figure 2.7 to the "maximum descent speed" at the `contact` initial conditions boundary ($\hat{y} = 0$).

The minimum safe exit velocity for the `contact` phase is the left facing triangle associated with the minimum liftoff velocity. Propagating backwards in time from this point for any value $0 \leq u \leq u_{max}$, one finds the state trajectory intersects the boundary of the singularity avoidance region (intersection not shown). As described previously, the control can be modulated between $u = u_{rtn}$ and $u = 0$ to follow the `SA` boundary backwards in time to point $\hat{y} = 0, \hat{y}' = 0$. Therefore, despite the change in exit conditions, the contact phase initial conditions remain unchanged.

Since the `contact` phase initial conditions are the same between iterations, both the `descent` phase exit and initial conditions are the same. Likewise for the `step`, `ascent`, and `liftoff` phases of motion. In this example, the algorithm terminates because there is no change between the W^{-2} and W^{-1} set of states.

In general, however, the algorithm will require more than two iterations through the phases of motion prior to termination. Figure 2.8 is a phase diagram of the same hopper in 2.7, but with $\Delta\hat{h}_{max} = 0.4$ and $\Delta\hat{h}_{min} = -0.4$ rather than ± 0.3. The first iteration through the phases of motion follows the previous discussion except that the higher step up requires a higher minimum liftoff velocity at the start of the `ascent` phase. After the first iteration through `liftoff`, the `contact` phase minimum safe exit velocity is marked by the lower most triangle on the `contact–liftoff` border. Propagating backwards in time from this point to start the second iteration of the algorithm, one finds that state trajectory just misses the singularity avoidance region. The backwards propagation is at $u = u_{max}$ for positive `contact` velocities, and at $u = 0$ for negative `contact` velocities. This control history minimizes the initial contact velocity and maximizes the range between minimum and maximum descent speeds. However, after the second iteration through the `contact` phase, the minimum initial contact speed, marked by the topmost right facing triangle along $\hat{y} = 0$ is non-zero. The change in `contact` phase initial conditions requires subsequent iteration through the phases of motion.

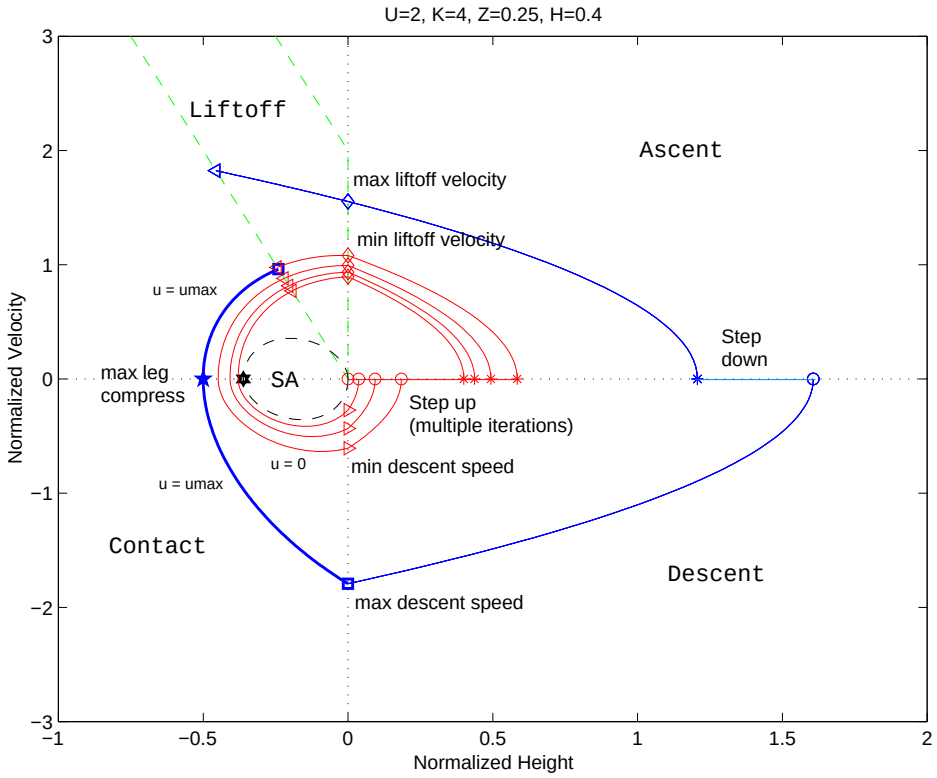


Figure 2.8: Maximal safe invariant set for a hopper with $u_{max} = 2$, $\hat{k} = 4$, $\zeta = 0.25$, and $\Delta\hat{h}_{max} = 0.4$. In this example the algorithm for finding the maximum safe invariant set requires multiple iterations through the phases of motion. After four steps up the hopper fails to clear the obstacle and the safe invariant set is empty.

Continuing backwards in time, the minimum initial height for the `descent` phase, marked by a circle in figure 2.8, moves out to the right from $\hat{y} = 0$. The higher minimum initial `descent` height, the higher the minimum initial height at the `step` phase and the higher the minimum liftoff velocity. The iterations generate an outward spiral shown in the figure. Finally, the minimum initial `liftoff` velocity, marked by left facing triangles at the `contact`–`liftoff` border, exceeds the maximum contact exit velocity, marked by a square in the figure. The algorithm terminates because the safe set is empty.

The iteration illustrated in Figure 2.8 can be interpreted in forward time as a

hopper trying to climb stairs. Assume the hopper starts at the bottom of the stair case in `descent` at the maximum safe height above ground. The hopper lands at maximum descent speed and immediately applies full thrust to avoid violating the maximum leg compression constraint. The hopper continues maximum thrust and through `contact` and `liftoff` phases and enters `ascent` just below the 'minimum liftoff velocity' marked on the figure. At the apex of flight the hopper's height above ground is indicated by the right most asterisk in the figure. Then the step up causes the hopper's height relative to ground to drop by 0.4 to the point marked by an open circle next to 'Step up' on the figure. At the top of the first step, the hopper lands at a speed just below the 'minimum descent speed'. Back in `contact`, the hopper waits for maximum leg deflection ($\hat{y}' = 0$) before applying thrust at $u = u_{max}$. However, the liftoff velocity is lower than after initial contact and the hopper lands on the second step with even less velocity than the first. The state space trajectory spirals inward (in forward time) until the hopper fails to clear the fourth step.

Each iteration through the phases of motion reduces the set of safe states. However, the ultimate outcome is not necessarily null. Figure 2.9 shows a result where the maximal safe invariant converges to a non-empty set. In this example the hopper leg has a non-dimensional stiffness of 10 and a non-dimensional damping ratio of 0.05. The leg is stiffer and less damped than in the previous examples but has the same thrust capability. As in Figure 2.8, the maximum terrain variation between steps is 40% of the uncompressed leg length ($\Delta\hat{h}_{max} = 0.4$). The hopper must land with sufficient speed to compress the leg so that it can apply thrust long enough to get the minimum required liftoff velocity. The minimum contact speed for the initial safe set of states is zero. However, by iterating through the phases of motion the initial minimum contact speed is increased until the difference between iterations is acceptably small. Starting with a contact speed greater than or equal to the minimum value, the hopper can safely climb an arbitrarily long stair case.

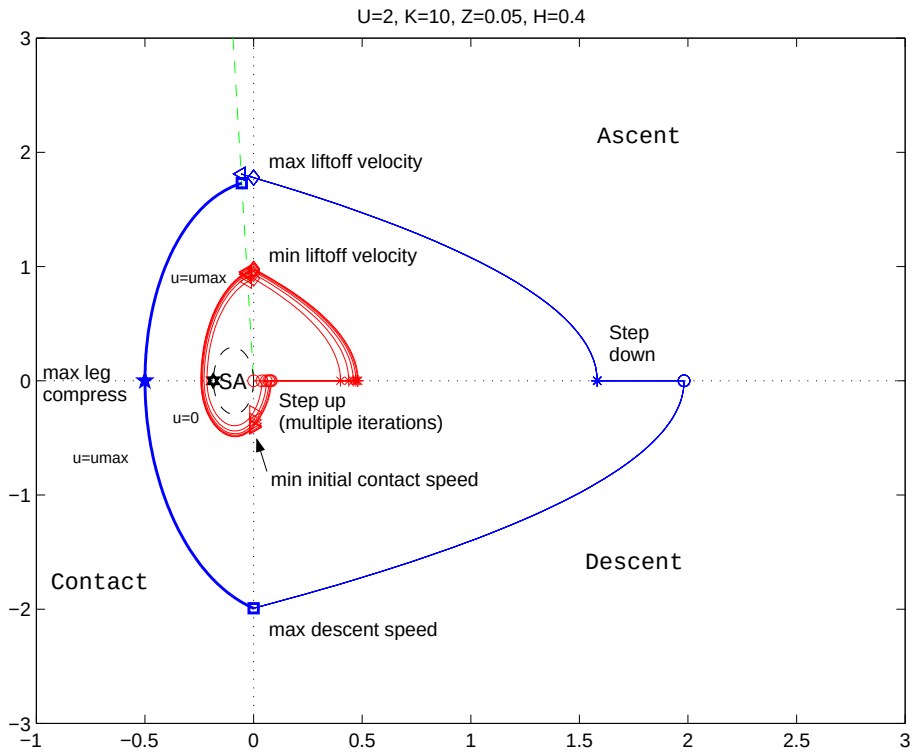


Figure 2.9: Maximal safe invariant set for a hopper with $u_{max} = 2$, $\hat{k} = 10$, $\zeta = 0.05$, and $\Delta\hat{h}_{max} = 0.4$. The algorithm for the maximum safe invariant set requires multiple iterations but converges to a non-empty solution space. In this example the hopper requires non-zero landing speeds to compress the leg so that the thrust can act over a long enough distance to achieve the required liftoff velocity.

2.7 Hopper Parameter Variation and Ruggedness

The previous section described how to determine the maximal safe set for a hopper in an environment where the change in terrain height between steps is bound between maximum and minimum values. This section determines the limit of those values which allows a non-empty safe solution. The maximum terrain change limit is called ruggedness and is a function of leg thrust, stiffness, and damping.

This study assumes $\Delta\hat{h}_{min} = -\Delta\hat{h}_{max}$. To determine ruggedness, the algorithm for the maximal controlled safe invariant is repeated applied but at different values for $\Delta\hat{h}_{max}$. If the safe set is non-empty, $\Delta\hat{h}_{max}$is increased by an increment. If the safe set is empty, $\Delta\hat{h}_{max}$is decreased until a non-empty safe set solution is found. In this study, the ultimate resolution for $\Delta\hat{h}_{max}$is 0.01 or one percent of the uncompressed leg length.

Hopper ruggedness for different leg thrust, stiffness, and damping is shown in Figure 2.10. The figure shows four subplots each with a different level for maximum leg thrust. The axes for each subplot are maximum terrain variability (that is, ruggedness) versus the non-dimensional damping coefficient and the curves in each subplot are for normalized spring stiffnesses of 2, 4, 6, 10, 14, and 20.

The results in Figure 2.10 show several general trends. First, ruggedness always increases with increasing leg thrust capability. Second, for a given leg thrust capability ruggedness increases with leg stiffness at low damping. Maximum ruggedness is achieved at high stiffness and near zero damping. But, at high stiffness ruggedness drops off very quickly with damping. Since all mechanical systems have some form of friction, this suggests that better performance will be achieved with moderate stiffness. Recall also that these results assume a massless leg. The energy loss at ground contact for non-zero mass leg is an equivalent damping. An interesting trend in Figure 2.10 is that for systems with low to moderate stiffness, increasing damping up to an optimum value increases ruggedness. The lower the stiffness, the higher the optimum value for damping. The damping dissipates energy from landing allowing the hopper to tolerate a step down change in terrain height without violating the leg compression constraint. However, as damping increases beyond the optimum value

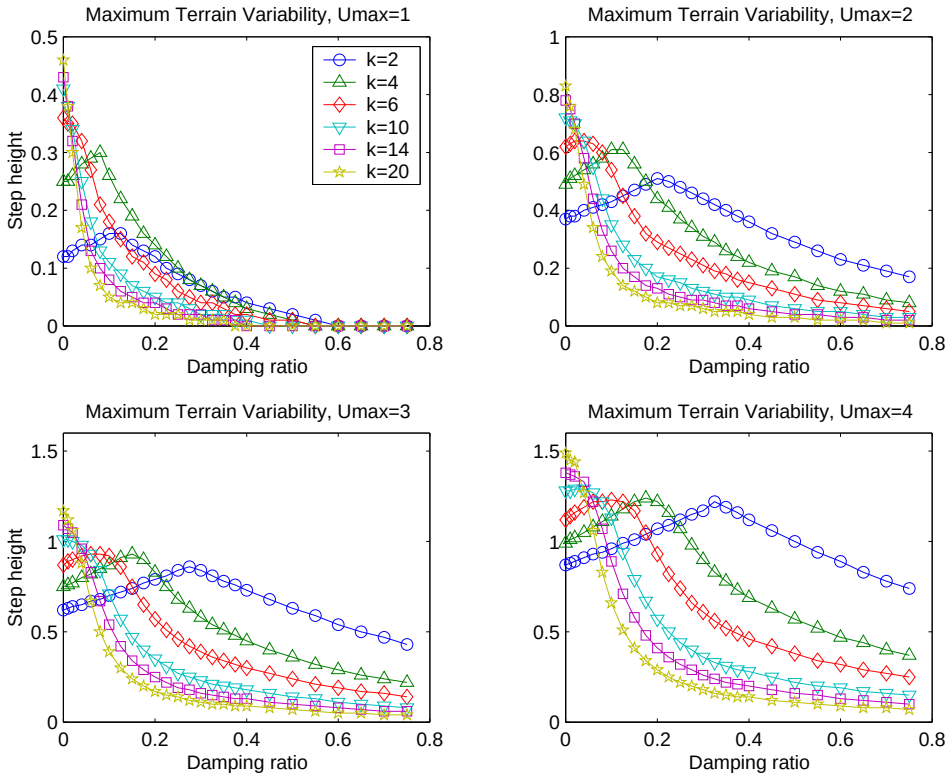


Figure 2.10: Hopper ruggedness as a function of leg thrust, stiffness, and damping. Ruggedness is a measure of the maximum variation in terrain height between steps that the system can safely handle. For zero and very low damping then high leg stiffness is optimal. For moderate and higher levels of damping moderate to low leg stiffness is best. Ruggedness increases uniformly with leg thrust capability and sensitivity to mechanical properties decreases with increasing leg thrust.

ruggedness is limited by the energy dissipated during thrust when the hopper tries to achieve a minimum liftoff velocity. A final trend observed in Figure 2.10 is that at higher thrust levels the difference between optimal ruggedness for hoppers with different leg stiffness diminishes. Hopper capabilities become more and more dominated by control rather than passive dynamics.

To further understand the effect of damping, Figure 2.11 plots hopper ruggedness versus damping for different leg stiffnesses at $u_{max} = 4$. Ruggedness is compared against "obstacle limited" and "compression limited" behaviors which are shown to be limiting values for hopper ruggedness. The "compression limited" curve, shown as a dashed line in the figure, is determined from the maximum height from which the hopper can safely fall. It limits hopper ruggedness at low to moderate leg stiffness and low damping. The "obstacle limited" curve, shown as a dotted line in the figure, is determined from the maximum obstacle height the hopper can clear. It limits hopper ruggedness at moderate to higher levels of damping. Figure 2.12 shows the maximal safe set in state space for hoppers with a dimensionless spring constant $\hat{k} = 4$ but different damping for maximum ruggedness at the "obstacle limited", "compression limited", and intermediate behaviors.

For compression limited behavior, ruggedness is one half the maximum height which the hopper can fall without violating the leg compression constraint. The compression limit is determined for a given set of leg parameters by assuming the leg at maximum compression is at rest and propagating the `contact` equations of motion backwards in time at $u = u_{max}$. This determines the maximum descent speed shown in Figure 2.12 (a). The maximum height above ground after the `step` phase is determined from the maximum descent speed. $\Delta\hat{h}_{max}$is one half the maximum safe height so a step up reduces height above ground to zero and a step down increases height above ground to the safe limit. Therefore, the maximum and minimum height prior to the `step` are equal. Likewise, the maximum and minimum liftoff velocities prior to `ascent` are equal. Extrapolating backwards in time from liftoff with $u = u_{max}$ (dotted curve between the triangle and hexagram), the state trajectory intersects the singularity avoidance boundary. Following along the singularity avoidance boundary in backwards time, the minimum velocity at contact is zero and the iteration for the

maximal safe set terminates. Note that if the step down height were reduced in Figure 2.12 (a), the hopper could safely handle a larger step up so ruggedness is step down or *compression* limited.

For obstacle limited behavior, ruggedness is the maximum height the hopper can clear from initial contact at zero velocity, as shown in Figure 2.12 (c). Starting at $\hat{y} = 0$ and $\hat{y}' = 0$ and propagating forward in time with $u = 0$, the hopper falls under gravity and follows the singularity avoidance boundary until maximum compression ($\hat{y}' = 0$). At this point full thrust is applied to achieve the minimum safe liftoff velocity which determines $\Delta\hat{h}_{max}$. After a step up, the hopper contacts the ground at zero velocity and the iteration for the maximal safe set terminates. For larger $\Delta\hat{h}$ the hopper must land with non-zero velocity, but the hopper cannot provide enough thrust to overcome the energy lost to damping so that the hopper lands with lower velocity on subsequent steps and the state space trajectory spirals inward as shown in Figure 2.8. Note the large distance between maximum and minimum safe height in `ascent` in Figure 2.12(c). The hopper can safely handle a larger step down so ruggedness is step up or *obstacle* limited.

Figure 2.11 shows that at moderate levels of damping ruggedness falls somewhere between the obstacle and compression limited behaviors. State space boundaries for the maximal safe set under these conditions are shown in Figure 2.12 (b). Here, the damping is such that the hopper can sustain stair climbing with a non-zero landing velocity so ruggedness is greater than the step up or "obstacle" limit. However, the gap between maximum and minimum safe height in `ascent` shows that the hopper could safely handle a larger step down so ruggedness is less than the step down or "compression" limited level. For moderate leg stiffness, ruggedness is maximized at this intermediate behavior.

2.8 Chapter Summary

The dynamics for a single legged hopper are derived and the equations of motion are converted to non-dimensional form. The hopper and its interaction with the environment is described as a hybrid system where discrete states describe different phases

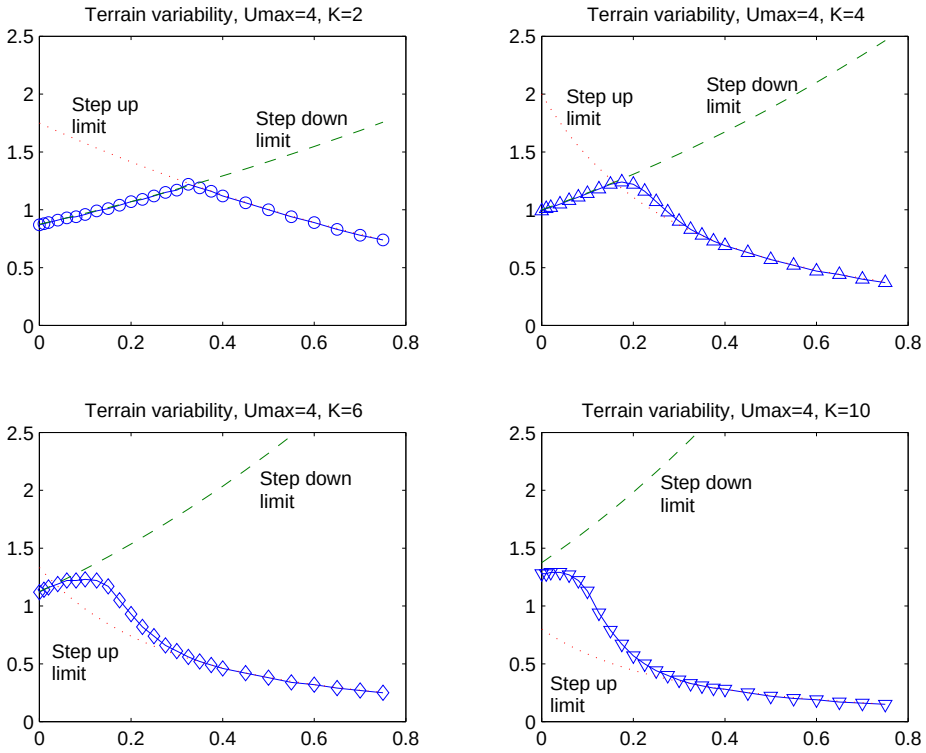
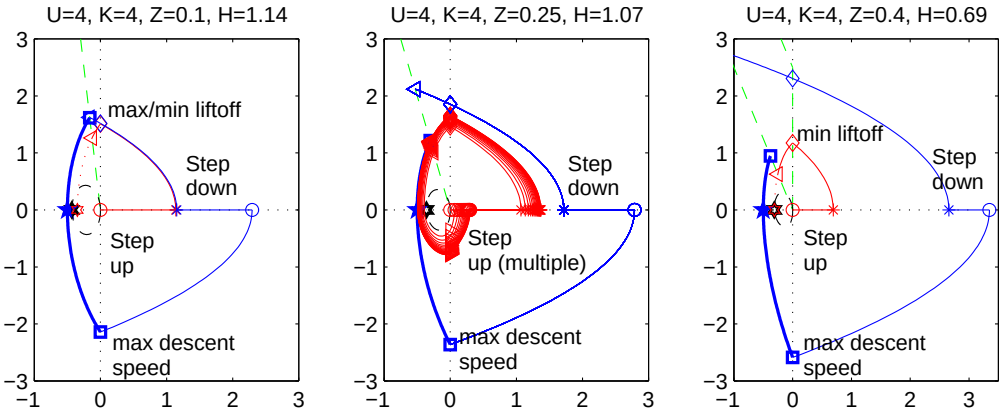


Figure 2.11: Hopper ruggedness and "step up" and "step down" limits. The "step down" limit is the maximum drop in height between successive steps. When step down limited the hopper applies maximum thrust at first contact to avoid joint limits and increasing damping increases ruggedness. The "step up" limit is the maximum rise the hopper can sustain over a succession of steps. When step up limited, the hopper applies maximum thrust at full compression to jump as high as possible and decreasing damping increases ruggedness. Optimal damping occurs near where the maximum step height for "step down" and "step up" limited behaviors intersect.



(a) compression limited (b) intermediate (c) obstacle limited

Figure 2.12: Maximal safe invariant for compression limited, obstacle limited and intermediate behaviors. For compression limited behavior (a) the "step up" plus "step down" height is equal to the height from which the hopper can fall without damage. For obstacle limited behavior (c) the maximum step up is the height which the hopper can clear and land at zero velocity and still compress the leg enough to clear the next step up. Intermediate behavior (b) requires a non–zero landing velocity to provide sufficient leg compression for the next step up. The intermediate behavior requires multiple iterations through the maximal safe invariant algorithm to find the minimum safe landing speed.

of motion and transitions between phases of motion, or system modes, are a function of the continuous state variables. The hybrid model permits terrain height changes at the discrete instant the hopper reaches the apex of flight. This treatment simulates running over uneven terrain as a problem of "hopping in place" on a treadmill. If the change in terrain height results in contact at the foot, the hopper stumbles and fails to clear the obstacle. Alternatively, the terrain can drop below the hopper causing it to land with excess velocity. Because the maximum change in terrain height between steps is constrained, the limiting hopper behaviors are climbing and descending stairs. However, the hopper has no knowledge of the environment and terrain height between steps can vary unpredictably within prescribed limits.

The chapter describes a general algorithm for determining the maximal controlled invariant safe set and the least restrictive control for a hybrid system (Tomlin, et. al., 2000, [48]). The algorithm is a form of dynamic programming (Bryson and Ho, 1975, [9]) and starts with an initial set of safe states for all discrete phases of motion and iterates backwards in time. The subset of states from which control inputs can force discrete transitions between modes and remain with the safe set comprise controllable predecessor states. States which are unsafe, and states from which disturbance inputs can force a discrete transition to the unsafe states comprise the uncontrollable predecessor states. Continuous time propagation between discrete transitions is accomplished using the *Reach* operation.

The *Reach* operation determines all states that can reach the uncontrollable predecessor states without first "escaping" to the set of controllable predecessor states. These states are eliminated from the initial set of safe states and the algorithm continues. The algorithm continues iterating backwards in time until either no safe states remain, or a fixed point solution set is found. Backwards time propagation for the *Reach* operation depends on optimal values for the continuous control and disturbance inputs. A detailed treatment of the optimization problem is deferred to Chapter 3. In this chapter, control inputs are determined by examination of the state plane trajectories.

The algorithm for determining the maximal safe set is applied to a single legged hopper where it is assumed that the controller has perfect feedback but no knowledge

of the environment. Two difficulties applying the algorithm are encountered. First, the hopper control variable, thrust, can directly control the transition between contact and flight phases of motion. Within a narrow portion of the state space varying the control cause undesirable transitions or thrashing between contact and flight. An additional phase, `liftoff`, is used to fix the thrust over this region of state space and prevent thrashing. Secondly, a provision for avoiding singularities in the continuous state space must be made in order to propagate the equations of motion backwards in time. This provision is enforced by defining a singularity avoidance boundary within the `contact` and `liftoff` phases of motion.

Applying the algorithm to the single legged hopper, one finds that the maximal safe set depends on leg thrust, stiffness, damping, and compression constraints, and the allowable terrain variation between steps. In some cases a fixed point solution can be found in a couple of iterations through the hopper phases of motion. In some cases no fixed point solution exists and the maximal safe set is null.

A numerical search was performed to determine the maximum change in height a hopper with given leg stiffness, damping, thrust and compression constraints can safely sustain. The magnitude of the this variation (the step may be either up or down by an equal amount)is a measure of the hopper's "ruggedness" against environmental disturbance. Ruggedness increases uniformly with thrust capability. For a leg with a given thrust capability and stiffness, an optimal damping exists. The stiffer the leg, the lower the optimal value for damping and the more sensitive ruggedness is to increased damping. For hoppers with low to moderate leg stiffness, ruggedness at damping below the optimum is compression limited; ruggedness is determined by how far the hopper can fall and safely dissipate the energy at impact. For damping sufficiently higher than optimum, ruggedness is obstacle limited; ruggedness is limited by the ability to successively clear obstacles (climb stairs). Near the optimal damping level ruggedness is balanced by both compression and obstacle limited behavior.

Chapter 3

Optimization of the Controlled Invariant

The algorithm for the maximal controlled safe invariant described in the previous chapter requires determination of the optimal control and disturbance inputs. Tomlin (1998, [47]) and Tomlin, et al. (1998 and 2000, [46, 48]) formulate the optimization problem using a pair of interconnected Hamilton–Jacobi equations. The Hamilton–Jacobi equations are first order non-linear partial differential equations. Their solution is complicated by the presence of discontinuities or "shocks" and numerical techniques are an active area of research (see Tomlin, et. al., 2000, [48]). In this work, we restrict the class of hybrid systems and associated safety constraints under consideration and show that explicit solution of the Hamilton–Jacobi equation is unnecessary. This result greatly simplifies the procedure and makes numerical solution to a large class of problems feasible.

3.1 A Restricted Class of Hybrid Systems

The hybrid system described in Definition 1 of the previous chapter is very general. The problems with applying the algorithm for the maximal safe invariant, encountered in section 2.6.1 and 2.6.2, point to the need for some additional restriction

and refinement. These additional restrictions are summarized below. These restrictions simplify the *Reach* and backward chaining operations. Justifications for these restrictions are elaborated in the following sections.

- The continuous control and disturbance inputs, u and d, for all continuous phases of motion, Φ_q, are bound.
- The field relationships, f_q, are separable in u and d so that $f_q(x,u,d) = f_{q_1}(x,u) + f_{q_2}(x,d)$ and are not explicit functions of time.
- Constraints on the control input, u, must be imposed to avoid any singularity at $f_q(x,u,d) = 0$, which halts time propagation of the continuous state, x.
- The initial or entrance conditions for each continuous phase of motion are constrained to an N-1 dimensional manifold satisfying $\psi_{Init}(x) = 0$ where ψ_{Init} is time invariant and is not a function of u or d.
- The set of unsafe states, G, within a mode consist of possibly intersecting compact, convex subspaces, G_i, with continuous boundaries which are time invariant but may be linear functions of u or d.
- The exit conditions for each continuous phase of motion (defined by the reset relation, R) are constrained to one or more N-1 dimensional manifolds satisfying $\psi_{Exit}(x) = 0$ where $\psi_{Exit}(x)$ is time invariant and is not a function of u or d.
- The transition from the current phase, q_i, to the subsequent phase, q_j, depends only on the current phase (not previous phases), the continuous states, x, and any discrete inputs, σ_1 and σ_2: $(q_j, x^+) = R(q_i, x, \sigma_1, \sigma_2)$.
- The mapping between continuous states through a phase transition is a linear function of the state and discrete inputs: $x^+ = Ax + B_1\sigma_1 + B_2\sigma_2$.

Limits on the control and disturbance inputs are required for finding extremal trajectories in the continuous state space when u and d are linear in $f_q(x,u,d)$. Separability of $f_q(x,u,d)$ simplifies determination of the optimal values for u and d during

the *Reach* operation. The constraints on singularity avoidance can be enforced by defining a singularity avoidance region, as described in section 2.6.2, or, for example, by creating additional modes which limit legal values of $u \in U_q$ so that $f_q(x, u, d) \neq 0$.

The restrictions imposed on initial and exit condition manifolds are required so that transitions between continuous modes are unambiguous in both forward and backward time (the transition may, however, be affected by a discrete control or disturbance input). Likewise, the restriction that phase or mode transitions depend only on the current mode and the continuous state vector eliminates any path dependence on the transition relationship. Note that these restrictions do not prevent transition from a predecessor mode to several possible successor modes, nor do the restrictions prevent transition to a successor mode from different possible predecessor modes. The restrictions on the set of unsafe states, G, will be described in the following section, but support calculation of the *Reach* operation.

The final restriction applies to discrete changes in the continuous state vector between modes. The linear mapping can describe a variety of transition phenomena including the `step` change for the hopper problem as well as elastic and inelastic collisions. The linear mapping is reversible and permits super-position which is required for the backward chaining portion of the algorithm.

3.2 The Reach Operation for Restricted Hybrid Systems

By restricting mode transitions to boundaries or manifolds within the continuous state space, the algorithm for finding the maximum invariant safe set can consider a single mode at a time (Tomlin, et. al., 2000, [48]). Assume that for each mode, Φ_q, the subset of the useable state space, $X_q \in \mathbb{R}^N$, is bound by an N-1 dimensional manifold, $\psi_{Init}(x) = 0$, defining initial conditions, an N-1 dimensional manifold, $\psi_{Exit}(x) = 0$, defining exit conditions, and one or more N-1 dimensional manifolds, $\psi_i(x) = 0$, connecting ψ_{Init} and ψ_{Exit}. The connecting manifolds are generated by integrating the continuous time equations of motion, $\dot{x} = f_q(x, u, d)$.

Now for each mode, Φ_q, assume a set of unsafe states, G, (which may be empty) and a desired set of escape states, E. The escape states transition to a new mode and accordingly are a subset of ψ_{Exit}. *Reach(G,E)* determines the regions of X_q from which trajectories will reach or pass through G or avoid E despite the best choice for u. The safe set is the complement of *Reach(G,E)* in X_q. Tomlin, et. al. (2000, [48]) solve this problem using dynamic programming techniques and the Hamilton–Jacobi equation from optimal control. Bryson and Ho (1975, [9]) also discuss optimal solutions to game theoretic problems.

Optimization requires a value or cost function. In this case there are two interconnected optimization problems: one is to avoid the unsafe states G, the other is to achieve the escape set E. These objectives are encoded in the value functions $J_G(x,t)$ and $J_E(x,t)$ below.

$$J_G(x,u,d,t) = \phi_G(x(t_f)) \quad \text{where} \quad \begin{cases} \phi_G(x) < 0 & \text{for } x \in G \\ \phi_G = 0 & \text{for } x \notin G \end{cases} \tag{3.1}$$

$$J_E(x,u,d,t) = \phi_E(x(t_f)) \quad \text{where} \quad \begin{cases} \phi_E(x) < 0 & \text{for } x \in E \\ \phi_E = 0 & \text{for } x \notin E \end{cases} \tag{3.2}$$

The functions ϕ_G and ϕ_E have a maximum value of zero. The player, or hopper in our previous examples, wishes to maximize J_G and minimize J_E. The environment wishes the opposite.

The value functions depend only on the continuous state vector, x, at the terminal time, t_f, which is arbitrary but fixed. To enforce transitions between modes we impose the terminal boundary constraints $\psi_{Exit}(x(t_f)) = 0$ and $\psi_{Init}(x(t_0)) = 0$. For $t < t_f$, J_G and J_E are functions of the continuous state vector, x, the control and disturbance inputs, u and d, respectively, and time, t. The state evolves according to $\dot{x} = f_q(x,u,d)$ and we assume that the control and disturbance inputs are separable so that $f_q(x,u,d) = f_{q_1}(x,u) + f_{q_2}(x,d)$.

The value functions in 3.1 and 3.2 contain no running cost because the safety constraints and exit or escape conditions depend only on the state vector. This leaves the possibility that $x(t) \in G$ for $t < t_f$, which is undesirable. Tomlin, [47], addresses

this problem with a modification to the Hamiltonian to be described shortly. Another approach described later in this chapter is to divide G into a collection of possibly intersecting convex subspaces and to enforce transversality conditions (Stengel, 1986, [43]) at the boundary of each subspace.

For the purpose of safety, the controller assumes worst case disturbance inputs. In the context of game theory, the worst case is determined by assuming the environment can anticipate the control. In other words, the control input is determined assuming the disturbance can assume any value within the legal range, but the disturbance input is determined after the control input is set. This gives an advantage to the environment and generates a conservative design. In game theory, this approach is known as a *Stackelberg solution* [9, 48] and is useful for problems where a saddle point solution is not guaranteed.

Assume for now that the optimization problems can be considered separately. The optimization for J_G finds the control which avoids the unsafe states G despite finite but unknown disturbances. Optimal values for J_G, u, and d are denoted with an asterisk below.

$$\begin{aligned} J_G^*(x,t) &= \max_{u\in U}\min_{d\in D} J_G(x,u,d,t) && (3.3)\\ u^* &= \arg\max_{u\in U}\min_{d\in D} J_G(x,u,d,t) && (3.4)\\ d^* &= \arg\min_{d\in D} J_G(x,u^*,d,t) && (3.5) \end{aligned}$$

Similarly, the second optimization problem is to find the control which reaches the escape set E despite disturbances. The optimal control minimizes J_E over the possible disturbance inputs:

$$\begin{aligned} J_E^*(x,t) &= \min_{u\in U}\max_{d\in D} J_E(x,u,d,t) && (3.6)\\ u^* &= \arg\min_{u\in U}\max_{d\in D} J_E(x,u,d,t) && (3.7)\\ d^* &= \arg\max_{d\in D} J_E(x,u^*,d,t) && (3.8) \end{aligned}$$

Assuming J_G^* and J_E^* exist, are continuous, and contain continuous first and second derivatives for all points of interest in the (x,t) space, the Hamilton–Jacobi equations, below, can be derived from arguments on the optimality of the value functions and the constraint that $\dot{x} = f_q(x,u,d)$, (Bryson and Ho, 1975 [9], and Tomlin, 1998, [47]).

$$\begin{aligned} -\frac{\partial J_G^*}{\partial t} &= \max_{u\in U}\min_{d\in D}\left\{\frac{\partial J_G^*}{\partial x} f_q(x,u,d)\right\} \\ &= \left(\frac{\partial J_G^*}{\partial x}\right) f_q(x,u^*,d^*) \qquad (3.9) \\ -\frac{\partial J_E^*}{\partial t} &= \min_{u\in U}\max_{d\in D}\left\{\frac{\partial J_E^*}{\partial x} f_q(x,u,d)\right\} \\ &= \left(\frac{\partial J_E^*}{\partial x}\right) f_q(x,u^*,d^*) \qquad (3.10) \end{aligned}$$

To address the possibility that $x(t) \in G$ for $t < t_f$, Tomlin, et. al. (2000, [48]) define the interconnected set of optimal Hamiltonians H_G^* and H_E^*.

$$H_G^*(x, \frac{\partial J_G^*}{\partial x}) = \begin{cases} 0 & \text{for } \{x \in X \mid J_E^*(x,t) < 0\} \\ \max_{u\in U}\min_{d\in D} \frac{\partial J_G^*}{\partial x} f_q(x,u,d) & \text{otherwise} \end{cases} \qquad (3.11)$$

$$H_E^*(x, \frac{\partial J_E^*}{\partial x}) = \begin{cases} 0 & \text{for } \{x \in X \mid J_G^*(x,t) < 0\} \\ \min_{u\in U}\max_{d\in D} \frac{\partial J_E^*}{\partial x} f_q(x,u,d) & \text{otherwise} \end{cases} \qquad (3.12)$$

Note that $J_E^* < 0$ if x can be forced to reach the escape set, E, despite worst case disturbances. Then the optimal Hamiltonian, H_G^*, is zero. Similarly, $J_G^* < 0$ if x can be forced to the unsafe condition G despite all efforts by the control, and the optimal Hamiltonian, H_E^*, is zero. Inserting the equations 3.11 and 3.12 into 3.9, Tomlin, et. al. (2000 [48]) create the interconnected set of Hamilton–Jacobi equations below.

$$-\frac{\partial J_G^*(x,t)}{\partial t} = \begin{cases} H_G^*(x, \frac{\partial J_G^*}{\partial x}) & \text{for } x \in X | J_G^* > 0 \\ \min\{0, H_G^*(x, \frac{\partial J_G^*}{\partial x})\} & \text{for } x \in X | J_G^* \leq 0 \end{cases} \qquad (3.13)$$

$$-\frac{\partial J_E^*(x,t)}{\partial t} = \begin{cases} H_E^*(x, \frac{\partial J_E^*}{\partial x}) & \text{for } x \in X | J_E^* > 0 \\ \min\{0, H_E^*(x, \frac{\partial J_E^*}{\partial x})\} & \text{for } x \in X | J_E^* \leq 0 \end{cases} \tag{3.14}$$

Tomlin, et. al. (2000, [48]) show that the *Reach* operation is obtained from the optimization described above. Their theorem is repeated below:

Theorem 1. ***Characterization of Reach:*** *Assume that $J_G^*(x,t)$ and $J_E^*(x,t)$ are smooth functions of x and t, that the satisfy the Hamilton–Jacobi equations in 3.13 and 3.14, and that they converge uniformly in x as $t \to -\infty$ to functions $J_G^*(x)$ and $J_E^*(x)$, respectively. Then,*

$$Reach(G,E) = \{x \in X | J_G^*(x) < 0\} \tag{3.15}$$

The solution for $Reach(G,E)$ requires numerical approximation for all but the simplest of problems. The numerical solution is complicated by the fact that the initial data may have a non-smooth boundary, that the control and disturbance may be discontinuous, and that $J^*(x,t)$ may not be continuous. These discontinuities are known as *shocks*. Tomlin, et. al. (2000, [48]) describe various numerical approximations for solving the Hamilton–Jacobi equations including a *viscosity solution* to address the discontinuous $J^*(x,t)$.

However, the value functions J_G and J_E in equations 3.1 and 3.2 depend only on the final states and do not contain any integral terms for running costs. They are not explicit functions of time, hence the left hand sides of equations 3.13 and 3.14 are zero. From equations 3.13 and 3.11, when $H_G^* = 0$ and $H_G^* = \frac{\partial J_G^*}{\partial x} f_q(x, u^*, d^*)$ and $f_q(x, u^*, d^*) \neq 0$, the gradient of the value function with respect to x is perpendicular to the derivative with the optimal control and disturbance inputs. A similar observation holds for J_E^*. If the interconnected optimization in equations 3.11 and 3.12 can be separated, the problem is greatly simplified.

3.2.1 Unsafe Set Optimization

Consider the optimization problems in equations 3.3 and 3.6 separately and allow that the final time for J_G, t_{f_G}, may be different than the final time for J_E, t_{f_E}, so that

$t_{f_G} \leq t_{f_E}$. In other words, the state trajectory reaches ψ_{Exit} and forces a transition to a new mode at time t_{f_E} but may enter or touch G at t_{f_G} prior to t_{f_E}. At time t_{f_G} we have that $J_G^* = \phi_G(x(t_{f_G}))$ and that:

$$\left.\frac{\partial J_G^*}{\partial x}\right|_{t=t_{f_G}} = \left(\frac{\partial \phi_G}{\partial x}\right)_{t=t_{f_G}} \tag{3.16}$$

The function ϕ_G is some constant less than zero for $x \in G$ and $\phi_G = 0$ for $x \notin G$. Thus $\frac{\partial \phi_G}{\partial x} = 0$ everywhere except along the border of G, designated by ∂G. In other words, at time t_{f_G} the gradient of the value function, $\frac{\partial J_G^*}{\partial x}$, and therefore the optimal Hamiltonian, H_G^*, is zero everywhere except on ∂G. The optimization starting at points away from ∂G is undefined (equation 3.9 reduces to $0 = 0$) and there is no optimal value for u or d. Accordingly, we restrict our attention to those trajectories which terminate on ∂G.

$\frac{\partial \phi_G}{\partial x}$ is the outward normal of G. Since $\frac{\partial J_G^*}{\partial t} = 0$, equation 3.9 requires that $f_q(x, u^*, d^*)$ be tangent to ∂G at t_{f_g}. Hence,

$$\frac{\partial \phi_G}{\partial x} f_q(x, u^*, d^*)_{t=t_{f_G}} = 0 \tag{3.17}$$

Otherwise, if $\frac{\partial \phi_G}{\partial x} f_q(x, u^*, d^*) < 0$ then for a small step forward in time $x \in G$. Likewise, if $\frac{\partial \phi_G}{\partial x} f_q(x, u^*, d^*) > 0$ then for a small step backward in time $x \in G$. Another interpretation is that at time t_{f_G} we require the state to be on the border, ∂G, then, equation 3.17 is the *transversality* condition for the terminal constraint (see Stengel, 1986, [43], page 243).

Equation 3.17 will not be satisfied at all points along ∂G. If u is bound the condition may not be satisfied anywhere on ∂G, implying that no invariant set of safe states exist within mode q. In general, the search for $\{x \in \partial G | \frac{\partial \phi_G}{\partial x} f_q(x, u^*, d^*) = 0\}$ can be difficult requiring a search over u and d. If f_q is linear in u and d the problem is simplified since from Pontryagin's *Maximum Principle* u^* and d^* will be at maximum or minimum limits so that only a finite set of values for u and d need to be considered (see Bryson and Ho, 1975, [9], page 135).

As an example, consider the vertical hopper from Chapter 2 in `contact` mode. The dynamics are a linear function of the leg thrust, u. The variable disturbance

input, d, is zero (gravity is a constant in the equations of motion). For simplicity, assume zero damping and that position and velocity are normalized so that the state space trajectories are circular arcs centered about equilibria points where $f(x, u) = 0$, as shown in Figure 3.1. The equilibria, or singularities, are a function of u, and two points, one each for $u = u_{min}$ and $u = u_{max}$, are shown. The solid arcs are the state space trajectories for $u = u_{max}$. The broken arcs are the state space trajectories for $u = u_{min}$. The state trajectories are shown only for positions less than zero because if greater than zero the system is no longer in `contact`. The singularity avoidance region, described in section 2.6.2, is shaded and shown for reference.

Figure 3.1 shows two overlapping subspaces of unsafe states. G_1 includes all states for which the position, x, is less than minimum value. G_2 is the set of states for which velocity, v, exceeds a maximum. Note that G_2 is an addition to the original hopper `contact` mode example and is included for purposes of example. The boundaries of these subspaces form the safety constraints $x \geq x_{min}$ and $v \leq v_{max}$ as shown in the figure. The outward normals, $\frac{\partial \phi_G}{\partial x}$, for both constraints are shown. For G_1, $f(x, u^*)$ is perpendicular to the outward normal at a single point. In fact, this point is independent of the value for u, but that is not generally the case. For G_2 there are a range of points over which $f(x, u^*)$ can be made perpendicular to the outward normal and the appropriate value for u^* changes with position. However, there is a point along ∂G_2 where $u^* = u_{min}$ and beyond which the transversality condition can no longer be maintained.

In general, the boundary, ∂G, is an N-1 dimensional manifold in state space. The set of states on ∂G that also satisfy transversality conditions form an N-2 dimensional manifold when solutions exist. For example, in Figure 3.1, the transversality constraint for ∂G_2 is shown at $u = u_{min}$ but can not be satisfied at $u = u_{max}$ because the solution occurs outside the `contact` region. The states for which transversality conditions are satisfied with extreme values for u and d are shown with an asterisk in Figure 3.1 and will be referred to as *extrema* states. In Figure 3.1, trajectories forward and backward in time from the extrema states are highlighted by heavy solid lines.

Propagation of the value function in backwards time ($t \leq t_{f_G}$) is governed by

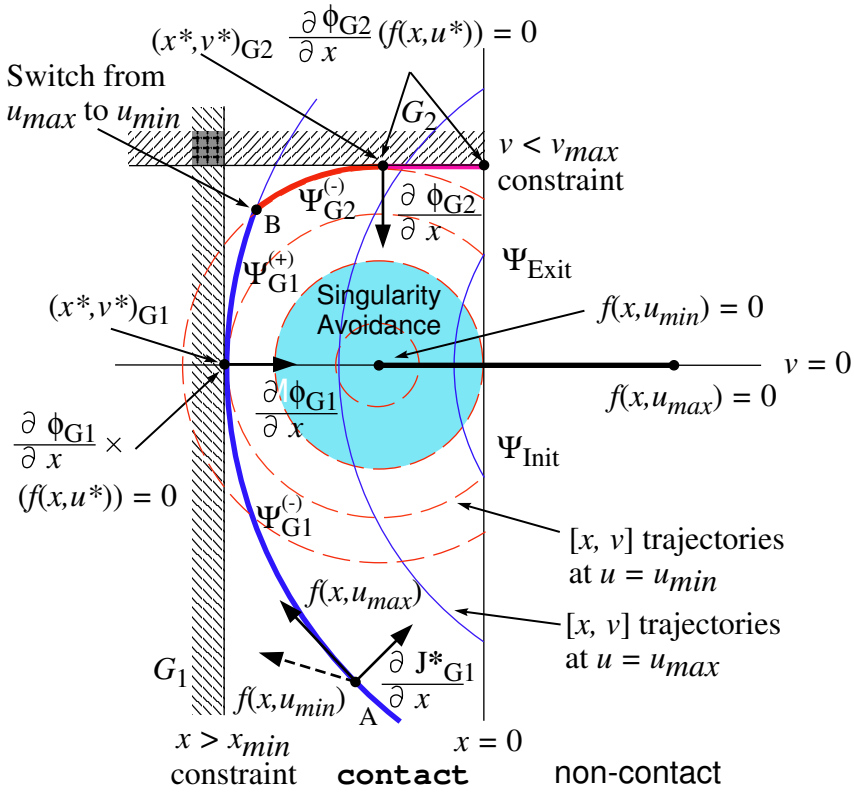


Figure 3.1: Safety constraint portion of the *Reach* operation, hopper `contact` mode example. G_1 is the set of states which violate the minimum length requirement. G_2 is the set of states which violate a maximum speed and is an addition to the original hopper `contact` problem. The unsafe states form constraints along their boundaries. State space trajectories for $u = u_{max}$ (solid lines) and $u = u_{min}$ (broken lines) are shown. Points along the constraint boundaries which can satisfy *transversality* conditions are identified. *Extrema* points satisfy transversality conditions at $u \in \{u_{min}, u_{max}\}$. Propagation forward and backward in time from the extrema points generate the extremal curves of the *Reach* operation. In backwards time propagation u^* is the value that maximizes the inner product between $f(x,u)$ and the surface normal, $\frac{\partial J^*_G}{\partial x}$. In forwards time propagation u^* is the value that minimizes the inner product.

equation 3.9. Propagation in forward time reverses the sense of the optimization so that:

$$-\frac{\partial J_G^*}{\partial t} = \max_{u\in U}\min_{d\in D}\left\{\frac{\partial J_G^*}{\partial x} f_q(x,u,d)\right\} \quad \text{for } t \le t_{f_G} \tag{3.18}$$

$$-\frac{\partial J_G^*}{\partial t} = \min_{u\in U}\max_{d\in D}\left\{\frac{\partial J_G^*}{\partial x} f_q(x,u,d)\right\} \quad \text{for } t > t_{f_G} \tag{3.19}$$

Since $\frac{\partial J_G^*}{\partial t} \equiv 0$, the gradient $\frac{\partial J_G^*}{\partial x}$ must remain perpendicular to $f_q(x,u^*,d^*)$ both forwards and backwards in time. This observation simplifies the optimization for the control and disturbance inputs. Optimal inputs can be determined from the geometry of the integral curve and explicit solution of the Hamilton–Jacobi equation is not required.

Intuitively, the control attempts to expand the region of safe states by pushing the state space trajectory 'outward' while the disturbance attempts to contract the region of safe states by pulling the trajectory 'inward'. Starting from the N-2 dimensional manifold of extrema states, propagation forward or backward in time will generate an N-1 dimensional surface, ψ_G^+ or ψ_G-. The perpendicular to this surface is along the direction of the gradient $\frac{\partial J_G^*}{\partial x}$. Accordingly, at each instant in time we determine the perpendicular to the surface of state trajectories and select u^* and d^* to satisfy equation 3.18 or 3.19. The time propagation continues until the state trajectory intersects either the initial condition manifold, ψ_{Init}, in backwards time, or the exit condition manifold, ψ_{Exit}, in forward time. Numerical approximation to the time propagation is described in more detail in Chapter 4.

Returning to the example illustrated in Figure 3.1, the point at $(x^*,v^*)_{G1} = (x_{min},0)$ is an extremum at ∂G_1 for any value $u_{min} \le u \le u_{max}$. Integrating $f(x,u)$ in backwards time, the inner product between $f(x,u)$ and the normal to the state trajectory is maximized at $u^* = u_{max}$. This is illustrated at point A on psi_{G1}^- (towards the bottom of Figure 3.1), where the surface normal at a point along the state trajectory is designated by $\frac{\partial J_{G1}^*}{\partial x}$ and the vectors $f(x,u_{max})$ and $f(x,u_{min})$ are shown. In forward time, the inner product between $f(x,u)$ and the normal to the state trajectory is also minimized at $u^* = u_{max}$. Thus, the ψ_{G1}^+ and ψ_{G1}^- extremal trajectories

start at $(x^*, v^*)_{G1}$ and continue with $u^* = u_{max}$ going both forward and backward in time.

Similarly, we propagate forward and backward in time from the extremum at $(x^*, v^*)_{G2}$. In this case, the backwards time the optimum value for u is $u^* = u_{min}$ and the resulting trajectory intersects the G_1 extremal at point B as shown in Figure 3.1. However, going forward in time the trajectory is constrained by ∂G_2 and the optimal control is determined to satisfy this constraint.

If G_i is convex, then since the derivative $f_q(x^*, u^*, d^*)$ at t_{f_G} is tangent to ∂G_i, the extremal curves will not re-enter G_i (provided G_i does not contain any singularities where $f_q(x, u, d) = 0$, and that $f_q(x, u, d)$ is otherwise well behaved – see Nemystskii and Stepanov, 1989, [35], pages 30-38, for discussion of regular dynamic systems). So, the forward and backward time propagation can proceed without concern over violating the ∂G_i constraint after leaving the surface. A composite of the G_i extremals can be constructed from the intersection between curves as illustrated in Figure 3.1.

Although not considered here, it is possible that G may depend on the control and disturbance inputs u and d. An example discussed in Chapter 6 is sliding contact in planar hopping. The onset of sliding motion is a function of contact forces which are directly influenced by leg thrust and hip torque. Provided that the boundary of G is a linear function of u and d we can apply the maximum principal and define $G_i = G_i(x, u_j, d_k)$ where $u_j \in \{u_{min}, u_{max}\}$ and $d_k \in \{d_{min}, d_{max}\}$. The solution for the unsafe set optimization proceeds as described above except that the extremum points for each G_i boundary are determined only at the appropriate values for u_j and d_k.

3.2.2 Escape Set Optimization

Now consider the optimization for the escape set E, given in equation 3.6. The optimization is similar to that described for the unsafe set G, except that E is a subset of the exit condition manifold ψ_{Exit}. At time t_{f_E} the state vector x is constrained to a point in ψ_{Exit}. To satisfy this constraint we modify the terminal condition on $\frac{\partial J_E^*}{\partial x}$ (see Bryson and Ho, 1975, [9], page 136) so that:

$$\left.\frac{\partial J_E^*}{\partial x}\right|_{t=t_{f_E}} = \left(\frac{\partial \phi_E}{\partial x} + \nu \frac{\partial \psi_{Exit}}{\partial x}\right)_{t=t_{f_E}} \tag{3.20}$$

Where ν is a positive scalar (for a single constraint ψ_{Exit}) so that equation 3.10 is satisfied at time t_{f_E}. Since $\frac{\partial J_E^*}{\partial t} = 0$, equation 3.10 requires that:

$$\left(\frac{\partial \phi_E}{\partial x} + \nu \frac{\partial \psi_{Exit}}{\partial x}\right) f_q(x, u^*, d^*)_{t=t_{f_E}} = 0 \tag{3.21}$$

Returning to the example of the vertical hopper in `contact` mode, Figure 3.2 shows a set of state space trajectories for $u = u_{max}$ and $u = u_{min}$, the terminal constraint, ψ_{Exit}, and the escape set E (the scale in Figure 3.2 is different than in 3.1 and G_2 has been removed for purposes of illustration). The terminal constraint, ψ_{Exit}, is at the vertical line $x = 0$ and $v > 0$ and labeled toward the top right of the figure. The escape set, E, is indicated with a heavy gray line along a portion of ψ_{Exit}. The extrema points $(x_1^*, v_1^*)_E$ and $(x_2^*, v_2^*)_E$ are shown at the boundaries of E. At the $(x_1^*, v_1^*)_E$ extremum, the gradient $\frac{\partial \phi_E}{\partial x}$ (where x is the state vector $[x\ v]^T$) is vertical upward in the [0 1] direction. At the $(x_2^*, v_2^*)_E$ extremum, the gradient $\frac{\partial \phi_E}{\partial x}$ is vertical downward in the [0 −1] direction. At both points the gradient for the terminal constraint, $\frac{\partial \psi_{Exit}}{\partial x}$ is in the [1 0] direction.

In Figure 3.2 the state vector derivatives, $f(x, u)$, are shown at the extrema points. At the $(x_1^*, v_1^*)_E$ extremum, the directions for both $f(x, u_{min})$ and $f(x, u_{max})$ are shown. $f(x, u_{min})$ is indicated with a solid line, and $f(x, u_{max})$ is indicated with a broken line. The inner product $\frac{\partial \phi_E}{\partial x} f(x, u)$ is minimized when $u = u_{min}$. Conversely, at the $(x_2^*, v_2^*)_E$ extremum, the gradient $\frac{\partial \phi_E}{\partial x} f(x, u)$ is in the opposite direction and $f(x, u_{max})$ minimizes the inner product. At both extrema, the gradient for the terminal constraint, $\frac{\partial \psi_{Exit}}{\partial x}$, points along the [1 0] direction. Also at both extrema, there exists a positive value for ν that satisfies equation 3.21.

Time propagation from the extrema points is only in backwards time since for $t > t_{f_E}$ a mode transition occurs. From point $(x_1^*, v_1^*)_E$ the state trajectory follows the heavy curve through points A, B, and C. At point A, the inner product between $\frac{\partial J_E^*}{\partial x}$ and $f(x, u)$ is minimized for $u = u_{min}$ per the optimization in equation 3.10. However, the optimal value for u changes at point B. As illustrated at point C, the

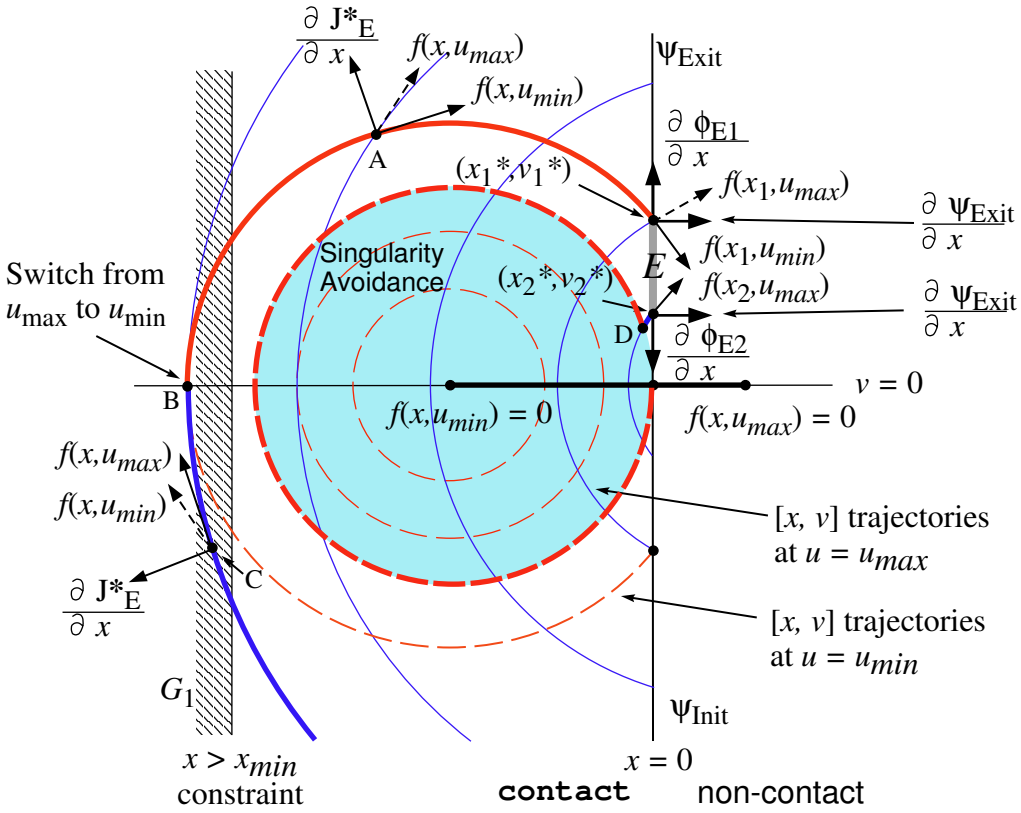


Figure 3.2: Escape set portion of the *Reach* operation, hopper `contact` mode example. The escape set, E, is a subset of the terminal constraint $\frac{\partial \psi_{Exit}}{\partial x}$. The optimal control at the (x_1^*, v_1^*) extremum is $u^* = u_{min}$. The optimal control at the (x_2^*, v_2^*) extremum is $u^* = u_{max}$. Over backward time propagation from (x_1^*, v_1^*) the optimal control remains at u_{min} through point A until switching to u_{max} at point B and passing through point C. The extremal path is indicated by heavy solid lines. Over backward time propagation from (x_2^*, v_2^*), the optimal control remains u_{max} only briefly before the state trajectory intersects the Singularity Avoidance region (shaded) at point D. Although legal trajectories exist through the region, the optimal control may be undefined. The algorithm avoids this difficulty by following the boundary of the Singularity Avoidance region marked by the heavy broken line. For reference with respect to the safety constraint problem in Figure 3.1, the unsafe region G_1 is included here, but G_2 has been removed for clarity.

inner product beyond point B is minimized at $u = u_{max}$. The propagation continues without regard to the safety constraint at ∂G_1 and continues to the initial condition manifold, ψ_{Init} (not shown).

From point $(x_2^*, v_2^*)_E$ the backwards time propagation starts at $u = u_{max}$ and very quickly intersects the singularity avoidance region at point D. Time propagation through the singularity avoidance region is possible, but the optimization is not well defined. For example at $v = 0$ any value for u meets the optimization criteria and there is a value for u^* for which $f(x, u^*) = 0$ so that the backwards time integration stops. Such difficulty can be avoided heuristically, but the heuristics will be problem specific and are beyond the scope of the present work. Rather, we assume the control is constrained to follow the singularity avoidance boundary and the state trajectory follows the heavy broken line to ψ_{Init}.

3.2.3 The Combined Optimization

The optimizations in equations 3.9 and 3.10 were considered separately by assuming different final times ($t_{f_G} \leq t_{f_E}$) for each problem. Since neither the value functions, J_G and J_E, the equations of motion, $f(x, u, d)$, nor the initial and exit conditions, ψ_{Init} and ψ_{Exit} are explicit functions of time, the J_G solution can be propagated forward in time and the optimizations can be combined.

Figure 3.3 shows the combined optimization [1]. The set of safe states is bound by the intersection of the extremal curves from the J_G and J_E optimizations. The intersection at point B in the figure is an example of a *shock* point in the combined optimization discussed earlier in this chapter. Starting from any point inside the intersection set, the continuous states will avoid the unsafe states in G and reach the escape set in E provided the specified control, u^*, is applied if the trajectory reaches an extremal boundary. With the exception of the Singularity Avoidance, the control is unspecified except at these boundaries so that the control is *least restrictive*. The Singularity Avoidance is a region where additional constraints must be placed on the control to force a transition to another mode. These constraints are in addition to

[1] The G_2 constraint in Figure 3.1 is omitted so the example more closely resembles the hopper problem in Chapter 2

the optimization problem and beyond the scope of current discussion. Alternatively, the boundary of the singularity avoidance region could be used to define transition to a new mode where control inputs are constrained in such a way to avoid singularities. Finally, the *Reach(G,E)* operation is the complement of the combined optimization (except for Singularity Avoidance) and is indicated by the cross hatched region in the figure.

In summary, the *Reach(G,E)* operation is accomplished by separating the optimization in equations 3.9 and 3.10 and executing the following steps:

1. Divide the unsafe set of states, G, into possibly intersecting, compact, convex subspaces, G_i, with continuous boundaries ∂G_i.

2. Define the outward normal along each boundary, $\frac{\partial \phi_{G_i}}{\partial x}$.

3. Identify points along ∂G_i where the derivative, $f(x,u,d)$, is perpendicular to the outward normal: $\{x_{G_i} \in \partial G_i \mid \exists u \in U,\ \forall d \in D,\ \left(\frac{\partial \phi_{G_i}}{\partial x}\right) f(x,u,d) = 0\}$. If the control is bound identify the extrema points, ${x_{G_i}}^*$, if any, where $u \in \{u_{min}, u_{max}\}$ and $d \in \{d_{min}, d_{max}\}$. (If x_{G_i} exist but there are no extrema points, the state trajectory can follow the constraint boundary over its entire surface and there is no need for time propagation of the J_G optimization.)

4. Determine u^* and d^* at the extrema points ${x_{G_i}}^*$ from equation 3.17.

5. Propagate backwards in time from each extrema $x^*_{G_i}(t_{f_G})$, generating the N-1 dimensional surface $\psi^{(-)}_{G_i}$ and determining u^* and d^* at each instant in time from equation 3.18, where $\frac{\partial {J_{G_i}}^*}{\partial t} = 0$ and $\frac{\partial {J_{G_i}}^*}{\partial x}$ is the outward normal of $\psi^{(-)}_{G_i}$ at ${x_{G_i}}^*(t)$. The propagation continues until ${x_{G_i}}^*(t) \in \psi_{Init}$ or until reaching a Singularity Avoidance region, at which point the control is prescribed by the SA boundary.

6. Propagate forwards in time from each extrema $x^*_{G_i}(t_{f_G})$, generating the N-1 dimensional surface $\psi^{(+)}_{G_i}$ and determining u^* and d^* at each instant in time from equation 3.19, where $\frac{\partial {J_{G_i}}^*}{\partial t} = 0$ and $\frac{\partial {J_{G_i}}^*}{\partial x}$ is the outward normal of $\psi^{(+)}_{G_i}$ at ${x_{G_i}}^*(t)$, The propagation continues until ${x_{G_i}}^*(t) \in \psi_{Exit}$ or until reaching a Singularity Avoidance region, at which point the control is prescribed by the SA boundary.

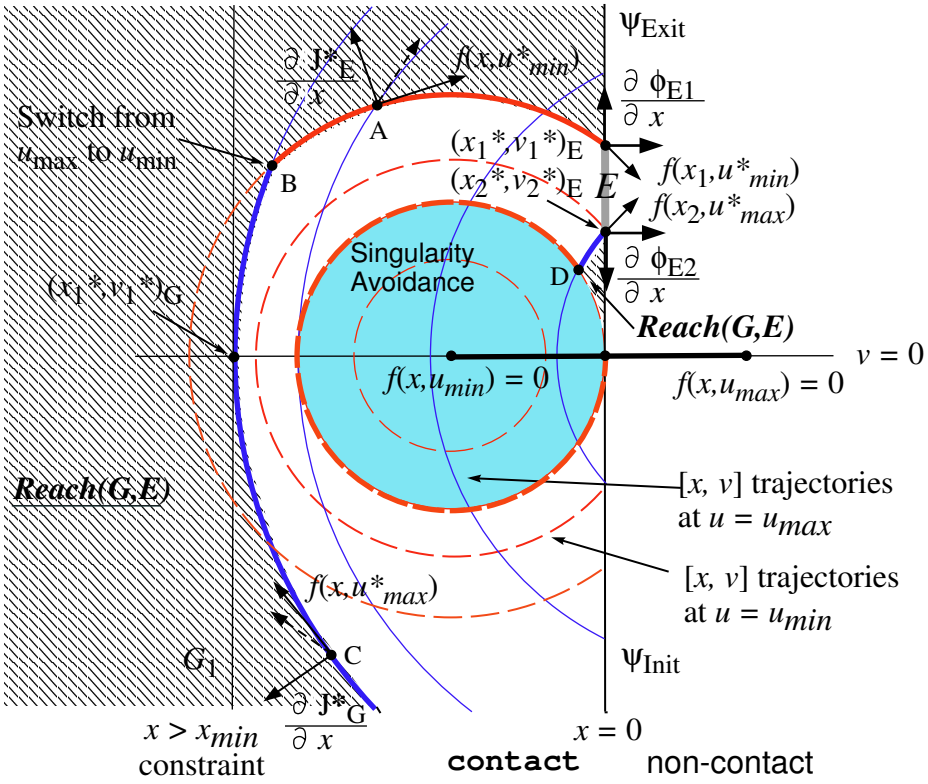


Figure 3.3: Combined unsafe and escape set optimizations, hopper `contact` mode example. The unsafe set optimization originates at the extremum labeled $(x_1^*, v_1^*)_G$ and progresses forward and backward in time with $u^* = u_{max}$. The escape set optimization optimization originates at the $(x_1^*, v_1^*)_E$ and $(x_2^*, v_2^*)_E$ extrema. Going backwards in time from $(x_1^*, v_1^*)_E$ the extremal curve follows the path through point A but intersects the J_G extremal at point B. Going backwards in time from $(x_2^*, v_2^*)_E$ the extremal curve intersects the Singularity Avoidance region (shaded) at point D and then follows the boundary marked by the heavy broken line. The intersection of the J_G and J_E optimizations is the initial safe set for the `contact` mode. The *Reach(G,E)* operation is the complement of this intersection (not including Singularity Avoidance) as indicated by the cross hatched regions.

7. Combine the extremal surfaces from the forwards and backwards propagation, $\psi_{G_i} = \psi_{G_i}^{(+)} \cup \psi_{G_i}^{(-)}$.

8. Repeat items 2-7 for each subspace, G_i, and combine the ψ_{G_i} surfaces as shown in Figure 3.1 (this operation is called a *resection* and is described in Chapter 4).

9. The escape set, E, may also be divided into compact N-1 dimensional subspaces, E_i, within the exit condition manifold, ψ_{Exit}. For each E_i, identify extrema points, ${x_{E_i}}^*$, along ∂E_i and define the outward normals $\frac{\partial \phi_{E_i}}{\partial x}$ and $\frac{\partial \psi_{Exit}}{\partial x}$.

10. Determine u^* and d^* at each extremum point ${x_{E_i}}^*$ from equation 3.21 where ν is a positive scalar.

11. Propagate backwards in time from each extremum, ${x_{E_i}}^*(t_{f_E})$, generating the N-1 dimensional extremal surface(s) ψ_{E_i} and determining u^* and d^* at each instant in time from equation 3.10, where $\frac{\partial {J_{E_i}}^*}{\partial t} = 0$ and $\frac{\partial {J_{E_i}}^*}{\partial x}$ is the outward normal of ψ_{E_i} at ${x_{E_i}}^*(t)$, The propagation continues until ${x_{E_i}}^*(t) \in \psi_{Init}$ or until reaching a Singularity Avoidance region, at which point the control is prescribed by the SA boundary.

12. Combine each extremal surface, ψ_{E_i} with the extremal surfaces ψ_{G_i} from item 8 as shown in Figure 3.3.

The result is a compact subset of X_q bound by subsets of the N-1 dimensional manifolds ψ_{Init}, ψ_{Exit}, ψ_{G_i}, and ψ_{E_i}. The complement of this subset (excluding Singularity Avoidance regions) is *Reach(G,E)*.

There are a couple of advantages to separating the J_G and J_E optimizations and calculating *Reach(G,E)* as enumerated above. One advantage, as has already been discussed, is that the Hamilton–Jacobi equations can be solved implicitly reducing computation. Another advantage is that since the unsafe states G are fixed in time, the J_G portion of the operation (steps 1-8) need only be performed once. During the backward chaining portion of the algorithm which covers transitions between modes, only the escape set E changes, so only the J_E portion of the operation (steps 9-12) is repeated.

3.3 Backward Chaining

Reach(G,E) is used to determine the reachable set of safe states within a particular mode, but does not capture the discrete evolution between modes. This evolution is defined by the reset relationship, $R(q, x, \sigma_1, \sigma_2)$, in Definition 1. The reset relationship establishes all possible transitions between the hybrid system modes and the criteria for each transition. The transition from one mode to another establishes a predecessor–successor relationship and the transition occurs at the interface between the exit and initial condition manifolds (ψ^q_{Exit} and ψ^{q+1}_{Init}). We assume that any given mode may have multiple successors and multiple predecessors.

Two examples of the reset relationship are described by phase transition diagrams in Figure 3.4. The predecessor-successor relationships are shown by arrows. For example, q_1 is a predecessor to q_2, $q_1 \succ q_2$, and q_1 is a successor to modes q_3 and q_5, $q_1 \prec (q_3, q_5)$. In both examples, the phase transition diagrams form closed loops so that the hybrid system can transition between modes indefinitely. Note that q_1 has multiple predecessors and that q_3 has multiple successors.

The algorithm for the maximal controlled safe invariant requires transitioning through the modes in backwards time. Assume that one or more compact subsets of the initial and exit condition manifolds of each mode are safe: $X^q_{Init} \subset \psi^q_{Init}$ and $X^q_{Exit} \subset \psi^q_{Exit}$. X^q_{Init} and X^q_{Exit} are determined by application of the *Reach(G,E)* operation to each mode. For the discrete evolution in forward time, the set of feasible initial conditions for a successor mode is bound by the union of the exit conditions of all the predecessor modes. Conversely, in backwards time, the set of safe exit conditions for a predecessor mode is bound by the union of the initial conditions of all the successor modes. The union of successor mode initial conditions is the set of desired escape states. However, the escape set, E_q, cannot include states that are part of the capture set of the unsafe states, G_q, as determined by the reach operation. Hence, E_q must be a subset of X^q_{Exit}:

$$E_q = X^q_{Exit} \cap (\bigcup X^s_{Init}) \tag{3.22}$$

X^s_{Init} are the successor mode initial conditions and X^q_{Exit} is the current mode's

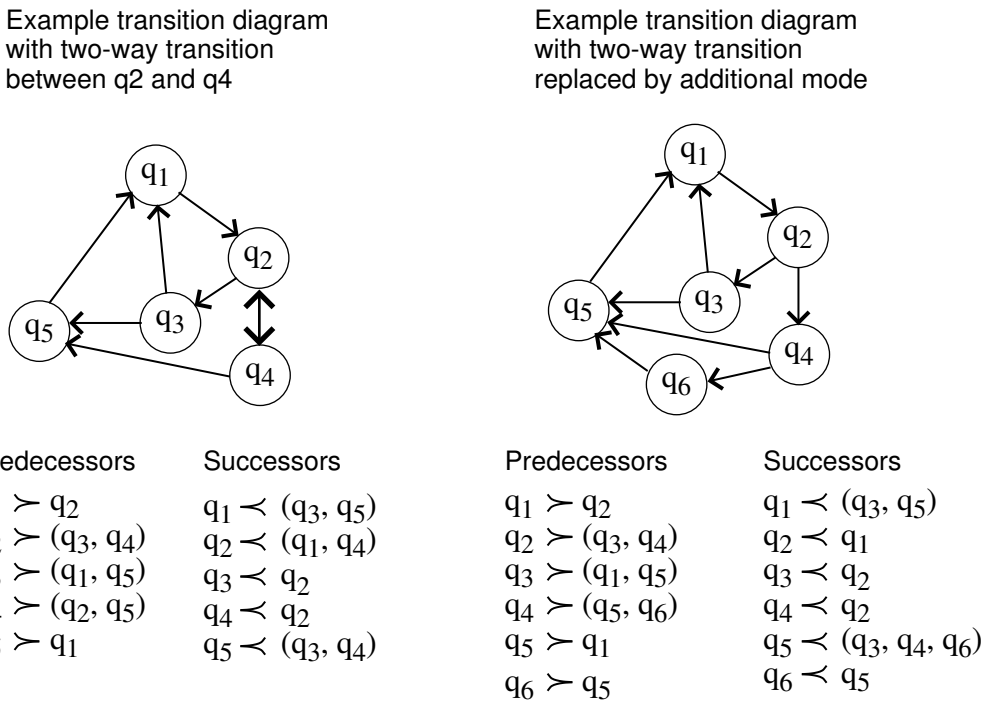


Figure 3.4: Example phase transition diagrams. The phase predecessor and successor relationships are shown. The first example includes a two-way transition between modes q_2 and q_4 which complicates the phase transition sequence. If the number of transitions between q_2 and q_4 through any one cycle is limited, a possible solution is to include additional modes as shown in the second example. Otherwise, the backward chaining sequence requires multiple iterations through q_2 and q_4 until the escape sets for these modes remain unchanged or become empty.

exit conditions. The backwards mode transition establishes a new set of escape states that replaces the exit conditions from the previous iteration:

$$X^q_{Exit} \leftarrow E_q \tag{3.23}$$

The updated escape set requires an iteration of the *Reach(G,E)* operation (though only a subset of the steps described in section 3.2.3 are required).

Returning to the example in Figure 3.4, establishing a feasible order for the backwards chaining sequence can sometimes be problematic. In the first example q_1 has two predecessors, q_3 and q_5, but q_3 is also a predecessor to q_5 so a valid sequence is $q_1 \prec q_5 \prec q_3$. However, a difficulty in this example is that q_2 and q_4 are both predecessor and successor modes to one another (indicated by the double arrow). If (in forward time) the transition back and forth between q_2 and q_4 occurs only once (e.g., $q_2 \succ q_4 \succ q_2 \succ q_3 \ \ldots$), then this difficulty can be overcome by adding an additional mode, q_6, as shown in the second example in the figure. Then a feasible sequence in backwards time becomes $q_1 \prec q_5 \prec q_6 \prec q_4 \prec q_3 \prec q_2 \prec q_1$. Alternatively, $q_1 \prec q_5 \prec q_3 \prec q_6 \prec q_4 \prec q_2 \prec q_1$ is also feasible.

On the other hand, if transitions between q_2 and q_4 may occur multiple times before transition to another mode (e.g. q_3 or q_5), then the mode q_2 to q_4 transition requires iteration. The iteration between modes, $q_4 \prec q_2 \prec q_4 \prec q_2 \ \ldots \Rightarrow (q_4 \prec q_2 \ \ldots)$, continues until the escape sets for both modes remain unchanged or become empty. Then a feasible modal transition sequence in backwards time is $q_1 \prec q_5 \prec (q_4 \prec q_2 \ldots) \prec q_3 \prec (q_2 \prec q_4 \ \ldots) \prec q_1$.

Interestingly, the backward chaining sequence through phase space is analogous to some robot motion planning algorithms in configuration space. Lazanas and Latombe (1995, [**?**]) describe a landmark-based navigation system that starts at a desired goal region and propagates backwards in time to a region of initial conditions. The backwards time propagation assumes that motion between landmarks is subject to unknown but bounded errors (disturbances). The landmarks themselves are analogous to the desired escape set, X_{Exit}, and the transition between landmarks is analogous to the transition between dynamic modes. The parallels suggest the possibility of a

unified approach to planning and control for navigating rough terrain.

3.3.1 Discrete Transitions in Reverse Time

The Reset relationship, $R(q, x, \sigma_1, \sigma_2)$, describes not only the discrete evolution between modes but also any discrete changes in the continuous state vector. These changes may be caused by discrete control or disturbance inputs or they maybe a consequence of a discrete event, such as a collision.

We restrict our attention to linear transformations so that:

$$x^s = Ax^p + B_1\sigma_1 + B_2\sigma_2 \tag{3.24}$$

Where x^s is the continuous state vector immediately after transition to the successor mode, x^p is the continuous state vector immediately prior to transition while in the predecessor mode, σ_1 is a discrete control input, and σ_2 is a discrete disturbance input. The discrete inputs are selected based on the game theoretic context of the maximal safe set problem. In forward time, if the successor mode initial conditions, X^s_{Init} are safe, then the controller will choose σ_1 to maximize the likelihood that $x^s \in X^s_{Init}$ despite the unknown disturbance σ_2. Also in forward time, the environment will select σ_2 to force $x^s \notin X^s_{Init}$ if possible.

If A in equation 3.24 is invertible, then the relationship between predecessor and successor states in reverse time is easily obtained:

$$x^p = A^{-1}(x^s - B_1\sigma_1 - B_2\sigma_2) \tag{3.25}$$

An example of the forward and reverse time transformations between modes is shown in Figure 3.5. In this example the discrete input is a two dimensional vector $\sigma = [\Delta x_1\ \Delta x_2]^T$ and values for each component can range from 0 and Δx_1^{max} and Δx_2^{max}, respectively. Figure (a) shows a mushroom shaped region of safe exit conditions for the predecessor mode. The range of possible initial conditions for the successor phase is shown in (b). Starting with the predecessor mode initial conditions (shown in dark gray), the set of possible initial conditions in the successor phase is the union of sets over the range of input values using the transformation in equation 3.24. In

reverse time, we begin with a set of safe initial conditions, X^s_{Init}, for the successor mode, (c). If σ is a disturbance input, the environment acts to minimize the set of possible predecessor states. The minimum set of predecessor phase exit condition states is the intersection of sets over the range of input values in the transformation in equation 3.25, and is indicated by the shaded region in (d). Conversely, if σ is a control input, the controller acts to maximize the set of possible predecessor states. The maximum set of predecessor mode exit condition states is the union of sets over the range of input values in equation 3.25, and is bound by the envelope of solid and broken curves in (d). Note that since the transformations in equations 3.24 and 3.25 are linear, superposition applies and the control and disturbance inputs can be considered separately.

In some problems of interest, the transformation matrix A in equation 3.24 is not invertible. For example, in an inelastic collision the relative velocity between two particles is set to zero and therefore A is not full rank. The transformation in reverse time is not defined, but it is possible to treat this problem by allowing all feasible velocities between particles in the predecessor mode (in other words, since the velocity prior to collision cannot be inferred from the state after collision assume all possible values). The approach is least restrictive in the sense that all feasible velocities are considered going backwards in time and the effects of any discrete control or disturbance input are inconsequential. If control or disturbance inputs act on other states, such as position states, then the transformation must be partitioned (or staged) so that a reverse time transformation for these inputs is defined.

3.4 Chapter Summary

This chapter begins by summarizing a set of restrictions on the general problem of the maximal safe invariant and controller synthesis problem presented in Chapter 2. These restrictions greatly simplify the calculation of the *Reach(G,E)* operation of the algorithm. The motivation and justification for these restrictions is demonstrated by the optimal control solution to *Reach(G,E)* first presented by Tomlin (1998, [47]). Tomlin's approach considers the optimization problem for the unsafe set, G, and the

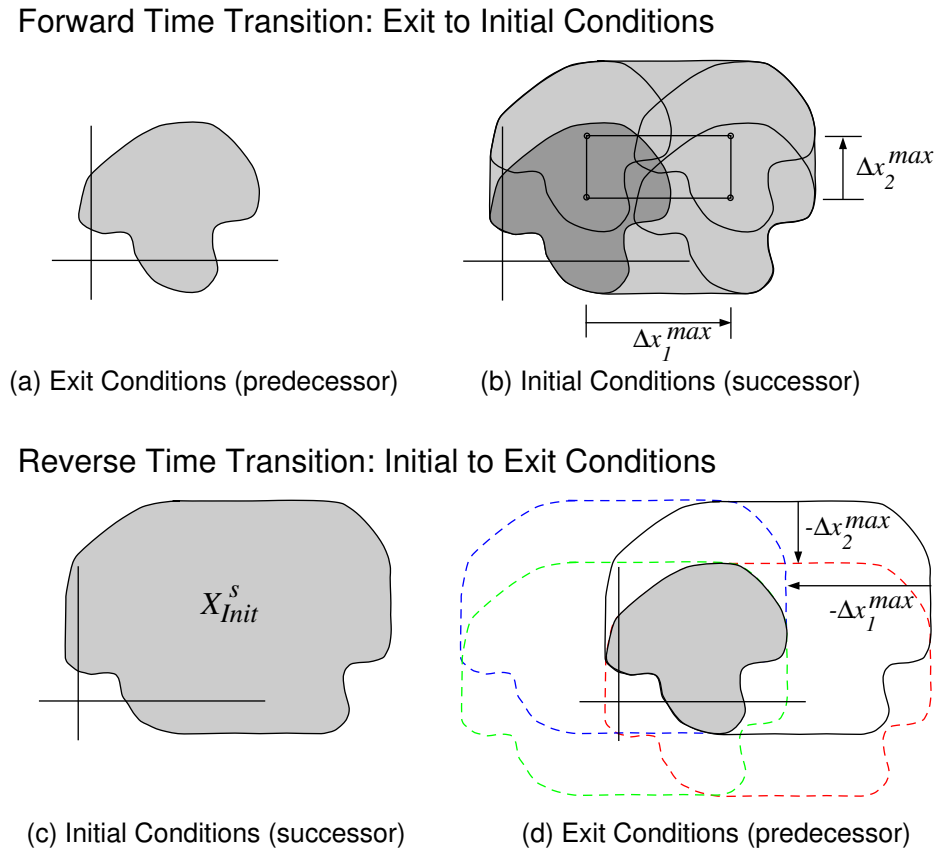


Figure 3.5: Discrete control and disturbance inputs. (a) is the set of predecessor mode exit condition states. (b) shows the possible range for the successor mode initial conditions given a discrete change in the x_1 state ranging from 0 to Δx_1^{max}, and a discrete change in the x_2 state ranging from 0 to Δx_2^{max}. In the backward chaining portion of the maximal safe invariant algorithm we proceed in reverse time from the successor mode safe initial conditions, X^s_{Init}, determined by the *Reach*(G^s, E^s) operation, (c). In reverse time the environment will act to minimize the set of possible successor mode exit condition states, and the control will act to maximize this set of states. The minimum set of states is found from the intersection of possible transformations as shown in the shaded portion of (d). The maximum set of states is the union of possible transformations which is bound by the external perimeter of solid and broken curves in (d).

escape set, E, simultaneously and solves an inter-connected pair of Hamilton–Jacobi equations which are expensive to calculate and often present numerical difficulties. The problem can be simplified by segregating the optimizations by allowing different final times, solving the optimization separately, and then combining solutions. This is possible when modal entrance and exit conditions are well defined and the safety constraints and equations of motion are not explicit functions of time. Segregating the problems allows exploitation of geometric properties of the optimizations. The optimal control and disturbance inputs, u^* and d^*, can be determined from constraint conditions at their respective final times and the geometry of the resulting integral curves. Solution to the Hamilton–Jacobi equation is thus determined implicitly. Furthermore, segregating the problem simplifies the backward chaining portion of the algorithm because the unsafe set remains unchanged and only the escape set optimization is repeated.

The chapter concludes with a description of the backward chaining sequence through the discrete modes or phases of the hybrid system. The algorithm allows modes to have multiple predecessors and successors but difficulties are encountered if multiple transitions are allowed in and out of two modes (that is the modes are both predecessor and successor to one another). Such situations require multiple iterations within the backward chaining sequence. The continuous state vector may change impulsively during discrete changes between modes. The discrete change may be influenced by discrete control and disturbance inputs, σ_1 and σ_2, respectively, and by the modal transition itself, such as the collision example. In reverse time, the control input will act to maximize the set of possible predecessor mode exit conditions and the disturbance input will act to minimize this set. In the collision example, the reverse time transformation between modes is not defined so that the set of possible predecessor mode states is all values over some predetermined range.

Chapter 4

Specification of the MISS Algorithm

A general algorithm for determining the maximal invariant safe set for a hybrid dynamic system was presented in Chapter 2 and computation of the *Reach* operator was discussed in Chapter 3. In Chapter 2, the algorithm for the maximal invariant safe set was applied without approximation in the vertical hopper example. That was possible because the system was piece-wise linear and the optimal control could be readily determined. However, practical problems generally require higher dimensionality and include significant non-linearities in the equations of motion. Therefore, a computational approach which can address these issues is desired.

This chapter describes a numerical algorithm for determining the maximum invariant safe subset. The algorithm is dubbed MISS (for maximal invariant safe subset) to distinguish it from Tomlin's more general algorithm given in section 2.5. The MISS algorithm specifies a numeric representation for a safe subset of continuous space and specifies the operations on these data objects. Algorithm execution consists of establishing the initial safe sets and then propagating backward in time through the continuous and discrete evolution of the hybrid system. In subsequent chapters, this numerical approach is applied to increasingly complex variants of the robot hopper problem but is suitable for a wide class of problems. As in all computer based

algorithms, there are trade-offs between different representations and memory, computational throughput, and accuracy. These trade-offs are discussed herein.

4.1 The Boundary Representation

Tomlin, et. al. (2000, [48]) use a grid to discretize the continuous state space. The subset of safe states is approximated by a subset of these grid points. Optimal control and disturbance inputs are calculated for each grid point using a discrete version of the Hamilton–Jacobi equations described in Chapter 3. Grid spacing must be set to provide an accurate approximation to the safe subset and provide an accurate solution for the Hamilton–Jacobi equations. This approach requires a tremendous number of points for high dimensional state spaces, driving both memory and computational requirements. Additionally, an appropriate grid spacing may not be known *a priori*, requiring some trial and error. Finally, we note from Chapter 3 that explicit solution to the Hamilton–Jacobi equations is not required for the restricted class of hybrid systems considered here. This fact opens the possibility of other representations for the safe subset of state space.

In this work, a *Boundary* is defined to be an N-1 dimensional surface surrounding a compact subset of state space. Boundaries are closed and divide the state space into two regions: those points which are either on or inside the boundary and hence a member of the subspace, and those points which are outside the boundary and not a member of the subspace. If in a given phase the safe state sets are disjoint, then a separate boundary surrounds each disjoint set.

A boundary representation of the subset of safe states has several advantages. Generally, fewer points (hence less memory) are required to describe the boundary of a volume rather than the volume itself. Secondly, points defining the boundary surface can be added or removed to provide the desired level of detail. Finally, the representation emphasizes the portion of the safe subspace of interest: that is, we are only concerned about the optimal control and disturbance inputs at the boundary of the safe subset because within the boundary any feasible control input is remains safe. A disadvantage to the boundary representation is that it can be difficult to determine

whether an arbitrary point in state space is inside or outside of the boundary surface (see discussions in sections 4.5.5 and 5.6).

Simplexes and Simplicial Complexes

The numerical representation of a boundary is based on N-1 dimensional simplicial complexes(Stillwell, 1993, [44]). Simplicial complexes were first studied by Poincaré in 1899 (see Stillwell, [44], page 3), whose many interests included dynamic systems. Simplicial complexes are an appropriate representation because they extend to N-dimensional space and are an arbitrarily accurate approximation of (and topologically equivalent to) any curved or polyhedral surface. Current applications of simplicial complexes include solid modeling (Popovic and Hoppe, 1997, [37])and medical imaging (Tohka, 2003, [45]).

A simplex is the simplest space element in N dimensional space, (Stillwell, 1993, [44]). Simplexes for 0 to three dimensional space are listed below:

- Dimension 0: point
- Dimension 1: line segment
- Dimension 2: triangle
- Dimension 3: tetrahedron

⋮

Simplex elements extend beyond 3 dimensions. An N-dimensional simplex, Δ^N, is created by forming the convex hull over a set of N+1 points which do not lie in the same N-1 dimensional hyperplane, $\{p_1, p_2, p_3, \ldots, p_N, p_{N+1}\}$. An N-dimensional simplex is built up from N-1 dimensional simplexes. Thus, a triangle is built up from line segments, a tetrahedron is built up from triangles, etc. Any subset of $m < N+1$ points within a simplex forms an $m-1$ dimensional simplex. A subset of N points of an N-simplex forms a *face* and every simplex, Δ^N, has N+1 faces. Thus a tetrahedron, Δ^3, has 4 triangular faces, Δ^2, each of which in turn have 3 line segment faces, Δ^1, etc. Each point of a simplex is a *vertex* and all vertices within a simplex are connected to one another by a line segment or *edge*.

Complex (and non-convex) geometric figures can be created by pasting together

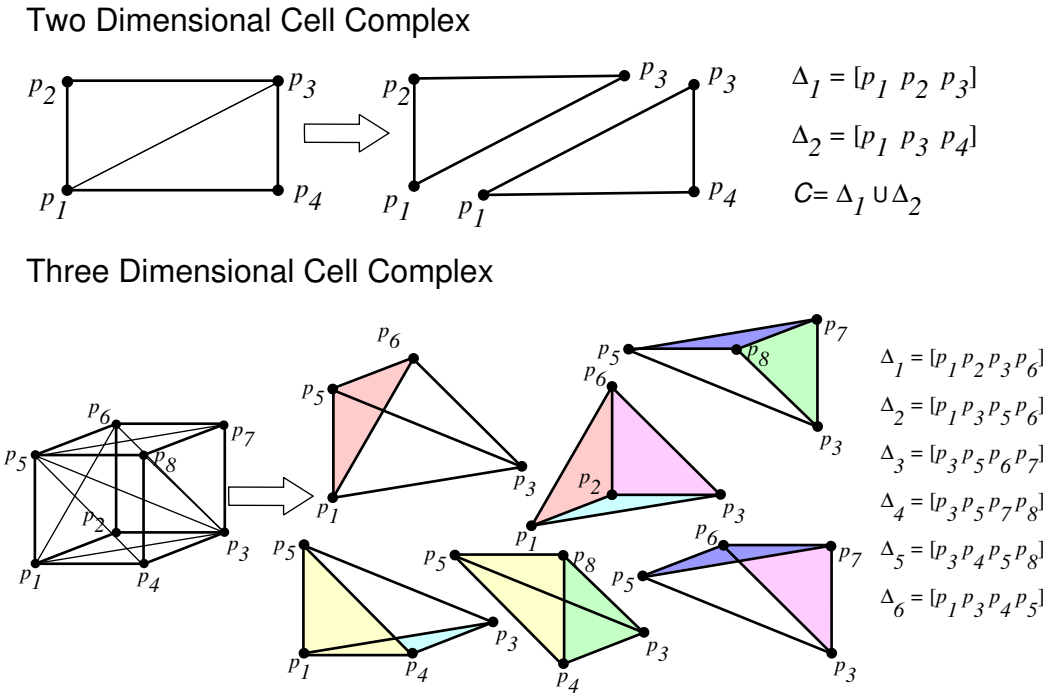


Figure 4.1: Simplex cells can be pasted together to form complex shapes. The simplex cell in two dimensions is a triangle: in three dimensions its a tetrahedron. Adjoining cells share the same face which is always an N-1 dimensional simplex. The *boundary* of a simplicial complex is composed of unmatched face cells. Boundary faces for the cubic cell complex are shaded to aid visualization.

simplexes so that the faces of each simplex either coincide or are disjoint. For example, a square can be created by pasting together two Δ^2 triangle simplexes. A cube can be created by pasting together six Δ^3 simplexes. Such figures are called *simplicial complexes* (Stillwell, 1993, [44]). As a counter example, pasting together two overlapping triangles (see Figure 4.12) does not constitute a simplicial complex because the cell faces intersect and do not coincide. The dimension of the simplicial complex is the largest dimension of any simplex within the complex. Examples in two and three dimensions are shown in Figure 4.1.

The *boundary* of an N-dimensional simplicial complex in $\mathbb{R}^N$ is the union of all

unmatched N-1 dimensional faces. The boundary is itself an N-1 dimensional simplicial complex. For example, in $\mathbb{R}^2$ a square is an N-dimensional simplicial complex and the boundary of the square is the union of the Δ^1 line segments along its perimeter. The diagonal across the square in Figure 4.1 is common to both Δ^2 simplexes and therefore not part of the boundary. In $\mathbb{R}^3$, the boundary of a cube is the union of triangular faces on the cube surface. The unmatched faces for the simplex cells comprising the cube in Figure 4.1 are shaded to aid visualization.

We refer to the individual N-1 dimensional simplexes that comprise the boundary as the boundary cells. As noted above, boundary cells can be decomposed into cell faces, and further decomposed to edges and points. The boundary is closed if all of the N-2 dimensional cell faces are shared by adjacent boundary cells. In Figure 4.1, the boundary for the square is a set of N-1 dimensional line segments and the N-2 dimensional faces of the line segment boundary cells are the 4 vertex points. Each vertex point is shared by 2 boundary cells and the boundary is closed in two dimensions. For the cube, the N-2 dimensional faces of the triangular boundary cells are the line segments between points on the same cube face. Each of these line segments is shared between 2 triangular cells on the cube surface and the boundary is closed.

A *subboundary* is a subset of the boundary cell complex. Subboundaries are N-1 dimensional cell complexes but are not closed. For example, a face of the cube in Figure 4.1 is a subset of the cube surface and constitutes a subboundary. A subboundary does not divide $\mathbb{R}^N$ into disjoint sets. However, subboundaries have a *border*. The border of a subboundary is an N-2 dimensional cell complex. The border is closed if each of the faces of the border face cells (N-3 dimensional simplexes) is shared by another border face cell. Returning to the cube example, the border of the cube face subboundary is the perimeter of the cube face and the border face cells are the line segments between verticies. The subboundary is closed because the 'faces of the border faces', that is, the vertex points, are shared between border faces. A valid subboundary has one and only one border. Thus, cube faces from opposite sides do not constitute a subboundary.

Boundaries, subboundaries, and subboundary borders are easily visualized in three

dimensions, but the concepts apply to higher dimensional spaces.

4.1.1 Boundary and Subboundary Objects

The geometry of any N-1 dimensional surface can be approximated by a cell complex, and, in particular, a boundary is a closed cell complex. However, in addition to geometry other information is needed to determine the maximal controlled invariant. To generate an extremal surface as described in Chapter 3, the control and disturbance input at each point must be determined. To determine the optimal control and disturbance input, a vector defining the inside direction from the boundary surface is used. A *boundary object* is a data structure for maintaining this information.

Additionally, it is useful identify certain subboundaries within the closed boundary. For the purposes of the MISS algorithm, every boundary has an initial conditions subboundary and an exit conditions subboundary. These subboundaries are used as part of the backward chaining portion of the algorithm.

Therefore, a data structure describing a boundary object consists of the following:

1. An ordered list of points in N dimensional space,

2. For each point, an N-dimensional vector specifying the inside directions at that point,

3. For each point, the m-dimensional continuous control and disturbance inputs required to remain on the extremal surface at that point,

4. A list of N-1 dimensional boundary cells, each cell consisting of a set of N points. For a closed boundary each face of each cell (a set of N-1 out of N points) will be contained in one and only one other cell (i.e. a closed boundary).

5. A list of N-1 dimensional Initial Condition cells which is a subset of the list of boundary cells. The Initial Condition cells comprise a subboundary.

6. A list of N-1 dimensional Exit Condition cells which is a subset of the list of boundary cells. The Exit Condition cells comprise a subboundary.

7. A list of N-1 dimensional unmatched cell faces constituting one or more boundary borders. For a closed boundary this list is empty, but is non-empty for a subboundary.

A *subboundary object* is similar to a boundary object except that the list of N-1 dimensional unmatched cell faces will not be empty and there are no identified Initial Condition or Exit Condition cells.

A boundary object can be created from two or more subboundaries using the *Merge* operation, $\oplus$, as illustrated in Figure 4.2. In this example the cells from subboundary B that are added to subboundary A form the Initial Condition subboundary, and cells from subboundary C that are added to $A \oplus B$ form the Exit Condition subboundary. Note that the inside directions of points in A used to determine which portions of B and C are included in the merge. Another operation involving boundaries and subboundaries is the *Resect* operation, $\oslash$, also illustrated

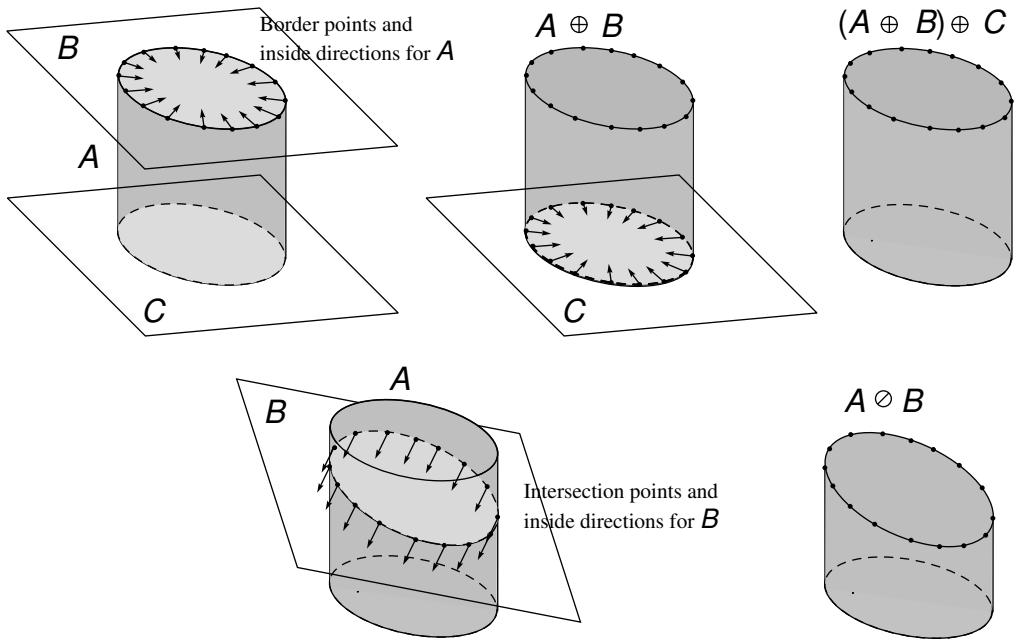


Figure 4.2: Closed boundaries can be created by merging subboundaries together using the *Merge* operation, $\oplus$. In this example subboundaries B and C represent initial and exit condition manifolds and subboundary A is the time propagation from a set of initial states to exit conditions. Inside directions for points in A are required to determine which portion of B and C belong to the merged boundary. A subboundary can also divide closed boundaries to form new ones. In the example below subboundary B resects subboundary A, $A \oslash B$. The inside direction of the points in B are used to determine which section of A is retained to form the closed set.

in the figure. Boundary operations are described in more detail in section 4.5.

The accuracy of the boundary representation is a function of cell size which is related to the distance between boundary points. Cell size is controlled by several parameters. One parameter sets the maximum cell diameter or distance between cell points. Limiting the maximum cell diameter increases accuracy and improves efficiency when searching for points in nearby cells. Another parameter is used to set the maximum error in the straight line approximation between cell points. This tolerance is applied only along integral curves where derivative information is available. There is no tolerance to control the spacing between integral curves and that has lead

to numerical difficulties in some cases.

Finally, since numerical computations are imperfect, separate calculations to determine the coordinates of the same point may yield slightly different answers. Therefore, a tolerance is applied to these calculations. If the distance between points is within this very small tolerance, the points are considered to be the same.

4.2 MISS Algorithm Inputs

Inputs for the Maximal Invariant Safe Subset (MISS) algorithm are a specification of the continuous state space, specification of the hybrid system given in Definiton 1 with additional constraints enumerated in section 3.1, and a set of termination conditions for the algorithm. The termination criterion includes a predetermined number of iterations through the backward chaining sequence and a tolerance on changes to phase exit condition subboundaries between iterations. If the change between iterations is small enough, the algorithm terminates.

Specification of the continuous state space includes the dimension, N, a description of any wrapped states (such as angles or clock states), a distance metric (typically a weighted 2-norm), maximum cell diameter, and tolerances establishing the accuracy of the boundary representation and minimum distance between distinct points.

The hybrid system is specified by defining each of the hybrid phases, Φ_q. Definition of a hybrid phase includes specification of: the equations motion, allowable control and disturbance inputs, initial and exit condition criteria, internal constraints, and predecessor and successor relationships. To generate the initial phase boundaries, the MISS algorithm uses initial and exit condition subboundaries which are finite but cover the expected range of safe states. Any singularity avoidance constraints must be identified for each mode. Safety constraints are enforced by forward and backward time propagation from extrema points indentified within the internal constraint subboundary.

Certain modes may only capture discrete events like collisions or the `step` disturbance input, as an example. Such modes are said to be 'discrete'. For discrete modes, the relationship between input and output states is algebraic and the full

boundary representation described in the preceding section is not required. Rather, the mapping from output to input states is specified by mode specific discrete actions.

4.3 Boundary Initialization

Each continuous mode or phase requires an N-1 dimensional closed boundary to represent the initial set of safe states. The initial boundary is created by time propagation between the initial and exit condition subboundaries for that phase.

The propagated subboundary is generated by forward or backwards time propagation. The choice of direction is driven by the relative complexity of the initial or exit condition subboundaries. The exit and initial condition subboundaries must be sized appropriately to accommodate the propagation subboundary for the merge operation to work properly.

Time propagation begins from the border of the initial or exit condition subboundary. The border points, inside directions, and edge connections between points are maintained in a structure called a *generator*. The time propagation is executed by the `CellEngine` function, described more fully in section 4.5.2. `CellEngine` determines optimal control and disturbance inputs and integrates the equations of motion. `CellEngine` determines when the integration should generate a new subboundary point based upon tolerances set as part of the state space specification. The new point is used to create new subboundary cells with the `BuildUpCells` function, described in section 4.5.3. The integration and cell build up process continue until all trajectories meet the termination condition criteria and the time propagation is complete.

The initial condition, exit condition, and propagated subboundaries must be merged together to create a single closed boundary. As illustrated at the top of Figure 4.2, the `Merge` operation assumes that the border points of one subboundary are contained within the manifold of the other subboundary. Care must be taken that the exit condition subboundary is large enough to 'cover' the border of the propagated subboundary at the exit condition manifold. Otherwise, the resulting border will be open, like a can with an open lid. The `Merge` operation is described in more

detail in section 4.5.8.

4.3.1 Safety Constraints and Singularity Avoidance

The initial phase boundary, generated by propagating forward or backward in time from a set of initial or exit conditions, may contain unsafe regions of the state space. These regions are characterized by a set of internal constraints for each phase. An example is the leg compression constraint for the single legged hopper in Chapter 2.

The internal constraints can be characterized by an N-1 dimensional subboundary. However, we are typically only interested in the subset of the constraint subboundary containing the extrema points. The extrema points define one or more N-2 dimensional borders within the constraint subboundary, and for N greater than 2 these borders are closed. Forward and backward time propagation from the extrema points generates N-1 dimensional subboundaries which approximate the extremal surfaces, ψ_G^+ and ψ_G^-, described in section 3.2.1. The numerical integration begins with a generator that includes the extrema points, inside directions, and edge connections between points. If the extremal surface is unconstrained (as for the ψ_{G1}^+ and ψ_{G1}^- and ψ_{G2}^- curves in Figure 3.1), the integration and subboundary generation and is executed by the `CellEngine` function to be described in section 4.5.2. If, on the other hand, the integration is constrained by the constraint surface (as for the ψ_{G2}^+ curve in Figure 3.1), then the safety constraint is itself the extremal curve and numerical integration is not necessary. However, the optimal control and disturbance inputs over the extremal portion of the constraint surface must be identified.

The forward and backward time extremal subboundaries are spliced together along the border defined by the extrema points using a version of the `Merge` operation. The combined subboundary spans from the initial to the exit condition manifolds. The combined subboundary intersects the initial phase boundary along a closed border. The unsafe states and those 'outside' of the extremal surface are removed from the initial phase boundary using the `Resect` operation shown in Figure 4.2. If the initial phase boundary includes multiple safety constraints, the subboundary generation and `Resect` operation are applied multiple times.

If the initial phase boundary includes singularities, then a subboundary delineating the singularity avoidance zone can be generated in a manner similar to the extremal surface subboundaries for the safety constraints. The singularity avoidance constraint is one or more integral surfaces that surround singular points (that is, where $f_q(x, u, d) = 0$) over the possible range for u and d. The integral surfaces are N-1 dimensional subboundaries that connect from the initial conditions manifold, ψ_{Init}, to the exit conditions manifold, ψ_{Exit}. In the sense of a'least restrictive control', the singularity avoidance should not constrain the desired escape set, E, or the safe range of initial conditions, X_{Init}. The construction of a singularity avoidance subboundary is thus very problem specific. However, once the subboundary has been generated, it can be incorporated into the initial phase boundary using the `Resect` operation.

4.4 Backward Chaining Between Boundaries

The initialization and internal constraint portions of the MISS algorithm result in a set of boundaries enveloping the initial set of safe states for each mode and removing from each all states that could be forced into an unsafe region prior to transition to another mode, *Reach(G,E)*. The backward chaining portion of the algorithm considers the transitions between modes and removes states that could eventually result in unsafe transitions. In the following discussion we assume a single boundary for each phase, but in fact each phase may have multiple distinct boundaries.

The backward chaining algorithm assumes a predecessor–successor relationship between modes and iterates through any loops. Because each mode may have multiple predecessors or successors an order to the mode propagation is specified so that all modes are considered through each iteration of the sequence. However, as discussed in section 3.3, the sequence may require internal iterations for tightly coupled modes that are both predecessor and successor to one another.

Starting with the first mode, Φ_q, we determine the successor modes and extract the initial condition subboundary, X^s_{Init}, from each successor. Note that the successor mode may have more than one initial condition subboundary. The exit conditions for

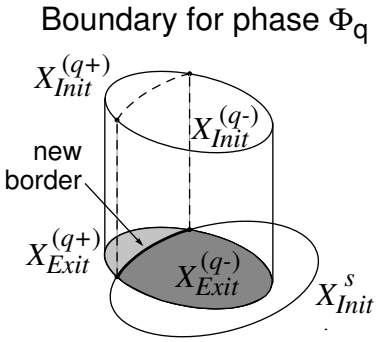


Figure 4.3: Backward Chaining operations on phase Φ_q. The $X_{Exit}^{(q+)}$ subboundary (light shaded) intersected with successor mode initial conditions, X_{Init}^{s}, forms a new set of exit conditions, $X_{Exit}^{(q-)}$. The new border for $X_{Exit}^{(q-)}$ forms a generator for backwards time propagation within the phase. The propagated subboundary, shown by broken lines, is resected from the boundary, affecting the set of initial conditions, $X_{Init}^{(q-)}$. The change to initial conditions for Φ_q ripples to other predecessor modes, Φ_p.

the current mode, X_{Exit}^{q}, must transition to safe initial conditions for the successor modes. Since we may have multiple successors, the backwards time transition is:

$$X_{Exit}^{(q-)} = \bigcup_{s} \left(X_{Init}^{s} \cap X_{Exit}^{q(+)} \right) \tag{4.1}$$

Where the superscripts $(q-)$ and $(q+)$ indicate a backward time transition. Clearly, $X_{Exit}^{(q-)} \subseteq X_{Exit}^{(q+)}$. The relationship between these subboundaries and the X_{Init}^{s} subboundary is illustrated in Figure 4.3. The numerical algorithm for determining the intersection between subboundaries is the `IntersectSubboundaries` operation described in section 4.5.7. Likewise, the numerical algorithm for the union of two subboundaries is `UnionSubboundaries`. Note that a union of multiple subboundaries can be non-manifold so that a condition of a legitimate simplicial complex is violated. This problem is avoided by performing the intersection prior to the union operation as implied in equation 4.1.

The intersection between between exit and initial condition subboundaries, $X_{Exit}^{(q-)}$, is one or more subboundaries that define the desired escape set, E. If Φ_q is a continuous mode, we repeat the *Reach(G,E)* operation for the updated escape set. The

border of the $X_{Exit}^{(q-)}$ defines a generator from which the `CellEngine` and `BuildUpCells` functions can propagate an N-1 dimensional subboundary in backwards time. Much of $X_{Exit}^{(q-)}$ will coincide with the exit conditions from the previous iteration $X_{Exit}^{(q+)}$. Therefore, the time propagation via the `CellEngine` and `BuildUpCells` functions need only be applied to that portion of the $X_{Exit}^{(q-)}$ border that does not coincide with the $X_{Exit}^{(q+)}$ border. Once, the propagated subboundary has been generated, the `Resect` operation removes the *Reach(G,E)* subspace from the boundary. Often this will result in a change to the initial conditions so that, $X_{Init}^{(q-)} \subseteq X_{Init}^{(q+)}$ which propagates backwards in time to other predecessor modes. If a phase contains more than one boundary, the intersection, backwards time propagation, and resection is repeated for each boundary.

If the phase Φ_q is discrete rather than continuous, then the relationship between initial and exit conditions is the result of a discrete action rather than the *Reach(G,E)* operation. Discrete transitions were discussed in section 3.3.1 and the discussion will not be repeated here.

The backward chaining sequence continues through all discrete and continuous phases either for a fixed number of iterations, or until the change in phase boundaries between iterations is small, or until a null solution is encountered.

4.5 Boundary Operations

This section describes the boundary operations introduced previously in more detail. First we discuss the boundary simplex and associated systems of equations. Then we describe the `BuildUpCells` and `CellEngine` functions. The `UnionSubboundaries` and `IntersectSubboundaries` functions are described along with `SubdivideCells` which creates new cells when one or more existing cells are divided at a point. The `InsideSimplex` and `OnBoundary` operations determine whether an arbitrary point is inside an N-1 dimensional simplex cell or on the surface of an N-1 dimensional cell complex. `InsideBoundary` determines whether a point is inside a closed boundary and hence whether the point belongs to the set defined by the boundary. Finally, we describe the `Merge` and `Resect` operations.

4.5.1 The Boundary Simplex and Systems of Equations

The simplex cell is the fundamental unit of the Boundary representation. An N-1 dimensional simplex, Δ^{N-1}, requires N points: $p_1, p_2, p_3, \ldots, p_N$. Boundary cells are limited in size so that the maximum distance between any vertex points is less than or equal a maximum cell diameter, d_{max}. Thus within a cell $|p_i - p_j| \leq d_{max}$ for $i \neq j$.

Intersections Between Cells and Line Segments

Many of the boundary functions search to determine whether a line segment intersects a boundary cell or a cell face. A line can be represented from the vector difference between two points, p_A and p_B:

$$p_0 = p_A + \lambda(p_B - p_A) \tag{4.2}$$

Where p_0 is any point on the line passing through p_A and p_B, and λ is a scalar. The point p_0 is between points p_A and p_B if λ is between 0 and 1.

A point, p_0 is inside the simplex cell if the vector equation 4.3 below holds. The points are N-dimensional vectors starting at the origin. Equation 4.3 is an N-dimensional vector equation with N-1 coefficients $\alpha_1,\ \alpha_2,\ \alpha_3, \ldots,\ \alpha_{N-1}$. Thus, p_0 is not arbitrary and must be a linear combination of the vectors $p_1,\ p_2,\ p_3, \ldots,\ p_N$.

$$\begin{aligned} p_0 \quad = \quad & p_1 + \alpha_1(p_2 - p_1) + \alpha_2(p_3 - p_2) \\ & + \alpha_3(p_4 - p_3) + \ldots + \alpha_{N-1}(p_N - p_{N-1}) \end{aligned} \tag{4.3}$$

and

$$0 \leq \alpha_1 \leq 1$$

and

$$\alpha_1 \geq \alpha_2 \geq \alpha_3 \ldots \geq \alpha_{N-1}$$

and

$$\alpha_i \geq 0 \ \forall i \in \{1, \ldots, N-1\}$$

In N-dimensional space, a line intersects an N-1 dimensional cell at a point, unless the line and cell share the same subspace, which is a degenerate condition. Equation 4.4 below combines equations 4.3 and 4.2 to a system of N equations with N unknowns $\alpha_1, \alpha_2, \alpha_3, \ldots, \alpha_{N-1}$, and λ. If the system of equations is full rank, the coefficient vector, x, can be solved for, $x = A^{-1}v$. The point of intersection, p_0, can then be determined from equation 4.2. p_0 is inside the cell simplex if the coefficients α_i satisfy the constraints listed under equation 4.3.

$$v = Ax \tag{4.4}$$

where from 4.3 and 4.2

$$\begin{aligned} v &= p_A - p_1 \\ A &= [\ p_A - p_B \ \ | \, p_2 - p_1 \ \ | \ \ldots \ \ | \, p_N - p_{N-1}\ ] \\ x &= [\ \lambda \ \ \alpha_1 \ \ \alpha_2 \ \ \ldots \ \ \alpha_{N-1}\ ]^T \end{aligned}$$

When the line through points p_A and p_B and the simplex cell share the same subspace, the intersection between the line and a cell face (an N-2 dimensional cell) may be desired. Such situations occur during intersection or union operations on subboundaries within the same N-1 dimensional manifold. The system of equations follows the same form as given in 4.4, but the matrix A is N by N-1. The solution for v is under determined or, conversely, x is over constrained (more equations than unknowns). A best fit for x can be found with least squares using the Moore-Penrose pseudo-inverse of A. The interpretation is that the intersection between the line segment and cell face is approximate. For example, in $\mathbb{R}^3$ a curved surface is approximated by a 2-dimensional cell complex. The cell faces are the line segments between cell vertices. Suppose p_A and p_B are on the curved surface but in different cells. A line segment from p_A to p_B will cross over a cell face, but won't exactly intersect it due to the approximation of the curved surface. However, if the minimum distance between a line p_A and p_B and a point on the cell face (determined by the Moore-Penrose pseudo-inverse) is within a tolerance, then the line segments are deemed to intersect. An additional check is made to identify whether the best fit point is near

function `IntersectSubCell`(p_A, p_B, $p_1, p_2, \ldots, p_{N-1}$)
returns Test = True/False, p_{int}
inputs: p_A, p_B, line segment end points, $p_1, p_2, \ldots, p_{N-1}$,
array of N-1 dimensional simplex points
initialization: $v = p_A - p_1$, $A = [p_A - p_B | p_2 - p_1| \ldots |p_{N-2} - p_{N-1}]$,
normal tolerance, δ, and tighter tolerance, ϵ.

1. Find least squares solution for x: $x = (A^T A)^{-1} A^T v$.
where $x \triangleq [\lambda \ \alpha_1 \ \ldots \ \alpha_{N-2}]^T$.
2. Determine if $p_q = p_A + \lambda(p_B - p_A)$ is within the line segment from p_A to p_B and if $p_r = p_1 + \alpha_1(p_2 - p_1) + \ldots \alpha_{N-1}(p_{N-1} - p_{N-2})$ is inside the simplex. If either test fails then **return** Test = False, $p_{int} = 0$.
3. If p_q is at p_A or p_B ($\lambda = 0$ or $\lambda = 1$) or if p_r is at a vertex ($\alpha_i = 1$ or $\alpha_i = 0$) then set the intersect radius to a small value $\rho = \epsilon$. Otherwise set the intersect tolerance to the normal value, $\rho = \delta$.
4. Solve for the vector difference $p_q - p_r = v - Ax$ and get the distance between points: $d = \| v - Ax \|$.
5. **if**($d \leq \rho$) **return** Test = True, $p_{int} = p_r$, **else return** Test = False, $p_{int} = 0$.

Figure 4.4: `IntersectSubCell` Function

a cell vertex but does not intersect the cell face. The algorithm for determining the intersection between a line and a cell face, dubbed `IntersectSubCell`, is summarized Figure 4.4.

The Boundary Simplex Normal Vector

A boundary simplex, Δ^{N-1}, has a normal direction which is the direction in N-dimensional space orthogonal to the hyperplane containing the simplex. In three dimensions the normal direction is along the cross product of the vectors $(p_2 - p_1)$ and $(p_3 - p_1)$. The simplex normal may be along either the positive or negative direction of the vector cross product. The inside directions associated with each boundary point are used to determine whether the normal vector is pointing outward

or inward from the boundary surface. For higher dimensions the simplex normal is determined from a generalized version of the cross product. Form an NxN-1 matrix, M, from the difference between simplex points:

$$M = [(p_2 - p_1)|(p_3 - p_1)|(p_4 - p_1)|\ldots|(p_N - p_1)] \tag{4.5}$$

The normal direction, ν, is orthogonal to each of the component vectors in M and can be determined from cofactor expansion:

for $k = 1, \ldots, N$
$I = \{1, \ldots, N\} \setminus k$
$J = \{1, \ldots, N\}$
$\nu(k) = -1^{k-1}\texttt{det}(M(I, J))$
end

Where $\texttt{det}(M(I, J))$ is the determinant of matrix M with the kth row removed. If Δ^{N-1} is non-degenerate (spans an N-1 dimensional hyperplane), then M is rank N-1, and the normal direction, ν, is non-zero, and ν can be normalized to unit length $\nu = \nu / \parallel \nu \parallel$. ν points inward if the dot product between and the inside direction of any of the points in the simplex, $\{p_1, p_2, \ldots, p_N\}$ is positive. Otherwise, ν points outward. The inward or outward sense of ν can be reversed by multiplying by -1.

4.5.2 `CellEngine`

`CellEngine` creates an N-1 dimensional subboundary by forward or backward time propagation from a set of initial conditions. The initial conditions include a set of points, a set of vectors associated with each point, and a relationship describing the edge connections between points. The initial conditions are provided in a data structure called a *generator* earlier.

Consider time propagation from a single point. Figure 4.5 shows both continuous propagation (broken lines) and a boundary approximation. Starting from the left most point, x_0, the first problem is to determine the optimal control and disturbance inputs so that the state derivative can be integrated. From our discussion in Chapter

3, the optimal inputs are determined from local geometry. The initial inside direction, ν_0, is either the outward normal to a safety constraint boundary, $\frac{\partial \phi_G}{\partial x}$ or the inward normal to the escape set boundary, $-\frac{\partial \phi_E}{\partial x}$. In backwards time, the optimal control, u^*, maximizes the dot product between ν_0 and $f(x_0, u, d)$; in forwards time, the optimal control minimizes the dot product, as shown in Figure 4.5. The optimal disturbance, d^*, has the opposite action but can be considered separately because $f(x, u, d)$ is separable in u and d. If $f(x, u, d)$ is linear in u and d, the search for u^* and d^* is simplified since $u \in \{u_{min}, u_{max}\}$ and $d \in \{d_{min}, d_{max}\}$.

Having determined the initial optimal control and disturbances, the integration is performed numerically with a small enough time step to ensure desired accuracy. Optimal values for u and d must be established at each time step and the integration follows the curve in Figure 4.5 labeled 'optimal propagation'. `CellEngine` uses the same normal vector over the propagation between boundary points to determine u^* and d^* (for example, ν_0 is used from points x_0 to x_1) . The approximation is acceptable over the tolerance between boundary points. Note that the inside direction at each boundary point is determined from the cell normal and not the tangent at the curve. This is because for a state space of dimension 3 or higher, an N-1 dimensional surface is required to establish a normal direction (see section 4.5.1.

`CellEngine` establishes boundary points, x_1, x_{k-1}, x_k, etc., at various points in the integration. The establishment of a boundary point is controlled by several criteria. One criterion is the distance between points which must be less than or equal to the maximum cell diameter, d_{max}, shown in Figure 4.5. Another criterion is the maximum error of the boundary approximation, h_{max}, shown between points x_{k-1} and x_k. The maximum error is estimated from the intersection of tangents between successive boundary points and the distance to the edge connecting these points. Other criteria for establishing a boundary point include a large enough change in the tangent direction between successive points (not shown) or a large enough change in the optimal control or disturbance inputs (also not shown). The distances and error between points in the figure are exaggerated for illustration.

For a state space dimension of 3 or higher the individual trajectories or integral

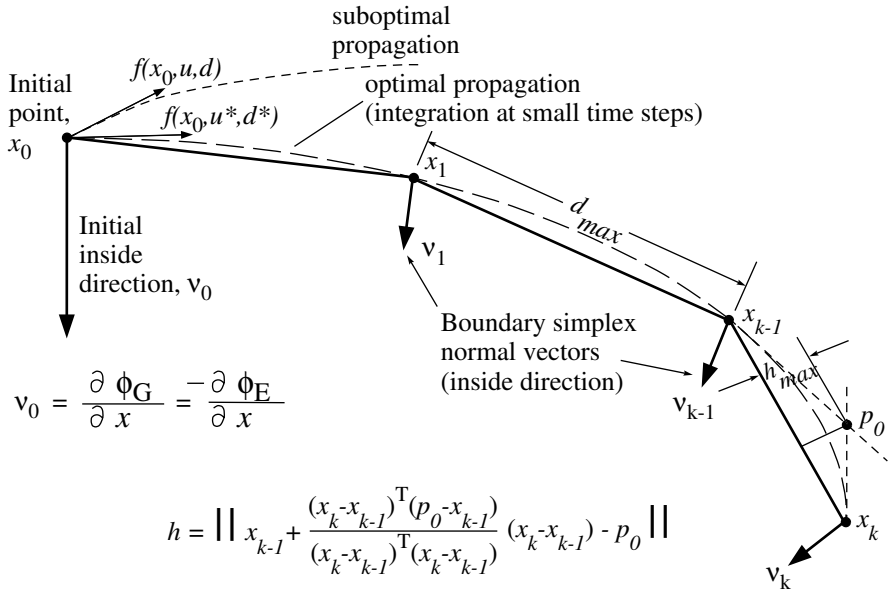


Figure 4.5: Boundary approximation of the continuous propagation for `CellEngine`. The propagation from x_0 starts with determination of the optimal control and disturbance inputs. Optimal control and disturbance inputs are determined at each time step but boundary points are established only when 1) the distance between successive points reaches a limit, d_{max}, 2) the approximation error, h, reaches h_{max}, 3) tangent vectors between successive points change direction in excess of a limit (not shown), or 4) the optimal control or disturbance input value changes significantly (not shown).

curves originating from each initial point must be connected together to from N-1 dimensional cells. The build up process is illustrated in Figure 4.6. The initial conditions are given in a data structure dubbed a 'generator' that includes the initial points, vectors defining inside directions for each point, and the edge connections between points. The time propagation from each initial point will result in a set of new boundary points when one of the conditions described in Figure 4.5 is triggered. Edge connections between the newly propagated and neighboring points are determined from the edge connections in the generator. After all points in the cell generator have been propagated to a new boundary point, the `BuildUpCells` function creates an N-1 dimensional cell complex from the edge connections in the generator and the new points. If an edge between the new points exceeds the maximum cell diameter a new point is added half way between and the cell is subdivided (see subfigure (f) in Figure 4.6). Finally, the border cells of newly propagated points become the cell generator for the next evolution of the boundary. The process repeats until all propagated points meet a termination condition.

4.5.3 `BuildUpCells`

The `CellEngine` function uses the `BuildUpCells` function to create an N-1 dimensional cell complex from the edge connections between points. The edge connections are 1-dimensional cells that become the faces of a 2-dimensional cell complex. To build up the 2-dimensional cells, we find a group of three 1-dimensional cells which share the same 3 points. For example, Figure 4.1 shows a rectangle composed of two 2-dimensional cells. The connections between points are $\{[p_1\ p_2]$, $[p_1\ p_3]$, $[p_1\ p_4]$, $[p_2\ p_3]$, $[p_3\ p_4]\}$. The sets $\{[p_1\ p_2]$, $[p_1\ p_3]$, $[p_2\ p_3]\}$ and $\{[p_1\ p_3]$, $[p_1\ p_4]$, $[p_3\ p_4[\}$ both repeat the same sets of points twice, hence $[p_1\ p_2\ p_3]$ and $[p_1\ p_3\ p_4]$ are 2-dimensional cells.

The process described above generalizes to higher dimensions. Every m-dimensional simplex contains m+1 unique points. Any subset of m out of m+1 cell points comprises a cell face, (Stillwell, 1993, [44]). Thus, a collection of m+1 cell faces contains only m+1 unique points and each point occurs m times in the collection. This process

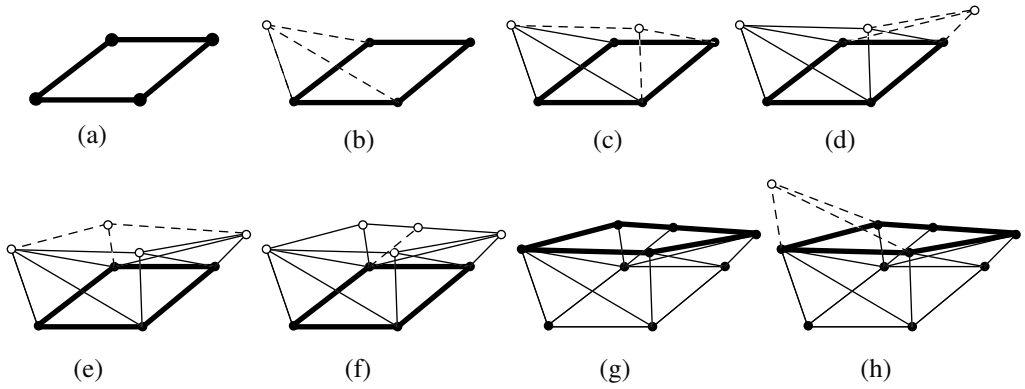


Figure 4.6: `CellEngine` generates cell complexes by time propagation over a set of initial conditions: (a) initial generator including points and edge connections between points, (b) time propagation from the first generator point with edge connections to generator points, (c) time propagation from second generator point with edge connection to propagated and unpropagated generator points, (d)-(e) time propagation of remaining generator points, (f) cell division for edges that exceed maximum cell diameter, (g) `BuildUpCells` function creates N-1 dimensional complex, generator layer transfers to the border containing newly propagated points, (h) time propagation from first point in the new layer.

is recursive. To generate an m+1-dimensional cell from a set of m dimensional cells, $\mathcal{C}^m$, it is sufficient to find a set of m+1 cell faces which use a combination of the same m+1 points. The process of decomposing and building up a cell from a collection of points is illustrated in Figure 4.7.

A problem with the build up process is that the search over combinations of cell faces to find a matching set of m+1 faces is combinatoric. For example over a collection of 10 2-dimensional cell faces (3 points each), the algorithm would have to consider up to $Comb(4, 10) = 210$ possible combinations of 4 out of 10 faces to find a matching set. For a collection of 11 2-dimensional cell faces, the algorithm would have to consider $Comb(4, 11) = 330$ possible combinations. Clearly, the search size is dramatically reduced by reducing the number of candidate cells. This can be facilitated by ordering the list of points in each cell so that a subset of cells over which to perform the search can be easily identified.

The `BuildUpCells` function recursively builds a collection of N-1 dimensional cells

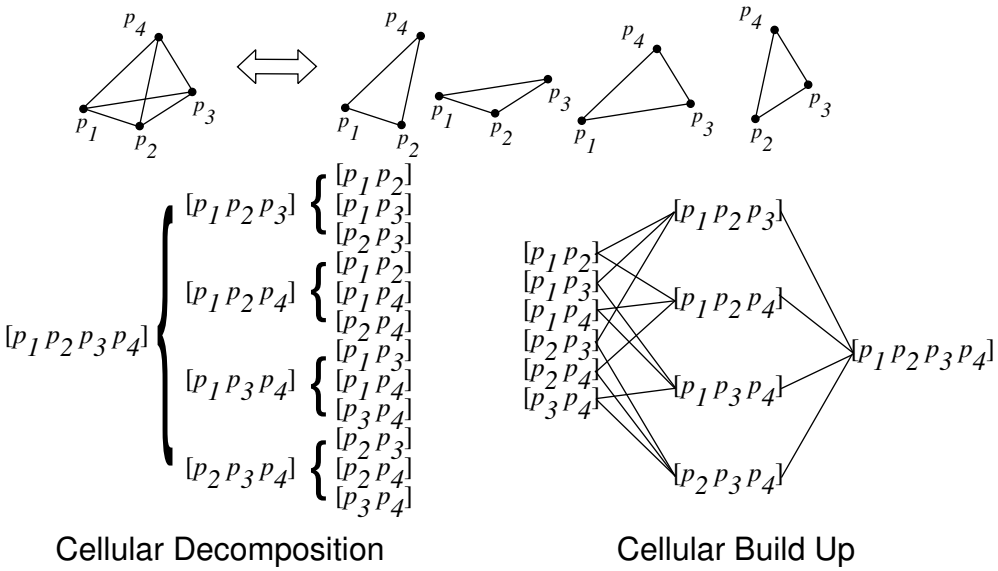


Figure 4.7: A cell complex can be decomposed progressively from a set of cells to a set of face cells and ultimately down to a set of edges. In this example, a single 3 dimensional cell has 4 cell faces each of which has 3 edges. Conversely, a cell complex can be built up from a set of edges. Edges with mutually shared points are matched to form 2-dimensional cells. These cells in turn become the faces of higher dimensional cells and the process continues recursively to dimension N-1.

from a list of points and the connections, or edges between points. The search is for all possible cells. If the input points and connections are well formed, the resulting collection of cells will form a cell complex in which faces between cells will be shared by no more than 2 cells. The `BuildUpCells` algorithm is summarized in Figure 4.8.

In Figure 4.8, note that the list of cells with p_i as a first element, $\mathcal{C}_i^m$, must contain at least m+1 cells in order to generate Δ_{cand}^n. Also, the search for the test cell Δ_{test}^m can be restricted to the subset of $\mathcal{C}^m$ with cells whose first element is a point greater than p_i. This step was omitted from Figure 4.8 for brevity.

4.5.4 `InsideSimplex` and `OnBoundary`

The functions `InsideSimplex` and `OnBoundary` determine whether an arbitrary point lies within an N-1 dimensional simplex cell, or within a cell of an N-1 dimensional cell complex or boundary. A point within an N-1 dimensional cell will satisfy the equations and criteria given in 4.3. This is a system of N equations with N-1 unknowns, α_i. However, any point "above" or "below" the simplex (as determined by the cell normal, ν) can satisfy this set of equations. On the other hand, if the test point, p_0 is within the hyperplane of the simplex, then the vector $p_0 - p_1$ will not have a component along the simplex normal, ν. If this test succeeds then a change in basis so that one direction is along the simplex normal reduces the problem to a unique set of N-1 equations.

The system of equations in 4.3 can be expressed in the form $v = Ax$ where $v = p_0 - p_1$, A is an NxN-1 matrix of vector differences, $A = [p_2 - p_1 | \ldots | p_N - p_{N-1}]$, and x is a vector of coefficients, α_i. The matrix A can always be decomposed by orthogonal triangularization into two matrices, $A = QR$. QR factorization is standard in many numerical commercial software packages such as Matlab®. R is an upper triangular matrix of the same dimension as A and Q is an orthornormal transformation and is always invertible. Thus $v = [QR]x$ and $Q^T v = Rx$. Since R is NxN-1 and upper triangular, the last row in R is all zeros. Thus we can eliminate the last row in R and $Q^T v$ and solve for x as shown below. The Q^T transformation provides the appropriate change of basis for reducing problem dimension.

function `BuildUpCells`$(\mathcal{C}^1, N-1)$ **returns** $\mathcal{C}^{N-1}$
inputs: $\mathcal{C}^1$, List of edges (1-dimensional simplexes),
$N-1$, Dimension of the returned Cell complex.
initialization: Create an ordered list of points, P, from $\mathcal{C}^1$.
Determine number of unique points in P: N_p.
Initialize cell faces, $\mathcal{C}^m = \mathcal{C}^1$ and $m = 1$.

1. **for** $n = 2, \ldots, N-1$, {execute steps 2-11 below}.
2. Set $m = n - 1$. Initialize new cell complex, $\mathcal{C}^n$.
3. **for** $i = 1, \ldots, N_p$, {execute steps 4-10 below}.
4. Create a list of cells from $\mathcal{C}^m$ with the first element of each cell equal to the ith element of P, p_i: $\mathcal{C}_i^m$.
5. The number of cells in $\mathcal{C}_i^m$ is M. Find all combinations of m+1 cells out of $\mathcal{C}_i^m$: $Comb(m+1, M)$.
6. **for** $k = 1, \ldots,$ size$(Comb(m+1, M))$, {execute steps 7-10 below}.
7. P_k is the collection of points from the kth combination of cells out of $\mathcal{C}_i^m$. **if** the number of unique points in P_k equals m+2 and each of the unmatched points in P_k, p_j (where $j > i$) occurs m+1 times, **then** {execute steps 8-10 below}.
8. Generate an n dimensional candidate cell from the m+2 unique points above: Δ_{cand}^n (to be added to $\mathcal{C}^n$).
9. Generate an m dimensional test cell from the m+1 unmatched points above: Δ_{test}^m.
10. Search $\mathcal{C}^m$ for a match to Δ_{test}^m. **if** Δ_{test}^m is contained in $\mathcal{C}^m$, **then** add the candidate cell, Δ_{cand}^n, to $\mathcal{C}^n$.
11. Replace $\mathcal{C}^m = \mathcal{C}^n$, **return** to step 1.
12. **return** $\mathcal{C}^{N-1}$, (i.e. $n = N - 1$).

Figure 4.8: `BuildUpCells` Function

$$
\begin{aligned}
R &= \begin{bmatrix} r_{1,1} & r_{1,2} & \dots & r_{1,N-1} \\ 0 & r_{2,2} & \dots & r_{2,N-1} \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \dots & r_{N-1,N-1} \\ 0 & 0 & \dots & 0 \end{bmatrix} = \begin{bmatrix} R_{red} \\ 0 \quad \dots \quad 0 \end{bmatrix} \\
Q^T v &= \begin{bmatrix} z_1 \\ \vdots \\ z_{N-1} \\ 0 \end{bmatrix} = \begin{bmatrix} z_{red} \\ 0 \end{bmatrix}
\end{aligned}
$$

and

$$
x = [R_{red}]^{-1} z_{red}
$$

The `InsideSimplex` function determines whether a point, p_0, is inside an N-1 dimensional simplex, specified by points $p_1,\ p_2, \dots,\ p_N$, is summarized in Figure 4.9.

Since the boundary representation limits cell size, if p_0 is inside a boundary cell it must be within one half a cell diameter of at least one cell point. Therefore, to determine whether the point p_0 is contained in any Boundary cell, one need only apply the `InsideSimplex` function to cells with points within one half a cell diameter of p_0. The `OnBoundary` function is summarized in Figure 4.10.

4.5.5 InsideBoundary

A point is inside a closed boundary if any line segment originating from the point and extending beyond the boundary surface in one direction intersects the boundary an odd number of times. The exception is a line which is just tangent to the boundary surface. The argument is an extension of the Jordan curve theorem (see Stillwell, [44]), and follows from the definition of a closed boundary. Since the boundary is closed and finite a line projecting from an initial point must be outside the boundary at some distance from that point. Also, since a closed boundary divides space in two,

function `InsideSimplex`(p_0, $p_1, p_2, \ldots, p_N$) **returns** Test = True/False
inputs: p_0, the test point,
$p_1, p_2, \ldots, p_N$, array of N-1 dimensional simplex points
initialization: $v_0 = p_0 - p_1$, $A = [p_2 - p_1 | \ldots | p_{N-1} - p_N]$

1. Determine whether p_0 is too close to any p_i:
   for $i = 1, \ldots, N$, {**ifany**($||p_i - p_0|| < \epsilon$) **return** Test=True}.
2. Get simplex normal, ν, from cofactor expansion of A:
   if($|v_0 \cdot \nu| > \epsilon$) **return** Test=False.
3. Perform triangular orthogonalization of A: $A = QR$ and $Q^T v_0 = Rx$.
4. Reduce dimensionality by eliminating the last row in $z = Q^T v_0$ and R to form z_{red} and R_{red}.
5. **if** R_{red}^{-1} exists, **then** solve for x: $x = R_{red}^{-1} z_{red}$, **else return** Test=False.
6. Determine whether the inside constraints of equation 4.3 are satisfied:
   if($x_1 < 0$ or $x_1 > 1$) **return** Test=False
   for $i = 2, \ldots, N-1$,
   {**ifall**($0 \leq x_i \leq x_{i-1}$) **return** Test=True, **else return**Test=False}.

Figure 4.9: `InsideSimplex` Function

function `OnBoundary`($p_0, \mathbf{S}$) **returns** Test = True/False, N_{cell} = cell index
inputs: p_0, the test point,
S, Boundary or Subboundary including cells and points.

1. Determine all points in **S** within 1/2 the maximum cell diameter, d_{max}, of p_0: P.
2. Find all cells with points in P: $\mathcal{C}$.
3. **for** $i = 1, \ldots,$ size($\mathcal{C}$),
   {**ifany**(`InsideSimplex`($p_0, \mathcal{C}_i$)) **return** Test=True, $N_{cell} = i$ **else return**Test=False, $N_{cell} = 0$}.

Figure 4.10: `OnBoundary` Function

every point along the line segment must be either inside or outside the boundary. Starting from the origination point, if the line crosses into the boundary it must also eventually cross outside of the boundary unless the intersection point is a tangent. If the origin point is inside the boundary then, mind-full of the tangency exception, the number of intersections between the line and the boundary must be odd.

For the boundary representation, the intersection between a line segment and a cell is given by equation 4.4. The test to see whether the intersection is a tangent is two part. The first part is to determine if the intersection occurs at the face of a cell. This occurs if one of the α_i coefficients of equation 4.3 is at an extreme value such as zero or α_{i-1}. If the intersection is not at a cell face then the line segment must not be within the hyperplane of the cell and cannot be a boundary tangent so the test is complete. If the intersection is at a cell face then it is a point common to two or more boundary cells and an additional test is required. The direction of the line emanating from the point in question is compared against the inside normal of all boundary cells sharing the intersection. If the dot product against the inside normals changes signs then the intersection is a tangent.

The determination of whether a line intersects a boundary cell requires the solution of the N simultaneous linear equations given in equations 4.3 and 4.2. The equations take the form $v = Ax$ where x is an Nx1 vector of the λ and α_i coefficients. To determine if a point is inside a Boundary, the intersection test must be applied to a number of feasible candidate cells. In matrix form the equation for multiple sets of linear equations takes the form below where v_i and x_i are Nx1 vectors and A_i are NxN matrices.

$$\begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_K \end{bmatrix} = \begin{bmatrix} A_1 & 0 & \dots & 0 \\ 0 & A_2 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \dots & A_K \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_K \end{bmatrix}$$

or

$$V = SX$$

where from equations 4.3 and 4.2

$$\begin{aligned} v_i &= p_0 - p_1 \\ A_i &= [\ -u_i \ \mid p_2 - p_1 \ \mid \ \ldots \ \mid p_N - p_{N-1}\ ] \\ x_i &= [\ \lambda \ \ \alpha_1 \ \ \alpha_2 \ \ \ldots \ \ \alpha_{N-1}\ ]^T \end{aligned}$$

Many commercial numerical packages such as Matlab® provide numerically efficient algorithms for solving large systems of equations that operate on sparse matrices. Again, the system of equations is decomposed by orthogonal triangularization so that $S = QR$. If R is of full rank than the entire system of equations can be solved simultaneously. The numeric solution with sparse matrices is typically more efficient than finding solutions to $v_i = A_i x_i$ individually. If R is not of full rank than one or more of the $v_i = A_i x_i$ system of equations is degenerate (e.g. the line extending from p_0 is embedded in the hyperplane of the simplex). Since the large matrix S is block diagonal the Q and R matrices are also block diagonal. Inspection of R can determine which of the $v_i = A_i x_i$ system of equations is degenerate and these can be removed. The reduced set of equations can then be solved for $X_{red} = [Q_{red} R_{red}]^{-1} V_{red}$ and the number of valid non-tangent intersection can be determined. A summary of the `InsideBoundary` function is summarized in Figure 4.11.

The `InsideBoundary` function is used in the `Resect` and `Merge` operations. Because of the large number of points that need to be checked, the numerical efficiency of the function is important. Unfortunately, the number of cells in $\mathcal{C}_k$ is large and a number of the tests in Figure 4.11 are computationally expensive. One potential improvement is to choose a new random unit vector, u_k, if the size of $\mathcal{C}_k$ is too large or degenerate cells are found (this eliminates the need for part of step 8 and all of step 9). These improvements seem to work well over 99 percent of the time but sometimes fail to correctly determine whether a point is inside or outside of a boundary.

4.5.6 `SubdivideCells`

When performing merge or intersect operations on boundaries and subboundaries, individual cells within each Boundary overlap and intersect. The intersecting cells

function `InsideBoundary`$(p_0, \mathbf{B})$ **returns** Test = True/False
inputs: p_0, the test point,
B, Boundary data object including simplex cells and associated points
initialization: $U = \{u_1, u_2, u_3, \ldots\}$, a random set of unit vectors,
P, a list of points in **B**.

1. Check to see if p_0 is a point in **B**. **if**$(\min(\| p_0 - P \|) < \epsilon)$, **then return** Test = True.
2. Check to see if p_0 is on Boundary. **if**(`OnBoundary`$(p_0, \mathbf{B})$), **then return** Test=True.
3. Select a direction u_k from U and $U = U \setminus u_k$.
if$(U = \emptyset)$, **then return** Test=False.
4. Find the set of all points, P_k, within 1/2 the maxi cell diameter $(0.5 * d_{max})$ of u_k from p_0.
$\{P_k \mid p_i \in P \wedge (p_i - p_0) \cdot u_k > 0 \wedge \| (p_i - p_0) - ((p_i - p_0) \cdot u_k) u_k \| < 0.5 * d_{max}\}$ **if**$(P_k = \emptyset)$ **return** Test=False.
5. Find all cells containing points P_k: $\mathcal{C}_k$
6. Generate $v_i = A_i x_i$ for each cell in $\mathcal{C}_k$ and form system of equations $V = SX$ in the sparse matrix equations above}.
7. Decompose $S = QR$.
8. Inspect diagonals of R for any degenerate solutions and eliminate associated A_i and v_i to form S_{red} and V_{red}. Eliminate associated cells (cells with u_k contained within their hyperplane) to get $\mathcal{C}_{red}$. If all solutions are degenerate **goto** step 3.
9. Repeat orthogonalization on reduced matrices: $S_{red} = Q_{red} R_{red}$.
10. Solve system of equations: $X_{red} = R_{red}^{-1} Q_{red}^T V_{red}$.
11. **for** $i = 1, \ldots,$ size$(\mathcal{C}_{kred})$, {determine from α coefficients in x_i if intersection is inside cell $\mathcal{C}_{kred_i}$}. Create a set of valid intersections: K.
12. Determine if any intersections in K are repeated. If so then intersection must be along a cell face. Check associated cell normals for tangency conditions. Eliminate repeated points and eliminate tangent points from K to get K_{red}.
13. **if**(mod(size$(K_{red}), 2) = 1$), **then return** Test=True, **else return** Test=False.

Figure 4.11: `InsideBoundary` Function

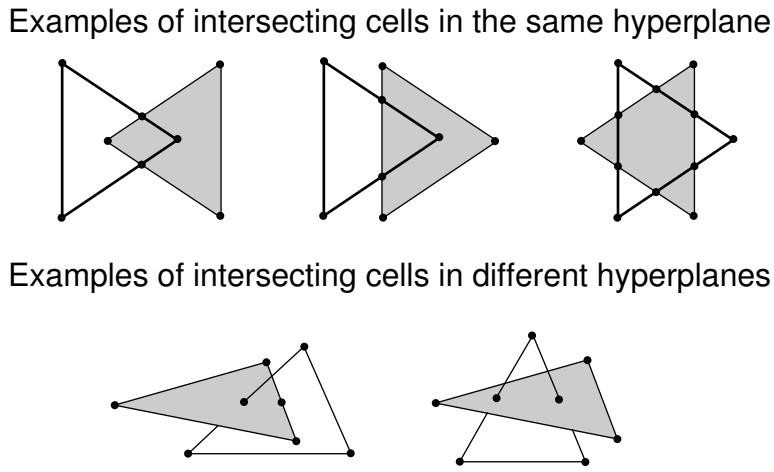


Figure 4.12: Example intersections for cells in the same hyperplane and cells in different hyperplanes. Within the same hyperplane points of intersection occur between the edges of one cell and the faces of the other and the vertices of one cell may be contained within the other. For intersections between cells in different hyperplanes points of intersection occur between cell edges and the other cell.

generate new points and cells which become part of the new Boundary. The way in which cells can intersect is varied. Figure 4.12 gives a couple of examples of how cells within the same plane in different planes can intersect.

A systematic approach to subdividing cells is to consider points and intersections one at a time and build the resulting cell complex. For example, consider the overlapping simplexes $\mathcal{A}$ and $\mathcal{B}$ at the top of Figure 4.14. The line segment between points p_a and p_b intersect $\mathcal{B}$ twice. The intersection points, p_1 and p_2, subdivide $\mathcal{A}$. At the first intersection $\mathcal{A}$ is divided into two: $\{[p_a\ p_c\ p_1],\ [p_b\ p_c\ p_1]\}$. At the second intersection $\mathcal{A}$ is divided further: $\{[p_a\ p_c\ p_1],\ [p_b\ p_c\ p_2],\ [p_c\ p_1\ p_1]\}$. Taken in succession, each intersection divides a cell in two. When an intersection occurs at a face between adjacent cells in $\mathcal{A}$, for example points p_7 and p_8 in Figure 4.14, then both cells will be divided.

When the cells are non-planar, then the intersection points between cells can occur inside the cell (i.e. not at the cell face). When the intersection is an interior point, the cell divides into N new cells. For example, if the point p_0 is an internal intersection

function `SubdivideCells`(p_0, Δ^{N-1}) **returns** $\mathcal{C}$
inputs: p_0, intersection point,
Δ^{N-1}, simplex cell of N points
outputs: $\mathcal{C}$, a set of new simplex cells
initialization: Get the N points of Δ^{N-1}: $p_1, p_2, \ldots, p_N$.

1. Determine if p_0 is within a tolerance, ϵ, of any of the simplex points. **ifany**$(|p_0 - p_i|)$, **return** $\mathcal{C} = \Delta^{N-1}$.

2. Form N permutations of new cells by substituting p_0 in for each point in Δ^{N-1}.
for $i = 1, \ldots, N$ $\{\Delta_i = \text{sort}(\{p_0, p_1, \ldots, p_N\} \setminus \{p_i\})$ $\}$.

3. For each Δ_i form a set of N-1 basis vectors, $[p_0 - p_1| \ldots |p_0 - p_{i-1}|p_0 - p_{i+1}| \ldots |p_0 - p_N]$. If the rank of the basis vectors is N-1 then add Δ_i to $\mathcal{C}$. **return**.

Figure 4.13: `SubdivideCells` Function.

then $\mathcal{A}$ divides into $\{[p_a\ p_b\ p_0]$, $[p_a\ p_c\ p_0]$, $[p_b\ p_c\ p_0]\}$.

In general, a point will divide an (N-1) dimensional simplex cell into N new simplex cells unless the point is inside a cell face. The new cells are generated by the permutation of the old cell points with the new intersection point. Each permutation is checked to verify that the cell spans an N-1 dimensional hyperplane. If the intersecting point is within a cell face then at least one of the new cell permutations will not span N-1 dimensional space and this permutation is degenerate. The determination of degeneracy is done numerically and requires tolerance for round off and floating point error. The `SubdivideCells` function is described in Figure 4.13.

The `SubdivideCells` function is readily modified to accommodate multiple input cells when, as often happens, the intersecting point p_0 subdivides multiple cells.

4.5.7 `IntersectSubboundaries` and `UnionSubboundaries`

We can now consider the problem of forming the intersection between subboundaries. The subboundaries are assumed to lie within the same N-1 dimensional manifold. Therefore, points in one subboundary may lie within the cells of the other subboundary and intersections between cells occur at the cell faces. The problem is to determine

which points are inside or outside of the cells in the complementary subboundary, determine intersections between edges and cell faces, subdivide the appropriate cells and build up a new subboundary from common points.

An example case of two overlapping cells is shown in Figure 4.14. Subboundaries $\mathcal{A}$ and $\mathcal{B}$ both consist of a single cell and none of the points of either subboundary is inside the other subboundary. However, the points in $\mathcal{A}$ and $\mathcal{B}$ are in close proximity and the edges between cells intersect. Starting with $\mathcal{A}$ form a set of edges between points: $E_A = \{[p_a\ p_b], [p_a\ p_c], [p_b\ p_c]\}$. For each edge find any intersections with cell faces in $\mathcal{B}$. For edge $[p_a p_b]$ we find the intersection points p_1 and p_2 which subdivides the cells in $\mathcal{A}$: $\{[p_a\ p_c\ p_1], [p_b\ p_c\ p_2], [p_c\ p_1\ p_2]\}$. Continuing with edge $[p_a\ p_c]$ we find intersections p_3 and p_4, and with edge $[p_b\ p_c]$ we find intersections p_5 and p_6. The original cell in $\mathcal{A}$ has new been subdivided into seven new cells, an additional cell for each intersection.

We repeat this process for $\mathcal{B}$. The intersection points along edges $[p_x\ p_y]$ and $[p_x\ p_z]$ are the same as for $[p_a\ p_c]$ and $[p_b\ p_c]$ and therefore don't change anything. However, the intersections along $[p_y\ p_z]$ contain two new points, p_7 and p_8. These points are added to $\mathcal{A}$ and corresponding cells are subdivided. The intersecting subboundary consists of points in $\mathcal{A}$ that are also in $\mathcal{B}$: $\{\ [p_1\ p_2\ p_7], [p_1\ p_3\ p_4], [p_1\ p_4\ p_8], [p_1\ p_7\ p_8], [p_2\ p_5\ p_6], [p_2\ p_6\ p_7]\}$.

The `IntersectSubboundaries` function is summarized in Figure 4.15. Note that the function only searches for intersections in the vicinity of the border between $\mathcal{A}$ and $\mathcal{B}$. There is no need to add additional points and cells well inside the region of space overlapped by both boundaries. Another point worth making is that intersections between edges and boundary faces can be approximate. In general the surfaces described by $\mathcal{A}$ and $\mathcal{B}$ are curved. However, the edges between points are straight lines and subcells are contained in an N-2 dimensional hyperplane. In $\mathbb{R}^3$, for example, the line segments defining an edge from one subboundary and a cell face from another may cross but not actually intersect. Rather than solve algebraically, the intersection is interpreted as a least squares fit to the shortest distance between points along the edge and inside the cell face. If the distance between points is within a tolerance and the end points are contained within the edge and cell face, then the

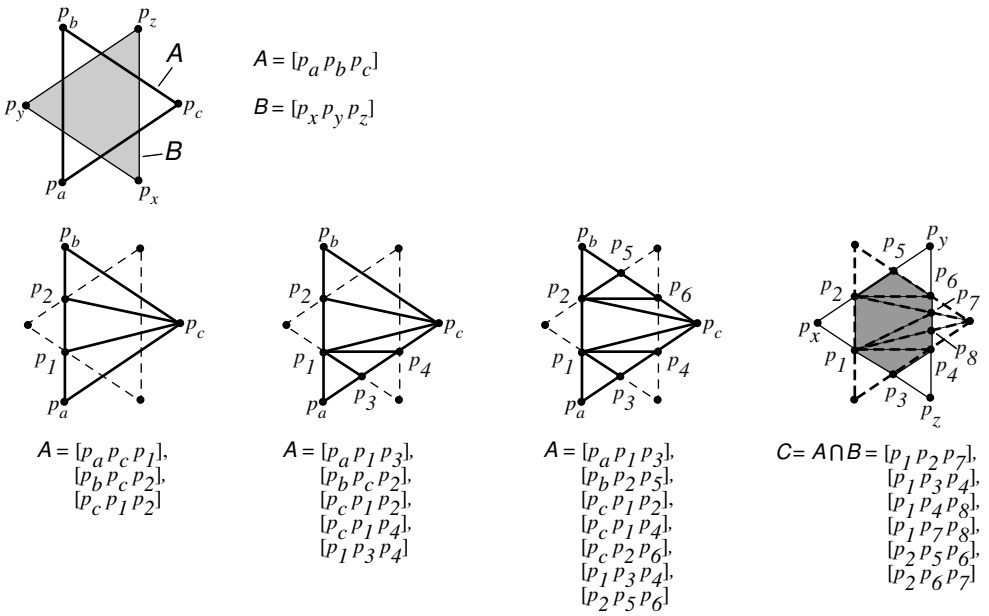


Figure 4.14: Two subboundaries intersect to form a new cell complex. Intersection points between edges in $\mathcal{A}$ and faces in $\mathcal{B}$ are found. Each intersection subdivides $\mathcal{A}$. Then intersection points between edges in $\mathcal{B}$ and faces in $\mathcal{A}$ are found to further subdivide $\mathcal{A}$. Intersecting cells consist only of points in $\mathcal{A}$ inside $\mathcal{B}$ and the intersections.

intersection point is the end point at the cell face.

The union between subboundaries is determined in a manner similar to the intersection between subboundaries except that the union contains all points in subboundary $\mathcal{A}$ and only those points in subboundary $\mathcal{B}$ that are not in $\mathcal{A}$. The union also contains the necessary intersection points common to $\mathcal{A}$ and $\mathcal{B}$. The search for intersections between edges and subcells is more involved because intersection points must be added to both subboundaries concurrently. This increases the number of edges and cell faces that must be searched for potential intersection. The `UnionSubboundaries` function is summarized in Figure 4.16.

4.5.8 `Merge`

`IntersectSubboundaries` and `UnionSubboundaries` apply when the two subboundaries lie within the same N-1 dimensional manifold. When the two subboundaries lie in different manifolds the algorithms must be modified. The `Merge` operation assumes that a closed portion of the subboundary $\mathcal{A}$ border lies inside $\mathcal{B}$. `Merge` takes the points and cells in $\mathcal{B}$ that are enclosed by the $\mathcal{A}$ border and adds them to $\mathcal{A}$. The analog in three dimensions is taking a sheet of paper, ($\mathcal{B}$), lying on top of an open can, ($\mathcal{A}$), and cutting the sheet to form a lid for the can. The `Merge` operation is illustrated in Figure 4.2.

Since the border points of $\mathcal{A}$ are inside $\mathcal{B}$, the intersections and subdivided cells can be found as described for `IntersectSubboundaries`. The difficulty of the `Merge` problem is determining what portion of $\mathcal{B}$ is inside or outside the closed curve defined by the border of $\mathcal{A}$. To solve this problem we use the inside direction of the points in $\mathcal{A}$. The inside direction of each point is part of the Boundary definition given in section 4.1. Take the cells in the now subdivided $\mathcal{B}$ with a face (N-1 points) that is contained within $\mathcal{A}$ and determine whether the remaining cell point is along the inside direction of the points in $\mathcal{A}$. If the point is inside add it to a list of points, P_{new}. Once this process is complete new points and cells can be added to $\mathcal{A}$. The process is repeated and cells are added in layers or rings until all in cells in $\mathcal{B}$ inside the border of $\mathcal{A}$ have been tested. After the first iteration, the cells with faces containing only

function `IntersectSubboundaries`($\mathcal{A}, \mathcal{B}$) **returns** $\mathcal{C}$
inputs: $\mathcal{A}, \mathcal{B}$, subboundaries with cells in the same hyperplane,
outputs: $\mathcal{C}$, subboundary of the intersection between $\mathcal{A}$ and $\mathcal{B}$.
initialization:

1. Determine points of $\mathcal{A}$ on $\mathcal{B}$, P_{AonB}, and points of $\mathcal{A}$ off $\mathcal{B}$, P_{AoffB}:
   forall p_i in $\mathcal{A}$ { **if**(OnBoundary(p_i, $\mathcal{B}$)), add p_i to P_{AonB}, **else** add p_i to P_{AoffB} }.
   ifall(p_i are in P_{AonB}), **return** $\mathcal{C} = \mathcal{A}$.

2. Determine points of $\mathcal{B}$ on $\mathcal{A}$, P_{BonA}, and points of $\mathcal{B}$ off $\mathcal{A}$, P_{BoffA}.
   forall p_i in $\mathcal{B}$ { **if**(OnBoundary(p_i, $\mathcal{A}$)), add p_i to P_{BonA}, **else** add p_i to P_{BoffA} }.
   ifall(p_i are in P_{BonA}), **return** $\mathcal{C} = \mathcal{B}$.

3. Add points in P_{AonB} to $\mathcal{B}$ and subdivide cells in $\mathcal{B}$. Add points in P_{BonA} to $\mathcal{A}$ and subdivide cells in $\mathcal{A}$.

4. Find all points in P_{AoffB} within the maximum cell diameter of any point in $\mathcal{B}$: P_{AnearB}.

5. Form edges from cells in $\mathcal{A}$ that contain points in P_{AnearB}: E_A. At least one of the edge points must be a member of P_{AnearB} and the other point may be a member of P_{AnearB} or P_{AonB}.

6. For each edge in E_A, create a set of subcells from $\mathcal{B}$ in the vicinity of E_A, $\mathcal{B}_{subcells}$, and search for intersection points, p_{int} = `IntersectSubCell`($E_A, \mathcal{B}_{subcells}$). If the intersection is unique add p_{int} to P_{AonB} and $\mathcal{A}$ and subdivide cells.

7. Repeat steps 3-5 for $\mathcal{B}$ substituting $\mathcal{B}$ for $\mathcal{A}$ except that we add unique intersection points to $\mathcal{A}$ and subdivide cells in $\mathcal{A}$.

8. Generate cells for $\mathcal{C}$ from cells in $\mathcal{A}$ that contain only points in P_{AonB}.
   return $\mathcal{C}$.

Figure 4.15: `IntersectSubboundaries` Function

function `UnionSubboundaries`($\mathcal{A}, \mathcal{B}$) **returns** $\mathcal{C}$
inputs: $\mathcal{A}, \mathcal{B}$, subboundaries with cells in the same hyperplane,
outputs: $\mathcal{C}$, subboundary of the union between $\mathcal{A}$ and $\mathcal{B}$.

1. Determine points of $\mathcal{A}$ on $\mathcal{B}$, P_{AonB}, and points of $\mathcal{A}$ off $\mathcal{B}$, P_{AoffB}:
   forall p_i in $\mathcal{A}$ { **if**(OnBoundary(p_i, $\mathcal{B}$)), add p_i to P_{AonB}, **else** add p_i to P_{AoffB} }.
   ifall(p_i are in P_{AonB}), **return** $\mathcal{C} = \mathcal{B}$.

2. Determine points of $\mathcal{B}$ on $\mathcal{A}$, P_{BonA}, and points of $\mathcal{B}$ off $\mathcal{A}$, P_{BoffA}.
   forall p_i in $\mathcal{B}$ { **if**(OnBoundary(p_i, $\mathcal{A}$)), add p_i to P_{BonA}, **else** add p_i to P_{BoffA} }.
   ifall(p_i are in P_{BonA}), **return** $\mathcal{C} = \mathcal{A}$.

3. Add points in P_{AonB} to $\mathcal{B}$ and subdivide cells in $\mathcal{B}$. Add points in P_{BonA} to $\mathcal{A}$ and subdivide cells in $\mathcal{A}$.

4. Find all points in P_{AoffB} within the maximum cell diameter of any point in $\mathcal{B}$: P_{AnearB}.

5. Form edges from cells in $\mathcal{A}$ that contain points in P_{AnearB}: E_A. At least one of the edge points must be a member of P_{AnearB} and the other point may be a member of P_{AnearB} or P_{AonB}.

6. For each edge in E_A, create a set of subcells from $\mathcal{B}$ in the vicinity of E_A, $\mathcal{B}_{subcells}$, and search for intersection points, p_{int} = `IntersectSubCell`($E_A, \mathcal{B}_{subcells}$). If the intersection is not already in $\mathcal{A}$ add p_{int} to P_{AonB} and subdivide cells. If the intersection is not already in $\mathcal{B}$ add p_{int} to P_{BonA} and subdivide cells in $\mathcal{B}$.

7. Repeat steps 3-5 for $\mathcal{B}$ substituting $\mathcal{A}$ for $\mathcal{B}$ and vice versa.

8. Generate points and cells for $\mathcal{C}$ from points and cells in $\mathcal{A}$. Add to $\mathcal{C}$ points from P_{BoffA} and all cells in $\mathcal{B}$ that contain at least one point in P_{BoffA}. **return** $\mathcal{C}$.

Figure 4.16: `UnionSubboundaries` Algorithm

points from the previous iteration must be inside the closed border and there is no need to check against inside directions. The `Merge` function is summarized in Figure 4.17.

4.5.9 Resect

Boundary resection is used during formation of the initial phase boundaries and during the backward chaining phase of the maximal invariant safe set algorithm. The resection is between a closed boundary and a subboundary that divides the closed boundary into multiple closed boundaries. The inside directions of the resecting subboundary are used to determine which portion of the subdivided boundary to keep. The `Resect` operation is illustrated at the bottom of figure 4.2.

This algorithm is similar to `Merge` except that the points in the subboundary are either inside or outside the boundary so no assumption about points on the border can be used. Intersections between edges and cells may be at cell faces, as in `Merge`, or inside the cell proper. A summary of the `Resect` operation is given in Figure 4.18 and continuing with Figure 4.19.

Steps 4 and 7 of the `Resect` function state that the border faces formed from the points in P_{AandB} should form a closed border. Due to the approximate nature of the boundary representation, this is not always guaranteed which has required a significant amount of extra logic and sometimes hand editing to fix. Although the `Resect` function should generalize to higher dimensions, it has not been demonstrated beyond 3 dimensions (described in Chapter 5).

4.6 Chapter Summary

This chapter describes a numerical implementation for the maximal invariant safe subset (MISS) algorithm. The implementation is based on a *Boundary* representation that envelopes a compact subset of the continuous state space. A boundary is closed if it divides the continuous state space into two disjoint sets: those points that are inside or on the surface of the boundary (hence safe), and those points outside of

function `Merge`$(\mathcal{A}, \mathcal{B})$ **returns** $\mathcal{C}$
inputs: $\mathcal{A}$ subboundary with border points, $P_{Aborder}$, contained in subboundary $\mathcal{B}$.
(note: $P_{Aborder}$ must either be closed in $\mathcal{B}$ or divide $\mathcal{B}$ in two).
outputs: $\mathcal{C}$, subboundary the result of the merge between $\mathcal{A}$ and $\mathcal{B}$.
initialization: $P_{A+B} = \emptyset$, list of points common to $\mathcal{A}$ and $\mathcal{B}$.

1. Find the border points of $\mathcal{A}$, $P_{Aborder}$, inside $\mathcal{B}$. Add the inside points to $\mathcal{B}$ and subdivide cells. **forall** p_i in $P_{Aborder}$, { **if**(OnBoundary(p_i, $\mathcal{B}$)), **then** add p_i to P_{AinB}, and P_{A+B}, and `SubdivideCells`$(p_i, \mathcal{B})$}.

2. Find points in $\mathcal{B}$ near P_{AinB}, P_{Bnear}, and determine if any are on the border of $\mathcal{A}$. Add these points to $\mathcal{A}$ and subdivide cells. **forall** p_i in P_{Bnear} {**if**(OnBoundary(p_i, $\mathcal{A}$)), **then** add p_i to P_{A+B} `SubdivideCells`$(p_i, \mathcal{A})$}.

3. Form edges between points in P_{A+B} from cells in $\mathcal{A}$: E_A (E_A should form a closed border or bisect $\mathcal{B}$).

4. For each edge in E_A, create a set of subcells from $\mathcal{B}$ in the vicinity of E_A, $\mathcal{B}_{subcells}$, and search for intersection points, p_{int} = IntersectSubCells$(E_A, \mathcal{B}_{subcells})$. Add p_{int} to $\mathcal{A}$, $\mathcal{B}$, and P_{A+B} and subdivide cells.

5. Initialize a set of points, P_{Old}, used to find which points in $\mathcal{B}$ that belong to the merged boundary: $P_{Old} = P_{A+B}$. Initialize the set of points to be added to $\mathcal{A}$, $P_{New} = \emptyset$. Set Search = True.

6. **while**(Search = True), {execute steps 7-11}.

7. Find cells in $\mathcal{B}$ that contain at least N-1 points in P_{Old}, $\mathcal{B}_{Test}$.

8. For each cell in $\mathcal{B}_{Test}$ if N-1 points are in P_{Old}, find the point, p_0, that is not a member of P_{Old}. Execute step 10 or 11.

9. If this is the first iteration through, determine whether p_0 is inside $\mathcal{A}$ from the inside directions and if test is positive add p_0 to P_{New}.

10. If this is not the first iteration, add p_0 to P_{New}.

11. If P_{New} is not empty, find cells in $\mathcal{B}$ containing only points in P_{Old} and P_{New}. Add the cells and the points in P_{New} to $\mathcal{A}$. Replace $P_{Old} = P_{New}$ and set $P_{New} = \emptyset$. Return to step 6.

12. If P_{New} is empty, set Search = False and stop iterations. **return** $\mathcal{C} = \mathcal{A}$.

Figure 4.17: `Merge` Function

function `Resect`$(\mathcal{A}, \mathcal{B})$ **returns** $\mathcal{C}$
inputs: $\mathcal{B}$, a closed Boundary,
$\mathcal{A}$, the intersecting subboundary.
outputs: $\mathcal{C}$, the resected Boundary.
initialization: $P_{AandB} = \emptyset$, list of points common to both $\mathcal{A}$ and $\mathcal{B}$.
$P_{AinB} = \emptyset$, list of points in $\mathcal{A}$ that are inside boundary $\mathcal{B}$

1. Determine whether points in $\mathcal{A}$ are inside $\mathcal{B}$ and add to P_{AinB}. If the point is on the boundary of $\mathcal{B}$ subdivide cells in $\mathcal{B}$ and add point to P_{AandB}: **forall** p_i in $\mathcal{A}$ { **if**(`InsideBoundary`$(p_i, \mathcal{B})$), **then** add p_i to P_{AinB}, **if**(`OnBoundary`$(p_i, \mathcal{B})$), **then** add p_i to P_{AandB} and to $\mathcal{B}$ and subdivide cells}.

2. Form edges from cells in $\mathcal{A}$ that connect points that are inside $\mathcal{B}$, P_{AinB}, with points that are not inside $\mathcal{B}$: E_{A1}.

3. For each edge in E_{A1}, search for intersections with cells in $\mathcal{B}$, p_{int} = `IntersectCell`$(E_{A1}(i), \mathcal{B})$. Add p_{int} to $\mathcal{A}$ and $\mathcal{B}$ and the P_{AandB} and P_{AinB} lists, and subdivide cells.

4. Identify cell faces in $\mathcal{A}$ and points in P_{AandB}, $\mathcal{A}^{AandB}_{border}$, that contain only points in P_{AandB}. The border faces should form a closed border in $\mathcal{A}$.

5. Form another set of edges connecting points in P_{AandB} from $\mathcal{A}^{AandB}_{border}$: E_{A2}.

6. For each edge in E_{A2}, create a set of subcells from $\mathcal{B}$ in the vicinity of $E_{A2}(i)$, $\mathcal{B}_{subcells}$, and search for intersection points, p_{int} = `IntersectSubCell`$(E_{A2}(i), \mathcal{B}_{subcells})$. Add p_{int} to $\mathcal{A}$ and $\mathcal{B}$ and the P_{AandB} and P_{AinB} lists, and subdivide cells.

7. Identify cell faces in $\mathcal{B}$ and points in P_{AandB}, $\mathcal{B}^{AandB}_{border}$, that contain only points in P_{AandB}. The border faces should form a closed border in $\mathcal{B}$.

8. Initialize a set of points, P_{Old}, used to find which points in $\mathcal{B}$ that belong to the merged boundary: $P_{Old} = P_{AandB}$. Initialize the set of points to be added to $\mathcal{A}$, $P_{New} = \emptyset$. Set Search = True.

Figure 4.18: `Resect` Function steps 1 through 8

9. **while**(Search = True), {execute steps 10-14}.

10. Find cells in $\mathcal{B}$ that contain N-1 points in P_{Old}, $\mathcal{B}_{Test}$.

11. For each cell in $\mathcal{B}_{Test}$ if N-1 points are in P_{Old}, find the point, p_0, that is not a member of P_{Old}. Execute step 12 or 13.

12. If this is the first iteration through, determine whether p_0 is inside $\mathcal{A}$ from the inside directions and if test is positive add p_0 to P_{New}.

13. If this is not the first iteration, add p_0 to P_{New}.

14. If P_{New} is not empty, find cells in $\mathcal{B}$ containing only points in P_{Old} and P_{New}. Add the cells and the points in P_{New} to $\mathcal{A}$. Replace $P_{Old} = P_{New}$ and set $P_{New} = \emptyset$. Return to step 9.

15. If P_{New} is empty, set Search = False and stop iterations. **return** $\mathcal{C} = \mathcal{A}$.

Figure 4.19: `Resect` Function steps 9 through 15

the boundary. A subset of the boundary (a subboundary) defines the safe initial or entrance conditions for the phase. Another subboundary defines the safe exit conditions for the phase. One or more subboundaries connect the initial and exit condition subboundaries along extremal paths to create a closed boundary.

The boundary is described by N-1 dimensional cell complex. A simplex cell is a list of N points. A connection or edge is made between each point in the cell. A subset of N-1 of the N simplex points defines a cell face. If two simplex cells share a common face they form a cell complex. Only two cells can share the same face in a proper cell complex, but one cell may share a face with up to N other cells. The cell complex is a finite approximation to the boundary surface. Accuracy of the representation can be controlled by tolerances and initial spacing between points. The primary boundary operations are `CellEngine`, `Merge`, and `Resect`.

The initial boundaries for each phase are generated by time propagation between the initial and exit condition subboundaries. The edge of these subboundaries are *borders* composed of the unmatched or open cell faces. The border is an N-2 dimensional cell complex. The `CellEngine` function can propagate forward or backward

in time from initial conditions established by the border to form an N-1 dimensional cell complex. The time propagation determines the optimal control and disturbance inputs derived in Chapter 3. Subboundaries can be joined together using the `Merge` operation to create a closed boundary. The `Resect` operation is used to eliminate unsafe states or those states within the set defined by the *Reach(G,E)* operation.

There are numerous other functions that support the `CellEngine`, `Merge`, and `Resect` operations. These functions are described to a fair level of detail in section 4.5. However, the algorithms presented are not perfect and eliminating all errors is beyond the scope of the current work. For example, sometimes intersections between cells are missed or points that are inside a boundary are erroneously considered to be outside. The greatest difficulties lie with boundaries that have narrow or sharp angled features as it becomes easy to miss intersections during `Resect` operations. These difficulties increase in higher dimensional spaces and point to the need for boundary based feature extraction and enhancement. This is a well studied problem in three dimensions (see for example, Henderson, et. al, [21], 1994) but less studied in higher dimensions. However, such improvements are beyond the current scope and are considered future work. The execution of the MISS algorithm on a 3-dimensional version of the vertical hopper problem and some of the numerical difficulties encountered are described in Chapter 5.

Chapter 5

Timer Based Control

The previous chapter described the MISS algorithm for determination of the maximal invariant safe subset. The algorithm assumes a finite set of phases or modes and uses an N-1 dimensional boundary to define a safe subset of the continuous state space for each phase. The boundary itself includes a subboundary of safe initial conditions, a subboundary of safe exit conditions, and one or more other subboundaries connecting the two.

In this chapter the numerical algorithm from Chapter 4 is applied to the problem of open loop or timer based control of a hopping robot. Thrust is applied periodically by a timer. The addition of the timer increases the dimensionality of the problem to 3. Ringrose, (1997, [39]) showed that on level ground a purely open loop or timer based system can achieve stable hopping motion. We use the MISS algorithm to validate Ringrose's results on level ground and extend them to climbing stairs and hopping over variable terrain. The results demonstrate that the open loop strategy accommodates constant stairs but can be easily rendered unsafe if the terrain varies.

5.1 Open Loop Hopping Model

The continuous time dynamics for the vertical hopping robot was described in section 2.2. With perfect feedback thrusting is achieved instantly and can occur any time and at any level. Under open loop control, thrusting occurs only at a fixed level and only

while the timer is within a specified range. The timer increases at a constant rate but rolls over to zero after reaching a limit forcing periodic behavior. The non-dimensional timer state, $\hat{s}$ has the following dynamics:

$$\hat{s}' = 1 \quad \textit{(all phases)} \tag{5.1}$$

$$\text{and } \hat{s} \leftarrow 0 \text{ when } \hat{s} = \hat{s}_{max}$$

Where the left facing arrow in the equation above reads "is replaced by". One way to implement the timer rollover is to add a new discrete phase to force a reset. The difficulty is that the rollover can occur in any mode but only affects dynamics if the hopper is in ground contact. Thus, multiple new 'rollover' phases are required, complicating the hybrid phase transition (see, for example, Figure 6.1). Another approach is to capture the rollover using a cylindrical state space. Timer values are mapped to an angle from 0 to 360 degrees and the rollover occurs naturally. Such a state space is locally Euclidian (Nemytskii and Stepanov, 1989, [35]) but the distance metric must properly account for rollover. For example, the distance angle between angles at 355 degrees and 5 degrees is 10 degrees.

For this problem, we choose to implement the rollover with a cylindrical state space. We control hopper thrust by dividing the `contact` phase shown in Figure 2.3 into three new phases: `compression`, `thrust`, and `expansion`. The transition between modes is illustrated in Figure 5.1. Thrust occurs when the hopper is in contact and the timer value is between 0 and the thrust on limit, s_{thrust}. If the hopper contacts the ground with the timer set to thruster off, ($s > s_{thrust}$), the hopper enters the `compression` phase. The timer increases to a maximum limit and rolls over triggering the transition from `compression` to `thrust`. The hopper remains in `thrust` until the timer indicates thrusting is disabled, $s > s_{thrust}$, or the hopper leaves contact. If the hopper completes thrusting while still in ground contact, the hopper enters the `expansion` phase.

Although the single legged hopper is a simple mechanical system, the phase transitions governing the dynamics are relatively complex. Note that the `descent`, `compression`, and `thrust` phases have multiple successor transitions and the `thrust`

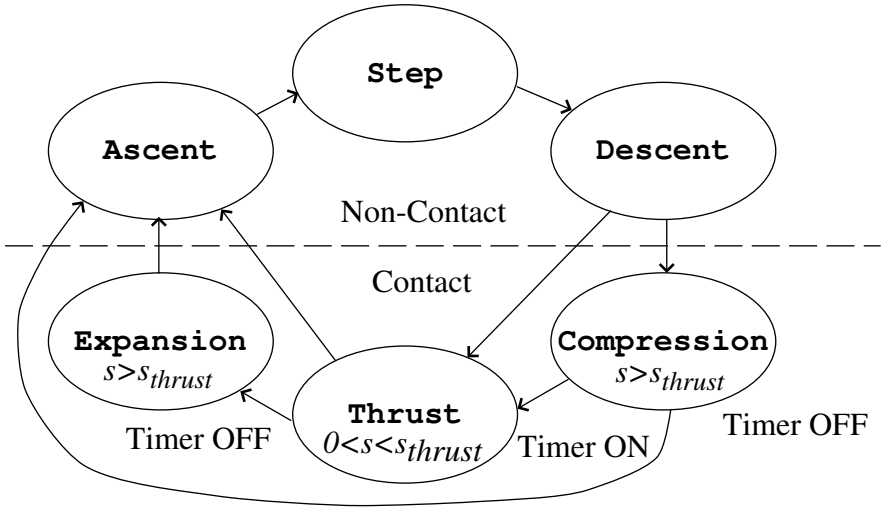


Figure 5.1: Hopper phases of motion under open loop control. The contact phase is divided into 3 new phases: `compression`, `thrust`, and `expansion`. The transition between phases is controlled by the timer state variable, $\hat{s}$. The timer is wrapped such that it resets to zero upon reaching a maximum value (not shown), thus forcing periodic motion.

and `ascent` phases have multiple predecessor phase transitions. The multiple transitions allow 5 possible paths to complete a hopping cycle. None of these paths allow the hopper to remain in contact over the entire cycle as that would be considered a failure.

$$\hat{y}'' = \begin{cases} -1 & \texttt{ascent}, \texttt{descent} \\ -\hat{k}\hat{y} - \hat{b}\hat{y}' - 1 & \texttt{compression}, \texttt{contact} \\ -\hat{k}\hat{y} - \hat{b}\hat{y}' - 1 + u_{max} & \texttt{thrust} \end{cases} \tag{5.2}$$

$$\hat{s}' = 1 \quad \text{for } 0 \leq \hat{s} < \hat{s}_{max} \quad \textit{(all phases)} \tag{5.3}$$

$$\hat{s} \leftarrow 0 \quad \text{for } \hat{s} = \hat{s}_{max} \quad \textit{(all phases)} \tag{5.4}$$

Since the control is timer based there is no continuous input, u, and there are no continuous disturbance inputs, d, for this problem. Hopper thrust is a fixed constant during `thrust` and zero for the `compression` and `expansion` phases. This eliminates

Parameter	*Value*
stiffness, $\hat{k}$	4.0
damping ratio, ζ	0.11
max thrust, $\hat{u}_{max}$	2.0
period, $\hat{s}_{max}$	3.0
on time, $\hat{s}_{thrust}$	0.6

Table 5.1: Open loop hopper non-dimensional parameter values

the need for the `liftoff` transition introduced for the closed loop problem in Chapter 2. Finally, since $\hat{s}' = 1$ for all time there are no singular points ($f(x, u, d) = 0$), and hence no singularity avoidance issues.

For the open loop problem we consider the timer limit, s_{max}, and the thrust duration, s_{thrust}, as fixed parameters not unlike leg stiffness and damping. This is not a strict limitation. It is possible to configure the problem to include an adaptive mechanism for modifying timer limits and thrust durations. However, this increases the dimensionality of the problem and is beyond the scope of the present discussion.

In Chapter 2 we studied how combinations of stiffness, damping, and thrust capability effect the ruggedness of a single legged hopper with perfect feedback control. For the open loop study in this chapter, we choose a hopper with parameter values that maximize ruggedness and straddle "step up" and "step down" limited hopping behavior discussed in Chapter 2. Thrust, leg stiffness, and damping were selected from the results in Figure 2.10. The timer limit or rollover is set to 3 in units of non-dimensional time. Thrust is enabled when the timer is between 0 and 0.6, a 20% duty cycle. The open loop hopper parameters are summarized in Table 5.1.

5.2 Initial Phase Boundaries for the Open Loop Hopper

The first step in the MISS algorithm is to establish an initial set of safe boundaries for each phase of motion. The process for defining the initial boundaries can be involved and generally requires some domain knowledge. This section describes generation of

the `compression` phase initial boundary as an example.

Referring back to Figure 5.1, initiation of the `compression` phase occurs on transition from `descent` when the hopper hits the ground and the timer state is off thrust ($\hat{s} > \hat{s}_{thrust}$). Hopper velocity is unspecified but it must be less than zero. The initial condition subboundary is therefore the N-1 dimensional surface defined by $\hat{y} = 0$, $\hat{y}' < 0$, and $\hat{s}_{thrust} < \hat{s} < \hat{s}_{max}$. However, the subboundary must be bound so a large negative velocity is selected ($\hat{y}'_{min} = -1.6$) as the minimum. The velocity limit is low enough that it will not restrict final solution to the maximal safe subset. Additionally, an initial `compression` phase velocity to close to zero can cause numerical problems. Although the dynamics do not include singularities, an initial zero velocity causes the state trajectory to spiral in tightly until the timer causes the hopper to thrust. The tight spiral creates a tortuous boundary with small scale features that cause numerical problems. Again, we resort to prior domain knowledge to set a maximum value for the initial velocity ($\hat{y}'_{max} = -0.05$). This value is far enough from zero to avoid the numerical problems described without ultimately restricting the maximal safe subset. The now finite (bounded) `compression` phase initial condition subboundary is shown in Figure 5.2.

The exit condition subboundary, also shown in Figure 5.2, is more complex. The `compression` phase will transition to `thrust` if the timer state variable exceeds $\hat{s}_{max}$ (at which time the timer state resets to zero), and the `compression` phase will transition to `ascent` if the ground reaction forces drop to zero prior to `thrust`[1]. Thus, the exit condition subboundary has two surfaces: one at $\hat{s} = \hat{s}_{max}$, and the other at $\hat{y} = -(\hat{b}/\hat{k})\hat{y}'$, which is the condition for zero ground reaction force in the absence of thrust. These two surfaces meet along the line where both conditions are satisfied. The position and velocity limits for the $\hat{s} = \hat{s}_{max}$ surface, as will be shown, are chosen to accommodate forward time propagation from the initial condition subboundary. The timer limits for the $\hat{y} = -(\hat{b}/\hat{k})\hat{y}'$ surface are $\hat{s}_{thrust} \leq \hat{s} \leq \hat{s}_{max}$. The minimum velocity for the ground reaction force surface is zero, and the maximum velocity is again chosen to accommodate forward time propagation of the initial conditions.

[1]This problem does not consider the possible but unlikely transition from `ascent` back to `thrust` if the timer were to reset during the brief period of time the ground reaction forces drop to zero while the hopper remains in ground contact.

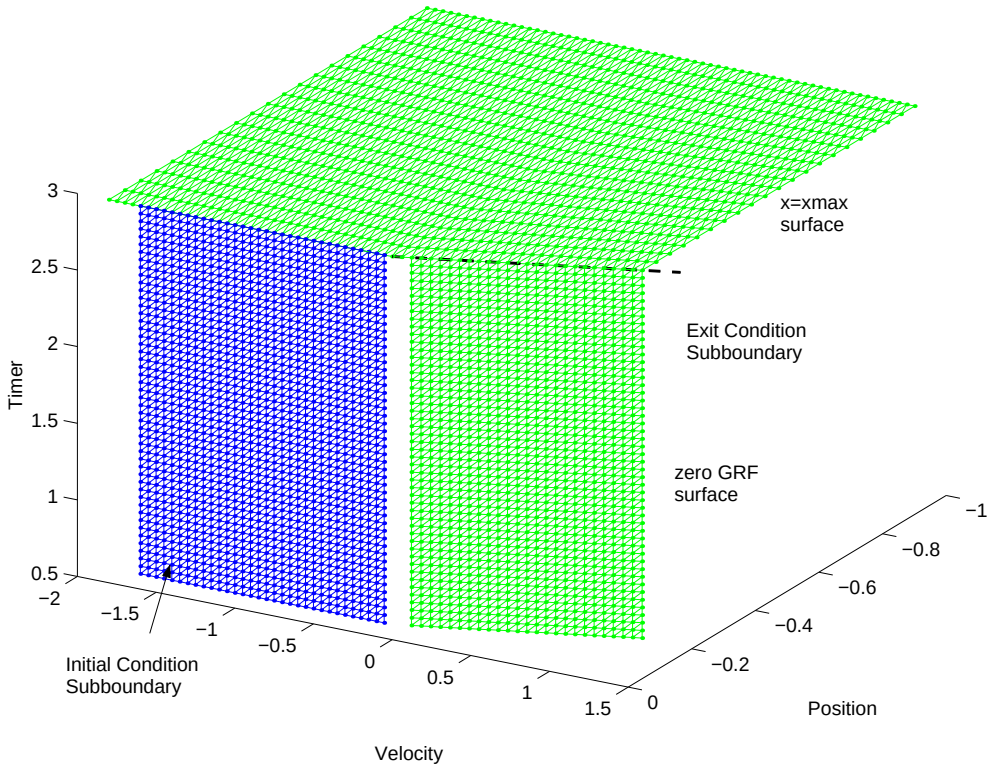


Figure 5.2: `Compression` phase initial and exit condition subboundaries.

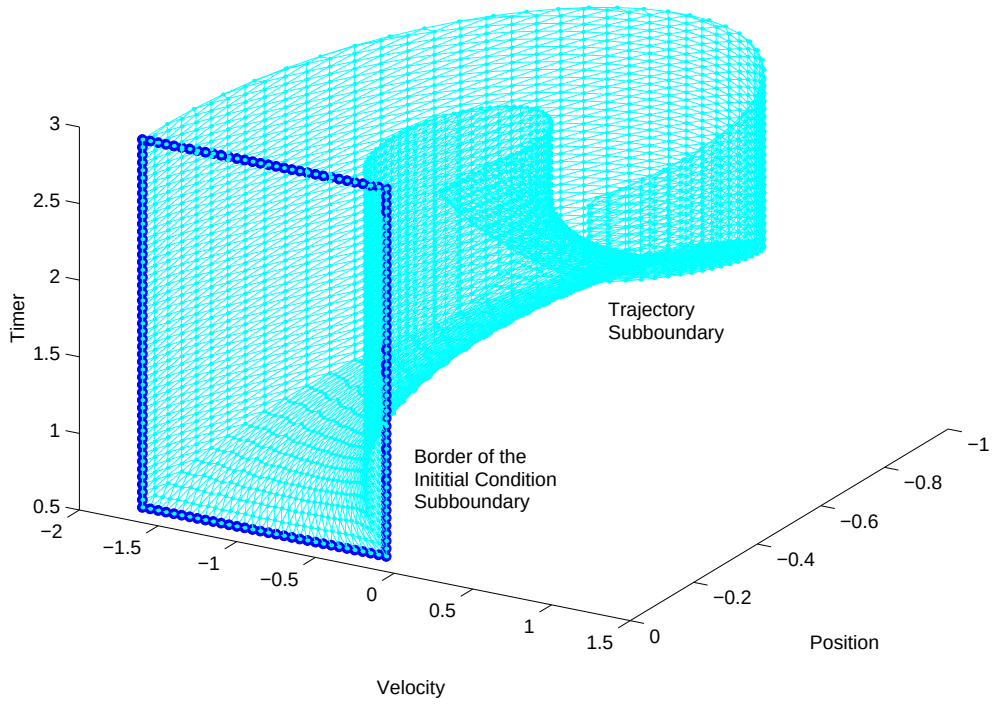


Figure 5.3: `Compression` phase "trajectory" subboundary. The subboundary is generated by propagating in forward time from the border of the `compression` phase initial conditions and connects the initial and exit condition subboundaries.

To generate the closed boundary we propagate forward in time from the border along the initial condition subboundary. Throughout the integration the `CellEngine` and `BuildUpCells` functions described in section 4.5.2 are invoked to create a "trajectory" subboundary, shown in Figure 5.3. The initial condition subboundary border is highlighted and the propagation is in forward time. In most cases the initial condition trajectories terminate at the $\hat{s} = \hat{s}_{max}$ surface. However, some of the trajectories reach the $\hat{y} = -(\hat{b}/\hat{k})\hat{y}'$ surface indicating that direct transition to `ascent` without thrusting is possible. Note that the timer state variable along the vertical axis causes the state trajectories to spiral upward in forward time.

The initial condition, exit condition, and trajectory subboundaries are merged to form a closed boundary for the `compression` phase. The `Merge` operation, described in section 4.5.8, is applied twice and the resulting boundary is shown in Figure 5.4. However, large portions of this boundary violate the $\hat{y} \geq -0.5$ leg compression constraint also shown on the figure. The safety constraints are enforced by forward and backward time propagation from a set of extrema points along the constraint subboundary. Since the control for this phase is by definition zero (no thrust) the extrema condition is met along the line $\hat{y} = -0.5$ and $\hat{y}' = 0$, which is the vertical set of points in the figure. The vertical line spans timer values $\hat{s} = 0.6$ to $\hat{s} = 3.0$ over which no thrusting occurs. However, this line itself is insufficient for generation of the resecting subboundary; forward and backward time propagation will not completely intersect the initial boundary. Accordingly, we augment the generator with a set of points along $\hat{y} = -0.5$, $\hat{y}' < 0$, and $\hat{s} = 3.0$ as shown.

The internal constraint subboundary is generated by propagating forward and backward in time from the internal constraint generator, applying the `CellEngine` and `BuildUpCells` functions as described for the trajectory subboundary, and then splicing the forward and backward propagated subboundaries together. The internal constraint subboundary and the `compression` phase boundary are shown in Figure 5.5. Note that for the `compression` phase forward time propagation stops at $\hat{s} = 3.0$ (transition to `thrust`), or the ground reaction forces drop to zero (transition to `ascent` phase). Backwards time propagation stops when $\hat{y} = 0$ (transition from `descent`).

The region of the initial boundary 'outside' of the leg constraint subboundary in Figure 5.5 is *Reach(G,E)* for the `compression` phase. A boundary containing only safe states is created by resecting the initial boundary with the internal constraint subboundary. The resulting boundary is shown in Figure 5.6. The boundary has a complex geometry and is non-convex. In this example the initial `compression` phase boundary consists of 4129 points and 8254 cells.

The process of generating initial boundaries, illustrated in figures 5.2 through 5.6, is repeated for all phases. However, only the `compression`, `thrust`, and `expansion` phases have have safety constraints. The `step` phase, is a discrete phase and does not require a boundary but only a set of initial and exit condition subboundaries. Initial

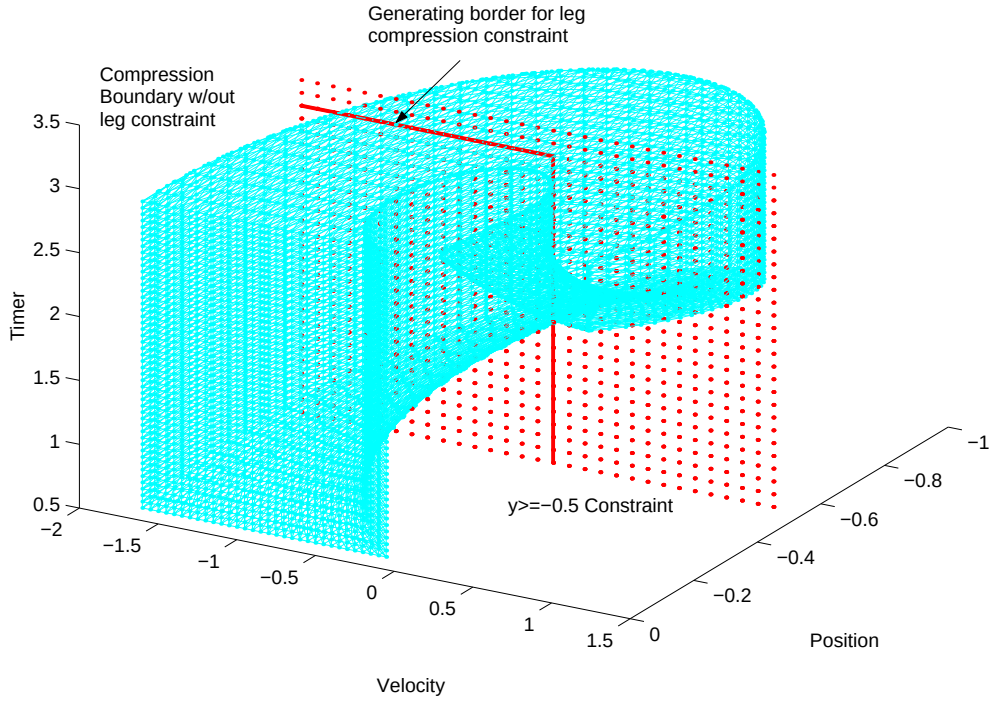


Figure 5.4: `Compression` phase initial boundary including unsafe states. The leg compression constraint, $\hat{y} \geq -0.5$, intersects the initial boundary. Extrema points at $\hat{y}' = 0$ are highlighted along a vertical line. Because of the wrapped nature of the timer, the extrema points also extend along a line at $s = 3.0$ and $\hat{y}' < 0$ to support backward time propagation.

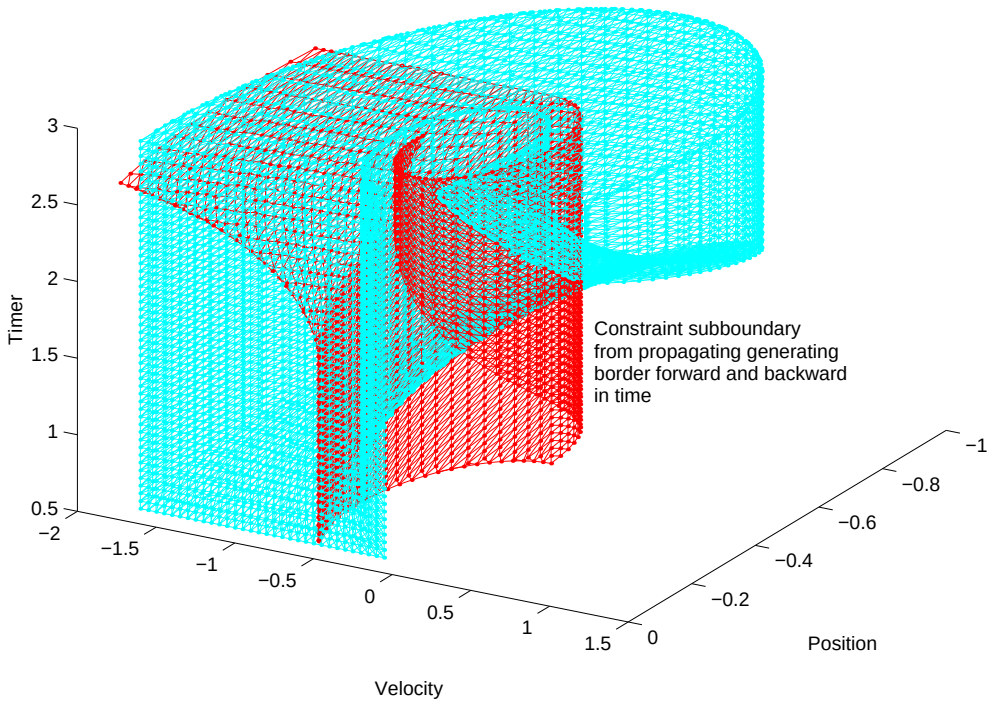


Figure 5.5: `Compression` phase initial boundary with leg constraint subboundary. The leg constraint subboundary is created by forward and backward time propagation from the extrema points shown in Figure 5.4.

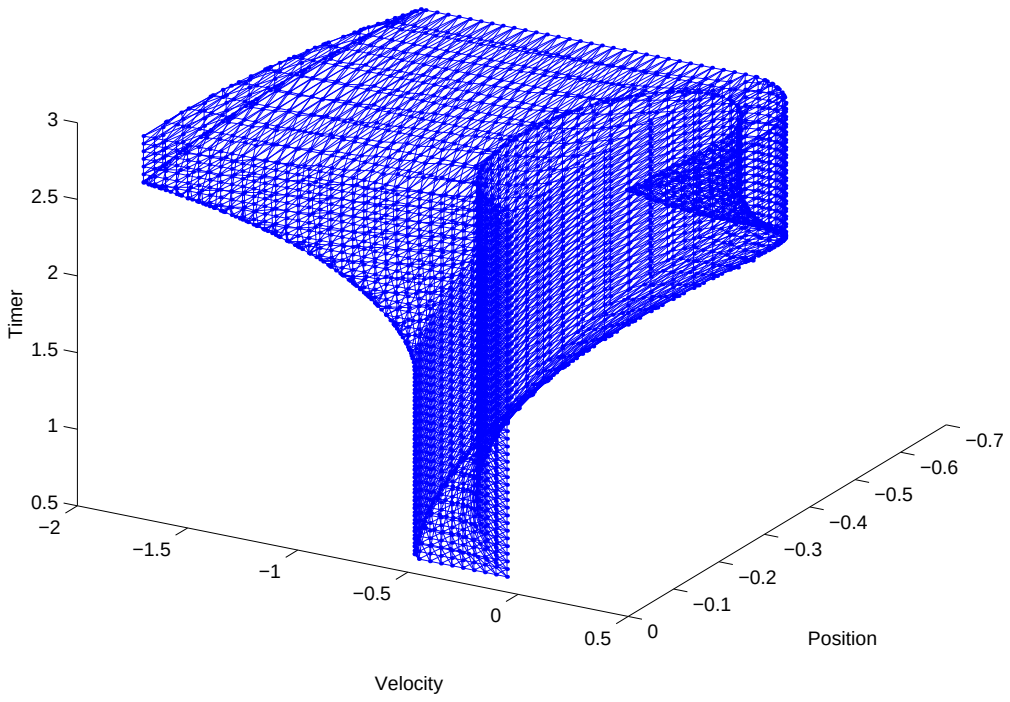


Figure 5.6: `Compression` phase initial boundary of safe states. The portion of the initial boundary satisfying the *Reach(G,E)* condition is removed by the `Resect` operation.

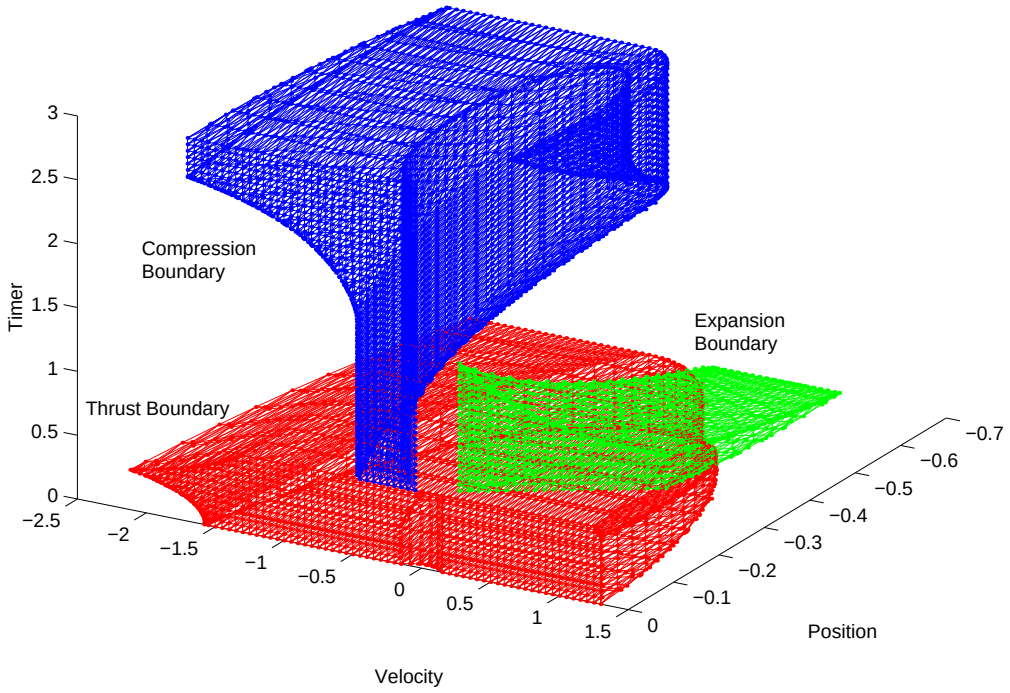


Figure 5.7: Initial boundaries for `compression`, `thrust`, and `expansion` phases. The `compression` phase exit conditions at the top of the figure ($\hat{s} = 3$) wrap to the `thrust` phase initial conditions at the bottom of the figure ($\hat{s} = 0$).

boundaries for safe states in the `compression`, `thrust`, and `expansion` phases are shown Figure 5.7. Since the timer state is wrapped, a state trajectory will exit the `compression` phase boundary at $\hat{s} = 3$ and enter the `thrust` phase boundary at $\hat{s} = 0$. The fact that the timer state is wrapped makes it difficult to illustrate the overlap between the `compression` phase exit conditions and the `thrust` phase initial conditions. The overlap between `thrust` and `expansion` phase exit and initial conditions at $\hat{x} = \hat{x}_{thrust}$ is more evident.

Initial boundaries for the `ascent` and `descent` phases are shown in Figure 5.8. The `step` phase initial and exit condition subboundaries are the same as the `ascent`

phase exit condition subboundary and are not shown. The wrapped nature of the timer state variable is evident from the shifted position of the top portion of the `ascent` boundary. In the time propagation from the `ascent` phase initial conditions at the right of the figure to the exit condition towards the middle, the timer variable reaches its maximum value and rolls over to zero causing the shift. Note that the rollover does not cause a change in mode because the hopper is not in contact.

Since the `ascent` phase exit conditions rollover, the `descent` phase initial conditions must also rollover. The rollover is difficult to show in a cartesian plot, however, the points at or near the timer value $\hat{s} = 3$ are connected to points at or near $\hat{s} = 0$ bridging the rollover. The `descent` phase initial conditions have an open border along $\hat{y} = 1.3$ and also at $\hat{y} = 0.005$, just above ground contact. The time propagation extends from these open borders to the `descent` phase exit conditions. The initial boundary appears hollow and open, but is actually a closed toroid. This is readily seen in a cylindrical coordinate space as shown in Figure 5.9. In this figure, velocity is along the vertical axis, position is a radial distance from the vertical axis, and the timer state is an angle about the vertical axis. The timer state is scaled so that 0 to $\hat{s}_{max}$ covers a circle. The position state is biased so that ground contact is at a radius of one and any position above ground is at a radius greater than one. The `descent` initial condition subboundary is the circular disk with a hole in it at zero velocity. The exit condition subboundary is the cylinder of radius one projecting downward from the hole in the center of the disk. The time propagation is the conic portion connecting the outside rim of the disk with the bottom rim of the cylinder. Cylindrical coordinates aid visualization of the wrapped timer variable but are not convenient for showing interactions between boundary initial and exit conditions and will not be used further.

Having generated the initial boundaries that define safe conditions for each phase, the next step is the backward chaining portion of the algorithm. The following section will describe the backward chaining for the case of level terrain: that is, no environmental disturbances are applied to the system. This establishes a useful baseline and the results can be compared against those reported by Ringrose (1997, [39]). Subsequent sections will describe the backward propagation results for steady state

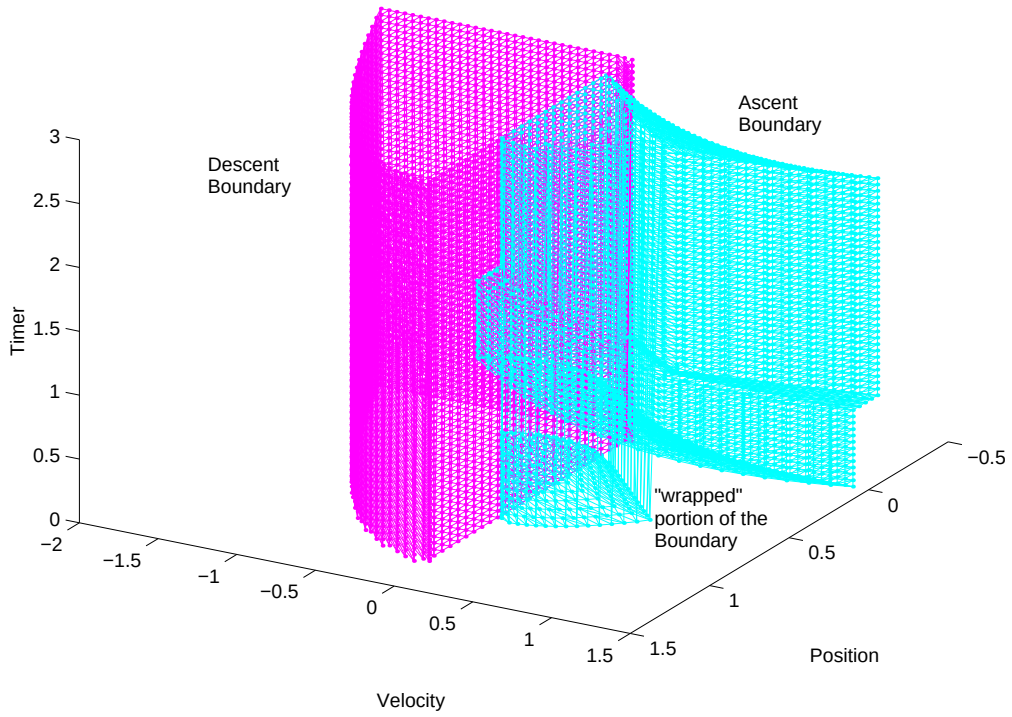


Figure 5.8: Initial boundaries for `ascent` and `descent` phases. The forward time propagation for the `ascent` phase initial boundary reaches the timer limit at $\hat{s} = 3$ and rolls over to zero. Since the hopper is not in contact, the leg cannot thrust and the rollover does not trigger a phase change. The `descent` phase initial condition subboundary is at $\hat{y}' = 0$ but for a timer value from 0 to 3 and hence wraps around.

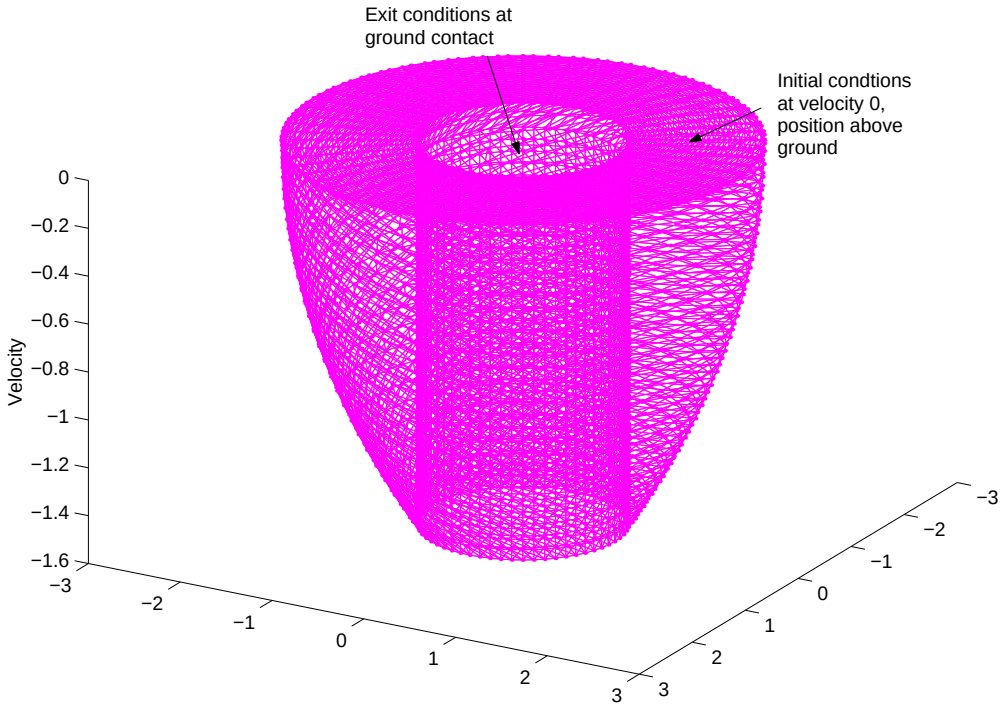


Figure 5.9: Initial `descent` boundary in cylindrical coordinates. Leg length is the radial distance from the center axis (not shown) and the timer variable is scaled as an angle about this axis so that 0 to $\hat{s}_{max}$ is 360 degrees. Velocity is along the vertical axis. Ground contact occurs at a radius of 1 which defines the exit condition subboundary. The initial condition subboundary is a circular disk at zero velocity, a radius from one to some bounding value, and over the possible range of timer values.

climbing where the terrain height changes by a constant value between hops, and finally variable terrain, where the terrain height can vary within a range values between hops.

5.3 Backward Chaining over Level Terrain

Prior to the backward chaining portion of the algorithm, the sequence through the phase transitions must be established. The sequence must step through all phases, though some transitions may become inactive as the algorithm progresses. The phase transition diagram in Figure 5.1 shows that the `expansion`, `thrust`, and `compression` phases are all predecessors to `ascent`. Additionally, the `thrust` phase is a predecessor to `expansion`, and `compression` is a predecessor for `thrust`. A backward chaining sequence that considers all predecessor/successor relationships in a feasible order is: `ascent`$\prec$ `expansion`$\prec$ `thrust`$\prec$ `compression`$\prec$ `descent`$\prec$ `step`, and then back to `ascent`. Note that for the case of level terrain, the `step` phase can be ignored so that the `descent`$\prec$ `ascent` transition is direct.

The `ascent`$\prec$ `expansion` transition and resection is shown in Figure 5.10. The upper left portion of the figure shows the `ascent` phase initial conditions in light grey, the `expansion` phase boundary in dark grey, and the intersection between `ascent` phase initial and `expansion` phase exit conditions in black. A portion of the `expansion` phase exit condition subboundary does not overlap the `ascent` phase initial condition subboundary. The border between overlapping and non-overlapping regions of the `expansion` phase exit conditions is shown in the upper right window of the figure. The border spans the exit conditions subboundary. Backwards time propagation from the new border generates a subboundary, shown in the lower left of portion of Figure 5.10, that bisects the old `expansion` phase boundary. The lower right portion of Figure 5.10 compares the `expansion` phase boundary after completing the resection operation (shown in dark grey) against the original `expansion` phase boundary. In this example the resection operation makes only a modest change to the `expansion` phase boundary.

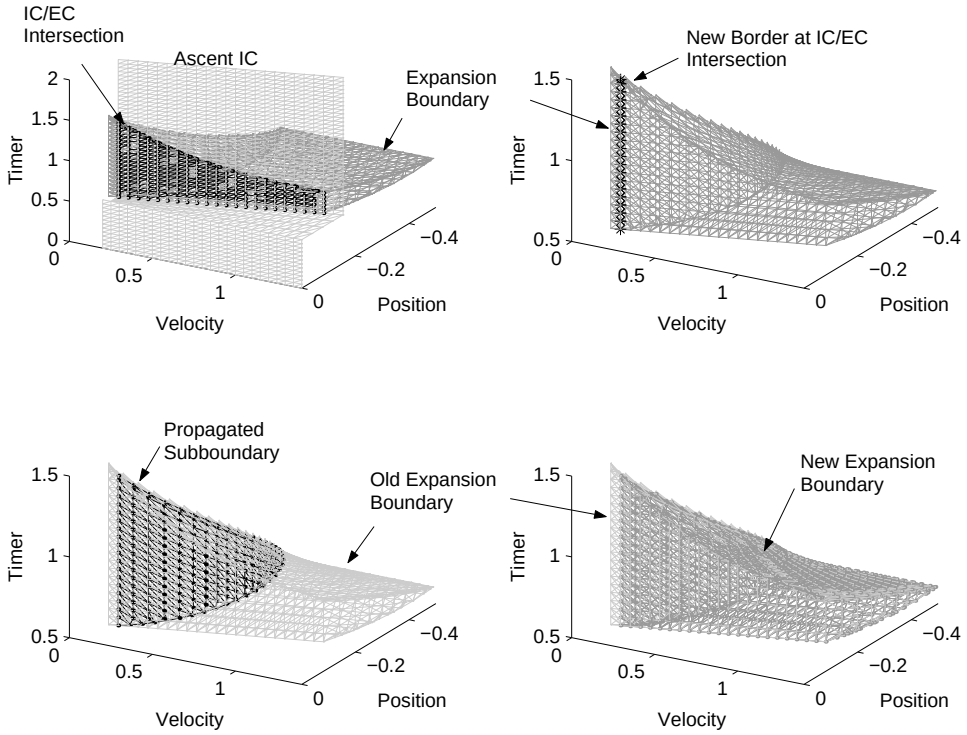


Figure 5.10: Backwards chaining through the `expansion` boundary. The time propagation portion of the chaining sequence begins by determining the intersection subboundary between `expansion` exit conditions and `ascent` initial conditions (upper left). If the intersection is a subset of the original exit conditions subboundary then the new border faces of the intersection subboundary are identified (upper right). The new border faces are the generator for backwards time propagation from exit to initial conditions. The propagated subboundary completely intersects the original `expansion` boundary (lower left). Finally, the old boundary is resected by the propagated subboundary to yield the new `expansion` boundary.

The backward chaining sequence can be understood by examining initial condition/exit condition subboundary interactions between predecessor and successor phases. A first iteration through the MISS algorithm is shown in Figure 5.11. In the figure, internal subboundary points and edges have been removed for clarity and only border points and edges are shown. The portion of the exit condition subboundaries that intersect with successor phase initial conditions are shaded dark grey. The remaining exit condition subboundary is shaded light grey. The initial condition subboundary is unshaded.

The upper left portion of Figure 5.11 starts with the same `ascent` initial conditions and `expansion` exit conditions as shown in the upper left of Figure 5.10. The `expansion` boundary resection is suppressed but the resulting `expansion` phase initial conditions are shown in the top right of Figure 5.11. Also shown in the top right is a portion of the `ascent` initial conditions (the portion abutting the `thrust` phase, i.e. timer states less than 0.6) and the `thrust` phase exit condition subboundary. The dark grey region in the top right figure is the new exit condition subboundary for `thrust`. The non-intersecting portion of the `thrust` phase boundary is resected as described for the `expansion` phase boundary and the new `thrust` phase initial conditions are shown in the middle left of the figure. The perspective of the middle left plot is straight down the Timer axis. All timer state values are zero (which is equivalent to a timer value of 3.0 due to rollover). The middle left plot also shows the `compression` phase exit conditions and the intersection with `thrust` phase initial conditions. Non-intersecting portions of the `compression` phase boundary are resected away and the new `compression` phase initial condition subboundary is shown in the middle right figure [2]. Here, almost the entire portion of the `compression` initial conditions intersect the `descent` phase exit conditions except for a small strip to the extreme right. The perspective for the middle right plot is straight down the position axis and all position states are at zero (initial ground contact). Finally, the lower left figure shows the resulting `descent` phase initial conditions and the pre-existing `ascent` phase exit conditions. The perspective of the lower left figure is straight up

[2]The middle right figure does not include `thrust` phase initial conditions for purposes of clarity. As the backward chaining sequence progresses, the transition from `descent` to `thrust` is eliminated from the invariant set.

the Velocity axis and all velocity states are zero (hopper at zenith). The first iteration completes after resecting non-intersecting portions of the `ascent` boundary and determining the new initial condition subboundaries.

The second iteration through the backward chaining sequence starts with a new `ascent` phase initial condition subboundary as shown in the top left of Figure 5.12. The top left window shows an errant point in the `ascent` phase initial conditions that was created during the resection operation in the previous iteration. The numerical functions described in Chapter 4 are imperfect. Occasional errors occur which must be examined and sometimes corrected by hand. In this case, the errant point is well outside the `expansion` phase exit conditions so the problem localized and will not affect the ultimate result.

In Figure 5.12, we repeat the sequence described for the first iteration but an interesting development occurs with the interaction between the `expansion` and `ascent` phase initial conditions and the `thrust` phase exit conditions shown in the top right. The intersection operation creates two separate exit condition subboundaries. The separate exit condition subboundaries generate separate and distinct boundaries within the `thrust` phase. Thus, what was once a compact and contiguous subspace of safe states is now disjoint. This division occurs once again with the interaction between `thrust` phase initial conditions and `compression` phase exit conditions shown in the middle left plot of the figure. Three distinct boundary subspaces propagate through the `compression`, `descent`, and `ascent` phases as shown in the middle right and lower left plots in Figure 5.12.

The third iteration through the backward chaining sequence is shown in Figure 5.13. The iteration starts with `ascent` phase initial condition subboundaries for three separate boundaries which intersect with the `expansion` phase exit condition subboundary to form three new exit condition subboundaries (top left plot). The three boundaries propagate through the `thrust` phase (top right plot). At the transition from `compression` phase exit conditions to `thrust` phase initial conditions the safe set subspace divides into 4 distinct boundaries (middle left plot). The top two exit condition subboundaries are very close to subboundaries determined in the previous iteration. This indicates the algorithm is beginning to reach steady state.

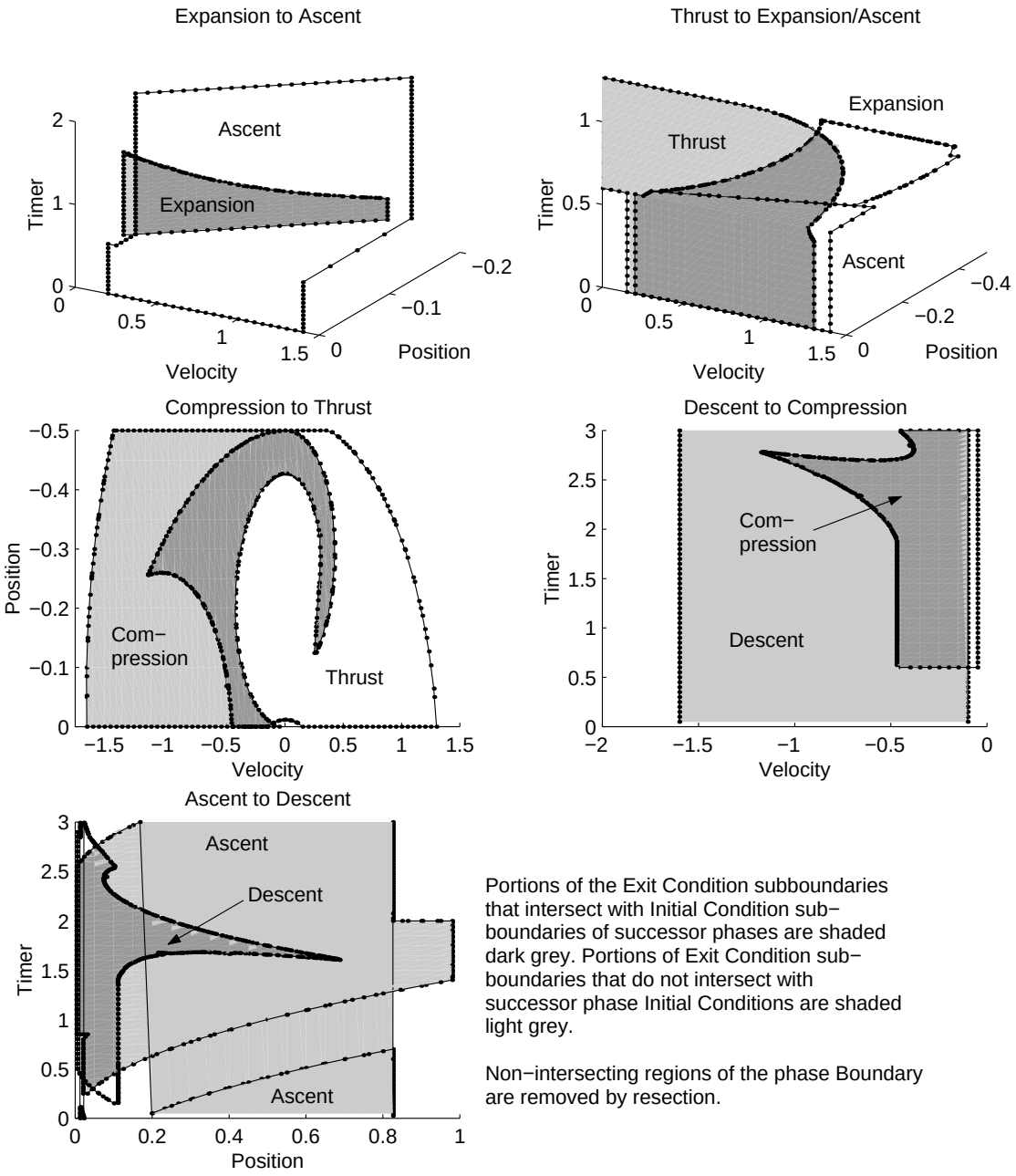


Figure 5.11: First iteration of the backward chaining sequence on level terrain. The sequence starts at the intersection of the `ascent` initial condition and `expansion` exit condition subboundaries (top left). The algorithm propagates in backwards time and creates a new initial condition subboundary. The intersection of the new `expansion` initial conditions with pre-existing `thrust` exit conditions forms a new exit condition subboundary (top right). The backwards chaining continues through to the `compression` phase (middle left), `descent` phase (middle right), and finally the `ascent` phase (lower left) exit condition subboundaries. The `thrust` phase initial conditions are omitted from `descent` phase exit conditions (middle right) for clarity.

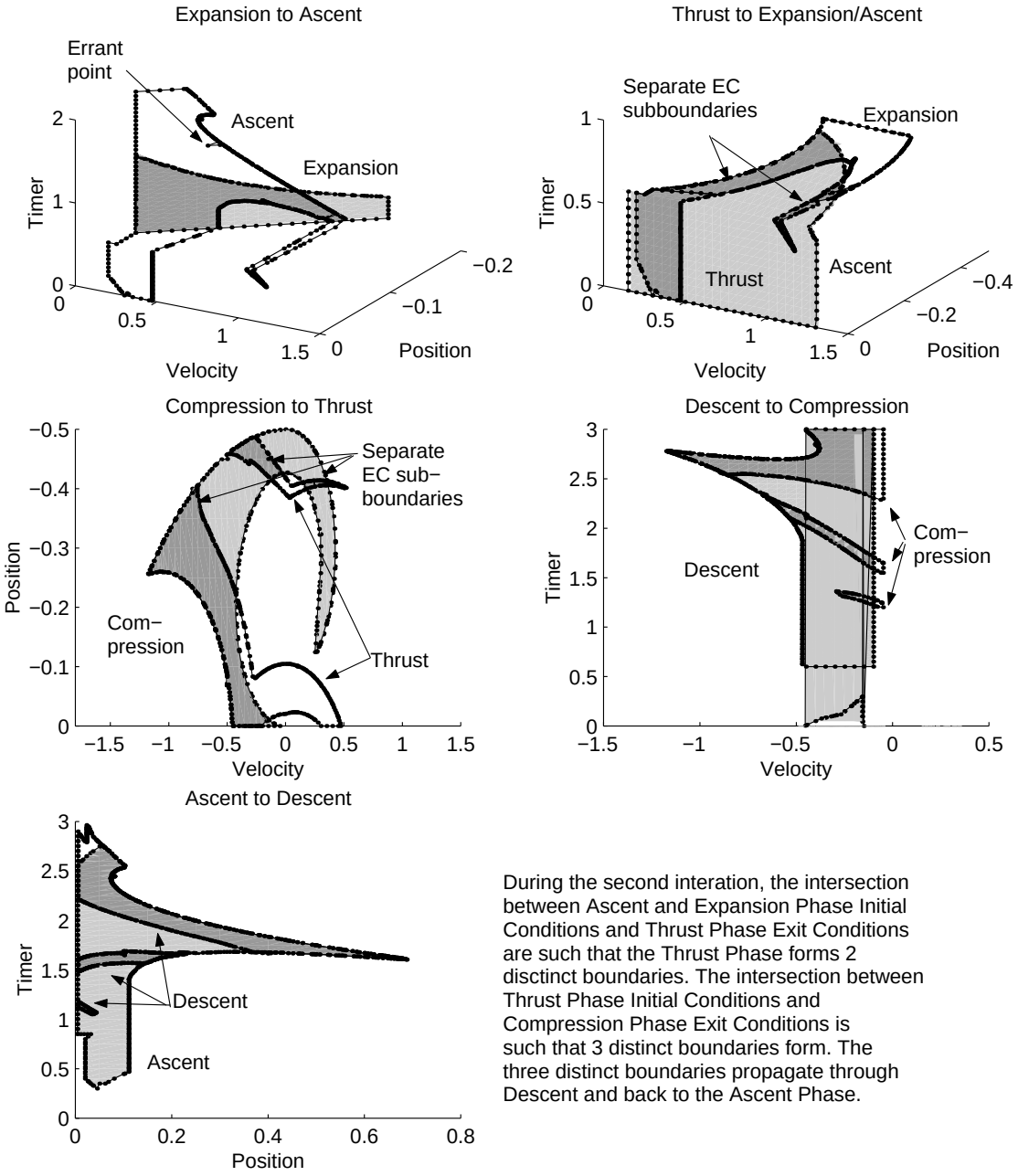


Figure 5.12: Second iteration of the backward chaining sequence on level terrain. The second iteration begins with a new `ascent` phase initial condition subboundary (top left). Subsequent `expansion` and `ascent` phase initial conditions intersect with `thrust` phase exit conditions forming two distinct exit condition subboundaries (top right). Subsequent `thrust` phase initial conditions intersect with `compression` phase exit conditions forming three distinct subboundaries (middle left). Multiple boundaries persist through the `descent` and `ascent` phases (middle right, bottom left).

The bottom two exit condition subboundaries are new. The pattern of two new subboundaries and two subboundaries very similar to subboundaries from the previous iteration continues through the `descent` and `ascent` phases.

By the fourth iteration through the backward chaining sequence, the algorithm has approached steady state. As shown in Figure 5.14, the successor phase initial condition subboundaries now match predecessor phase exit condition subboundaries quite closely. An exception is in the `ascent`$\prec$ `expansion` phase transition shown in the upper left window. The upper most initial condition subboundary does not overlap an `expansion` phase exit subboundary. The `expansion` phase has three exit conditions subboundaries here labeled A, B, and C, compared to four `ascent` phase initial condition subboundaries. The upper right plot follows the backwards time propagation from the `expansion` phase exit to initial condition subboundaries. The `thrust` phase exit conditions in the upper right plot corresponding to the `expansion` phase exit conditions in the upper left are labeled A, B, C. Additionally, the C' exit condition subboundary results from the `thrust`$\prec$ `ascent` phase transition. The `compression`$\prec$ `thrust` phase transition is shown in the middle left plot and the mapping from `thrust` phase exit conditions to `compression` phase exit conditions is again indicated by the labels. The separation between the C and C' exit condition subboundaries is caused by a limit on the `compression` phase initial velocity limit. Referring back to the middle left plot in Figure 5.11, the inside border of the `compression` exit condition subboundary starts from an initial velocity of -0.05. The inside border curves around near zero velocity and intersects the `thrust` phase initial condition subboundary twice slicing it in two. The four exit condition subboundaries propagate through the `compression`, `descent`, and `ascent` phases.

The bottom right plot of Figure 5.14 shows the `ascent` phase initial conditions for the next iteration of the backward chaining sequence. The initial conditions in the upper left and lower right hand plots are nearly identical so the backward chaining sequence terminates. These plots show the initial and exit subboundaries for the maximal invariant safe subset. If a trajectory passes through a point outside of these subboundaries, the hopper will reach an unsafe condition within 4 hops. If a trajectory passes through a point within these subboundaries, the hopper will remain

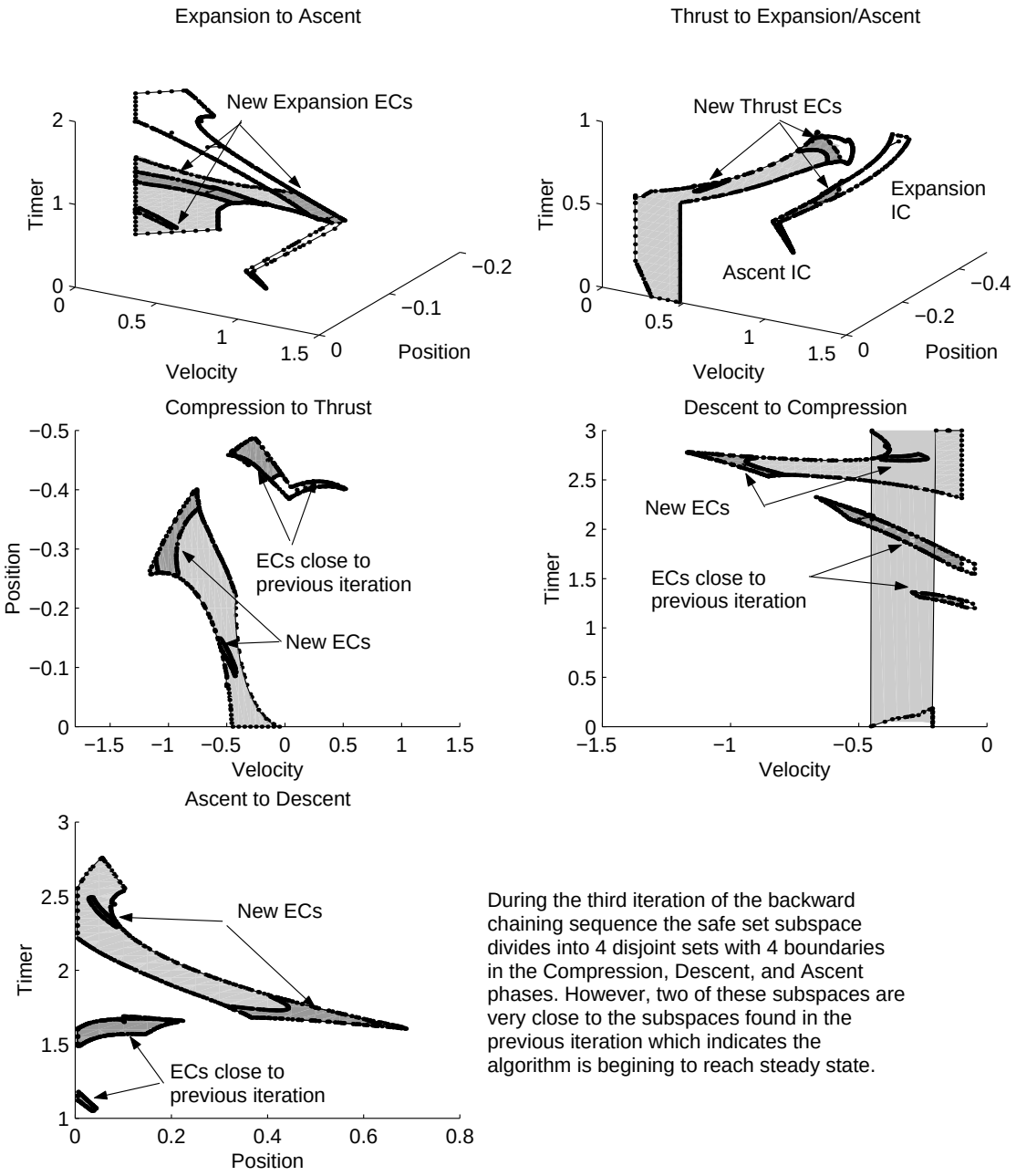


Figure 5.13: Third iteration of the backward chaining sequence on level terrain. The third iteration begins with three separate phase initial and exit condition subboundaries at the `ascent`← `expansion` phase transition (top left). The `compression` phase exit condition subboundaries split again and the number of safe set subspaces increases from three to four (middle left). However, two of the four exit conditions subboundaries are substantially the same as subboundaries determined in the previous iteration, indicating the algorithm is approaching steady state.

safe indefinitely.

In Figure 5.14, note the differences between subboundary labels in the the upper left and lower right plots. There are two interesting differences. One is the switch between the initial condition subboundaries labeled A and C. The other difference is between the B and C' initial condition subboundaries. Going forward in time (from lower right to upper left) a trajectory begining from a point in the A subboundary of the lower right plot will return to the `ascent` phase at a point in the C initial condition subboundary (lower right hand plot) and vice versa. A trajectory propagating forward in time and starting from a point the B subboundary in the shown in the lower right plot will return to the `ascent` phase in the C' initial condition subboundary (lower right hand plot). Finally, a trajectory propagating forward in time and starting from a point in the C' subboundary will return to the `ascent` phase in the A initial condition subboundary (lower right hand plot).

The switch between the A and C `ascent` phase initial condition subboundaries indicates that the hopper alternates between boundaries on successive hops. This is shown in the 2-dimensional projections of the state space trajectory in Figure 5.15. The figure shows several orthographic views of the state space trajectory through two hopping cycles. Beginning in this example with the `ascent` phase and at point X, indicated by an asterisk, the hopper cycles through the `descent`, `compression`, `thrust`, and `expansion`, and returns to the `ascent` phase at point Y. Rollover of the timer state variable is indicated by the broken vertical lines in the upper right and lower left hand plots. In the second hopping cycle starting at point Y, the hopper transitions through the `descent`, `compression`, and `thrust` phases and transitions directly from `thrust` to `ascent`, skipping `expansion`, and returning to the `ascent` phase at point X. Point X was chosen so that the trajectory forms a closed loop with periodicity 2. Points X and Y are fixed point solutions to the hopper dynamics on level terrain, but are contained in different `ascent` phase initial condition subboundaries. In the lower right hand plot of Figure 5.14 point X is inside subboundary A, point Y is inside subboundary C. Trajectories originating from points in the B or C' initial condition subboundaries transition to the A and C subboundaries after only a couple of hops and trend toward the fixed points X and Y.

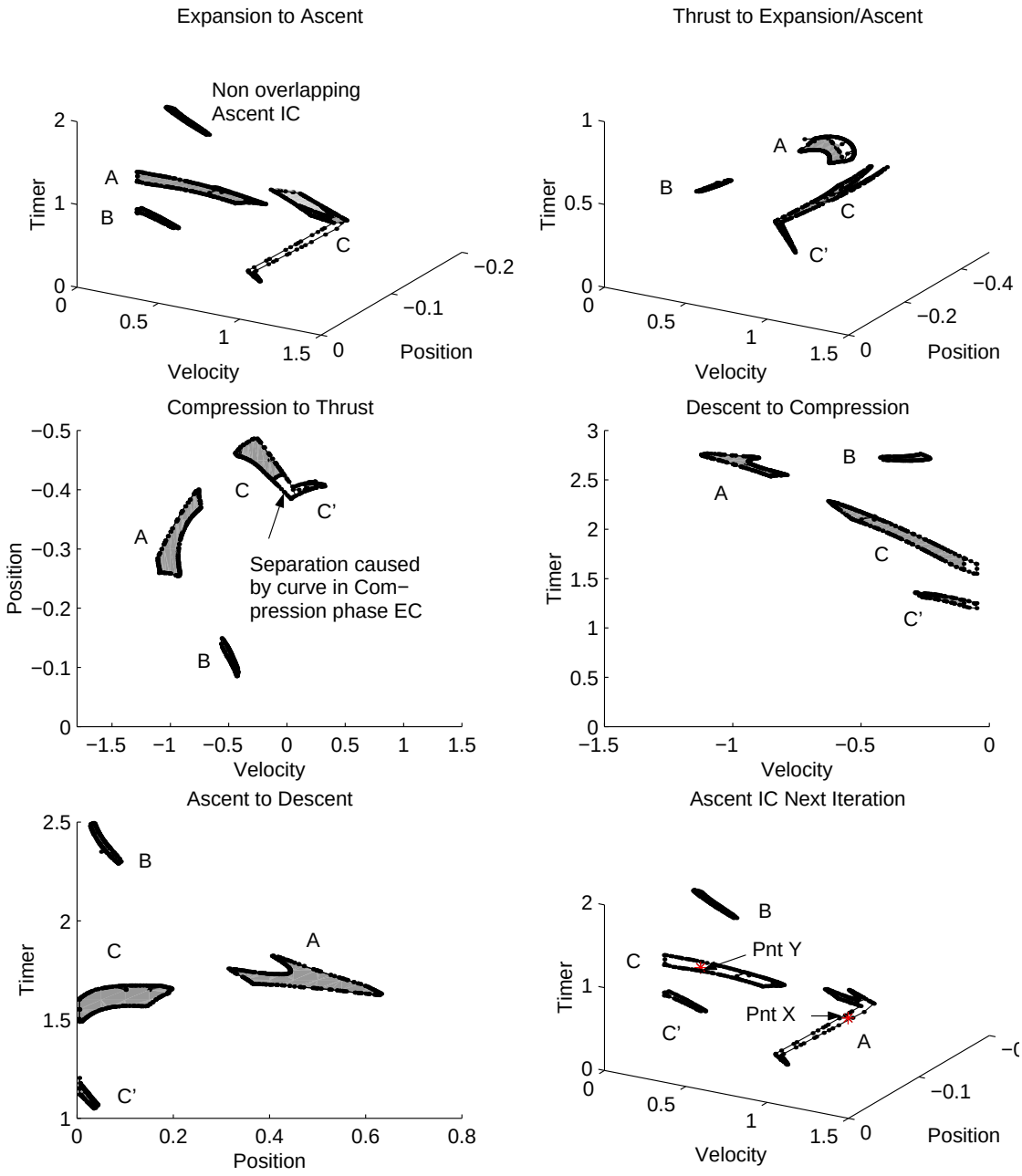


Figure 5.14: Fourth iteration of the backward chaining sequence on level terrain. The backward chaining sequence begins with `ascent` phase initial condition subboundaries in the upper left plot. The three subboundaries labeled A, B, C intersect `expansion` phase exit conditions and continue the backward chaining sequence. The three subboundaries split into four and return to the `ascent` phase at the initial condition subboundaries shown in the plot at the lower right. These subboundaries are nearly identical to the initial conditions subboundaries in the upper left plot, indicating the algorithm has reached steady state. Although the subboundaries are nearly the same, the corresponding labels through one iteration of the sequence are different indicating relatively complex dynamics.

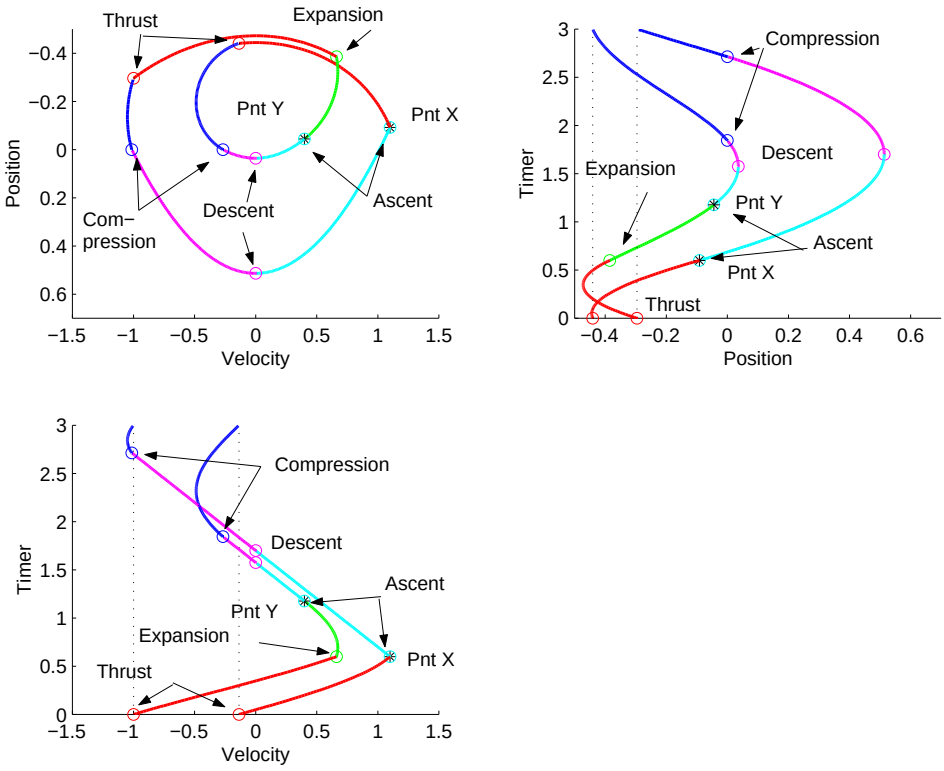


Figure 5.15: Fixed point state space trajectory for timed hopper on level ground. The state space trajectory is shown in several orthographic views. Points X and Y are `ascent` phase initial conditions that form a closed loop in two hopping cycles. The broken vertical line in the plots to the upper right and lower left indicate the timer state variable roll over. Points X and Y alternate between the `ascent` phase initial condition subboundaries in Figure 5.14.

Interestingly, the results shown in Figure 5.15 are similar to results reported by Koditschek and Bühler (1991, [27]). They reported steady state period 2 orbits, described as "limped gaits", but for a system with a non-linear spring and active feedback control. More recently, Cham [14] describes steady state period 2 solutions for a clock driven hopper with a linear spring on level ground.

5.4 Backward Chaining for a Stair Climbing Sequence

The previous section determined the maximal invariant safe subset for a clock driven hopper on level terrain and worked through the algorithmic steps in some detail. In this section we consider the exact same hopping robot, but this time with a constant increase in terrain height between hops as if climbing stairs. The change in terrain height is accommodated by inclusion of the `step` phase which was ignored for level terrain. The magnitude of the step is ten percent of the unsprung leg length so that $\Delta\hat{h} = 0.1$ in non-dimensional units.

We start the backward chaining sequence with the same initial phase boundaries as for the case with level terrain. The upper left plot in Figure 5.16 shows the discrete transition during the `step` phase. By construction the `step` phase occurs at the apex of flight where velocity is zero. The safe initial condition subboundary, shown in dark gray, range from a height of 0.1 to 1.4 for all timer values from 0 to 3. After the step, the height relative to ground drops 0.1 and the exit condition subboundary height ranges from 0 to 1.3 and timer values remain unchanged. Proceeding backwards in time, the plot in the upper right shows the `step`$\prec$ `ascent` transition. The `ascent` exit condition subboundary is shown in light gray and the intersection between `ascent` and `step` phase subboundaries is shown in dark gray. Note that relative to the flat terrain example, the `step` phase exit condition has moved to the right and this eliminates the portion of the `ascent` subboundary less than 0.1.

The middle and lower level plots in Figure 5.16 continue through the backward chaining sequence. The change to the `ascent` phase exit condition subboundary as a

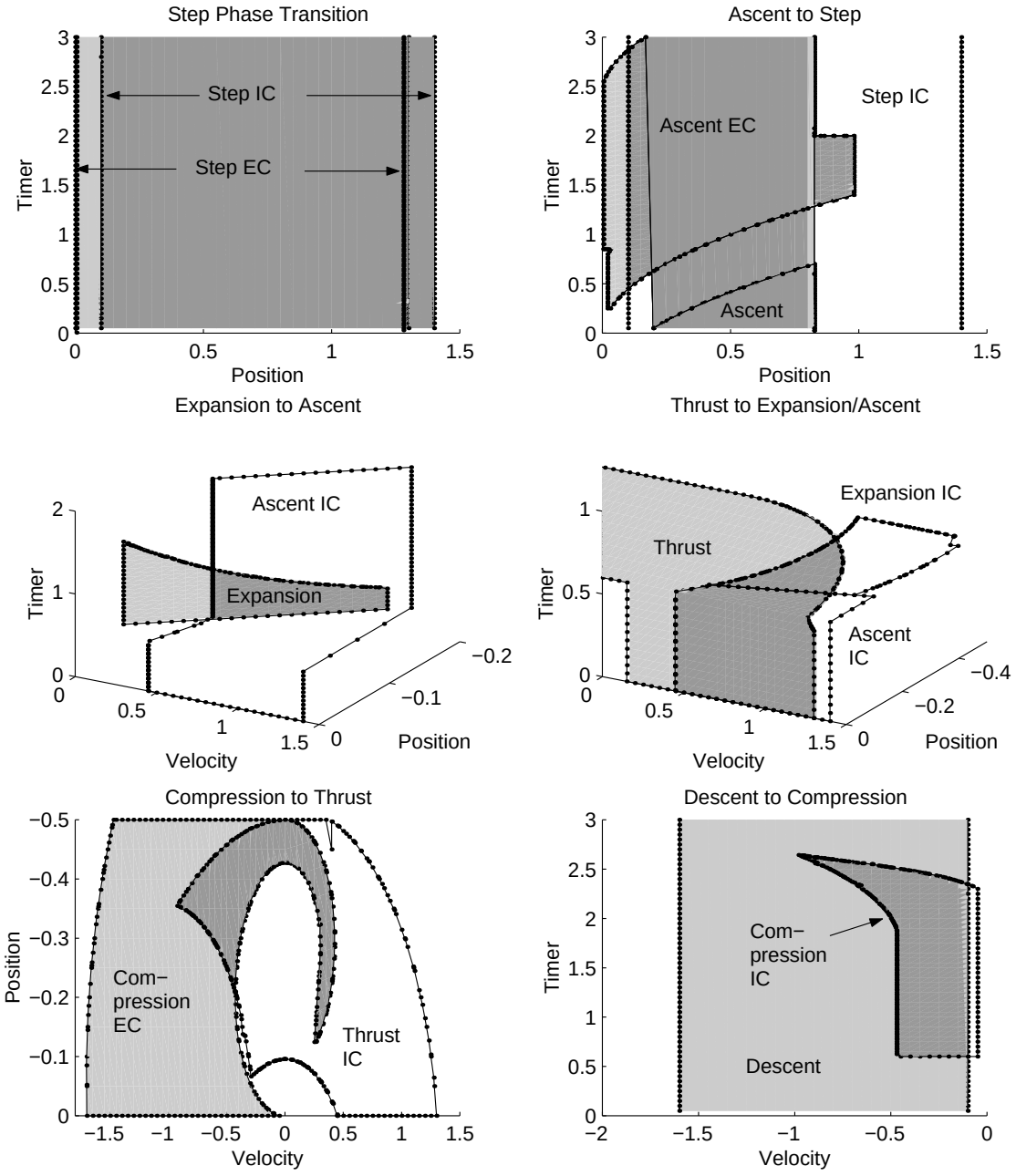


Figure 5.16: First iteration of the backward chaining sequence for stair climbing. The change in height ($\Delta\hat{h} = 0.1$) during the step is illustrated by the change in `step` phase initial and exit condition subboundaries in the upper left plot. The step reduces the size of the `ascent` phase exit condition subboundary relative to the flat terrain case, and this change propagates through the backward chaining sequence.

result of the step transition propagates through the sequence. In particular compare the bottom two plots in 5.16 with the middle plots in 5.11. Note that with the step, the intersections between `thrust` phase initial conditions and `compression` phase exit conditions and `compression` phase initial conditions and `descent` phase exit conditions are reduced indicating a reduced set of safe states relative to flat terrain.

Figure 5.17 shows the second iteration through the backward chaining sequence. Again the upper left plot shows the discrete change in height shifts the `step` phase exit condition subboundary below the initial condition subboundary so that the height above ground upon entering the `step` phase must be at least 0.1. The algorithm proceeds through the `step`$\prec$`ascent`, `ascent`$\prec$`expansion`, `expansion`$\prec$`thrust`, `thrust`$\prec$`compression`, and `compression`$\prec$`descent` phase boundary transitions. Note that the intersection between initial and exit subboundaries in the `thrust`$\prec$`compression` transition, shown in the lower left plot, generates two separate `compression` phase exit condition subboundaries which propagate to the `compression`$\prec$`descent` transition (lower right plot).

Figures 5.18 and 5.19 show the third and fourth iterations through the backward chaining sequence for the stair climbing problem. In Figure 5.18 the second subboundary generated during the `thrust`$\prec$`compression` phase tranisition in the second iteration does not intersect with the `ascent` phase exit conditions (upper right plot). The second boundary is basically a dead end and drops out of the sequence. Also in Figure 5.18 the `thrust` phase exit condition shows a jagged "sawtooth" border over a portion of the subboundary. This is an artifact due to some missing cells in the exit condition subboundary but does not affect overall results since this portion of the subboundary does not intersect with `expansion` or `ascent` phase initial conditions and is discarded.

In Figure 5.19 the plot scales have been adjusted for clarity. The sequence repeats through a 4th iteration. By the `compression`$\prec$`thrust` and `descent`$\prec$`compression` transitions (bottom two plots in figure) the `compression` and `descent` phase exit conditions are very similar between the third and fourth iterations and the algorithm reaches steady state.

Note that in the final iteration of the backward chaining sequence for the stair

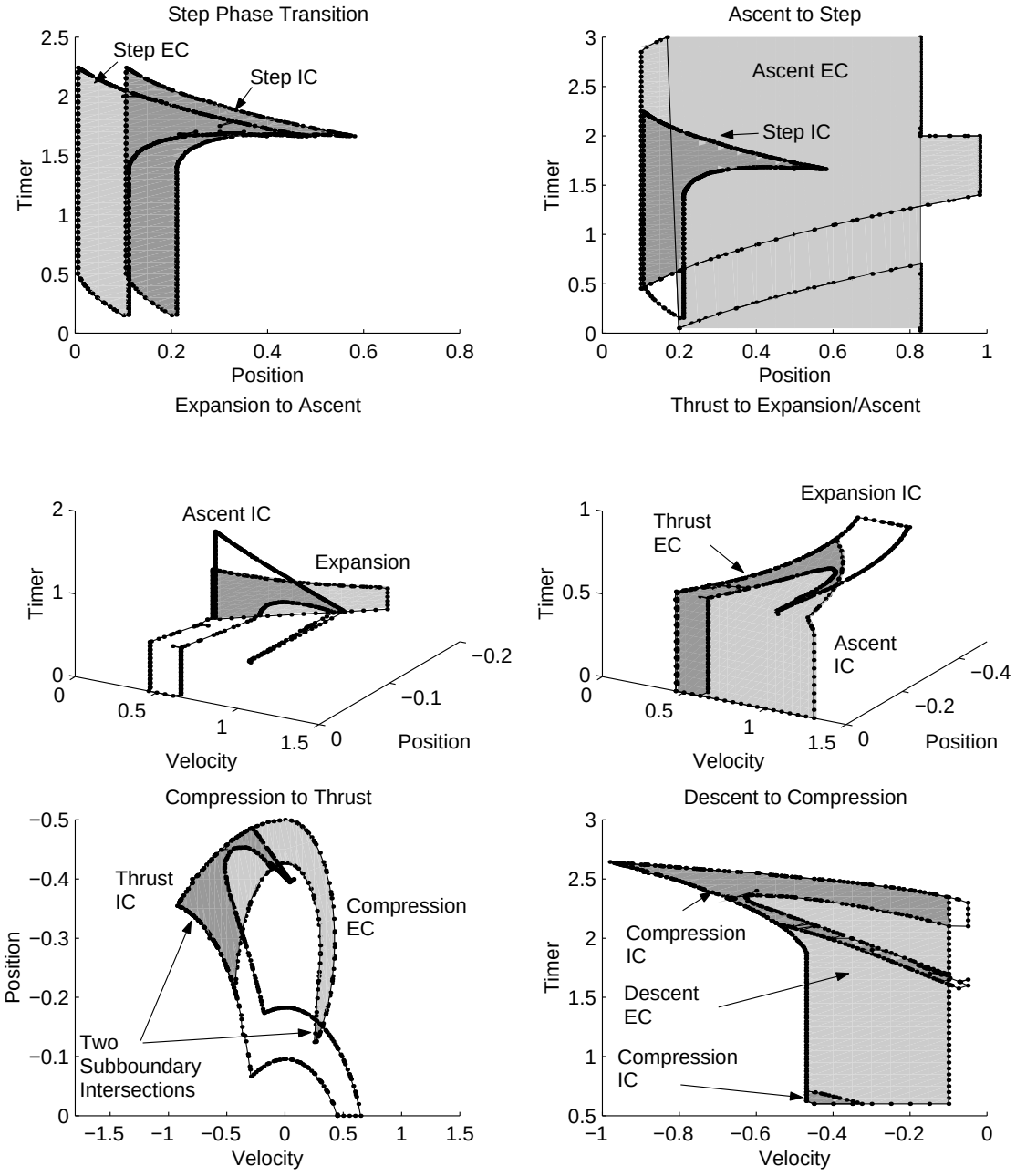


Figure 5.17: Second iteration of the backward chaining sequence for stair climbing. The upper left plot shows the $\Delta\hat{h} = 0.1$ change in height between the `step` phase initial and exit condition subboundaries. The backward chaining sequence proceeds as before. Note the `thrust`$\prec$`compression` transition (lower left plot) generates two separate `compression` phase exit condition subboundaries.

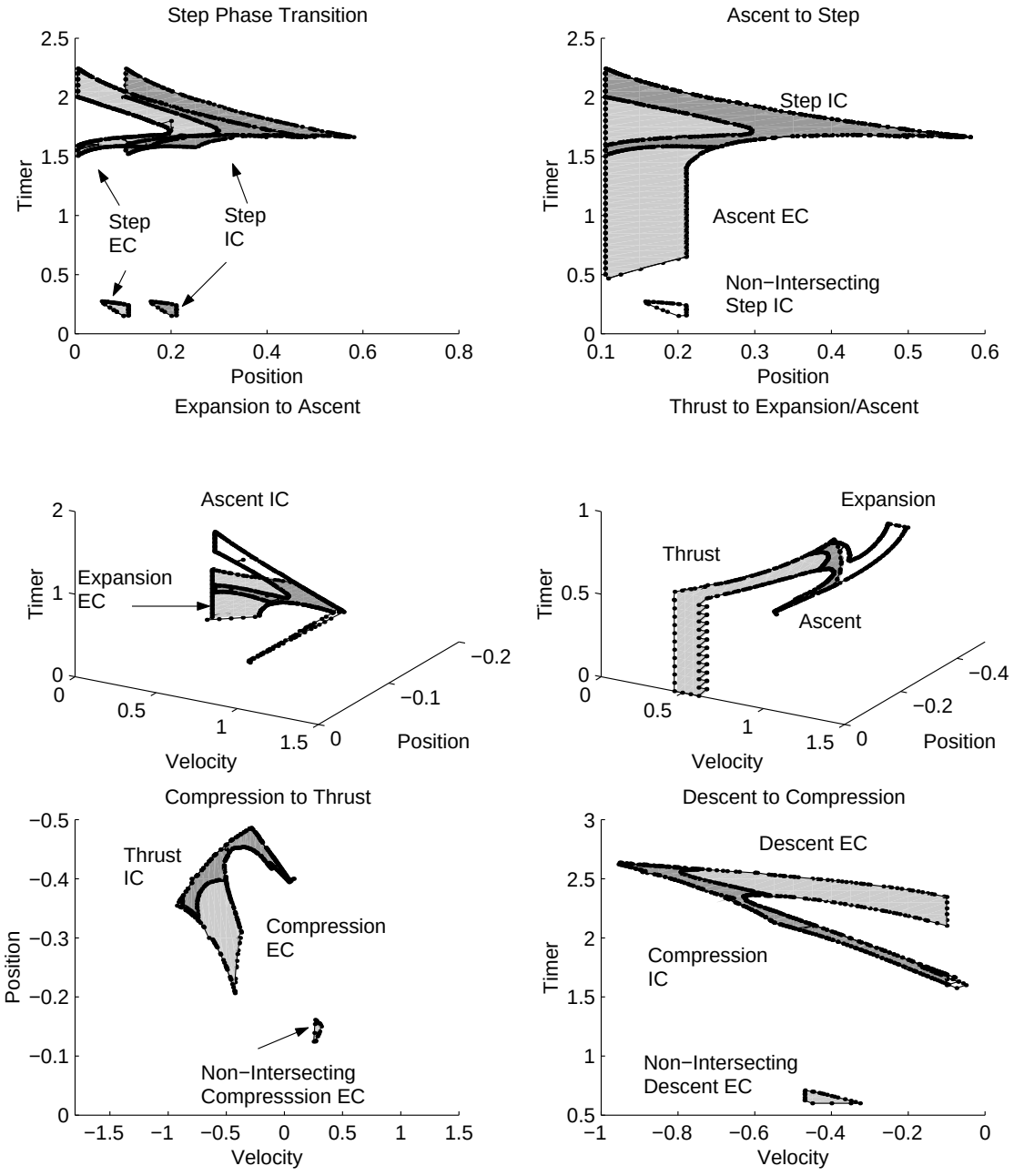


Figure 5.18: Third iteration of the backward chaining sequence for stair climbing. The second boundary created during the `compression`$\prec$`thrust` phase in the second iteration disappears because it does not intersect with `ascent` phase exit conditions (upper right plot).

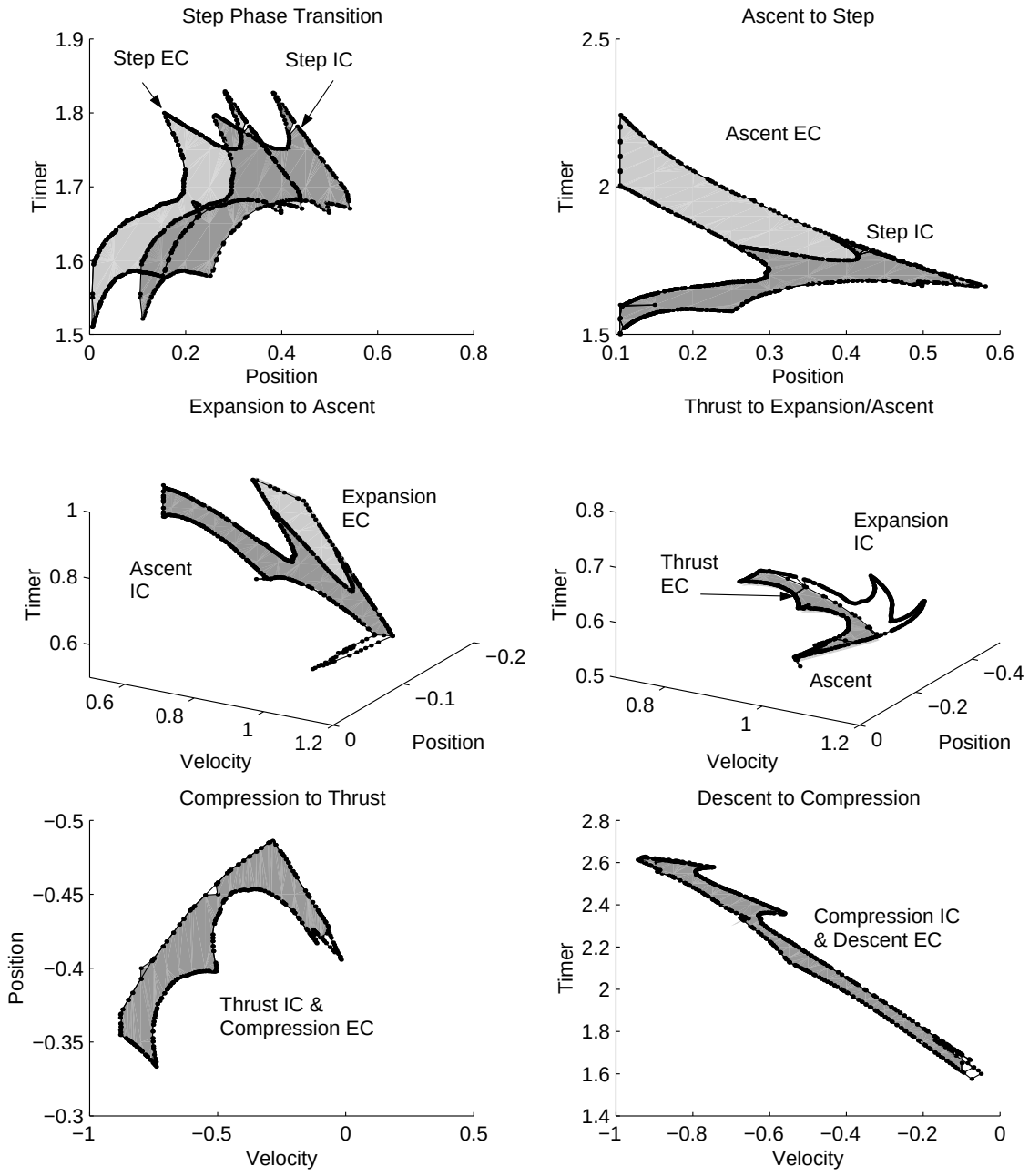


Figure 5.19: Fourth iteration of the backward chaining sequence for stair climbing. The `compression` and `descent` phase exit condition subboundaries (bottom two plots) closely match the exit condition subboundaries from the previous iteration and the algorithm finally reaches steady state.

climbing problem, each phase has only a single boundary in contrast to the results of the flat terrain problem. Exploration of the safe state boundary reveals a stable period two steady-state state trajectory and an unstable period one steady-state state trajectory. The period one and period two fixed points in the `step` phase exit condition subboundary are shown in Figure 5.20. For the period two solution, a hopper dropped from a height and with its timer set to values corresponding to point X will cycle through the phases of motion and cross over the first step at point Y. On the second hop, the hopper will again cycle through the phases of motion and cross over the second step at point X repeating the sequence. The steady-state state trajectory and corresponding fixed points are stable because trajectories originating within the initial condition subboundary will migrate over several iterations towards the fixed points. For the period one solution, a state trajectory starting from point Z will return to point Z after one hopping cycle, but any small perturbation will migrate the trajectory to the period two solution.

5.5 Hopping over variable terrain

In this section we consider the problem of variable terrain where the height between hops varies unpredictably but within some bound. Again, we assume the same robot hopper as used in the previous two sections. The step height variation between hops is +/- five percent of the unsprung leg length so that $-0.05 \leq \Delta\hat{h} \leq 0.05$. In forward time the transition from `step` phase initial to exit conditions is a change in height that can vary by +/- 0.05. In backwards time, as discussed in Chapter 4 and illustrated in Figure 3.5, the initial condition subboundary is determined from the intersection of the inverse transformed exit conditon subboundaries.

Figure 5.21 shows the first iteration of the backward chaining sequence for the variable terrain problem. The plot at the upper left shows the `step` phase transition. The exit condition subboundary (light gray) spans from a height above ground of 0 to 1.3. The initial condition subboundary (dark gray) spans from a height above ground of 0.05 to 1.25. Note that for both subboundaries the timer state variables wraps

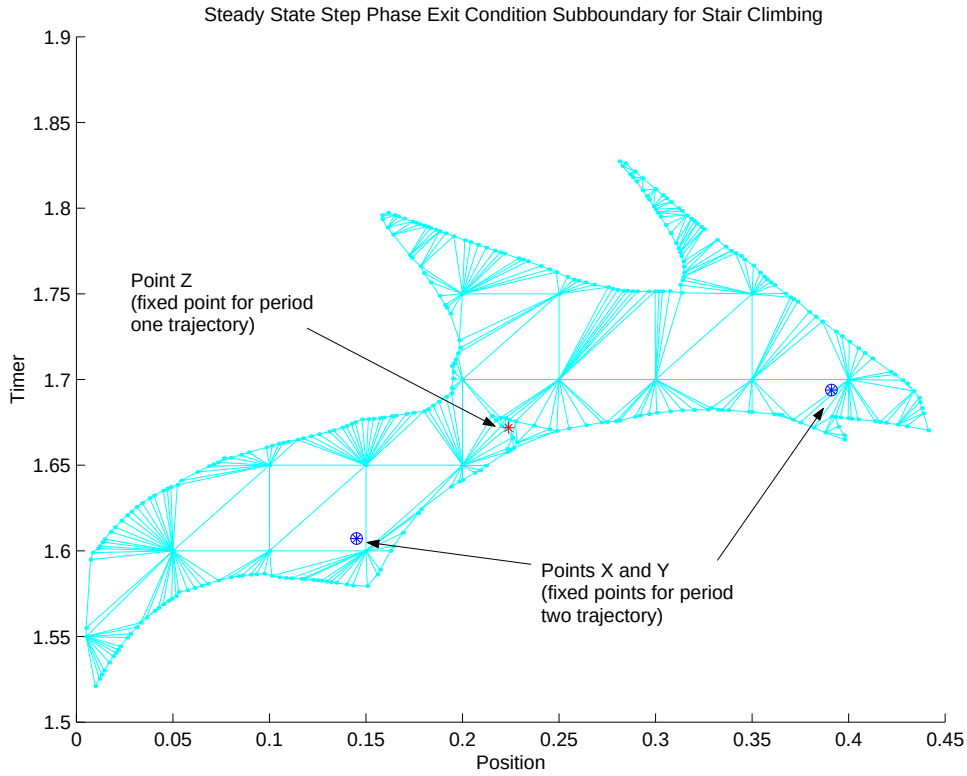


Figure 5.20: Steady state `step` phase exit condition subboundary for stair climbing. State space trajectories originating within this subboundary will remain safe and return to points within this boundary. The hopper will eventually settle into a steady-state cycle that alternates between points X and Y at the `step` phase exit condition (period two solution). Trajectories originating at point Z will return to point Z but small perturbations or errors will drive the hopper to the period two solution.

from 3 to 0 forming a continuous band which is not shown in the figure. The backward chaining sequence progresses through the `step`$\prec$`ascent`, `ascent`$\prec$`expansion`, `expansion`$\prec$`thrust`, `thrust`$\prec$`compression`, `compression`$\prec$`descent`, and `descent`$\prec$`step` phase transitions as in the previous examples.

Figure 5.22 shows the second iteration through the backwards chaining sequence. Here the plot in the upper left more clearly shows the `step` phase exit and initial condition subboundaries. The initial conditions are the set of states which will remain within the exit condition subboundary for any change in terrain height from $-0.05 \leq \Delta\hat{h} \leq 0.05$. The set of initial condition states which are guaranteed safe is therefore smaller than the set of exit condition states. For example, states with timer values greater than about 2.2 or less than 0.5 are eliminated from the initial condition subboundary. The backwards chaining sequence proceeds as in the previous examples, but with smaller and more distorted boundaries.

The third iteration through the backward chaining sequence with variable terrain does not complete. Figure 5.23 shows the third iteration up to the `expansion`$\prec$`thrust` phase transition. The lower right hand plot shows that the ascent and expansion phase initial conditions from the third iteration do not intersect the `thrust` phase exit conditions from the previous iteration thus the backwards chaining sequence ends. The maximum invariant safe set for the timed hopper on variable terrain is null and there is no steady state solution.

The results imply that a sequence of terrain variations within the range $-0.05 \leq \Delta\hat{h} \leq 0.05$ can be found to force the hopper to an unsafe condition within three hops. These results are radically different than the results presented in Chapter 2, Figure 2.10, which show a hopper with the same mechanical properties (max thrust = 2, spring stiffness = 4, damping ratio = 0.11) can tolerate a terrain variation of up to 60 percent of the unsprung leg length with perfect state feedback. These results also contrast with the results for stair climbing presented in the previous section. The difference is that the open loop system can be easily forced to unsafe regions of state space by modulating the terrain height up and down so that a fixed duty cycle either fires too early or too late to provide sufficient thrust to sustain hopping.

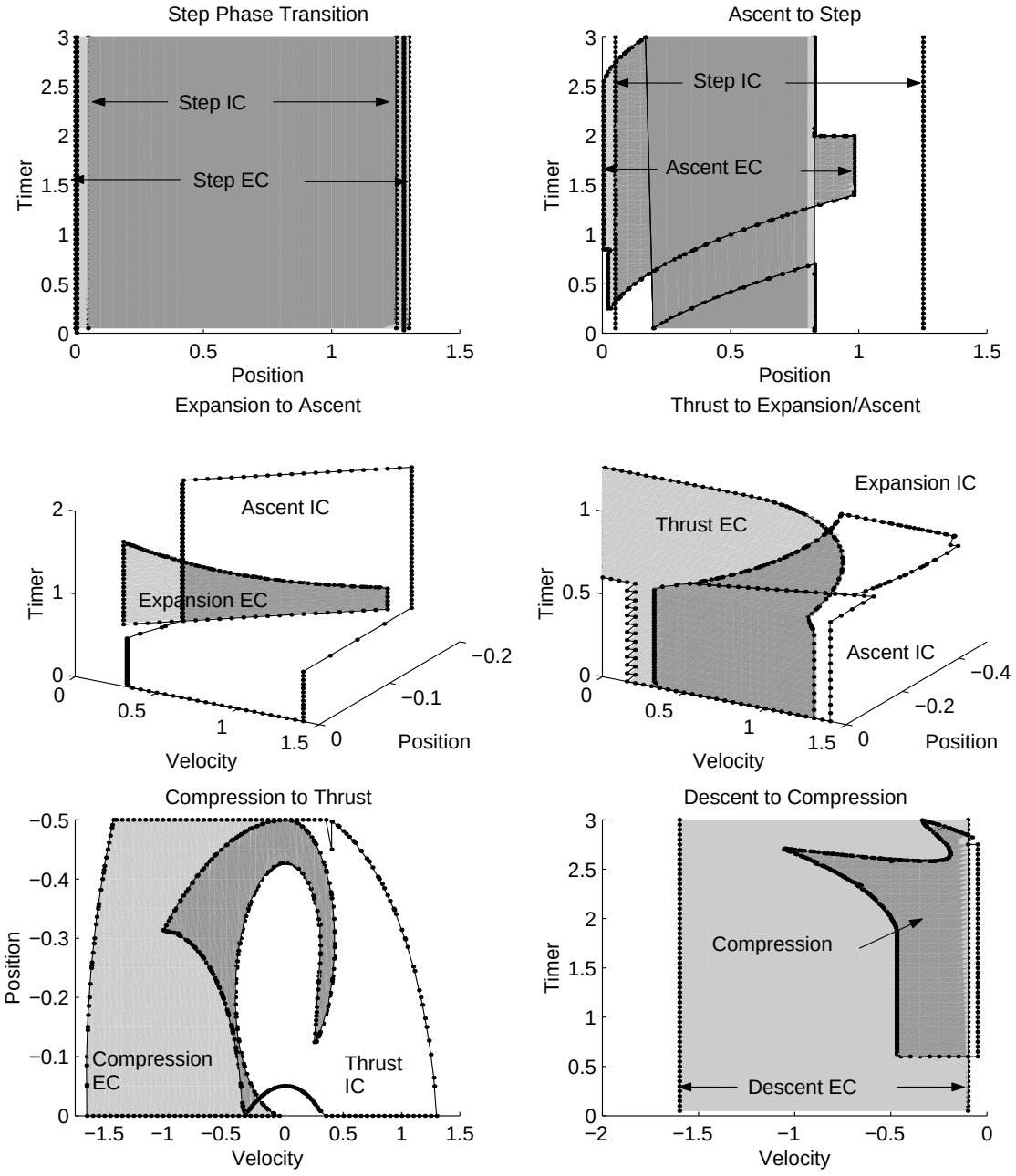


Figure 5.21: First iteration of the backward chaining sequence for variable terrain. In the upper left plot `step` phase exit conditions (light gray) range from an above ground height of 0 to 1.3 and `step` phase initial conditions (dark gray) range from 0.05 to 1.25. Subsequent transitions through the backward chaining sequence are similar to previous examples.

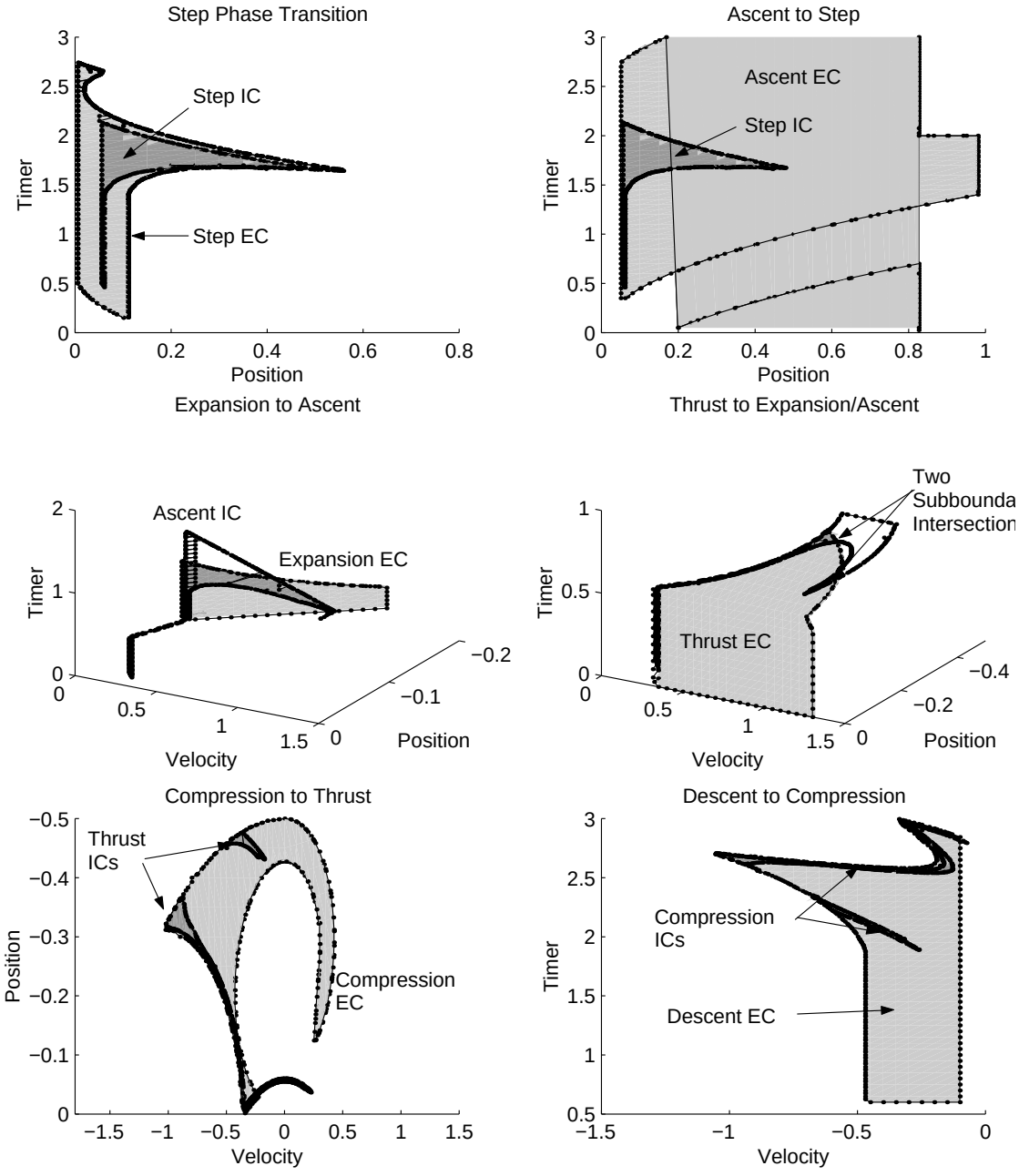


Figure 5.22: Second iteration of the backward chaining sequence for variable terrain. The upper left plot clearly shows the `step` phase exit to initial condition transition in backwards time and the consequent contraction of the initial condition subboundary. Subsequent transitions through the backward chaining sequence are similar to previous examples.

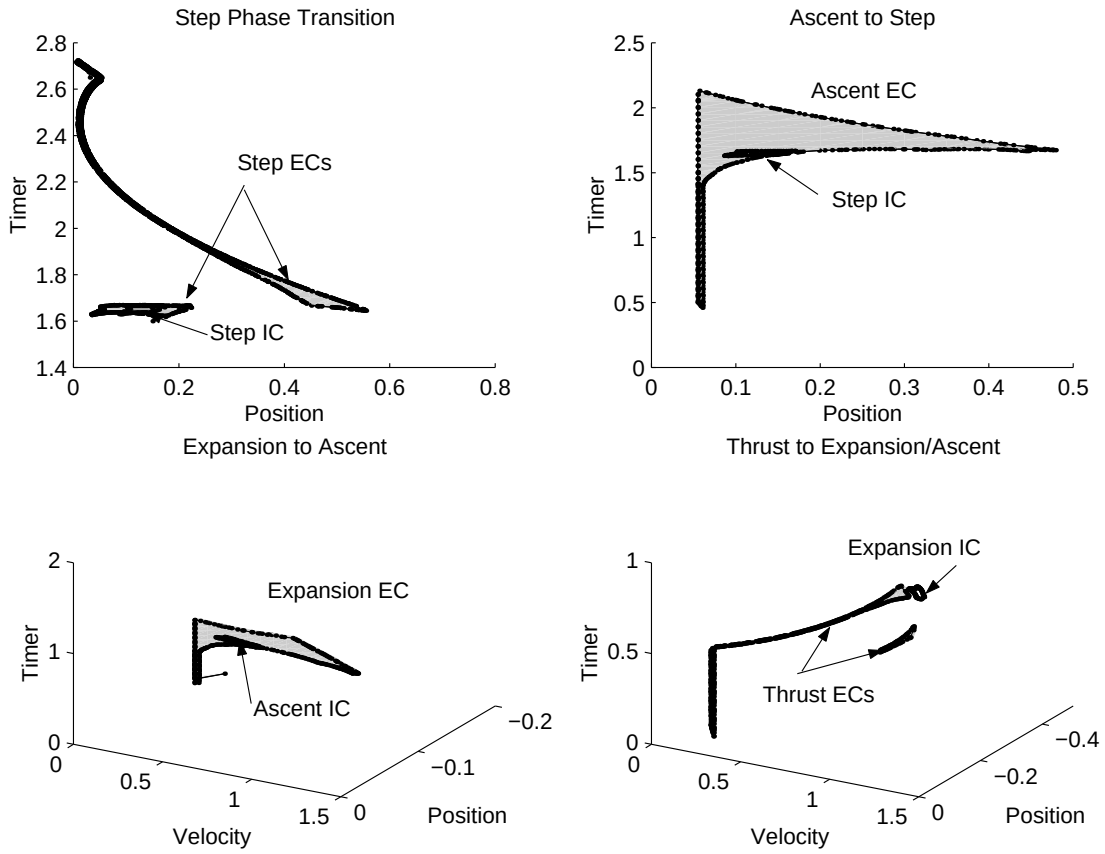


Figure 5.23: Third iteration of the backward chaining sequence for variable terrain. The backward chaining sequence ends at the `expansion`←`thrust` phase transition because the `ascent` and `expansion` phase initial conditions do not intersect the `thrust` phase exit conditions. The maximal invariant safe subset in this example is null.

5.6 Numerical Difficulties

Numerical difficulties leading to errant points and missing cells were pointed out in Figures 5.12 and 5.18. In fact, close examination of figures 5.12 through 5.23 will reveal some small errors in almost every figure. The errors often do not have a significant effect on the final answer, but the fact that they exist is problematic.

There are several principle causes for these errors. One has been mentioned previously. The boundary representation is a finite approximation. In the intersection between two curved surfaces, for example, the straight line connection between cell points in one boundary may not intersect with cell surfaces in the other boundary, resulting in 'missing' intersection points. This problem can be controlled to a large extent by limiting the maximum cell diameter and by allowing a tolerance for computing intersection points. However, too small a cell diameter increases memory and computational needs, and too large a tolerance on the intersection will generate spurious points and cells. The difficulty is aggravated by the lack of derivative information along directions orthogonal to $f(x, u, d)$. For example, the generating borders along the exit condition subboundaries are often non-smooth and non-convex. Therefore, it is difficult to determine a reasonable density of generating points that will maintain an accurate representation of the boundary surface throughout the time propagation. A possible approach is to automatically identify boundary features and add or eliminate vertex points to optimize the boundary representation. Although feature identification is well established in two and three dimensions (see for example Henderson, et. al., 1994, [21]) it is not well studied in higher dimensions. Feature extraction and optimization are an active area of research in imaging and solid modeling (see for example Tohka, 2003 [45]).

Other sources of error can be attributed to programming implementations intended to optimize computational speed. As an example, the algorithm for `InsideBoundary` given in section 4.5.5 was implemented in Matlab® script. Matlab provides convenient functions for handling large vectors and matrices including orthogonal triangularization. The `InsideBoundary`algorithm determines the number of times a vector

projecting from a point intersects the boundary surface and the orthogonal transformation greatly speeds up these computations. However, if the vector is tangent to (or contained within the subspace of) a boundary cell, then the system of equations is degenerate. The algorithm handles this problem by selecting another vector direction at random and repeating the intersection calculations. To prevent a continuous loop, the algorithm aborts if a non-degenerate solution cannot be found after numerous tries and assumes the candidate point is outside the boundary. Very rarely a point that is in fact inside the boundary is erroneously considered to be outside the boundary. Unfortunately, given the thousands of points that are considered during a resect operation this error occurs often enough to cause problems. One approach is to eliminate the degenerate cells from the simultaneous solutions since tangency does not change whether a point is inside or outside of a boundary. However, implementing this fix was considered difficult and instead problems caused by whether a point was incorrectly determined to be outside of a boundary (as well as problems due to missing intersections described above) have been fixed by manually editing output data files. This has proved to be a slow and cumbersome solution!

Errors resulting from finite approximation and computational expediency become even more significant in higher dimensional spaces. This is the primary reason why the work in this chapter is restricted to $\mathbb{R}^3$. Several, somewhat ad hoc, 'garbage collector' type routines have been created to identify and fix errors but even these measures are not 100 per cent successful. Improving and correcting algorithms are an object of future work. Fortunately, there is reason to be optimistic. Popovic and Hoppe, 1997, [37], show an approach for progressive simplicial complexes that permits variable levels of detail through a process of transformations. Silva, 2003, [17], provides a Matlab library for determining topological invariants of simplicial complexes that can be used to detect errors and optimize the boundary. A more disciplined approach to the problems of intersection, resection, and boundary optimization will limit and prevent the numerical difficulties discussed.

5.7 Chapter Summary

This chapter presented a hybrid model for a purely open loop hopper that includes an additional timer variable controlling thrust and the appropriate phase transitions. The timer increases at a constant rate, but rolls over at a maximum value forcing periodic behavior. The rollover is implemented in a cylindrical state space by mapping timer values to an angle from 0 to 360 degrees. The hybrid model does not include continuous control or disturbance inputs. Rather, the control is a discrete event conditioned on the timer value. The lack of continuous control or disturbance inputs obviates the need for singularity avoidance or a `liftoff` transition phase.

The chapter described the development of the initial phase boundaries for the MISS algorithm in some detail. The procedure included: establishment of initial and exit condition subboundaries, generation of a trajectory subboundary by time propagation, merging the trajectory subboundary with the initial and exit conditions, identification of any constraint conditions and time propagation from the constraint boundary generator, and finally resection of constraints from the merged boundary. Singularity avoidance for this problem was not required.

After establishment of the initial phase boundaries, backward chaining over different terrain conditions was examined. Over flat terrain there are no discrete disturbances between hops. For stair climbing the terrain changes by a fixed amount between steps. Selected step height is ten percent of the unsprung leg length. Over variable terrain the height between steps changes over a range from plus to minus five percent of the unsprung leg length.

On flat terrain the timed hopper can achieve stable hopping. The hopping cycle and hopper mechanical characteristics (ie body mass and leg stiffness, damping, and thrust) for this example are such that the steady state cycle has a periodicity of two with a high hop followed by a short hop and repeating. This limped gait result has been reported in previous work (Cham 2002, [14] and Koditchek and Buehler, 1991, [27]). Furthermore, the hopper can safely negotiate constantly ascending stairs. However, the MISS algorithm shows that for variable terrain the hopper can always be forced to an unsafe condition within 3 hops. Thus, despite the self stabilizing behavior

identified by Ringrose (1997 [39]) relatively small variations in the environment can force the open loop hopper out of a safe or steady operating regime.

The results for variable terrain contrast with the demonstrated terrain crossing ability of clock driven multi–legged robots, such as *RHex* (Saranli, et. al., 2000, [40]) and *Sprawlita* (Cham, et. al., 2000, [11]). One explanation for this disparity is the additional stability obtained from an alternating tripod rather than a bouncing gait. A more fundamental reason, however, is that the safety criteria for a multi–legged and single legged robots are criteria are quite different. Although any robot will have joint limits that must be respected, a multi–legged robot will not necessarily fail if a leg stumbles while clearing an obstacle. On the other hand, running robots, including multi-legged robots, are constrained by friction limits. Issues concerning planar running and multi-legged robots are discussed in more detail in Chapter 6.

Finally, this chapter discusses several sources of error that have required time intensive manual fixes and thus restricted the present work with the MISS algorithm to three dimensions. Some of these errors are the result of a finite representation but could be eliminated with algorithms that can detect errors and refine the boundary representation to sufficient detail. Some of these errors are the result of programming expediencies and can be eliminated with corrections and improvements to the software. Despite some of the difficulties encountered with the algorithms in a boundary representation, there are significant computational advantages over a grid-based representation used in previous work (Tomlin, et. al., 2000, [48]).

Chapter 6

Extension to Planar and Multi–legged Running

This chapter describes application of the MISS algorithm to more complex problems. Because of their high dimensionality, detailed study of these problems is beyond the scope of the current work. However, it is important to show that the MISS algorithm can be applied to more practical and complex problems. We start by considering several variations to the single legged hopper problem, then single and multi–legged planar running.

6.1 Extensions of the open loop hopping problem

6.1.1 Variable hopper parameters and duty cycle

A principal weakness of the analysis of the open loop hopper in Chapter 5 is that the parameters defining the hopper dynamics, that is the leg stiffness, damping, thrust, and hopper period, were all preselected. The analysis was structured to answer the question of whether a particular hopper design can operate safely in the specified environment. However, it would be useful to know, for example, whether variation of a design parameter, such as hopping cycle period or thruster duty cycle, affects the ruggedness of the hopper. A brute force method, which was applied in Chapter 2,

is to simply repeat the analysis over a range of parameter values. However, another possibility is to increase the dimensionality of the problem: the parameter of interest is augmented to the state vector. The advantage to this approach is it allows for actively controlling these parameters. The change permits investigation of adaptive tuning approaches such as those proposed by Cham (2002, [14]) and Bailey (2004, [1]). The disadvantage is that the higher dimensionality complicates the algorithm and increases the computational burden.

Consider the open loop hopper problem with fixed thruster duty cycle presented in Chapter 5. The phases of motion and the transition between phases is shown previously in Figure 5.1. The equations of motion for the system are repeated below where $\hat{y}$ is the normalized height above ground, and $\hat{s}$ is the normalized timer variable, and τ_{Per} is the timer period in units of normalized time. The prime symbol ($'$) indicates differentiation with respect to normalized time, τ.

$$\hat{y}'' = \begin{cases} -1 & \texttt{ascent}, \texttt{descent} \\ -\hat{k}\hat{y} - \hat{b}\hat{y}' - 1 & \texttt{compression}, \texttt{contact} \\ -\hat{k}\hat{y} - \hat{b}\hat{y}' - 1 + u_{max} & \texttt{thrust} \end{cases} \tag{6.1}$$

$$\hat{s}' = 1 \quad \text{for } 0 \leq \hat{s} < \tau_{Per} \quad \textit{(all phases)} \tag{6.2}$$

$$\hat{s} \leftarrow 0 \quad \text{for } \hat{s} = \tau_{Per} \quad \textit{(all phases)} \tag{6.3}$$

To consider the effect of varying the duty cycle, we augment the continuous state vector, $x = [\hat{y}, \hat{y}', s, \tau_{Per}, \tau_{Off}]^T$ where τ_{Per} is a parameter defining the clock timer period and τ_{Off} is a parameter defining the thrust duration from time $\tau = 0$. An important change to the problem formulation presented in Chapter 5 is that the reset of the timer variable is no longer implemented in cylindrical space but instead requires a phase transition (see section 5.1).

Figure 6.1 considers two alternatives for incorporation of a reset phase transition. One approach is to create an additional mode, `reset`, which resets the timer state variable s back to 0 whenever $s = \tau_{Per}$. The `reset` is a discrete or instantaneous phase and returns immediately back to `ascent` or `descent` or transitions from `compression`

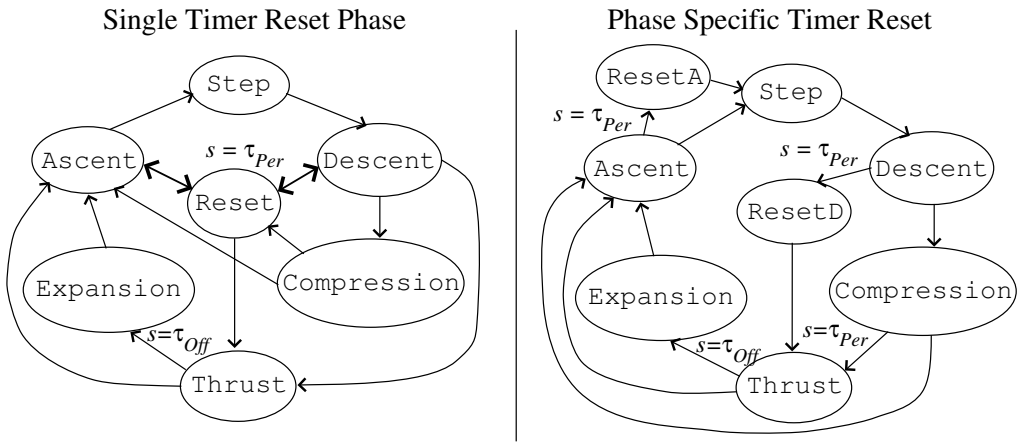


Figure 6.1: Alternative phase transitions for the timed hopper with variable period, τ_{Per}. A reset can occur during `ascent`, `descent`, or `compression` phases. To the left, the `reset` phase sets the timer state to zero and returns to `ascent` or `descentor` transitions from `compression` to `thrust`. On the right, the timer resets on transition to separate phases `resetA`, `resetD`, and `thrust` eliminating internal loops in the backward chaining sequence.

to `thrust`. Note that the values for τ_{Per} and τ_{Off} are assumed to be such that the timer will not expire during `expansion` and that reset cannot occur during `step` which is also a discrete phase.

The problem with the single `reset` phase approach is that `ascent` and `descent` are both predecessor and successor phases to `reset`. This creates a couple of inner loops, indicated in the figure by heavy double arrow lines, that complicate the backward chaining sequence. Instead a preferred approach, shown on the right side of the figure, is to add a couple of continuous phases, `resetA` and `resetD`. These are continuations of the `ascent` and `descent` phases, but reset the timer variable upon transition. Similarly, the timer is reset upon transition from `compression` to `thrust`. Note that this approach assumes that τ_{Per} is long enough that a reset could not occur more than once during either `ascent` or `descent`.

Adopting the phase specific timer reset approach on the right side of Figure 6.1, the equations of motion for the open loop hopper with a parameterized period, τ_{Per} and thruster duration τ_{Off} are shown below, where the left hand arrow, $\leftarrow$, indicates

a reset of the timer state variable, and '$\succ$' indicates a phase transition in forward time.

$$\hat{y}'' = \begin{cases} -1 & \texttt{ascent}, \texttt{descent}, \texttt{resetA}, \texttt{resetD} \\ -\hat{k}\hat{y} - \hat{b}\hat{y}' - 1 & \texttt{compression}, \texttt{contact} \\ -\hat{k}\hat{y} - \hat{b}\hat{y}' - 1 + u_{max} & \texttt{thrust} \end{cases} \tag{6.4}$$

$$\hat{s}' = 1 \tag{6.5}$$

$$\tau'_{Per} = 0 \tag{6.6}$$

$$\tau'_{Off} = 0 \tag{6.7}$$

$$\hat{s} \leftarrow 0 \text{ when } \begin{cases} \texttt{ascent} \succ \texttt{resetA} \\ \texttt{descent} \succ \texttt{resetD} \\ \texttt{compression} \succ \texttt{thrust} \end{cases} \tag{6.8}$$

This formulation can be used to investigate the effects of varying hopper period and thruster duty cycle by considering a range of values: $\tau_{Per}^{min} \leq \tau_{Per} \leq \tau_{Per}^{max}$ and $\tau_{Off}^{min} \leq \tau_{Off} \leq \tau_{Off}^{max}$ provided $\tau_{Off}^{max} < \tau_{Per}^{min} - \tau_{wait}$. In other words, the minimum hopper period is greater than the maximum thruster duration by some value, $\tau_{wait} > 0$, so that thrusting doesn't occur twice during the same contact.

Adaptive control is implemented by adjusting τ_{Per} and τ_{Off} upon transition to `resetA` or `resetD` or `thrust`, as shown in equations 6.9 and 6.10 below. The $(\cdot)^-$ and $(\cdot)^+$ superscripts in the equations indicate a change in parameter value immediately before and after the phase transition. Note that equations 6.9 and 6.10 do not prescribe any algorithm for determining τ_{Per}^+ or τ_{Off}^+ and only restrict the range of legal values.

$$\tau_{Per}^- \leftarrow \tau_{Per}^+ \text{ where } \{\tau_{Per}^+ : \tau_{Per}^{min} \leq \tau_{Per}^+ \leq \tau_{Per}^{max}\} \text{ for } \begin{cases} \texttt{ascent} \succ \texttt{resetA} \\ \texttt{descent} \succ \texttt{resetD} \\ \texttt{compression} \succ \texttt{thrust} \end{cases} \tag{6.9}$$

$$\tau_{On}^{-} \leftarrow \tau_{Off}^{+} \text{ where } \{\tau_{Off}^{+} : \tau_{Off}^{min} \leq \tau_{Off}^{+} \leq \tau_{Off}^{max}\} \text{for } \begin{cases} \texttt{ascent} \succ \texttt{resetA} \\ \texttt{descent} \succ \texttt{resetD} \\ \texttt{compression} \succ \texttt{thrust} \end{cases} \quad (6.10)$$

Figure 6.2 shows a hypothetical backwards time transition for τ_{Per}. In the figure, the boundary of safe states just after the mode transition is shown by the shaded region: $\tau_{Per}^{min} \leq \tau_{Per}^{-} \leq \tau_{Per}^{max}$, and position and velocity vary with τ_{Per}^{-}. Thruster duration, τ_{Off}, and the timer state variable, s, which transitions from 0 to τ_{Per}^{+} in backwards time are not shown in the example. If the problem under consideration assumes the hopping period remains fixed, then the boundary region is unchanged through transition. If, however, the problem permits τ_{Per} to change, then for any safe pairing of position and velocity the range for τ_{Per}^{+} is τ_{Per}^{min} to τ_{Per}^{max}. Through the backwards time transition, the shape of the boundary changes to a prismatic projection over the range of τ_{Per}. The projection maximizes the possible safe set and the results can be used to synthesize a control logic.

6.1.2 Sensing the environment

One important way to overcome environmental variations and improve the ruggedness of a system is to measure the environment. For example, for the closed loop vertical hopping robot in Chapter 2 a measurement of the change in height at the upcoming step could be useful. However, the quality and timing of the measurement affects its utility.

In the context of the hybrid automaton of Definition 1, an environmental measurement becomes a constraint on the discrete disturbance input. This constraint can be formulated different ways depending on the nature of the measurement. For example, suppose that a vertical hopping robot has a sensor that accurately measures the change in terrain height at the next hop and that the measurement is made at the initiation of contact. The situation is illustrated in Figure 6.3. If we augment the state vector with the measurement value, $\Delta\hat{h}$, the hopper can use this value to determine thrust levels during `contact`. After the hopper leaves contact and during the subsequent `step` phase, the height above ground changes by minus the measured

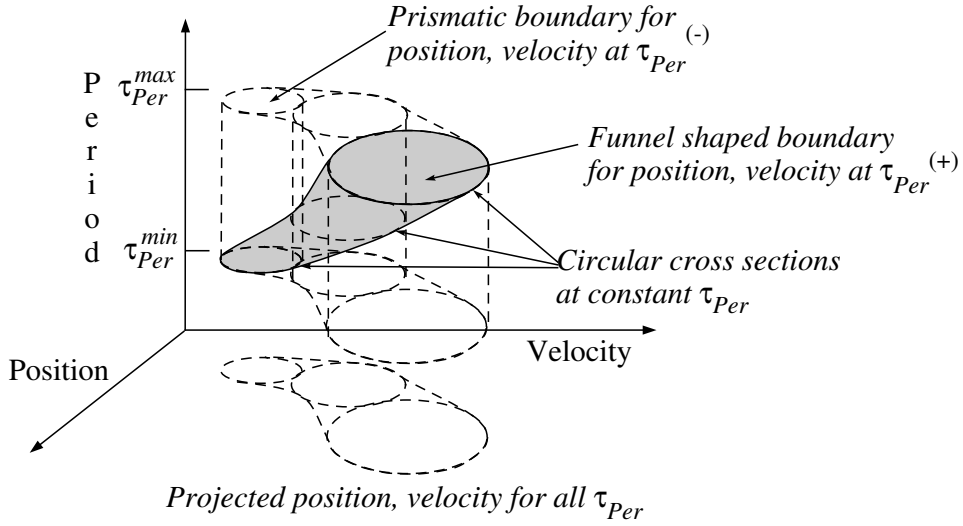


Figure 6.2: Example backward time transition for variable timer period. The shaded region shows allowable position and velocity changes as a function of the hopper period, τ_{Per}. Assuming the ability to change timer period, the backward time transition is to any value $\tau_{Per}^{+} \leftarrow \tau_{Per}^{-} : \{\tau_{Per}^{min} \leq \tau_{Per}^{-} \leq \tau_{Per}^{max}\}$. The volume of the safe state region changes to the prismatic projection outlined by broken lines.

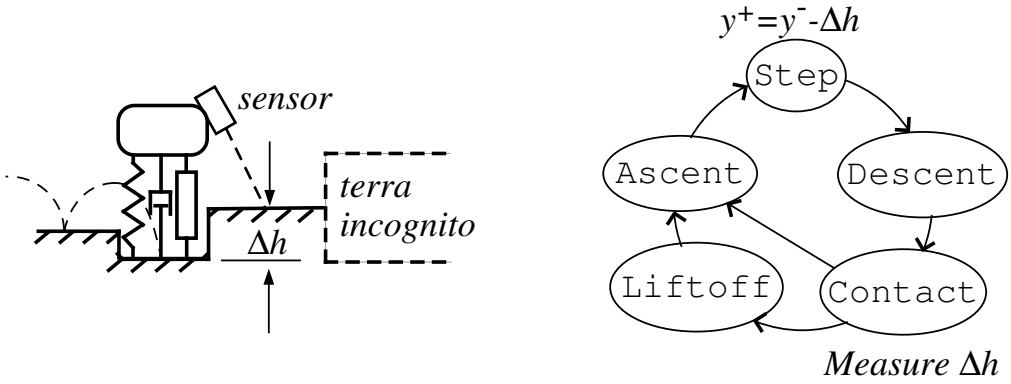


Figure 6.3: Measuring the environment. The change in terrain height, $\Delta\hat{h}$, is captured at the initiation of the `contact` phase. This knowledge is available for control during `contact` and constrains the discrete transition at the subsequent `step` phase.

value, -$\Delta\hat{h}$. Upon initiation of the subsequent `contact` phase a new measurement is taken and the process repeats.

For simplicity suppose that the change in terrain height, $\Delta\hat{h}$, can be only one of two possible values, $\Delta\hat{h} \in \{\Delta\hat{h}_1, \Delta\hat{h}_2\}$. Hopper parameters, including values for $\Delta\hat{h}_1$ and $\Delta\hat{h}_2$, are shown in the Table 6.1. $\Delta\hat{h}_1$ has a large negative value which indicates a large step down. $\Delta\hat{h}_2$ has a smaller positive value which indicates a smaller step up. Asymmetric limits like these maybe encountered, for example, if the hopper is traveling down an irregular but generally descending hillside.

Figure 6.4 compares the safe position and velocity states for $\Delta\hat{h}_1$ and $\Delta\hat{h}_2$. The figure highlights the maximum compression constraint at $[-0.5\ 0]$ with a star. The extremal trajectory passing through the compression constraint is a heavy solid curve. The singularity avoidance region is indicated by the curve with a broken curve between the point at [0 0] and the 'Q point' (the point of maximum compression starting contact at rest and with gravity only) marked by a hexagram. The border between `contact` and `liftoff` phases is shown by the light colored broken line. For $\Delta\hat{h} = -2$, shown at the top of the figure, the hopper must limit thrust to prevent landing too hard and violating the maximum compression constraint. For $\Delta\hat{h} = 1$, the hopper must maintain a minimum hopping height to avoid eventually colliding with the step

Parameter	*Value*
stiffness, $\hat{k}$	10.0
damping ratio, ζ	0.1
thrust, $\hat{u}_{max}$	4.0
min change in terrain, $\Delta\hat{h}_1$	-2.0
max change in terrain, $\Delta\hat{h}_2$	1.0

Table 6.1: Physical parameters and environmental ranges for the hopper measurement problem.

in subsequent hops. The minimum safe hopping height for $\Delta\hat{h} = 1$ is greater than the maximum safe hopping height when $\Delta\hat{h} = -2$ so measurement is required for a feasible solution.

The example can be extended to a continuous range for $\Delta\hat{h}$ by adding the measurement to the continuous state vector. The problem can be further extended to accommodate measurement error by allowing a difference between actual and measured terrain variation. However, it is worth making a couple observations. First, although measurement of the environment is never detrimental, it is not always beneficial. For example, if $\Delta\hat{h}_2$ were slightly larger ($\Delta\hat{h}_2 = 1.2$) then the hopper would have insufficient thrust to sustain stair climbing and would ultimately fail, measurement or not. Alternatively, if the maximum step down was smaller ($\Delta\hat{h}_1 = -1$) then a feasible solution exists without need of an environmental sensor.

A second observation is that due to the time delay between measurement and application, the game theoretic nature of the problem has changed. It is difficult to determine an optimal or 'worst case' strategy for the environmental disturbance because the 'payoff' with respect to the J_E optimization in Chapter 3 is unknown until the hopper uses the measurement information. Determination of an optimal strategy requires projection in forward time based on possible control responses. However, if we assume the measurement value takes on an arbitrary value (ranging from $\Delta\hat{h}_1 \leq \Delta\hat{h} \leq \Delta\hat{h}_2$) rather than a worst case value, then the MISS algorithm can be used unmodified. The discrete transition for the measurement variable is similar to the prismatic projection shown in Figure 6.2 with the replacement of $\Delta\hat{h}$ for τ_{Per}.

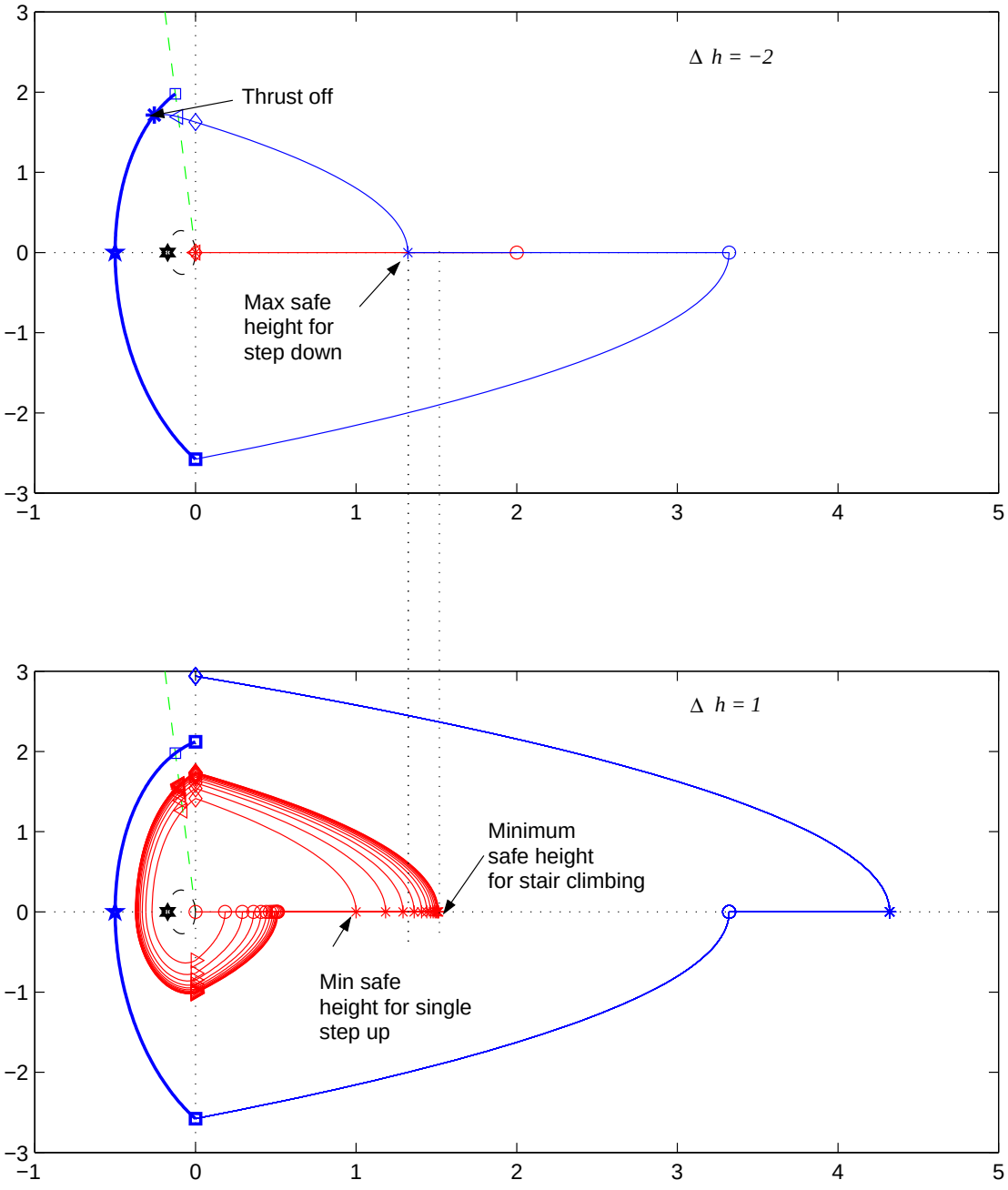


Figure 6.4: Phase plane plots for hopper with perfect state feedback and environmental sensor. The top figure shows safe position and velocity when the subsequent step is $\Delta\hat{h} = -2$. Note that the hopper must limit thrust to avoid exceeding the compression constraint on the subsequent step. The bottom figure shows safe position and velocity when the subsequent step is $\Delta\hat{h} = 1$. The hopper must maintain a minimum height above ground to avoid collision in subsequent steps. The minimum safe height for $\Delta\hat{h} = 1$ is greater than the maximum safe height for $\Delta\hat{h} = -2$ so without an environmental measure this problem is infeasible.

6.2 Planar Running

Extension from simple hopping to planar running complicates the hopper dynamics and the problem formulation in several ways. First, additional states are required for the additional degrees of freedom. Secondly, the change in terrain is no longer be constrained to the apex of flight, but instead can occur multiple times during the flight phase, or not at all. Finally, because the hopper leg can contact the surface at an angle, slipping is possible. Thus, friction plays an important role in the dynamics.

6.2.1 Running with a Single Leg

The problem of running over variable terrain with a single legged hopper is diagrammed in Figure 6.5. The figure shows a sequence of positions and orientations over a single hop. The hopper has an elliptical body with mass, m, and inertia, I, and a massless leg with spring constant, k, damping coefficient, b, and a maximum thrust capability, f_{max}. The body is at an angle θ with respect to horizontal and the hopper leg is at an angle ϕ with respect to the body. The terrain is stepped with variable changes in terrain height, Δh, separated by a fixed distance, Δx. Changes in terrain height are bound between Δh_{max} and Δh_{min}. Alternative models of the terrain are certainly possible but the stepped terrain is most analogous to the vertical hopper studied previously.

During the flight phase the hopper positions the leg to a desired angle for contact. Once in contact, and assuming no slipping, the toe remains fixed to the surface while the leg compresses and rotates about the hip. Prior to liftoff the leg extends from the combination of spring and any thrusting forces. After liftoff the hopper resumes a ballistic trajectory and repositions the leg for the next contact.

The position of the hopper mass center can be tracked relative to the relevant terrain features by adjusting the reference or origin with a discrete phase transition. Figure 6.5 marks changes in terrain height with the letters 'A' and 'B'. These features are separated by a set distance, Δx. During flight, the hopper toe may cross over one or more terrain features as indicated by the broken vertical lines in the figure. The cross over triggers a discrete phase transition that shifts the origin of the reference

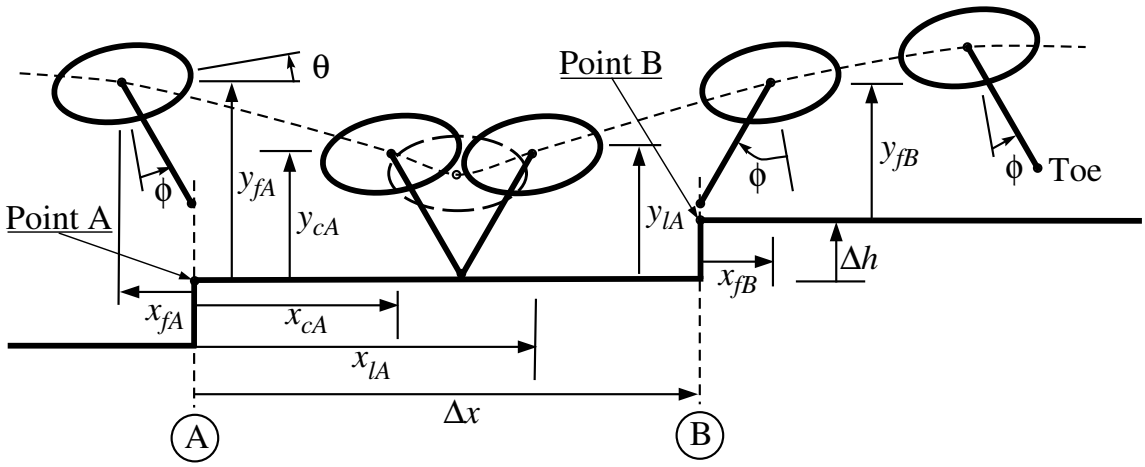


Figure 6.5: Diagram of planar running with a single leg. The environment is stepped with variable changes in terrain height, Δh, separated by a fixed distance, Δx. To track hopper position relative to terrain features a discrete phase transition shifts the origin of the coordinate system when the hopper toe crosses over feature locations. The planar problem also includes rotational degrees of freedom for the body attitude, θ, and the hopper leg angle, ϕ, which is measured relative to the hopper body.

coordinate system. For example, when the toe crosses feature A, the origin of the reference system becomes point A and the horizontal and vertical location of the mass center with respect to point A is x_{fA}, which is negative, and y_{fA}. At the time of contact mass center position has changed to x_{cA} and y_{cA}, and at the time of lift off mass center position has changed to x_{lA} and y_{lA}. The hopper resumes flight and when the toe crosses feature B the origin abruptly shifts to point B and the mass center location becomes x_{fB} and y_{fB}. The relationship between points A and B is the distance between features, Δx, and the change in terrain height, Δh.

The phases of motion for planar running over variable terrain are shown in Figure 6.6. The `contact` and `liftoff` phases are directly analogous to the vertical hopper phases. The flight phases of motion are divided into `ascent`, and `descent`. The `reposition` and `cross` phases capture discrete events. The `cross` phase resets the origin of the coordinate system when the abscissa of the hopper toe reaches Δx. At this instant terrain height changes and collision may occur. Because the distance

traveled in flight varies from hop to hop, the phasing between terrain changes and the flight phase is also variable. Thus, a `cross` may occur one or more times during a hop, or not at all. Also during flight the hopper must reposition the leg from the angle at liftoff to the angle for landing. Since the leg is massless the reposition can occur instantaneously and has no affect on hopper dynamics in flight unless a collision occurs. For simplicity the leg reposition is modeled as an instantaneous slew at the apex of flight. However, it should be noted that the leg angle at contact is an important control variable. The `reposition` phase should allow a range of feasible leg angles and results obtained during the backward chaining sequence of the algorithm will determine the safe subset of reposition angles. Also during `reposition` checks are made for any `cross` event or possible collision.

If feature distance, Δx, is small compared to the length of a hop then more than one `cross` event may occur in flight. The phase diagram on the left side of Figure 6.6 indicates multiple `cross` events during `ascent` and `descent` with the two way arrow between mode transitions. The two way transitions create internal loops which must be iterated through during the backward chaining sequence. However, if feature distance is large compared to the length of a hop, then no more than one `cross` event will occur per hopping cycle. This constraint can be enforced by adding modes and modifying predecessor, successor relationships as shown in the right side of Figure 6.6. Despite the additional modes and more complex looking phase diagram, execution through the backward chaining sequence is simplified.

The equations of motion while in contact can be derived from free body diagrams of the forces and torques acting on the hopper leg and body. Consider the forces and moments acting on the leg as shown on the left of Figure 6.7. The leg is in contact with the ground at the toe and supports the hopper body at the hip. The reaction forces R_N and R_T act at the toe. The leg force, F, which is a combination of leg thrust, u_1, and spring and damping forces, acts along the length of the leg. A normal force, F_N, acts perpendicular to the leg at the hip. Also at the hip, the hopper exerts a hip torque, u_2, on the leg. The forces and moments acting on the body are equal and opposite to the leg forces and moments at the hip. Newton's third law is observed by reversing the direction of F, F_N, and u_2 as shown on the right hand side of Figure

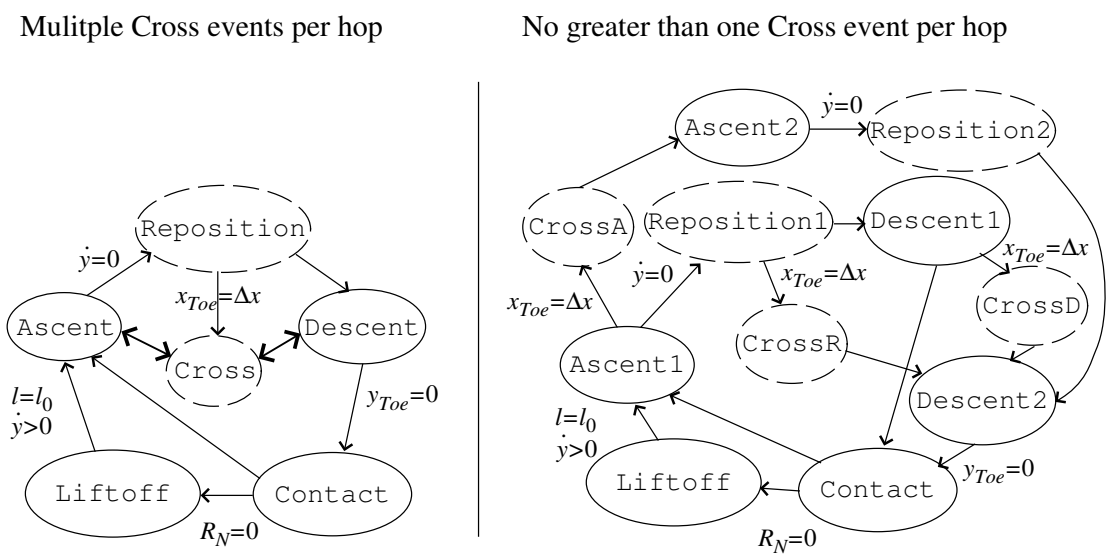


Figure 6.6: Phases of motion for planar running with a single leg. The `ascent`, `descent`, `contact`, and `liftoff` phases are analogous to the like named vertical hopper modes. The `step` phase is replaced by `cross` which moves the origin of the coordinate system to accommodate a change in terrain height. `Reposition` slews the leg to a new angle to prepare for landing and may also trigger a `cross`. If feature distance, Δx, is small then more than one `cross` event may occur in flight and the hopper returns to `ascent` or `descent` as indicated by the two-way transition to the right. If feature length is large compared to the length of a hop then no more than one `cross`event will occur per hop which can be enforced by creating new modes with one way predecessor/successor relationships.

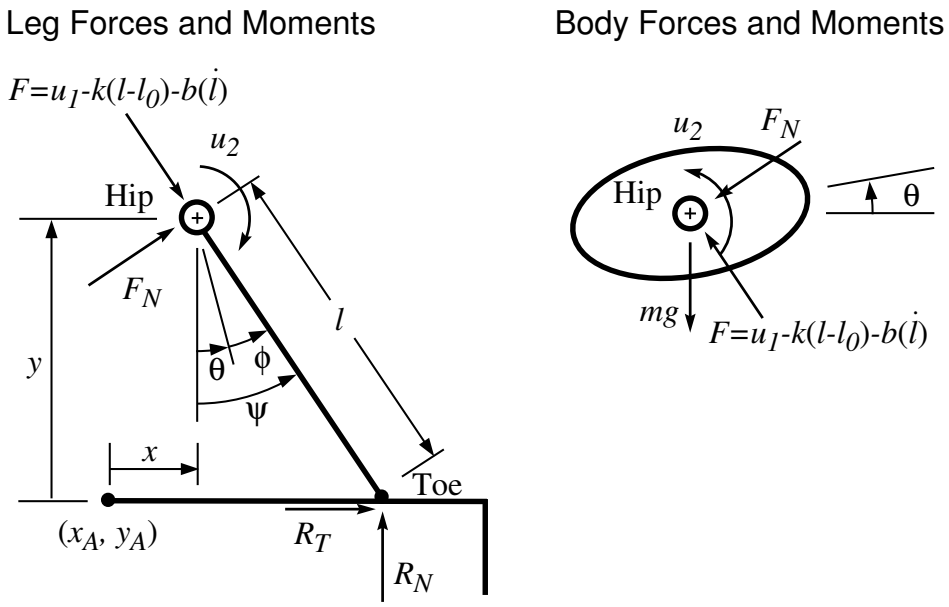


Figure 6.7: Forces acting on hopper leg and body in planar running. Hip forces and moments act equal and opposite on the body and leg. Additionally, gravity acts through the body mass center located, in this example, at the hip. Thrust, u_1, and spring and damping forces act along the leg axis at the hip. Reaction forces R_N and R_T act at the toe. The onset of slipping occurs when $R_T = \mu R_N$ or $R_T = -\mu R_N$. Slipping conditions are safety constraints and are functions of leg angle, ψ, hip height, y, hip velocity, and the control variables u_1 and u_2.

6.7. Additionally, gravity acts through the mass center of the hopper body which, in this example, is located at the hip.

Figure 6.7 also shows the coordinates of the hopper body mass center, (x, y), with respect to a reference point, (x_A, y_A), the hopper leg length, l, hopper body angle, θ, the leg angle with respect to the body reference, ϕ, and the leg angle with respect to vertical, ψ, which is the sum of θ and ϕ. For no slip contact, the toe is fixed and leg length is a function of hopper height and the leg angle: $l = y/\cos(\psi)$.

Because the leg is massless, the sum of the forces and moments acting on the leg must be zero. The system of equations, 6.11, can be used to eliminate non-contributing forces F_N, R_N, and R_T. Thus $F_N = -(1/l)u_2$. However, note that contact is maintained only while R_N is positive and the relationship between forces

is valid only for no-slip contact. Assuming Coulomb friction, sliding contact occurs when the tangential reaction force, R_T, reaches a threshold proportional to the normal reaction, R_N. The coefficient of friction, μ, is a function of material properties and is typically near 0.3 for metal to metal contact.

$$
\begin{aligned}
(\sum F_x)_{leg} &= F_N \cos(\psi) + (u_1 - k(l - l_0) - b(\dot{l})) \sin(\psi) + R_T = 0 && (6.11)\\
(\sum F_y)_{leg} &= F_N \sin(\psi) - (u_1 - k(l - l_0) - b(\dot{l})) \cos(\psi) + R_N = 0 && (6.12)\\
(\sum M)_{leg} &= -u_2 - lF_N = 0 && (6.13)\\
R_N &\geq 0 \quad \text{contact condition}\\
|R_T| &\leq \mu R_N \quad \text{no-slip contact}
\end{aligned}
$$

The sum of the forces acting on the body is equal to the body mass times acceleration at the mass center. The sum of the moments acting on the body is equal to body inertia about the hip times angular acceleration, given in equations 6.14 below.

$$
\begin{aligned}
m\ddot{x} &= (1/l)u_2 \cos(\psi) - (u_1 - k(l - l_0) - b(\dot{l})) \sin(\psi) && (6.14)\\
m\ddot{y} &= (1/l)u_2 \sin(\psi) + (u_1 - k(l - l_0) - b(\dot{l})) \cos(\psi) - mg && (6.15)\\
I\ddot{\theta} &= u_2 && (6.16)
\end{aligned}
$$

During contact, and assuming no-slip, the leg angle is kinematically constrained so that the toe remains fixed. Thus $x_{toe} = x + y\tan(\psi)$ is constant so that $\dot{x} + \dot{y}\tan(\psi) + y\sec^2(\psi)\dot{\psi} = 0$. Substituting $l = y/\cos(\psi)$ for $y > 0$ and $\psi = \theta + \phi$ and adding the kinematic constraint we derive the following equations of motion in contact, 6.17.

$$
\begin{aligned}
\ddot{x} &= (1/m)[-b\sin^2(\theta+\phi)\dot{x} + k\tan(\theta+\phi)y + b\sin(\theta+\phi)\cos(\theta+\phi)\dot{y}\\
&\quad - \sin(\theta+\phi)u_1 + (\cos^2(\theta+\phi)/y)u_2 - kl_0\sin(\theta+\phi)] && (6.17)\\
\ddot{y} &= (1/m)[b\sin(\theta+\phi)\cos(\theta+\phi)\dot{x} - ky - b\cos^2(\theta+\phi)\dot{y}
\end{aligned}
$$

$$+\cos(\theta+\phi)u_1 + (\cos(\theta+\phi)\sin(\theta+\phi)/y)u_2$$
$$+kl_0\cos(\theta+\phi)] - g \tag{6.18}$$
$$\ddot{\theta} = (1/I)u_2 \tag{6.19}$$
$$\dot{\phi} = -(\cos^2(\theta+\phi)/y)[\dot{x} + \tan(\theta+\phi)\dot{y}] \tag{6.20}$$

The equations of motion during `ascent` and `descent` phases are trivial: the hopper follows a ballistic trajectory and the leg angle, ϕ remains fixed. However, the `cross` phase instantaneous shifts hopper position to a new frame of reference:

$$x \leftarrow x - \Delta x \quad (\texttt{cross}) \tag{6.21}$$
$$y \leftarrow y - \Delta y \quad (\texttt{cross}) \tag{6.22}$$

where Δx is a fixed value and Δy is a disturbance input. A collision occurs if the y coordinate of the toe falls below zero: $y - l_0\cos(theta + \phi) \leq 0$, where l_0 is the unsprung leg length (we assume the leg does not retract in flight). During `reposition` the hopper instantaneously repositions the leg angle from the value at liftoff to a new value to prepare for landing. A collision will occur if over this range of motion the y coordinate of the toe falls below zero. Additionally, the `reposition` may trigger a `cross` event which can also generate a collision.

The collision condition is a safety constraint on the system. Additional safety constraints include the minimum leg length constraint, $y/\cos(\theta+\phi) \geq l_{min}$, and constraints on the hip angle, $\phi_{min} \leq \phi \leq \phi_{max}$, which would be imposed by mechanical limits on the joint. To avoid singularities it is useful and reasonable to impose constraints on the body angle, θ, so that the hopper doesn't perform cartwheels. Example constraints are $\theta_{max} = \pi/2 - \phi_{max}$ and $\theta_{min} = -\pi/2 - \phi_{min}$.

A final constraint on the system is non-slip contact at the toe. Returning to the force and moment balance on the leg, slipping begins when $R_T = \mu R_N$ or $R_T = -\mu R_N$. Substituting these values back into equations 6.11 we solve for the slipping condition as a function of the state and control variables. Using $\psi = \theta + \phi$, the constraint equation for positive slipping ($R_T = \mu R_N$) is:

$$
\begin{aligned}
0 \;=\; & b[\sin^2(\psi) + \mu\cos(\psi)\sin(\psi)]\dot{x} - k[\tan(\psi) + \mu]y \\
& -b[\sin(\psi)\cos(\psi) + \mu\cos^2(\psi)]\dot{y} + [\sin(\psi) + \mu\cos(\psi)]u_1 \\
& -[\cos(\psi) - \mu\sin(\psi)](u_2/l) + kl_0(\sin(\psi) + \mu\cos(\psi))
\end{aligned} \tag{6.23}
$$

The constraint equation for negative slipping ($R_T = -\mu R_N$) is:

$$
\begin{aligned}
0 \;=\; & b[\sin^2(\psi) - \mu\cos(\psi)\sin(\psi)]\dot{x} - k[\tan(\psi) - \mu]y \\
& -b[\sin(\psi)\cos(\psi) - \mu\cos^2(\psi)]\dot{y} + [\sin(\psi) - \mu\cos(\psi)]u_1 \\
& -[\cos(\psi) + \mu\sin(\psi)](u_2/l) + kl_0(\sin(\psi) - \mu\cos(\psi))
\end{aligned} \tag{6.24}
$$

The slipping constraints are linear functions of the control variables u_1 and u_2. This problem was briefly discussed in section 3.2.1. From the extreme value theorem of elementary calculus we know that the bounding constraint conditions must be at the extreme values of the control variables. Thus, the constraint for positive slipping in equation 6.23 is bound by 4 constraint surfaces: one each for ($u_{1_{min}}$, $u_{2_{min}}$), ($u_{1_{max}}$, $u_{2_{min}}$), ($u_{1_{min}}$, $u_{2_{max}}$), and ($u_{1_{max}}$, $u_{2_{max}}$). As described in section 3.2.1, we perform the *Reach(G,E)* operation for each constraint surface by propagating forward and backward and time from extremma points on the constraint surface and then resecting the boundary. The process is repeated for the negative slipping constraint given in equation 6.24. If over the range of interest the constraint surfaces defined by equations 6.23 and 6.24 are non-convex, they can be approximated with multiple constraint surfaces.

In summary, we've discussed how to apply the MISS algorithm to the problem of planar running with a single leg. The extension requires attention to several issues. First, the `step` phase is replaced by a `reset` phase which may occur more than once, or not at all during the hop. Secondly, the equations of motion are more complex requiring seven state variables for the additional degrees of freedom. The seventh state variable, leg angle ϕ, is controlled in flight during the `reposition` phase and

is kinematically constrained during contact. Finally, the planar running problem has additional safety constraints on the leg joint angles, body orientation, and sliding contact. The algorithm can be applied to determine the ruggedness of a given hopper design against a variable environment and to establish the least restrictive control that will keep hopper safe. Solution to this problem, however, is beyond the scope of the current work.

6.2.2 Running with Multiple Legs

Applying the MISS algorithm to the problem of running with multiple legs approaches a practical application and introduces a number of difficulties. The algorithm can be used to investigate what effect different configurations have on ruggedness. For example, robot legs may be telescoping, such as the sprawl family of robots (Cham, et. al., 2000, [11]), rigid with flexible hip joints, such as the RHex robot (Saranali, et. al. 2000, [41](, or multi-link such as Tekken (Zhang, et. al., 2005, [50]). Robot posture maybe upright or sprawl. Robot control maybe clock driven or multi-input/multi-output based feedback with environmental sensors. The MISS algorithm is a unified method for assessing performance of these different configurations in an unknown, variable environment.

Figure 6.8 shows two instances of a two-legged hopper negotiating a step in the terrain. On the left side of the figure the rear leg of the hopper is in contact while the front leg is free. On the right side of the figure the front leg is in contact while the rear leg is free. Other phases of motion include non-contact and dual leg contact.

Figure 6.9 is a simplified diagram of the phase transitions for two-leg running. Alternating between phases of motions generates different gaits. For example, a running gait may start with Leg 1 in contact then transition to a non-contact phase, then transition to Leg 2 in contact, then transition to another non-contact phase, and repeat. The transition diagram uses two distinct non-contact phases between Leg 1 and Leg 2 contact to force alternating between legs. A single leg hopping mode would be permitted by adding a transition from 'No contact 1' back to 'Leg 1 contact' and from 'No contact 2' to 'Leg 2 contact'. For a walking gait, both legs are in contact

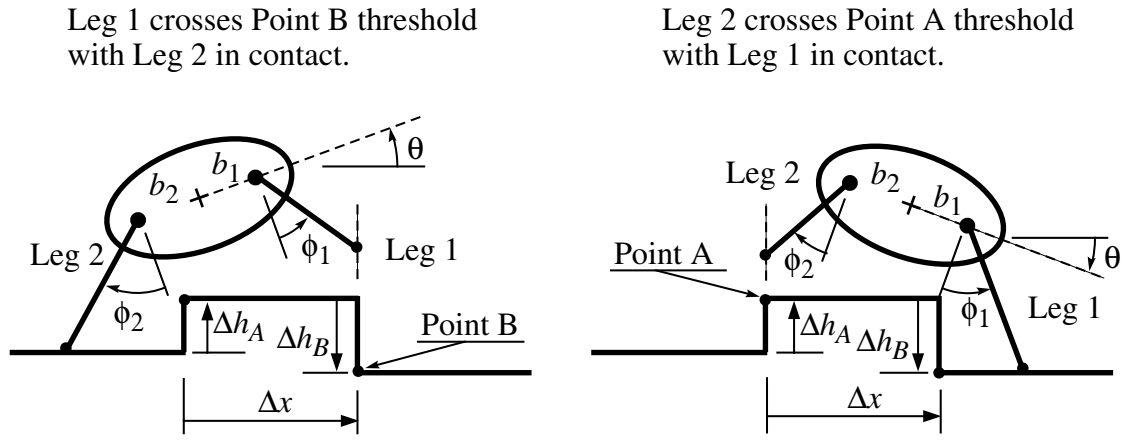


Figure 6.8: Planar running with two legs. An additional variable, ϕ_2, is required to track the position of the second leg relative to the body. Multiple legs also require retaining a "moving window" picture of the environment. When the toe of the front leg reaches a distance Δx, the environment injects a disturbance,Δh_B, to change terrain height and the origin of the coordinate systems shifts to the new reference point. When the toe of the back leg reaches $-\Delta x$ the new disturbance replaces the old terrain change, Δh_A, as a buffered value in the continuous state vector.

between the Leg 1 and Leg 2 only contact phases. Again, separating dual leg contact phases forces alternating legs between steps.

If the legs are massless and flexible[1], the equations of motion for running with two or more legs are similar to the equations of motion for a single legged running hopper, except that the additional forces and torques from each leg in contact must be added. Also, additional states are required for specifying the position of each leg.

However, when considering variable terrain, multiple legs complicate the injection of environmental disturbances. Returning to Figure 6.8, the left hand diagram shows the toe of the front leg of the hopper crossing a change in terrain height, Δh_B at Point B. As for the single legged hopper, this change in terrain is assumed to be a worst case disturbance input over a bound range. The right hand side of the figure shows the toe of the rear leg of the hopper cross a change in terrain height at Point

[1]If the legs are rigid (fixed length), then each leg in contact imposes kinematic constraints that affect the dynamics and complicate the mode transition.

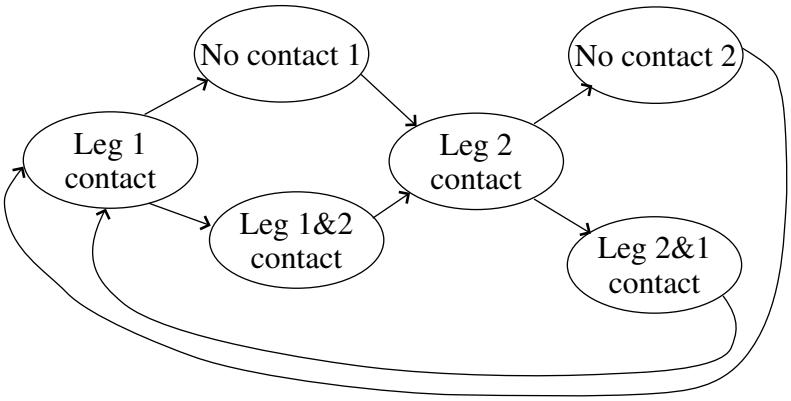


Figure 6.9: Simplified phase diagram for running with two legs. Phases between Leg 1 only contact and Leg 2 only contact can be separated by a no contact phase (running gait), or by phases with both legs in contact (walking gait), or some combination of both. Separate no contact and dual leg contact phases enforce alternating legs between steps.

A. This feature has already been encountered and the change in height, Δh_A, was established previously. A change in terrain height at the front leg is fundamentally different then a change in terrain height for the rear leg. Therefore, the system must add dynamic states to track the terrain.

Suppose that the maximum span between Leg 1 and Leg 2 is less than twice the terrain feature distance, Δx. Then the maximum number of terrain features between toes is two and the change in terrain height can be characterized by two state variables: Δh_1 is the change in terrain height between rear and front toes prior to a front leg cross over, and Δh_2 is the change in terrain height at the front leg cross over. When the toe of the front leg crosses a feature, the terrain height changes by an input value, Δh, where $\Delta h_{min} \leq \Delta h \leq \Delta h_{max}$. The coordinate system shifts to the new feature point and the change in the continuous time states is:

$$x \leftarrow x - \Delta x \tag{6.25}$$

$$y \leftarrow y - \delta h \tag{6.26}$$

$$\Delta h_1 \leftarrow \Delta h_1 \tag{6.27}$$

$$\Delta h_2 \leftarrow \delta h \tag{6.28}$$

Where x and y are the position of the hopper body mass center. The change in ground height between toes 1 and 2 is $\Delta h = \Delta h_1 + \Delta h_2$. When the toe of the rear leg crosses a terrain feature there is no disturbance input since the terrain feature has already been encountered. Rather a rear leg crossing triggers a shift in the terrain states:

$$\Delta h_1 \leftarrow \Delta h_2 \tag{6.29}$$

$$\Delta h_2 \leftarrow 0 \tag{6.30}$$

The position of toes 1 and 2 with respect to the new reference point can be determined from a combination of the state variables, including the hopper leg angles ϕ_1 and ϕ_2, and hopper geometry:

$$\begin{aligned} x_{toe1} &= x + b_1\cos(\theta) + l_1\sin(\theta+\phi_1) \\ y_{toe1} &= y + b_1\sin(\theta) - l_1\cos(\theta+\phi_1) \\ x_{toe2} &= x - b_2\cos(\theta) + l_2\sin(\theta+\phi_2) \\ y_{toe2} &= y - b_2\sin(\theta) - l_2\cos(\theta+\phi_2) \end{aligned}$$

where b_1 and b_2 are distances from mass center to the respective hip joints and the leg lengths, l_1 and l_2, can be determined in contact or out of contact from the following

$$l_1 = \begin{cases} (y + b_1\sin(\theta))/\cos(\theta+\phi_1) & \text{in contact} \\ l_0 & \text{out of contact} \end{cases}$$

$$l_2 = \begin{cases} (y - b_2\sin(\theta) + \Delta h)/\cos(\theta+\phi_2) & \text{in contact} \\ l_0 & \text{out of contact} \end{cases}$$

Note that substituting expressions for l_1 and l_2 back into the equations for each toe yields $y_{toe1} = 0$ and $y_{toe2} = -\Delta h$ when each respective leg is in contact. A front leg cross occurs when leg 1 is not in contact and $x_{toe1} = \Delta x$. A rear leg crossing occurs when leg2 is not in contact and $x_{toe2} = -\Delta x$.

Figure 6.10 is a more detailed diagram of the phase transitions for two legged running with the addition of discrete modes for front and rear leg crossing events. The number of phases explodes to account for the number of different precedent relationships considered. The figure indicates discrete phases by broken boundaries and front leg crossing events are shaded. For example when leg 2 is in contact and leg 1 is not in contact (phase B), a front leg crossing event may occur (phase C1), upon which the system transitions back to a continuous phase (phase C2). The transition is to a new phase (C2) rather than the previous phase (B) to avoid an inner loop during backward chaining. From motion with leg 2 in contact (phase B or C2) the system can transition to a phase with both legs in contact (phase D) or with neither leg in contact (phase E). While neither leg is in contact a front leg crossing may occur (phase G1) or a rear leg crossing may occur (phase H1) or both may occur. If both front and rear leg crossings occur with neither leg in contact, the front leg crossing may occur first (G1 $\succ$ G2 $\succ$ H3 $\succ$ G4) or the rear leg crossing may occur first (H1 $\succ$ H2 $\succ$ G3 $\succ$ G4). A similar set of possible mode transitions occurs during the non-contact following leg 1 only contact (phases J, K1-K4, L1-L3).

Figure 6.10 does not include `liftoff` type transition phases or any explicit `reposition` type phases for the out of contact leg. These phases are omitted to avoid additional clutter. Liftoff phases prevent negative ground reaction forces and maybe omitted at a small price to model fidelity depending on the significance on damping forces in the leg. Reposition of the out of contact leg can occur during or upon transition into the single leg contact phases. For example, leg 1 out of contact reposition can occur during phase B and leg 2 out of contact reposition can occur during phase F. Note that any cyclic phase sequence must include phases B and F so that leg repositioning cannot be skipped over.

The number of phases in the two-legged running example begins to approach the practical limits of a phase transition diagram. An alternative representation is the

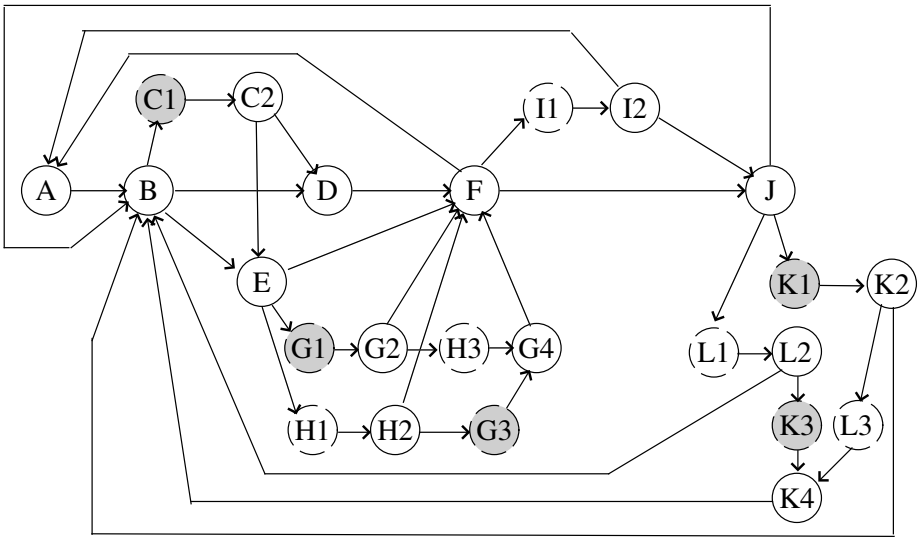
predecessor/successor relationship matrix at the bottom of the of Figure. Predecessor phases are shown in the left hand column and successor phases are shown along the top row. An 'x' in the matrix indicates a predecessor-successor relationship. Each relationship can be annotated to show transition conditions. The predecessor-successor matrix can be readily analyzed to find loops and determine reachability.

6.3 Chapter Summary

This Chapter describes the formulation of several high dimensional problems in the MISS algorithm. The actual solutions require improvements to the MISS algorithm, as noted in section 5.6, and are outside the scope of the current work. However, assuming these numerical difficulties can be addressed, the MISS algorithm will provide a uniform means of evaluating complex systems in uncertain environments.

The open loop control problem in Chapter 5 can be extended to include parameter variation by augmenting the continuous state vector. Parameter variation can be used to optimize a design (to increase ruggedness for example) or to provide a mechanism for adaptive control. In the example, timer period and thrust duration can be tuned by adding these parameters to the state vector and providing the corresponding discrete controls. The adaptation mechanism is not discussed but could be a timer feedback scheme as described in Cham, (2002, [14]).

Environmental knowledge is another means to improve ruggedness. Section 6.1.2 considers the effect of a sensor that can measure the change in terrain height at the next hop. The measurement can be used to advantage over a range of terrain height variations. But the measurement may add no value if the disturbances are small or if disturbance is so large that there is insufficient control authority to maintain safety. In the example in section 6.1.2, the measurement fixes the disturbance input at the next hop, but the environment can still adjust the terrain for the subsequent hop. From the environment's perspective the measurement causes a one cycle delay to the disturbance input. This changes the nature of the 'game' because a worst disturbance input depends on the control system response which is unknown. However, the MISS algorithm can be used to determine the maximal invariant safe subset under the



A: Leg 1&2 contact
B: Leg 2 contact only
C1: Leg 1 cross after (B)
C2: Leg 2 contact after (C1)
D: Leg 2&1 contact
E: No contact after (B)
F: L1 contact only
G1: Leg 1 cross after (E)
G2: No contact after (G1)
G3: Leg 1 cross after (H2)
G4: No contact after (G3, H3)
H1: Leg 2 cross after (E)
H2: No contact after (H1)
H3: Leg 2 cross after (G2)
I1: Leg 2 cross after (F)
I2: Leg 1 contact after (I1)
J: No contact after (F)
K1: Leg 1 cross after (J)
K2: No contact after (K1)
K3: Leg 1 cross after (L2)
K4: No contact after (K3, L3)
L1: Leg 2 cross after (J)
L2: No contact after (L1)
L3: Leg 2 cross after (K2)

	A	B	C1	C2	D	E	F	G1	G2	G3	G4	H1	H2	H3	I1	I2	J	K1	K2	K3	K4	L1	L2	L3
A		x																						
B			x			x																		
C1				x																				
C2					x	x																		
D							x																	
E							x	x			x													
F	x														x		x							
G1									x															
G2							x							x										
G3											x													
G4							x																	
H1													x											
H2							x			x														
H3											x													
I1																x								
I2	x																x							
J		x																	x			x		
K1																			x					
K2		x																						x
K3																					x			
K4		x																						
L1																							x	
L2		x																		x				
L3																					x			

Figure 6.10: Detailed phase diagram for two legged running. The number of modes explodes to track the number of possible predecessor/successor relationships. The typical phase transition diagram can be replaced by a matrix which shows the predecessor/successor relationship and is amenable to analysis for inner loops and reachability.

assumption of an arbitrary rather than an adversarial worst case disturbance input.

Extension from simple hopping to planar running complicates the hopper dynamics and the MISS problem formulation in several ways. First, additional states are required for horizontal and rotational motion and leg angle. If running with multiple legs, additional states are required not only for the angular position of each leg, but to track the difference in terrain height between legs. Secondly, additional safety constraints are required to satisfy hopper orientation, hip joint limits, and non-slip contact. The contact constraints are linear functions of the leg thrust and hip torque control inputs. These constraints generate multiple subboundaries at the extreme values of the control inputs. Finally, the change in terrain is no longer constrained to the apex of flight, but instead can occur multiple times during the flight phase or not at all. This increases the number of possible phases and complicates the phase transition.

Chapter 7

Conclusions and Future Work

This thesis explores the notion of safety as a criterion for legged robot performance in uneven terrain. The robot and environment are modeled as a hybrid system. Unsafe regions of the state space, for example joint constraint violations, collisions, or sliding contact, are identified for each phase of motion. A game theoretic approach described by Tomlin, et. al. (2000, [48]) is used to determine the maximal subset of safe states assuming that the environment behaves in an adversarial or worst case manner. The approach also determines the least restrictive control to keep the system safe.

The analysis method was first applied to a single legged vertical hopping robot under feedback control and then to an open loop hopping robot with an additional timer state to control thrusting. Under feedback control, a measure termed 'ruggedness', that is the maximum step to step variation in terrain height that the robot can safely tolerate, was used to compare robot performance with different leg stiffness, leg damping, and thrust level. The results show that for a given leg stiffness and thrust level there is an optimum value for damping. The results are in contrast with some of the early previous work which assumed zero or very light damping (Koditschek and Buehler, 1991, [27], Ringrose, 1997, [39], Berkemeier and Desai, 1999, [3]). Under open loop control, safe and stable operation can be maintained over level terrain and even climbing stairs. However, if terrain height can vary either up or down, then even small step to step variations can quickly force the hopper to an unsafe state. The results over level terrain are consistent with previous work (Ringrose, 1997, [39],

Cham, 2002, [14]). The results over variable terrain are counter to the experimental success of multi-legged open loop robots such as *Sprawl* (Cham, et. al., 2000, [11]) and *RHex* (Saranali, et. al., 2000, [41]). Likely reasons for this difference include the additional stability and different gaits available with multiple legs.

The analysis method for the closed loop hopping robot in Chapter 2 was a graphical implementation of Tomlin, et. al.'s (2000, [48]) game theoretic approach. The graphical approach was feasible because the hopping robot is a second order system with piece–wise linear dynamics. For higher order and non-linear problems a numerical method is required. However, generalized numerical solutions to optimization problems are extremely difficult. Chapter 3 provides conditions under which direct solution of the Hamilton-Jacobi equations can be avoided which greatly simplifies the numerical aspects of the problem. Chapter 4 provides a detailed description of the Maximal Invariant Safe Subset (MISS) numerical algorithm. Chapter 5 applies the MISS algorithm to the third order problem of open loop hopping with an additional timer state variable. Chapter 6 shows how the MISS algorithm can be applied to hopping problems with adaptive mechanisms and additional measurements and to problems in planar running with single or multiple legs.

The MISS algorithm can be applied to determine safe robot running speeds and terrain variability. The concept of safety and an associated measure, such as ruggedness, can be used to evaluate design alternatives such as different leg configurations or mechanical properties, control strategies, and additional measurements. The approach therefore offers a comprehensive framework for the analysis of the dynamic interaction between legged robots and the environment from which design principles can be derived.

Suggestions for future work are mentioned throughout this document. Currently, the MISS algorithm is imperfect and numerical difficulties were described in Chapter 5. Many of the numerical problems will become worse in more complex problems with higher dimensional state spaces. Memory and processing requirements tend to increase exponentially. Additionally, there is a risk that additional numerical problems may manifest in higher dimensional state spaces. There is also a general need

to develop tools to support visualization and boundary initialization in higher dimensions. These problems require a significant amount of time and effort but are surmountable. Additional future work should include application of the MISS algorithm to the problems in Chapter 6 and experimental confirmation of these results. For most systems safety is not the only criterion and practical demonstration of safe and efficient controllers, as described by Lygeros, et. al. (1997, [32]) is required. Finally, it is my belief that the notion of safety and the approaches described in this thesis have broad application and the future work includes identifying other domains where this work can make a contribution.

Appendix A

Solutions to Hopper Equations of Motion

The normalized non-dimensional equations of motion for the single leg hopping robot are repeated in equation A.1 below, where the primes indicate differentiation with respect to normalized time, τ. (Note that I am redefining yhat so its equal to zero at touchdown).

$$\hat{y}'' = \begin{cases} -1 & \text{ascent, descent} \\ -\hat{k}\hat{y} - \hat{b}\hat{y}' + u - 1 & \text{contact} \end{cases} \tag{A.1}$$

A general solution for position and velocity as a function of normalized time is given in equation A.4 below.

$$\hat{y} = Ae^{-\zeta\omega_n\tau}\sin(\omega_d\tau + \phi_0) + (u-1)/\omega_n^2 \tag{A.2}$$

$$\hat{y}' = -Ae^{-\zeta\omega_n\tau}[\zeta\omega_n\sin(\omega_d\tau + \phi_0) + \omega_d\cos(\omega_d\tau + \phi_0)] \tag{A.3}$$

where

$$\omega_n = \sqrt{1/\hat{k}}$$

$$\zeta = 0.5\hat{b}/\omega_n$$

$$\omega_d = \omega_n\sqrt{1-\zeta^2}$$

and A and ϕ_0 are a function of initial conditions, $\hat{y}_0$ and $\hat{y}'_0$

Find A and ϕ_0 as a function of initial conditions using the relationships in equations A.7.

$$
\begin{aligned}
A\sin(\phi_0) &= \hat{y}_0 - (u_0 - 1)/\omega_n^2 && \text{(A.4)}\\
A\omega_d\cos(\phi_0) &= -(\zeta\omega_n(\hat{y}_0 - (u_0 - 1)/\omega_n^2) + \hat{y}'_0) && \text{(A.5)}\\
\text{for } \omega_d \neq 0 & \\
A &= \pm\{(\hat{y}_0 - (u_0 - 1)/\omega_n^2)^2 + \\
&\quad ((\zeta\omega_n(\hat{y}_0 - (u_0 - 1)/\omega_n^2) + \hat{y}'_0)/\omega_d)^2\}^{1/2} && \text{(A.6)}\\
\phi_0 &= \arctan\left(\frac{-\omega_d(\hat{y}_0 - (u_0 - 1)/\omega_n^2)}{\zeta\omega_n(\hat{y}_0 - (u_0 - 1)/\omega_n^2) + \hat{y}'_0}\right) && \text{(A.7)}
\end{aligned}
$$

For the singularity avoidance constraint, assume contact starting with zero initial velocity and thrust to maximum deflection. That is, $\hat{y}_0 = \hat{y}'_0 = u_0 = 0$. Then the amplitude and phase from equation A.7 are given below. Note the sign for A_0 is negative because position decreases with time.

$$
\begin{aligned}
A_0 &= \frac{-1}{\omega_n^2\sqrt{1-\zeta^2}}\\
\phi_0 &= \arctan\left(\frac{-\sqrt{1-\zeta^2}}{\zeta}\right)
\end{aligned}
$$

At the point of maximum deflection $\hat{y}' = 0$ so $\tau = \pi/\omega_d = \pi/(\omega_n\sqrt{1-\zeta^2})$. Note that $\sin(\pi+\phi_0) = -\sin(\phi_0) = \sqrt{1-\zeta^2}$. From equation A.4 the position at maximum deflection, $\hat{y}_1$ is given below.

$$\hat{y}_1 = \frac{-1 - e^{-\pi\zeta/\sqrt{1-\zeta^2}}}{\omega_n^2} \qquad \text{(A.8)}$$

To determine the constant control input, u_{rtn} that returns the system to $\hat{y}_2 =$

$\hat{y}'_2 = 0$, we have that

$$\begin{aligned} A_1 \sin(\phi_1) + (u_{rtn} - 1)/\omega_n^2 &= \frac{-1 - e^{-\pi\zeta/\sqrt{1-\zeta^2}}}{\omega_n^2} \qquad \text{(A.9)} \\ \text{and since } \hat{y}'_1 = 0 & \\ \sin(\phi_1) &= -\sqrt{1-\zeta^2} \qquad \text{(A.10)} \end{aligned}$$

At $\tau = \pi/\omega_d$ the desired position and velocity are $\hat{y}_2 = \hat{y}'_2 = 0$ and $\sin(\pi + \phi_1) = -\sin(\phi_1)$. Substituting into the previous equations we get the relation in A.11.

$$\begin{aligned} u_{rtn} &= \omega_n^2 A_1 \sin(\phi_1) + 1 \\ u_{rtn} &= -(u_{rtn} + e^{-\pi\zeta/\sqrt{1-\zeta^2}})e^{-\pi\zeta/\sqrt{1-\zeta^2}} + 1 \\ u_{rtn} &= \frac{1 - e^{-2\pi\zeta/\sqrt{1-\zeta^2}}}{1 + e^{-\pi\zeta/\sqrt{1-\zeta^2}}} \qquad \text{(A.11)} \end{aligned}$$

To find the $\hat{y}$ – $\hat{y}'$ trajectory at which the u_{rtn} constraint in equation A.11 applies, it is convenient to transform position and velocity as shown in equation A.12 below.

$$\begin{bmatrix} w_1 \\ w_2 \end{bmatrix} = \begin{bmatrix} \zeta\omega_n & -(\omega_d - 1) \\ \zeta^2\omega_n^2 + \omega_d(\omega_d - 1) & \zeta\omega_n \end{bmatrix} \begin{bmatrix} \hat{y} - (u-1)/\omega_n^2 \\ \hat{y}' \end{bmatrix} \qquad \text{(A.12)}$$

Substituting in equation A.4 into A.12 gives the canonical form below which simplifies to equation A.15.

$$\begin{bmatrix} w_1 \\ w_2 \end{bmatrix} = Ae^{-\zeta\omega_n\tau} \begin{bmatrix} \zeta\omega_n\omega_d \sin(\omega_d\tau + \phi_0) + (\omega_d^2 - \omega_d)\cos(\omega_d\tau + \phi_0) \\ (\omega_d^2 - \omega_d)\sin(\omega_d\tau + \phi_0) - \zeta\omega_n\omega_d \cos(\omega_d\tau + \phi_0) \end{bmatrix} \qquad \text{(A.13)}$$

$$\begin{bmatrix} w_1 \\ w_2 \end{bmatrix} = Re^{-\zeta\omega_n\tau} \begin{bmatrix} \sin(\omega_d\tau + \theta_0) \\ \cos(\omega_d\tau + \theta_0) \end{bmatrix} \tag{A.14}$$

where

$$\begin{aligned} R &= A\omega_d \left(\omega_n^2 - 2\omega_d + 1\right)^{1/2} \\ \theta_0 &= \phi_0 + \arctan\left(\frac{\omega_d - 1}{\zeta\omega_n}\right) \end{aligned}$$

The dependence on normalized time, τ, is eliminated by conversion to polar coordinates, (ρ, θ) where $\theta = \omega_d\tau + \theta_0$. This gives the logarithmic spiral relationship $\rho = Re^{\frac{-\zeta(\theta-\theta_0)}{\sqrt{(1-\zeta^2}}}$ where $w_1 = \rho\sin(\theta)$ and $w_2 = \rho\cos(\theta)$. The equations for R and $theta_0$ is given in A.15 and the dependence on initial conditions is given in A.7. When $\omega_n = 0$ or $\zeta = 0$ and $\omega_n = 1$ the determinant for the transformation in equation A.12 is zero. Otherwise, the transformation back to $\hat{y}$ and $\hat{y}'$ is given by equation A.15 below.

$$\begin{bmatrix} \hat{y} - (u-1)/\omega_n^2 \\ \hat{y}' \end{bmatrix} = \frac{1}{\omega_d(\omega_n^2 - 2\omega_d + 1)} \begin{bmatrix} \zeta\omega_n & (\omega_d - 1) \\ -\zeta^2\omega_n^2 - \omega_d(\omega_d - 1) & \zeta\omega_n \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} \tag{A.15}$$

Bibliography

[1] Sean Ashley Bailey. *Biomimetic Control with a Feedback Coupled Nonlinear Oscillator: Insect Experiments, Design Tools and Hexapedal Robot Adaptation Results*. PhD thesis, Stanford University, 2004. Department of Mechanical Engineering.

[2] John E. Bares and David S. Wettergreen. Dante II: Technical description, results, and lessons learned. *The International Journal of Robotics Research*, 18(7):621–649, July 1999.

[3] Matthew D. Berkemeier and Kamal V. Desai. Control of hopping height in legged robots using a nerual-mechanical approach. In *IEEE International Conference on Robotics and Automation*, pages 1695–1701, Detroit, Michigan, 1999.

[4] R. Blickhan. The spring–mass model for running and hopping. *Journal of Biomechanics*, 22(11–12):1217–1227, 1989.

[5] R. Blickhan and R. J. Full. Similarity in multilegged locomotion: Bouncing like a monopode. *Journal of Comparative Physiology*, 173:509–517, 1993.

[6] M. S. Branicky, V. S. Borkar, and S. K. Mitter. A unified framework for hybrid control: Background, model, and theory. In *Proceedings of the 33rd IEEE Conference on Decision and Control*, pages 4228–4234, Buena Vista, Florida, 1994.

[7] R. Brockett. Hybrid models for motion control systems. In H. L. Trentleman and J. C. Willems, editors, *Essays on Control: Perspectives in the Theory and its Application*. Birhauser, 1993.

[8] I. E. Brown and G. E. Loeb. A reductionist approach to creating and using neuromusculoskeletal models. In J. M. Winter and P. E. Cargo, editors, *Biomechanics and Neural Control of Posture and Movement*. Springer–Verlag, 1999.

[9] Authur E. Bryson and Yu-Chi Ho. *Applied Optimal Control*. Hemisphere Publishing, 1975.

[10] G. A. Cavagna, N. C. Heglund, and C. R. Taylor. Mechanical work in terrestrial locomotion: Two basic mechanisms for minimizing energy expenditure. *American Journal of Physiology*, 233(5):243–261, 1977.

[11] J. G. Cham, S. A. Bailey, and M. R. Cutkosky. Robust dynamic locomotion through Feedforward–Preflex interaction. In *ASME IMECE Proceedings*, Orlando, Florida, 2000.

[12] J. G. Cham, J. K. Karpick, and M. R. Cutkosky. Stride period adaptation for a biomimetic running hexapod. *The International Journal of Robotics Research*, 23(2):141–153, 2004.

[13] Jorge G. Cham, Jonathan Karpick, Jonathan E. Clark, and Mark R. Cutkosky. Stride period adaptation for a biomimetic running hexapod. In *International Symposium of Robotics Research*, Lorne, Victoria, Australia, 2001.

[14] Jorge Gabriel Cham. *On Performance and Stability in Open–Loop Running*. PhD thesis, Stanford University, 2002. Department of Mechanical Engineering.

[15] A. Church. Logic, arithmetic, and automata. In *Proceedings of the International Congress of Mathematicians*, pages 23–35, 1962.

[16] J. E. Clark, J. G. Cham, S. A. Bailey, E. M. Froehlich, P. K. Nahata, R. J. Full, and M. R. Cutkosky. Biomimetic design and fabrication of a hexapedal running robot. In *IEEE International Conference on Robotics and Automation*, volume 4, pages 3643–3649, 2001.

[17] Vin de Silva. Plex – a matlab library for studying simplicial homology. Unpublished technical manual available at http://math.stanford.edu/comptop/programs/plex/, 2003.

[18] Yashuhir Fukuoka, Hirosi Kimura, and Avis H. Cohen. Adaptive dynamic waling of a quaddruped robot on irregular terrain based on biological concepts. *The International Journal of Robotics Research*, 22(3–4):187–202, 2003.

[19] R. J. Full, K. Autumn, J. I. Chung, and A. Anh. Rapid negotiation of rough terrain by the death–head cockroach. *American Zoologist*, 38, 1998.

[20] R. J. Full and D. E. Koditschek. Templates and anchors: Neuromechanical hypothesis of legged locomotion on land. *Journal of Experimental Biology*, 202(23):3325–3332, 1999.

[21] M. R. Henderson, G. Srinath, R. Stage, K. Walker, and W. C. Regli. Boundary-representation based feature identification. In J. J. Shah, M. Mäntylä, and D. S. Nau, editors, *Advances in Feature Based Manufacturing*, pages 15 – 38. Elsevier Science B.V., 1994.

[22] Michael Heymann, Feng Lin, and George Meyer. Control synthesis for a class of hybrid systems subject to configuration–based safety constraints. Technical Memorandum 112196, NASA Ames Research Center, June 1997.

[23] Shigeo Hirose, Kan Yoneda, and Hideyuki Tsukagoshi. TITAN VII: Quadruped walking and manipulating robot on a steep slope. In *IEEE International Conference on Robotics and Automation*, Albuquerque, New Mexico, 1997.

[24] Jessica K. Hodgins and Marc H. Raibert. Adjusting step length for rough terrain locomotion. *IEEE Transactions on Robotics and Automation*, 7(3):289–298, June 1991.

[25] R. Isaacs. *Differential Games*. John Wiley, 1965.

[26] Sangbae Kim, Jonathan E. Clark, and Mark R. Cutkosky. isprawl: Autonomy, and the effects of power transmission. In *7th International Conference on Climbing and Walking Robots*, Madrid, Spain, 2004.

[27] Daniel E. Koditschek and Martin Bühler. Analysis of a simplified hopping robot. *The International Journal of Robotics Research*, 10(6):587–605, December 1991.

[28] H. Komsuoglu and D. E. Koditschek. Preliminary analysis of a biologically inspired 1–DOF 'clock' stabilized hopper. In *Proceedings of World Multiconference on Systemics, Cybernetics and Informatics*, volume IX, pages 670–675, Orlando, Florida, 2000.

[29] T. M. Kubow and R. J. Full. The role of the mechanical system in control: a hypothesis of self stabilization in hexapedal runners. *Philosophical Transactions of the Royal Society of London*, 354:849–861, 1999.

[30] Arthur D. Kuo. The relative roles of feedforward and feedback in the control of rhythmic movements. *Motor Control*, 6:129–145, 2002.

[31] G. Labinaz, M. M. Bayoumi, and K. Rudie. Modeling and control of hybrid systems: A survey. In *IFAC 13th Triennial World Congress*, pages 293–304, San Francisco, California, 1996.

[32] John Lygeros, Claire Tomlin, and Shankar Sastry. Multi-objective hybrid controller synthesis. In O. Maler, editor, *Hybrid and Real–Time Systems*, volume 1201 of *Lecture Notes in Computer Science*. Springer Verlag, 1997.

[33] E. Z. Moore, D. Campbell, F. Grimminger, and M. Buehler. Reliable stair climbing in the simple hexapod 'rhex'. In *IEEE International Conference on Robotics and Automation*, Washington, D.C., 2002.

[34] Pieter J. Mosterman, Gautam Biswas, and Janos Szitpanovits. A hybrid modeling and verification paradigm for embedded control systems. In *Control Engineering Practice*. Elsevier Science Ltd., 1998.

[35] V. V. Nemytskii and V. V. Stepanov. *Qualitative Theory of Differential Equations*. Dover, 1989.

[36] N. Neville and M. Beuhler. Towards bipedal running of a six–legged robot. In *12th Yale Workshop on Adaptive and Learning Systems*, 2003.

[37] Jovan Popovic and Hugues Hoppe. Progressive simplicial complexes. In *SIGGRAPH*, pages 217–224, 1997.

[38] Marc H. Raibert. *Legged Robots that Balance*. MIT Press, 1986.

[39] Robert Ringrose. Self–Stabilizing running. In *IEEE International Conference on Robotics and Automation*, Albuquerque, New Mexico, 1997.

[40] Uluc Saranli, Martin Buehler, and Daniel Koditschek. RHex – a simple and highly mobile hexapod robot. *The International Journal of Robotics Research*, 20(7):616–631, 2001.

[41] Uluc Saranli, Martin Bühler, and Daniel E. Koditschek. Design, modeling and preliminary control of a compliant hexapod robot. In *IEEE International Conference on Robotics and Automation*, San Francisco, California, 2000.

[42] Ching-Long Shih. Ascending and descending stairs for a biped robot. *IEEE Transactions on Systems, Man, and Cybernetics – Part A: Systems and Humans*, 29(3):255–268, May 1999.

[43] Robert F. Stengel. *Stochastic Optimal Control*. John Wiley and Sons, 1986.

[44] John Stillwell. *Classical Topology and Combinatorial Group Theory*. Springer–Verlag, 1993.

[45] Jussi Tohka. *Global Optimization–Based Deformable Meshes for Surface Extraction from Medical Images*. PhD thesis, Tampere University of Technology, 2003. Digital Media Institute/Signal Processing.

[46] Claire Tomlin, John Lygeros, and Shankar Sastry. Synthesizing controllers for nonlinear hybrid systems. In T. A. Henziger and S. Sastry, editors, *Hybrid Systems: Computation and Control*, volume 1386 of *Lecture Notes in Computer Science*. Springer Verlag, 1998.

[47] Claire J. Tomlin. *Hybrid Control of Air Traffic Management Systems*. PhD thesis, University of California, Berkeley, 1998. Department of Electrical Engineering.

[48] Claire J. Tomlin, John Lygeros, and Shankar Sastry. A game theoretic approach to controller design for hybrid systems. *Proceedings of the IEEE*, 88(7), July 2000.

[49] K. J. Waldron, V. J. Perry, and R. B. McGhee. Configuration design of the adaptive suspension vehicle. *The International Journal of Robotics Research*, 3(2), 1984.

[50] Z. G. Zhang, Y. Fukuoka, and H. Kimura. Adaptive running of a quadruped robot using delayed feedback control. In *IEEE International Conference on Robotics and Automation*, Barcelona, Spain, 2005.