

**École polytechnique de Louvain**

# **Scalability and Robustness of Abstraction-Based Control for Bipedal Robots**

Author: **Baptiste GOIES**  
Supervisor: **Raphaël JUNGERS**  
Readers: **Julien CALBERT, Nicolas DOCQUIER, François WIELANT**  
Academic year 2025–2026  
Master [120] in Electro-mechanical Engineering



# Acknowledgements

I would like to thank Professor Raphaël Jungers for supervising this thesis. His remarks and guidance helped me better structure and position my work within the broader context of abstraction-based control, and his interest in my ideas encouraged me to further develop them.

I would especially like to express my sincere gratitude to Julien Calbert and François Wielant for their continuous support throughout this work. Their availability, guidance, and the many discussions we shared during the project were invaluable and greatly contributed to the completion of this thesis. Julien Calbert's expertise in abstraction-based control and `Dionysos.jl` provided essential scientific insight throughout the study, while François Wielant's knowledge of the electromechanical aspects of the robot and bipedal robotic systems greatly helped connect the theoretical and practical dimensions of the project.

I also thank Riccardo for his help regarding the degrees of freedom of the robot, and John for his explanations and advice concerning the computing clusters.

My thanks also go to Véronique Duran for her help with the structure and writing of this thesis.

Finally, I would like to acknowledge the use of ChatGPT during the preparation of this thesis.

Computational resources have been provided by the supercomputing facilities of the Université catholique de Louvain (CISM/UCL) and the Consortium des Équipements de Calcul Intensif en Fédération Wallonie-Bruxelles (CÉCI), funded by the Fonds de la Recherche Scientifique de Belgique (F.R.S.-FNRS) under convention 2.5020.11 and by the Walloon Region.

# Abstract

Abstraction-based control has emerged as a promising framework for the synthesis of correct-by-design controllers for nonlinear and hybrid systems. However, applying such methods to bipedal robots remains challenging due to the high dimensionality and complexity of their dynamics. This thesis investigates the scalability and robustness of abstraction-based control for a planar five-link biped robot using the Dionysos.jl framework.

Since the computational cost of abstraction generation grows exponentially with the dimension of the state space, several strategies are explored to mitigate the curse of dimensionality. These include distributed transition computation through job arrays, exploitation of left-right symmetry, pruning techniques, and adaptive refinement strategies based on local sensitivity and Lipschitz estimates.

The robustness of the generated abstractions is analysed through discretization error propagation and local Lipschitz bounds, with the objective of improving the reliability and precision of the synthesized controllers while limiting computational complexity. Different abstraction-generation strategies and discretization choices are analysed to study the trade-offs between robustness, scalability, and computational cost.

The results highlight both the potential and the limitations of abstraction-based control for bipedal locomotion, and provide insights into practical strategies for improving the scalability of symbolic control methods applied to high-dimensional robotic systems.

# Contents

|          |                                                                   |           |
|----------|-------------------------------------------------------------------|-----------|
| <b>1</b> | <b>State of the Art</b>                                           | <b>1</b>  |
| 1.1      | Introduction and Motivation .....                                 | 1         |
| 1.2      | Bipedal Locomotion as a Control Problem .....                     | 2         |
| 1.2.1    | From Quasi-Static Walking to Dynamic Locomotion .....             | 2         |
| 1.2.2    | Planar Bipedal Robots as Benchmark Systems .....                  | 4         |
| 1.3      | Abstraction-Based Control for Nonlinear Systems .....             | 5         |
| 1.3.1    | Principle of Symbolic Abstractions.....                           | 5         |
| 1.3.2    | Controller Synthesis on Symbolic Models.....                      | 6         |
| 1.4      | Scalability Limits of Symbolic Abstractions .....                 | 6         |
| 1.4.1    | Curse of Dimensionality.....                                      | 6         |
| 1.4.2    | Approximation Errors and Conservatism .....                       | 7         |
| 1.4.3    | Hybrid Sensitivity in Legged Locomotion .....                     | 7         |
| 1.5      | Scalable Abstraction Generation Strategies .....                  | 7         |
| 1.6      | Research Gap and Thesis Objectives.....                           | 8         |
| <b>2</b> | <b>Framework and Previous Developments</b>                        | <b>10</b> |
| 2.1      | Robot Model Used in This Work.....                                | 10        |
| 2.1.1    | Continuous-Time Dynamics of the Bipedal Robot .....               | 10        |
| 2.1.2    | Generalized Coordinates and State Variables .....                 | 11        |
| 2.1.3    | State-Space Formulation .....                                     | 12        |
| 2.1.4    | Discrete-Time Successor Map .....                                 | 12        |
| 2.2      | State Dimension Reduction .....                                   | 13        |
| 2.2.1    | Reduction Through Stance-Foot Constraints.....                    | 13        |
| 2.2.2    | Reduction Through Orientation Fixing .....                        | 14        |
| 2.2.3    | Additional Reductions Used in Previous Work .....                 | 14        |
| 2.3      | Symbolic Abstraction Framework .....                              | 15        |
| 2.3.1    | Finite State-Space Partition .....                                | 15        |
| 2.3.2    | Controller Synthesis on the Abstract Graph .....                  | 16        |
| 2.3.3    | Over-Approximation and Robustness Guarantees.....                 | 17        |
| 2.3.4    | Cell-Center Abstraction as a Computational Compromise ....        | 19        |
| 2.3.5    | Shortest-Path Versus Weighted-Cost Synthesis .....                | 20        |
| 2.3.6    | Application to the Continuous Robot .....                         | 21        |
| 2.4      | Previous Abstraction-Based Controllers and Their Limitations..... | 22        |
| <b>3</b> | <b>Scalable Abstraction Generation</b>                            | <b>24</b> |
| 3.1      | Introduction .....                                                | 24        |

|          |                                                                         |           |
|----------|-------------------------------------------------------------------------|-----------|
| 3.2      | Exploiting Left–Right Symmetry .....                                    | 25        |
| 3.3      | Distributed Transition Computation Through Job Arrays .....             | 27        |
| 3.3.1    | Independent Transition Evaluations .....                                | 28        |
| 3.3.2    | Deterministic Workload Partitioning .....                               | 30        |
| 3.3.3    | Recovering State–Input Pairs from Linear Indices .....                  | 31        |
| 3.3.4    | Implementation with Independent Job Arrays .....                        | 33        |
| 3.3.5    | Scalability of Transition Generation .....                              | 37        |
| 3.3.6    | Memory Bottleneck for Controller Synthesis .....                        | 40        |
| 3.3.7    | Conclusion .....                                                        | 42        |
| 3.4      | Pruning Strategies for Reducing the Effective Abstraction Domain ...    | 42        |
| 3.4.1    | Principles for Designing Pruning Rules .....                            | 43        |
| 3.4.2    | Generic Implementation Pattern .....                                    | 44        |
| 3.4.3    | Swing-Foot Clearance as a Directional Feasibility Constraint .          | 46        |
| 3.5      | Conclusion .....                                                        | 47        |
| <b>4</b> | <b>Sensitivity and Error Propagation in Abstraction-Based Control</b>   | <b>49</b> |
| 4.1      | Introduction .....                                                      | 49        |
| 4.2      | Sources of Approximation Errors in Symbolic Abstractions .....          | 49        |
| 4.3      | Lipschitz-Based Error Propagation .....                                 | 53        |
| 4.3.1    | Weighted norm and local Lipschitz bound .....                           | 53        |
| 4.3.2    | Finite-difference estimation of local amplification .....               | 55        |
| 4.3.3    | Why exhaustive Lipschitz estimation is infeasible .....                 | 56        |
| 4.4      | Empirical Sensitivity Estimation .....                                  | 57        |
| 4.5      | Dimension-Wise Sensitivity .....                                        | 64        |
| <b>5</b> | <b>Global Sensitivity-Based Grid Optimization</b>                       | <b>68</b> |
| 5.1      | Motivation .....                                                        | 68        |
| 5.2      | Why Not a Fully Adaptive Cell-Wise Partition? .....                     | 69        |
| 5.3      | Global Anisotropic Reformulation .....                                  | 70        |
| 5.4      | Grid Update Under a Fixed Budget .....                                  | 71        |
| 5.5      | Algorithm .....                                                         | 72        |
| 5.6      | Results .....                                                           | 73        |
| 5.6.1    | Preservation of the Transition Budget .....                             | 74        |
| 5.6.2    | Convergence of the Error-Score Estimate .....                           | 75        |
| 5.6.3    | Independent Initial–Final Validation .....                              | 77        |
| 5.7      | Interpretation, Generality, and Limitations .....                       | 79        |
| <b>6</b> | <b>Further Investigations: Robust and Physically Informed Synthesis</b> | <b>82</b> |
| 6.1      | From Error Propagation to Transition Costs .....                        | 82        |
| 6.2      | Why the Quantization Term Matters .....                                 | 84        |
| 6.3      | Physical Cost: Hip-Torque-Aware Synthesis .....                         | 85        |
| 6.4      | Combined Robust and Physical Synthesis .....                            | 87        |
| 6.5      | Discussion .....                                                        | 87        |
| <b>7</b> | <b>Conclusion</b>                                                       | <b>89</b> |



# Introduction

The rapid integration of autonomous systems in modern technology has created a growing demand for advanced control strategies for cyber-physical systems (CPS). These systems are often nonlinear, high-dimensional, and underactuated, requiring sophisticated methods to ensure stability and robustness. Within this context, abstraction-based control has emerged as a promising framework for the synthesis of correct-by-design controllers for nonlinear systems. By replacing continuous dynamics with finite symbolic models, abstraction methods enable graph-theoretic reachability analysis and formal controller synthesis.

Bipedal robots constitute a particularly challenging application domain for such methods. Their dynamics combine nonlinearities, underactuation, intermittent foot-ground contacts, and hybrid phase transitions. These characteristics produce highly sensitive behaviors in which small perturbations may generate large trajectory deviations, especially near impact events. Consequently, abstraction generation for bipedal locomotion rapidly becomes computationally intractable when realistic state dimensions and discretization resolutions are considered.

Although abstraction-based control traditionally relies on conservative reachable-set over-approximations providing formal guarantees, the scalable implementations investigated in this thesis primarily rely on simulation-based center abstractions and empirical robustness metrics. Consequently, the objective of this work is not to preserve strict formal guarantees, but rather to investigate practical scalability-robustness trade-offs for high-dimensional robotic systems.

This positioning does not make the abstraction-based approach equivalent to a standard simulation or trajectory-optimization method. Even when formal guarantees are not yet preserved at the continuous level, the symbolic model remains useful because it transforms the locomotion problem into a graph problem over a finite state space.

This differs from MPC-based approaches such as those studied in [7]. In MPC, the controller does not explicitly explore all possible input branches. Instead, it solves an optimization problem over a finite prediction horizon, starting from the current state, and applies only the first input of the optimized sequence before repeating the procedure at the next sampling instant. In the simplified LIPM formulation, this optimization can be posed as a convex finite-horizon problem. However, when the full nonlinear robot dynamics are included, the resulting problem becomes non-convex and the computed solution is generally only locally optimal, depending on the initialization, the chosen

cost function, the constraints, and the prediction horizon.

By contrast, the abstraction-based approach explicitly constructs a finite transition graph over a discretized state-input space. Controller synthesis is then performed on this graph and yields a policy defined over the abstract domain, rather than a single finite-horizon trajectory from the current state. The policy can therefore be globally optimal with respect to the chosen graph objective, such as reachability, shortest-path distance, or cumulative transition cost. This optimality remains limited to the abstraction and does not imply global optimality for the original continuous robot, but it provides a reusable global structure that is fundamentally different from repeated finite-horizon nonlinear optimization.

Moreover, once the abstraction has been generated, it is not tied to a single trajectory or objective. The same symbolic transition system can in principle be reused to synthesize controllers for different reachability, safety, robustness, or cost-based objectives. This reusability is one of the main motivations for investigating scalable abstraction generation, even before full formal guarantees are recovered.

Within the UCLouvain Learning to Control (L2C) project, previous works demonstrated the feasibility of abstraction-based controller synthesis for planar bipedal robots using the Dionysos.jl framework. However, these results were obtained under strong modeling simplifications and highly restricted abstraction domains in order to maintain computational tractability. Even under these reductions, abstraction generation remained computationally expensive and difficult to scale toward larger operating domains or more physically realistic robot models.

The present thesis therefore investigates computational strategies for improving the scalability and robustness of abstraction-based control for nonlinear planar bipedal robots. Particular emphasis is placed on reducing the computational cost of abstraction generation while preserving sufficiently accurate symbolic representations for locomotion control.

Several complementary strategies are explored throughout this work. First, distributed abstraction generation using job-array parallelization is investigated in order to accelerate large-scale transition computation. Second, symmetry exploitation and pruning strategies are introduced to reduce the effective abstraction domain and eliminate physically irrelevant regions of the state space. Third, local sensitivity estimation and Lipschitz-based analyses are used to study discretization-error propagation and identify dynamically sensitive regions of the abstraction. Finally, adaptive and anisotropic refinement strategies are investigated in order to concentrate computational effort where abstraction errors are most likely to amplify.

The objective of this thesis is therefore not only to evaluate the applicability of abstraction-based control to bipedal locomotion, but also to better understand the practical limitations that arise when symbolic abstractions are applied to nonlinear hybrid robotic systems with black-box dynamics. In particular, the work aims to clarify the trade-offs between abstraction fidelity, robustness, computational complexity, and scalability.

The thesis begins with Chapter 1, which presents the state of the art in bipedal robotics, symbolic abstraction methods, scalability limitations, and adaptive abstraction strategies.

Chapter 2 introduces the abstraction framework used throughout the thesis and reviews previous developments performed on the considered planar bipedal robot.

Chapter 3 investigates several strategies for scalable abstraction generation, including symmetry exploitation, distributed transition computation, and pruning techniques.

Chapter 4 studies sensitivity amplification and discretization-error propagation in non-linear hybrid systems through local Lipschitz analysis and empirical sensitivity estimation.

Chapter 5 introduces a global sensitivity-based grid optimization method. The method uses the dimension-wise error information developed in Chapter 4 to redistribute a fixed transition budget across the state variables while preserving a Cartesian grid structure.

Chapter 6 discusses further investigations toward robust and physically informed controller synthesis. It shows how transition-error estimates and physical criteria such as energy consumption requirements could be incorporated into weighted graph costs.

Finally, Chapter 7 summarizes the main contributions, discusses the limitations of the simulation-based and center-based approach, and outlines future research directions.

# Chapter 1

## State of the Art

### 1.1 Introduction and Motivation

Bipedal locomotion is one of the most challenging problems in robotics because it combines nonlinear dynamics, underactuation, intermittent contacts, and balance constraints [32, 11, 22]. Unlike wheeled robots, bipedal systems cannot rely on a continuously stable support base. Instead, they must coordinate their motion across successive contact phases while maintaining dynamic stability.

Over the past decades, important progress has been achieved through nonlinear control, trajectory optimization, whole-body control, and predictive control methods [32, 11, 33]. These approaches have enabled increasingly dynamic walking and running behaviors. However, providing formal guarantees of safety and convergence remains difficult, especially for hybrid robotic systems subject to impacts, contact switches, and model uncertainty [28, 23].

Abstraction-based control offers a complementary perspective. Instead of directly synthesizing a controller on the continuous nonlinear dynamics, the state space is partitioned into finitely many symbolic states. Controller synthesis can then be formulated as a graph-search or reachability problem on a finite transition system [28, 21, 37, 23]. This makes it possible, at least in principle, to combine nonlinear control with formal correctness guarantees.

The main obstacle is scalability. For high-dimensional robotic systems, the number of symbolic states and transitions grows rapidly with the dimension and with the discretization resolution. This curse of dimensionality is especially severe for bipedal robots, where accurate modeling requires several coupled degrees of freedom and where hybrid contact transitions may strongly amplify local errors [28, 31, 23].

This thesis investigates scalable abstraction-based controller synthesis for a planar bipedal robot using the `Dionysos.jl` framework. The objective is not only to synthesize walking controllers, but also to study how abstraction generation can be made

more tractable through symmetry exploitation, parallelization, pruning, and sensitivity-aware discretization.

The remainder of this chapter is organized as follows. Section 1.2 reviews the evolution of bipedal robotics and explains why planar bipeds remain useful benchmark systems. Section 1.3 introduces the general principles of abstraction-based control. Section 1.4 discusses the main scalability and robustness limitations of symbolic abstractions. Section 1.5 reviews existing strategies for reducing abstraction-generation cost. Finally, Section 1.6 identifies the research gap addressed by this thesis and states its main objectives.

## 1.2 Bipedal Locomotion as a Control Problem

### 1.2.1 From Quasi-Static Walking to Dynamic Locomotion

The historical evolution of bipedal robotics progressively shifted from quasi-static walking strategies toward highly dynamic locomotion control. Early humanoid systems such as WABOT-1 demonstrated the feasibility of coordinated bipedal motion but relied primarily on slow quasi-static balance strategies.

Subsequent developments introduced dynamic stabilization techniques based on feedback control, inertial sensing, and trajectory generation. In particular, Zero Moment Point (ZMP) stabilization enabled dynamically balanced walking on robots such as Honda’s ASIMO [13, 17, 30].



WABOT-1 (1973)



ASIMO robot evolution

Figure 1.1: Examples of early and dynamically balanced humanoid robots.

Optimization-based whole-body control and predictive control methods have enabled

highly dynamic locomotion behaviors on humanoid robots such as Atlas. These systems exploit real-time optimization and full-body dynamic models to perform complex motions including jumping, running, and terrain adaptation [22, 29].



Figure 1.2: Boston Dynamics' Atlas robot.

More recently, the humanoid robot Lightning, developed by Honor, completed the Beijing Half-Marathon on 19 April 2026 in 50 minutes and 26 seconds [19, 34]. This achievement highlights the continuing progress toward long-duration dynamic locomotion, while also illustrating the persistent challenges associated with orbital stability, energetic robustness, and sustained high-speed walking. Such challenges motivate control approaches able to reason about global performance criteria, including energy consumption, control effort, and robustness, which abstraction-based control can address through cost-based synthesis on a finite transition graph.

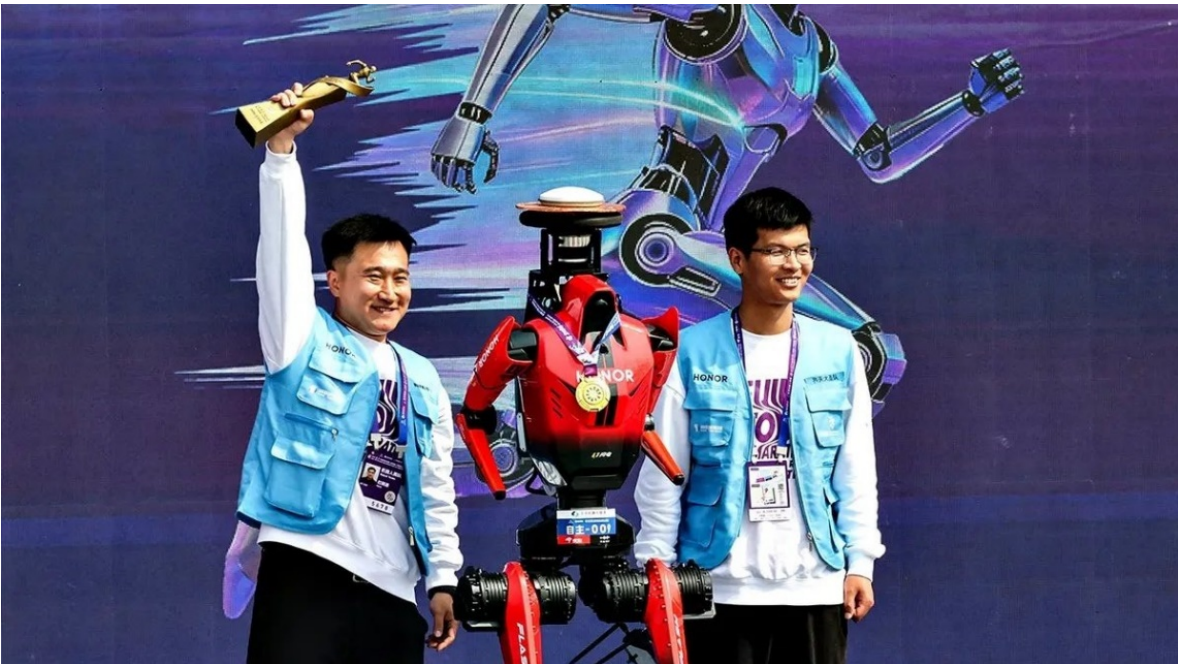


Figure 1.3: Lightning by Honor.

### 1.2.2 Planar Bipedal Robots as Benchmark Systems

Although full humanoid robots demonstrate increasingly sophisticated locomotion capabilities, their high-dimensional dynamics make formal controller synthesis particularly difficult. For this reason, planar bipeds have long been used as benchmark systems for studying the fundamental mechanisms of dynamic walking.

Planar bipeds are constrained to move in the sagittal plane. This removes lateral dynamics and reduces the number of degrees of freedom, while preserving the essential difficulties of legged locomotion: nonlinear coupling, underactuation, intermittent contacts, impact events, and orbital stability. They therefore provide an intermediate level of complexity, rich enough to capture the main hybrid nature of walking but simple enough to remain computationally accessible.

Classical experimental platforms such as RABBIT [5], ERNIE [35], Amber 1 [15], Amber 2, Amber 3M [2], and Durus-2D [16] illustrate this role. These robots typically use five-link or closely related architectures composed of two legs and a torso, with actuation at the hips and knees and passive or lightly actuated feet.

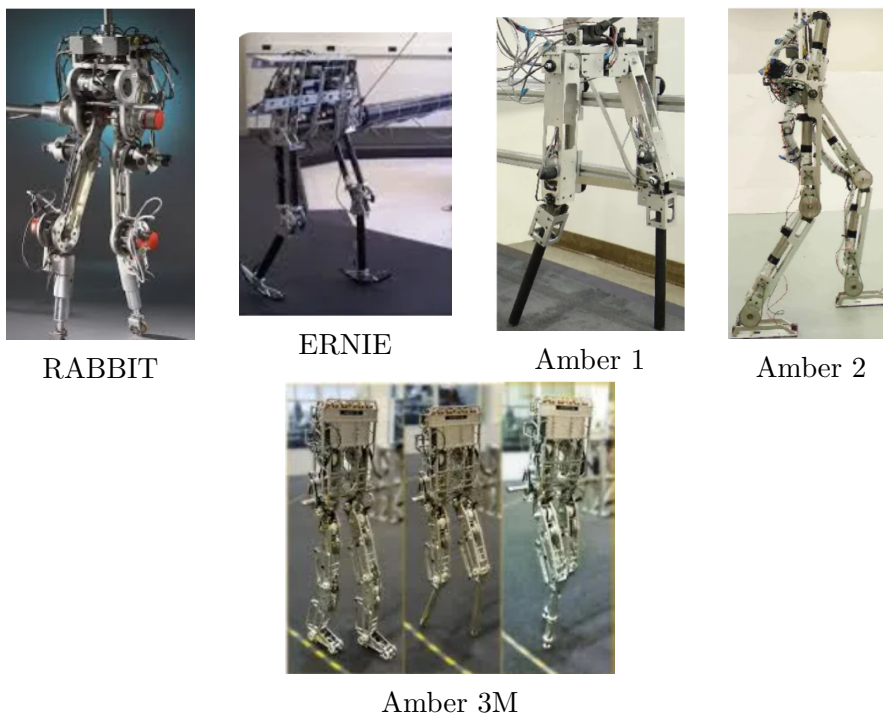


Figure 1.4: Examples of planar bipedal robots used as experimental platforms for dynamic walking and underactuated locomotion.

Such platforms have been central in the development of hybrid zero dynamics, virtual constraints, passive dynamic walking, and underactuated locomotion control [32, 29]. For abstraction-based control, they are particularly relevant because they expose the main challenges of hybrid locomotion while limiting the dimension of the symbolic state space. The detailed robot model used in this thesis is introduced in Chapter 2.

## 1.3 Abstraction-Based Control for Nonlinear Systems

### 1.3.1 Principle of Symbolic Abstractions

Abstraction-based control replaces a continuous nonlinear system by a finite symbolic transition system suitable for formal verification and controller synthesis [28, 21, 20]. The continuous state space is partitioned into a finite collection of cells, each associated with a symbolic state.

The abstraction-based control procedure is classically organized into three principal steps: abstraction of the continuous system, controller synthesis on the abstract model, and concretization of the symbolic controller for the original dynamics.

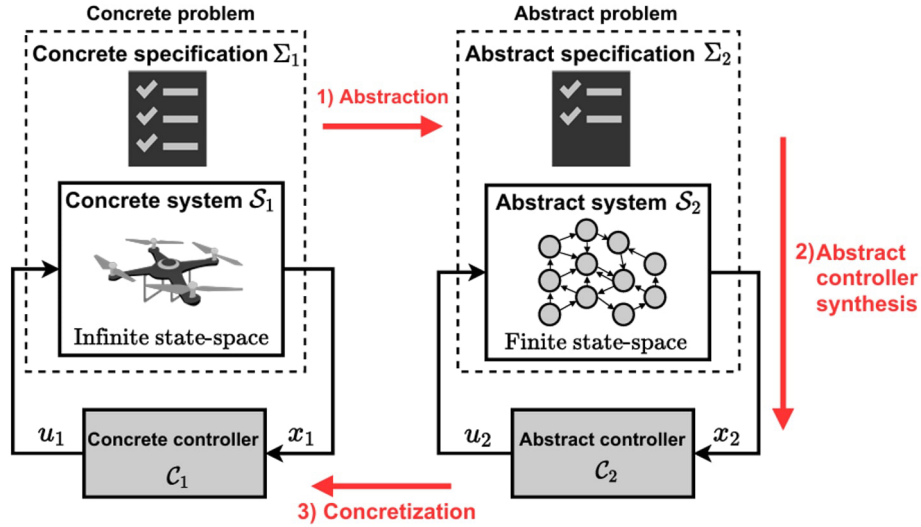


Figure 1.5: Three-step abstraction-based control procedure: abstraction of the concrete system, controller synthesis on the abstract model, and concretization of the symbolic controller. Adapted from [4].

The strength of this approach is that the original nonlinear control problem is converted into a finite graph problem. Reachability objectives, safety constraints, and target-reaching controllers can then be studied using algorithmic methods from formal verification and graph theory.

When the abstraction is constructed conservatively, the symbolic controller can inherit correctness guarantees with respect to the original continuous system. This makes abstraction-based control attractive for safety-critical nonlinear systems. However, as discussed below, the computational cost of constructing the symbolic model rapidly becomes the main bottleneck.

### 1.3.2 Controller Synthesis on Symbolic Models

Once the symbolic transition system has been constructed, controller synthesis is typically formulated as a reachability problem. Given a target set, the algorithm computes the set of symbolic states from which the target can be reached while satisfying the required transition conditions.

A symbolic controller is then extracted by associating each winning state with an admissible input leading closer to the target. This controller is finally applied to the original system by mapping each measured continuous state to its corresponding symbolic state.

This general synthesis procedure is presented here only at the conceptual level. The concrete implementation used in this thesis, including the discrete-time successor map, the winning-set computation, and the weighted-cost formulation, is detailed in Chapter 2.

## 1.4 Scalability Limits of Symbolic Abstractions

Despite their theoretical appeal, symbolic abstractions remain difficult to apply to robotic locomotion because their size grows combinatorially with the state and input dimensions.

### 1.4.1 Curse of Dimensionality

If the  $i$ -th state dimension is discretized into  $N_i$  intervals, the number of symbolic states scales approximately as

$$|Q| \sim \prod_{i=1}^n N_i.$$

Similarly, the total number of transition evaluations grows proportionally to the number of symbolic states and admissible inputs. As a consequence, refining the discretization resolution or increasing the system dimension rapidly produces extremely large transition graphs.

This phenomenon is particularly limiting for bipedal robots. Even simplified planar models require several joint positions and velocities to describe the locomotion state. Increasing the resolution, extending the admissible state ranges, or adding additional physical degrees of freedom therefore rapidly makes exhaustive abstraction generation computationally prohibitive [28, 31, 23].

### 1.4.2 Approximation Errors and Conservatism

Symbolic abstractions rely on finite discretizations of continuous state and input spaces. Consequently, all abstraction procedures introduce approximation errors between the continuous dynamics and their symbolic representation.

A conservative abstraction accounts for these errors by over-approximating the reachable set from each abstract state. This can provide robustness guarantees, but it may also introduce spurious successors that do not correspond to actual concrete trajectories. Such conservatism may prevent some states from being classified as winning, even when a valid concrete control action exists.

Conversely, approximate or simulation-based abstractions reduce computational cost by evaluating only selected representative states. These methods are often more tractable for nonlinear robotic systems, but they weaken the formal guarantees provided by the abstraction.

This creates a fundamental trade-off between computational tractability, abstraction fidelity, and robustness guarantees [31, 1, 10].

### 1.4.3 Hybrid Sensitivity in Legged Locomotion

Legged locomotion introduces additional difficulties due to intermittent contact transitions and strongly nonlinear dynamics. Near impact events or phase transitions, neighboring initial conditions may generate different contact sequences or significantly different successor states.

These local sensitivity amplification phenomena considerably complicate transition approximation and robustness analysis within symbolic abstractions. They are particularly important in bipedal locomotion, where contact switches, ground impacts, and underactuated phases are intrinsic to the walking cycle [26, 9, 11].

Together, dimensionality, approximation errors, and hybrid sensitivity explain why abstraction-based control remains difficult to scale to realistic robotic locomotion systems.

## 1.5 Scalable Abstraction Generation Strategies

Several strategies have been proposed to reduce the computational cost of symbolic abstraction generation.

A first direction consists of parallel and distributed abstraction generation. Since transition evaluations are often independent from one another, they can be partitioned across multiple processors or computing nodes. This is particularly useful for high-dimensional

systems, where transition generation dominates the total computational cost.

A second direction concerns adaptive and nonuniform discretization methods. Rather than refining the entire state space uniformly, these approaches allocate resolution according to local error estimates, sensitivity indicators, or transition variability. The objective is to concentrate computational effort in dynamically sensitive regions while keeping coarser cells in smoother parts of the state space [18].

A third family of methods reduces the number of transitions that need to be computed. This can be achieved by exploiting structural properties of the system, such as symmetries, sparsity, or reachable-domain restrictions. For robotic systems, such reductions are important because many regions of the state-input space may correspond to physically irrelevant or dynamically uninformative configurations.

Finally, recent approaches increasingly rely on approximate or simulation-based abstractions instead of conservative reachable-set over-approximations. In particular, center-based transition evaluation may substantially improve computational tractability for nonlinear robotic systems, although at the cost of weaker formal guarantees [3].

These approaches motivate the scalability strategies investigated in the present thesis: symmetry exploitation, parallelized transition generation, domain pruning, and sensitivity-aware refinement.

## 1.6 Research Gap and Thesis Objectives

Previous work has shown that abstraction-based control can synthesize walking controllers for simplified planar bipedal robots [3]. However, these results were obtained under strong modeling reductions, restricted abstraction domains, and computationally expensive transition-generation procedures.

The remaining challenge is therefore not only to apply abstraction-based control to a bipedal robot, but to make this approach scalable enough to handle larger domains, richer dynamics, and more robust controller synthesis.

This thesis addresses this challenge through four complementary directions:

- exploiting the left-right symmetry of the biped in order to reduce redundant transition computations;
- distributing transition generation across independent processors or machines;
- pruning physically irrelevant or uninformative regions of the abstraction domain;
- studying local sensitivity in order to guide future adaptive discretization strategies.

The next chapter introduces the concrete robot model, the symbolic abstraction framework, and the previous developments on which the present work builds.

# Chapter 2

## Framework and Previous Developments

The previous chapter introduced abstraction-based control as a promising but computationally demanding framework for nonlinear hybrid systems. This chapter now specializes these ideas to the planar bipedal robot considered in this thesis.

The objective is twofold. First, the robot model and the reduced state representations used for abstraction generation are introduced. Second, the symbolic abstraction and controller-synthesis procedures used in previous work are reviewed, in order to identify the precise limitations addressed in the remainder of the thesis.

Previous studies demonstrated that walking controllers could be synthesized for the considered biped using the `Dionysos.jl` framework. However, these controllers relied on strong modeling simplifications, restricted state domains, and center-based transition evaluations. As a result, abstraction generation remained computationally expensive, while the obtained controllers were sensitive to discretization errors and limited operating ranges.

This chapter therefore provides the technical basis for the extensions developed later in the thesis.

### 2.1 Robot Model Used in This Work

#### 2.1.1 Continuous-Time Dynamics of the Bipedal Robot

A planar bipedal walking robot is described by a vector of generalized coordinates  $q$ , containing the joint coordinates and the floating-base coordinates. Its dynamics are governed by the Euler–Lagrange equations [27, 32]:

$$M(q)\ddot{q} + C(q, \dot{q})\dot{q} + g(q) = B(q)u + J(q)^T F_{\text{ext}}(q, \dot{q}, u), \quad (2.1)$$

where  $M(q)$  is the inertia matrix,  $C(q, \dot{q})\dot{q}$  contains Coriolis and centrifugal effects,  $g(q)$  is the gravity vector,  $B(q)$  maps the control inputs to generalized forces, and  $J(q)^T F_{\text{ext}}$  represents external contact forces.

This model captures the main characteristics of bipedal locomotion: nonlinear mechanical coupling, contact forces, and state-dependent dynamics. The hybrid nature of walking arises because the contact configuration changes during the motion, for example when the swing foot impacts the ground.

### 2.1.2 Generalized Coordinates and State Variables

The planar robot considered in this work is represented by the generalized coordinates

$$q = \begin{bmatrix} x_R & y_R & \theta_R & \theta_{LH} & \theta_{RH} & \theta_{LK} & \theta_{RK} \end{bmatrix}^T,$$

where  $x_R$  and  $y_R$  denote the position of the reference foot,  $\theta_R$  denotes the global orientation of the robot,  $\theta_{LH}$  and  $\theta_{RH}$  are the hip angles, and  $\theta_{LK}$  and  $\theta_{RK}$  are the knee angles.

The control inputs correspond to the motor voltages:

$$u = \begin{bmatrix} u_{LH} & u_{RH} & u_{LK} & u_{RK} \end{bmatrix}^T.$$

The complete state of the robot is defined as

$$x = \begin{bmatrix} q \\ \dot{q} \end{bmatrix}. \tag{2.2}$$

Since the robot contains seven generalized coordinates, the corresponding continuous state space is 14-dimensional.

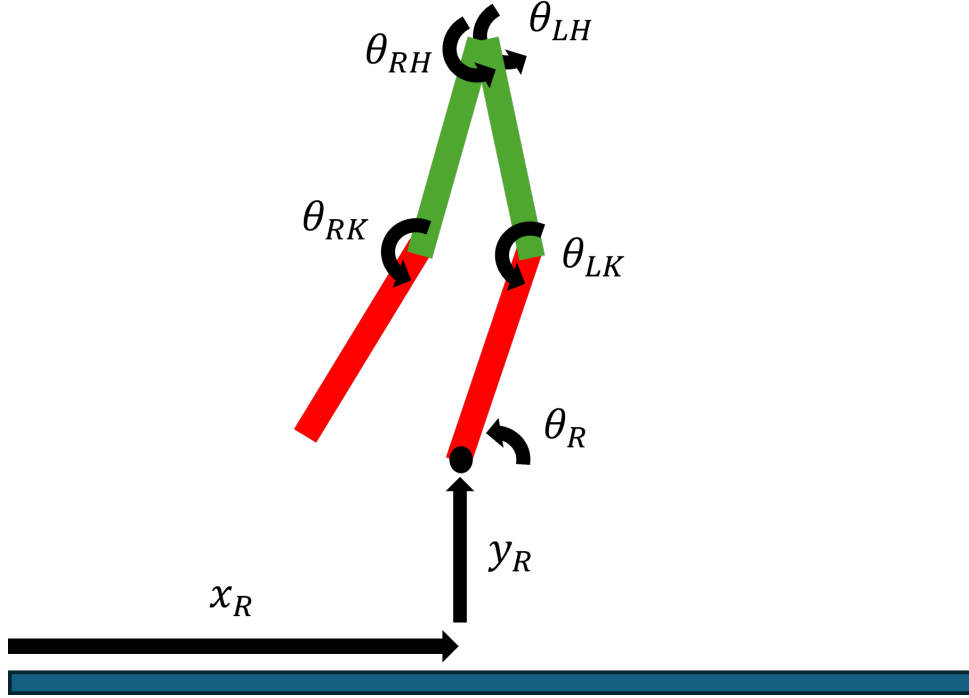


Figure 2.1: Planar bipedal robot model and generalized coordinates. The complete floating-base model contains seven generalized coordinates and therefore a 14-dimensional state space.

### 2.1.3 State-Space Formulation

To apply abstraction-based control, the mechanical dynamics are rewritten as a first-order control system. Assuming that  $M(q)$  is invertible, the acceleration is obtained from (2.1):

$$\ddot{q} = M(q)^{-1} \left( B(q)u + J(q)^T F_{\text{ext}}(q, \dot{q}, u) - C(q, \dot{q})\dot{q} - g(q) \right). \quad (2.3)$$

The dynamics can therefore be rewritten as

$$\dot{x} = f(x, u) = \begin{bmatrix} \dot{q} \\ M(q)^{-1} \left( B(q)u + J(q)^T F_{\text{ext}}(q, \dot{q}, u) - C(q, \dot{q})\dot{q} - g(q) \right) \end{bmatrix}. \quad (2.4)$$

This formulation shows that, for a given control input  $u$ , the evolution of the robot only depends on its current state  $x$ .

### 2.1.4 Discrete-Time Successor Map

The symbolic controller is not built directly on the continuous-time dynamics. Instead, the system is observed at discrete instants separated by a fixed sampling time  $T_{\text{step}}$ .

For a constant input  $u_k$  applied during one sampling period, the dynamics are integrated from  $t = 0$  to  $t = T_{\text{step}}$ . This defines the successor map

$$x_{k+1} = F(x_k, u_k), \quad (2.5)$$

where  $F$  is obtained by simulating

$$\dot{x} = f(x, u)$$

during one sampling interval.

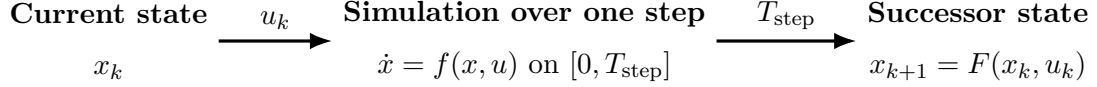


Figure 2.2: Discrete-time successor computation. Starting from  $x_k$ , the continuous dynamics are simulated during one sampling period  $T_{\text{step}}$  using a constant input  $u_k$ .

## 2.2 State Dimension Reduction

The combinatorial growth discussed in Chapter 1 makes it impossible to build a symbolic abstraction directly on the full floating-base model. Several reductions are therefore introduced before constructing the abstraction.

### 2.2.1 Reduction Through Stance-Foot Constraints

During a single-support walking phase, one foot is assumed to remain in contact with the ground without slipping. This contact imposes holonomic constraints on the stance-foot position. Consequently, the translational coordinates

$$x_R, \quad y_R$$

are no longer independent degrees of freedom.

The configuration can therefore be reduced to

$$q = [\theta_R \quad \theta_{LH} \quad \theta_{RH} \quad \theta_{LK} \quad \theta_{RK}]^T,$$

leading to a 10-dimensional state space.

This reduction is mechanically justified during the single-support phase and is commonly used in planar bipedal locomotion models.

### 2.2.2 Reduction Through Orientation Fixing

A further simplification used in previous abstraction-based studies consists in fixing the global orientation of the robot:

$$\theta_R = 0, \quad \dot{\theta}_R = 0.$$

The remaining generalized coordinates are then only the four actuated joint angles:

$$q_{\text{red}} = [\theta_{LH} \quad \theta_{RH} \quad \theta_{LK} \quad \theta_{RK}]^T.$$

The reduced state becomes

$$x_{\text{red}} = \begin{bmatrix} q_{\text{red}} \\ \dot{q}_{\text{red}} \end{bmatrix},$$

which corresponds to an 8-dimensional state space.

Unlike the previous reduction, fixing the global orientation does not result from the natural contact constraints of the walking robot. Instead, the hip orientation is artificially constrained in order to reduce the abstraction complexity.

This simplification also changes the actuation structure of the system. Keeping  $\theta_R$  leads to five generalized coordinates for four motor inputs, corresponding to an underactuated robot. By contrast, the reduced model contains four generalized coordinates and four actuators, making the system fully actuated.

Although this reduction significantly decreases the abstraction size, it also removes part of the floating-base dynamics that are important in physical bipedal locomotion.

### 2.2.3 Additional Reductions Used in Previous Work

Previous studies introduced even more aggressive reductions by constraining specific joints, in particular the right knee, through low-level controllers. Such reductions further decrease the abstraction complexity but also move the model farther away from the physical robot dynamics.

These simplifications are useful for obtaining tractable symbolic abstractions, but they also motivate the need for more scalable abstraction-generation methods. Without such methods, increasing the physical realism of the model or extending the admissible state ranges rapidly becomes computationally prohibitive.

## 2.3 Symbolic Abstraction Framework

### 2.3.1 Finite State-Space Partition

The continuous state space of the robot contains infinitely many states and cannot be solved directly as a finite graph. The main idea of symbolic abstraction is therefore to partition the state space into a finite number of cells [28, 21, 37, 23].

Each abstract state  $S_i$  represents a set of concrete states:

$$S_i \subset X.$$

A concrete state  $x$  is associated with the abstract cell containing it. Instead of manipulating infinitely many continuous states, the control problem is approximated by a finite-state graph.

For each abstract state  $S_i$  and each admissible control input  $u$ , the successor map  $F$  is evaluated and mapped back to the discretized state space. This creates the abstract transition relation

$$\text{Post}(S_i, u) = \{S_j \in \mathcal{S} \mid S_i \xrightarrow{u} S_j\},$$

where  $\text{Post}(S_i, u)$  denotes the set of abstract successor cells reachable from  $S_i$  under input  $u$ .

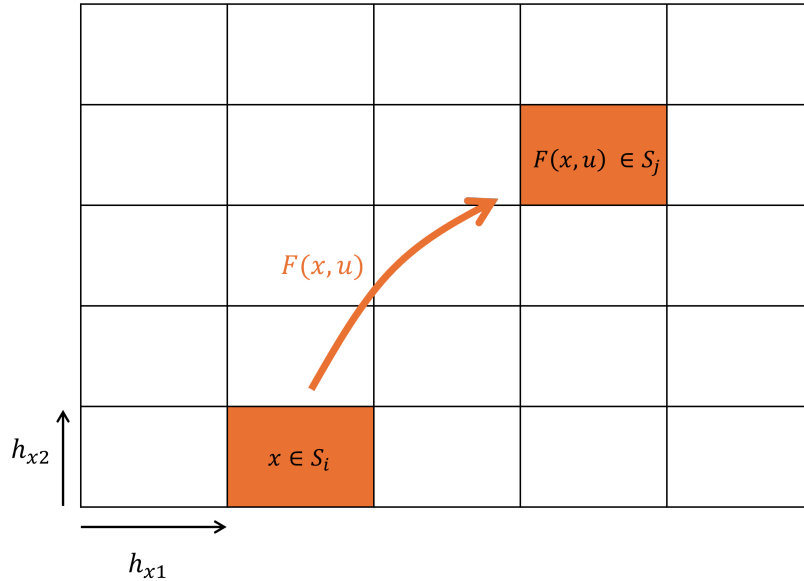


Figure 2.3: Abstract transition between two cells of the discretized state space. The continuous successor  $F(x, u)$ , obtained by simulating the dynamics from  $x \in S_i$ , is mapped to the abstract successor cell  $S_j$ .

### 2.3.2 Controller Synthesis on the Abstract Graph

Once the abstract transition relation has been constructed, the control problem becomes a graph-search problem. Given an initial abstract state  $S_{\text{init}}$  and a target set  $T$ , the objective is to find a sequence of control inputs that drives the system from  $S_{\text{init}}$  to  $T$ .

The target region is initialized as

$$W_0 = T.$$

Then, the winning sets are computed backward. A cell is added to  $W_{k+1}$  if there exists one control input  $u$  such that all possible successors generated under this same input already belong to  $W_k$ :

$$W_{k+1} = W_k \cup \{S_i \in \mathcal{S} \mid \exists u \in \mathcal{U} \text{ such that } \text{Post}(S_i, u) \subseteq W_k\}.$$

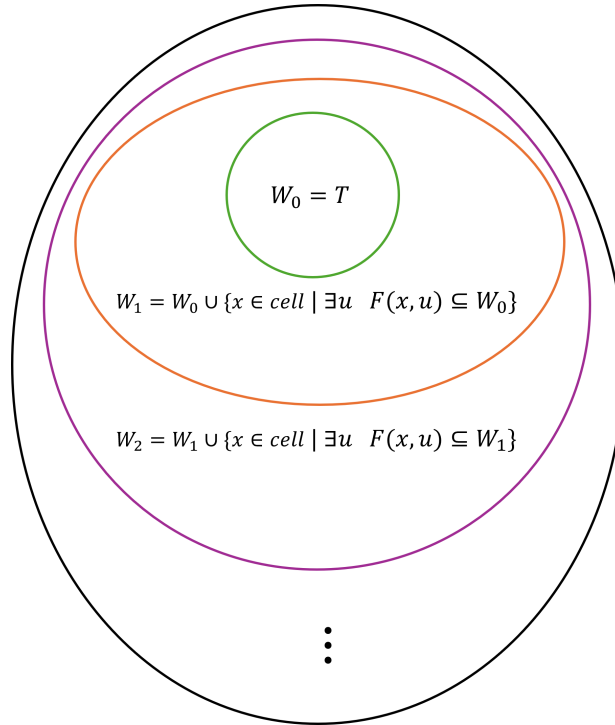


Figure 2.4: Iterative construction of the winning sets. The target set is initialized as  $W_0 = T$ . At each iteration, a cell  $S_i$  is added to  $W_{k+1}$  if there exists a control input  $u$  such that all possible successors generated under this same input belong to  $W_k$ .

The order of the quantifiers is essential. The controller must choose one input for the current abstract state. This same input must drive all possible successors into the previous winning set. However, once a successor cell has been reached, another control input may subsequently be selected according to the symbolic controller associated with that new cell.

The winning set computation can be implemented as a backward breadth-first search from the target. The target is assigned value

$$V(T) = 0.$$

Then, the algorithm propagates values backward through the predecessor relation. The value  $V(S_i)$  represents the minimum number of abstract transitions required to reach the target from  $S_i$ .

At the same time, the algorithm stores the corresponding control input, therefore defining the symbolic controller

$$\pi(S_i) = u.$$

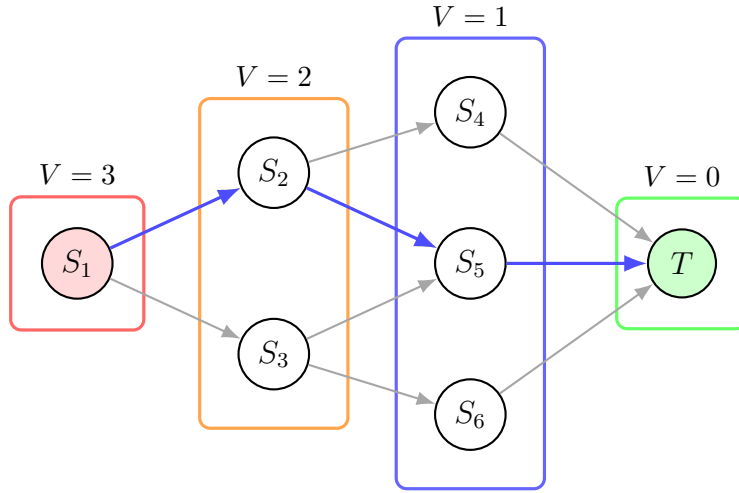


Figure 2.5: Backward search on the abstract graph. Starting from the target state  $T$ , the algorithm propagates values backward and computes a control policy leading to the target.

### 2.3.3 Over-Approximation and Robustness Guarantees

In practice, an abstract state  $S_i$  represents a whole set of concrete states, not a single point. Therefore, applying an input  $u$  to  $S_i$  may generate several possible successors.

A conservative abstraction accounts for this by over-approximating the reachable set from the whole cell. This means that every possible concrete successor is included in the abstract successor set.

This provides a robustness guarantee: if a controller is synthesized on the conservative abstraction, then it is intended to remain robust with respect to the considered abstraction approximations.

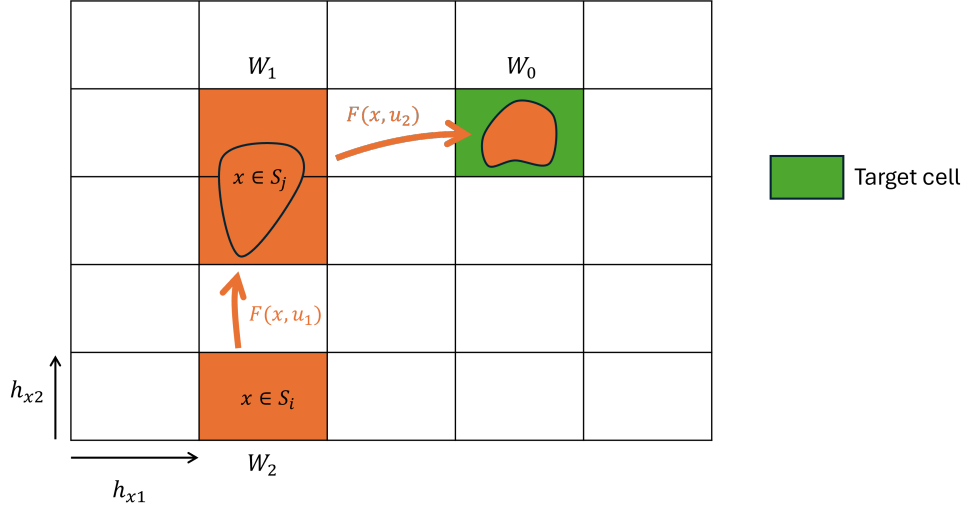


Figure 2.6: Robustness guarantee provided by over-approximation. Since all possible successors remain inside winning cells, the controller is intended to remain robust for every concrete state represented by the abstract cells.

However, this guarantee comes at the cost of conservatism. The over-approximation may introduce spurious successors that do not correspond to actual concrete trajectories.

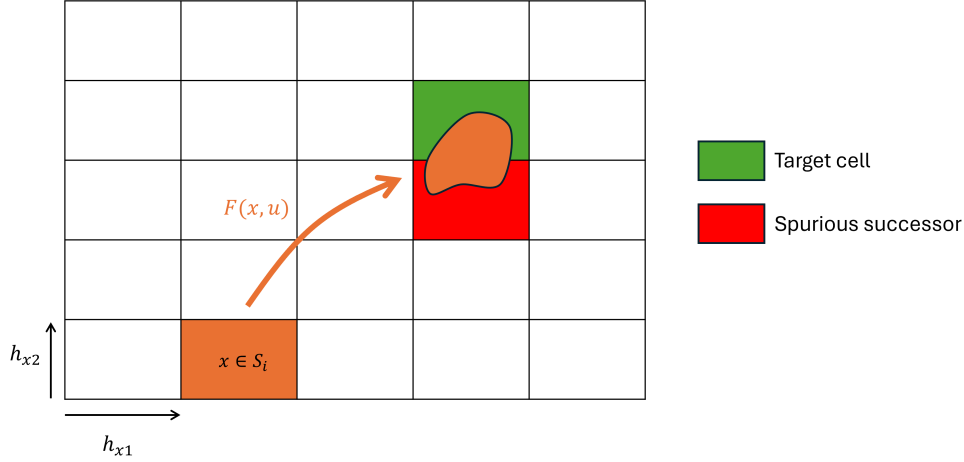


Figure 2.7: Conservatism introduced by over-approximation. The over-approximated successor intersects an additional spurious successor cell, preventing the initial cell from being classified as winning for this input.

This illustrates the conservative nature of over-approximation: by accounting for all possible successors from a cell, the abstraction may also include cells that are only reached by the over-approximated set and not by the actual trajectory. As a result, some admissible control inputs may be rejected, and the computed winning set may become smaller than the set of states that are controllable in practice.

This conservatism can be reduced by making the approximation more precise. One possibility is to compute a tighter reachable set, closer to the true image of the cell.

Another possibility is to use smaller cells, so that the set of possible successors is less spread out. In both cases, the abstraction becomes more expensive to generate: tighter reachable-set computations are more costly per transition, while smaller cells increase the total number of symbolic states and transitions.

### 2.3.4 Cell-Center Abstraction as a Computational Compromise

To reduce the computational burden, one can use a simpler abstraction where only the center  $x_c$  of each cell is simulated.

The winning set recursion then becomes

$$W_{k+1} = W_k \cup \{x_c \mid \exists u \in \mathcal{U} \text{ such that } F(x_c, u) \in W_k\}.$$

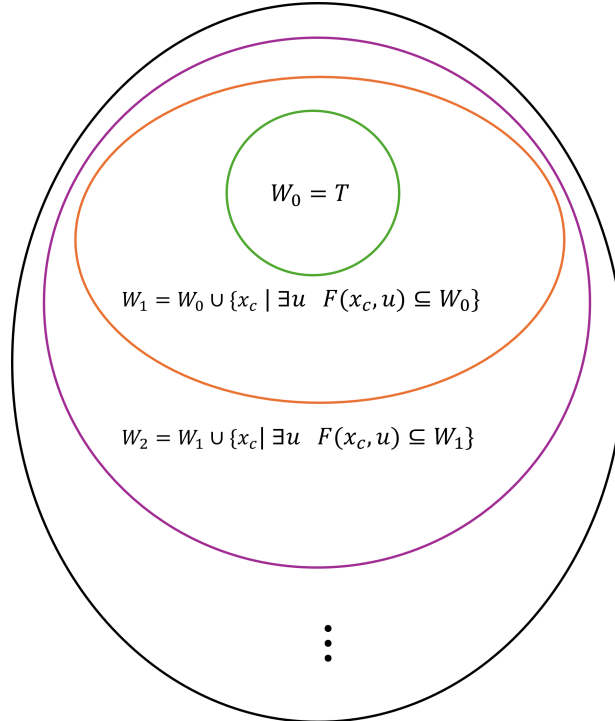


Figure 2.8: Winning set propagation using only the cell centers.

This approach drastically reduces the computational cost since only one simulation is required per cell and per input.

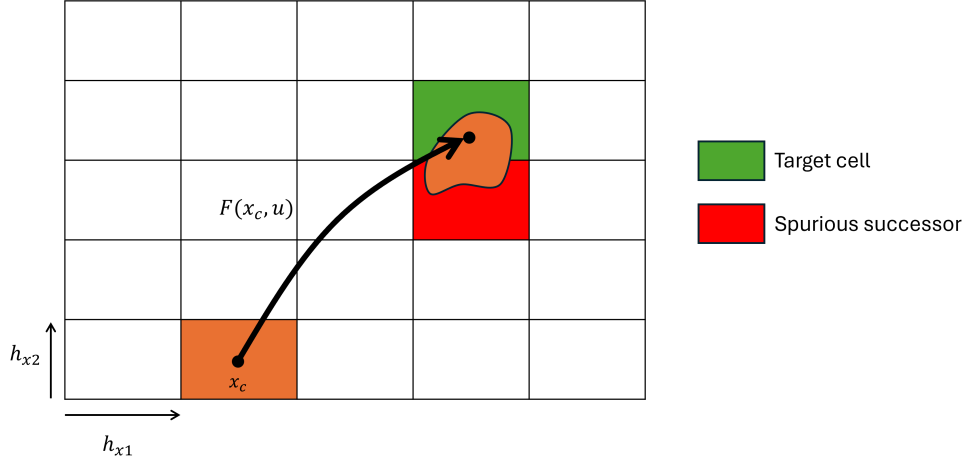


Figure 2.9: Center-based abstraction. Only the center state  $x_c$  of the cell is simulated.

However, the robustness guarantee is lost. Since only the center trajectory is simulated, trajectories starting from other concrete states inside the same cell may behave differently.

The center-based abstraction therefore constitutes a computational compromise: it significantly reduces the number of simulations and transitions, but without guaranteeing that the synthesized controller remains valid for all concrete states represented by the abstract cells.

### 2.3.5 Shortest-Path Versus Weighted-Cost Synthesis

The previous formulation computes the winning sets through a backward breadth-first search. In this case, the value function

$$V(S_i)$$

represents the minimum number of abstract transitions required to reach the target. The resulting controller therefore minimizes the number of discrete steps.

However, minimizing only the number of transitions is not always desirable for robot locomotion. Different transitions may correspond to motions with very different properties, such as control effort, energy consumption, tracking error, robustness margins, proximity to unsafe regions, or transition uncertainty.

Instead of assigning a uniform cost equal to 1 to every transition, one may associate a weighted cost

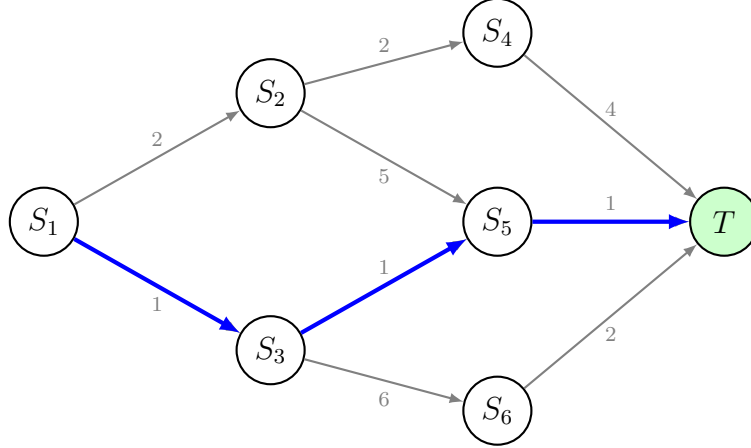
$$c(S_i, u, S_j)$$

to each abstract transition. The value function then becomes

$$V(S_i) = \min_u (c(S_i, u, S_j) + V(S_j)),$$

where the minimization is performed over all admissible successor transitions.

The controller synthesis problem therefore becomes a shortest-path problem on a weighted graph. In this context, Dijkstra's algorithm can be used instead of a standard breadth-first search [8].



Dijkstra selects the path with minimum cumulative cost:

$$S_1 \rightarrow S_3 \rightarrow S_5 \rightarrow T$$

Figure 2.10: Weighted shortest-path synthesis on the abstract graph. Each edge is assigned a transition cost. Dijkstra's algorithm minimizes the cumulative cost instead of only minimizing the number of abstract transitions.

This weighted formulation is particularly useful in abstraction-based locomotion control because it allows the controller to prefer trajectories that are more robust or dynamically favorable, even if they require more abstract transitions to reach the target.

### 2.3.6 Application to the Continuous Robot

The symbolic controller is applied to the original continuous robot in closed loop. At each sampling time, the current continuous state  $x_k$  is measured or estimated and mapped to its corresponding abstract state  $S_i$ . The controller then selects the input

$$u_k = \pi(S_i).$$

This input is applied during one sampling period  $T_{\text{step}}$ , after which the procedure is repeated.

In this way, the nonlinear and hybrid dynamics of the bipedal robot are not controlled directly on the continuous state space. Instead, they are controlled through a finite symbolic controller.

## 2.4 Previous Abstraction-Based Controllers and Their Limitations

Previous works [3] demonstrated that abstraction-based control could synthesize controllers for planar bipedal locomotion. However, these results were obtained under strong modeling simplifications and highly restricted abstraction domains, while giving up formal guarantees.

| Variable                                                                 | Range                      | Discretization                        |
|--------------------------------------------------------------------------|----------------------------|---------------------------------------|
| $\theta_{LH}$                                                            | $[-13, 1]^\circ$           | $h_{\text{pos}} = 2^\circ$            |
| $\theta_{RH}$                                                            | $[-1, 13]^\circ$           | $h_{\text{pos}} = 2^\circ$            |
| $\theta_{LK}$                                                            | $[-1, 15]^\circ$           | $h_{\text{pos}} = 2^\circ$            |
| $\dot{\theta}_{LH}$                                                      | $[-39, 13]^\circ/\text{s}$ | $h_{\text{vel}} = 8.6^\circ/\text{s}$ |
| $\dot{\theta}_{RH}$                                                      | $[-13, 39]^\circ/\text{s}$ | $h_{\text{vel}} = 8.6^\circ/\text{s}$ |
| $\dot{\theta}_{LK}$                                                      | $[-13, 39]^\circ/\text{s}$ | $h_{\text{vel}} = 8.6^\circ/\text{s}$ |
| $u_{LH}$                                                                 | $[-3, 3] \text{ V}$        | $h_u = 1 \text{ V}$                   |
| $u_{RH}$                                                                 | $[-3, 2] \text{ V}$        | $h_u = 1 \text{ V}$                   |
| $u_{LK}$                                                                 | $[-3, 3] \text{ V}$        | $h_u = 1 \text{ V}$                   |
| <b>Number of computed transitions <math>\approx 2 \times 10^7</math></b> |                            |                                       |
| <b>Abstraction computation time <math>\approx 21 \text{ h}</math></b>    |                            |                                       |

Table 2.1: State and input ranges used in previous abstraction-based controllers.

The previous controllers were synthesized on reduced abstraction domains corresponding to highly constrained walking motions. In particular, the state ranges associated with the left and right hip angles were calibrated only for a single half-step. Consequently, the resulting abstraction was not symmetric and required the generation of two distinct controllers in order to alternate between left and right support phases.

Furthermore, the abstraction did not explicitly include a progress variable or a clearance-related constraint. As a consequence, the synthesized controller was not directly encouraged to generate forward locomotion and could produce motions with limited stepping progression, as discussed in Section 3.4.3.

Several additional modeling simplifications were also introduced in order to keep the abstraction computationally tractable.

First, the right knee angle was constrained through a low-level controller, which reduced the number of independent state variables. However, this simplification also restricted the ability of the robot to redistribute its mass and modify the center-of-mass dynamics during walking.

Second, the global hip orientation was fixed, removing the floating-base rotational de-

gree of freedom. Although this reduction significantly decreased the abstraction complexity, it also transformed the robot into a fully actuated system. The resulting model therefore no longer captured part of the underactuated behavior characteristic of physical bipedal locomotion.

In addition, the admissible velocities and motor voltages remained relatively small. The abstraction was therefore restricted to low-speed motions and limited dynamic behaviors.

Despite these simplifications, the abstraction remained expensive to construct: approximately  $2 \times 10^7$  transitions were computed, requiring around 21 hours of computation. This illustrates the central difficulty addressed in the present thesis: even strongly reduced abstraction domains already lead to large transition graphs and significant computation times.

The following chapters therefore investigate several strategies for extending the abstraction-based approach to larger and more robust domains while reducing the computational burden of transition generation.

# Chapter 3

## Scalable Abstraction Generation

### 3.1 Introduction

The previous chapter showed that abstraction-based control can successfully generate walking controllers for planar bipeds, but also highlighted the severe computational limitations that appear as soon as the abstraction domain is enlarged.

The objective of this chapter is therefore to investigate how the admissible state and input ranges can be extended while keeping abstraction generation computationally tractable. In particular, the goal is to reduce some of the previous modeling restrictions and enlarge the set of reachable walking motions without causing a prohibitive increase in computation time.

To address this issue, several complementary strategies are investigated. Rather than modifying the abstraction framework itself, these approaches aim at reducing redundant computations, distributing the workload more efficiently, and restricting the abstraction to physically meaningful regions of the search space.

Three complementary strategies structure the remainder of the chapter:

1. Exploiting the left–right symmetry of the biped in order to reduce the number of explicit transition computations.
2. Distributing the abstraction-generation process across many processors to accelerate large-scale transition evaluation.
3. Introducing pruning strategies to reduce the effective abstraction domain by excluding physically irrelevant or uninformative regions of the search space.

The three strategies are summarized in Table 3.1. The gains indicated below are approximate orders of magnitude intended only to provide a global intuition regarding the expected computational impact of each strategy [25, 14].

| Strategy                         | Principle                                                                     | Expected gain             |
|----------------------------------|-------------------------------------------------------------------------------|---------------------------|
| Left–right symmetry exploitation | Reuse symmetric transitions and controllers instead of recomputing them       | $\approx 2$               |
| Distributed computation          | Distribute transition evaluations over many processors using SLURM job arrays | $\approx n_{\text{proc}}$ |
| Predicate-based pruning          | Discard irrelevant states and inputs before simulation                        | Problem-dependent         |

Table 3.1: Overview of the strategies investigated to enlarge the abstraction domain while limiting abstraction-generation cost.

## 3.2 Exploiting Left–Right Symmetry

A key structural feature of the considered biped is its left–right symmetry. This property can be exploited to reduce the redundancy of the abstraction process.

Figure 3.1 shows that several characteristic postures of the gait can be paired through a simple exchange of the left and right legs. This observation motivates the introduction of a symmetry operator  $S$ , illustrated in Figure 3.2, which maps a state  $\mathbf{x}$  to its mirrored counterpart  $S(\mathbf{x})$  by permuting the coordinates associated with the two legs. The same permutation naturally applies, when relevant, to the control inputs.

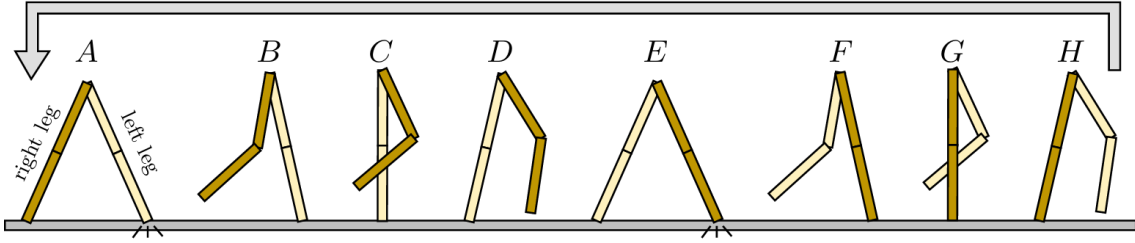


Figure 3.1: Representative walking configurations showing the left–right symmetry of the planar biped gait.

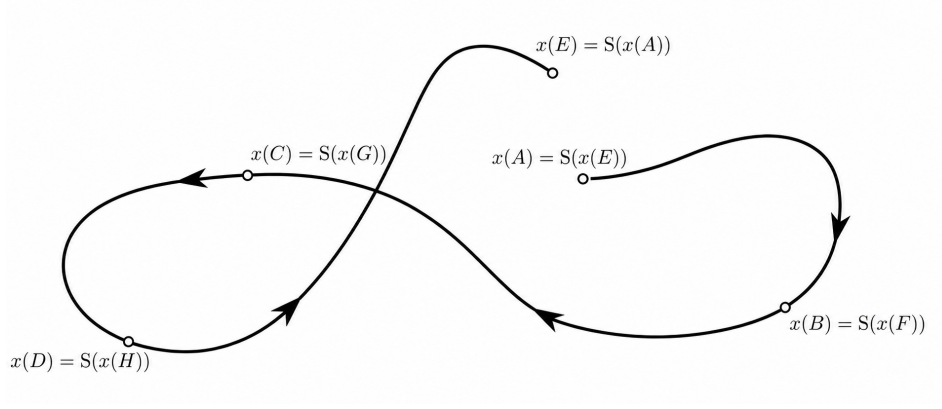


Figure 3.2: State-space symmetry induced by the left-right swap operator  $S$ .

Let the state vector be defined as

$$\mathbf{x} = [\theta_R \quad \theta_{LH} \quad \theta_{RH} \quad \theta_{LK} \quad \theta_{RK} \quad \dot{\theta}_R \quad \dot{\theta}_{LH} \quad \dot{\theta}_{RH} \quad \dot{\theta}_{LK} \quad \dot{\theta}_{RK}]^T$$

The left-right swap operator is then simply the permutation

$$S(\mathbf{x}) = [\theta_R \quad \theta_{RH} \quad \theta_{LH} \quad \theta_{RK} \quad \theta_{LK} \quad \dot{\theta}_R \quad \dot{\theta}_{RH} \quad \dot{\theta}_{LH} \quad \dot{\theta}_{RK} \quad \dot{\theta}_{LK}]^T$$

which exchanges the variables associated with the two legs:

$$\theta_{LH} \leftrightarrow \theta_{RH}$$

$$\theta_{LK} \leftrightarrow \theta_{RK}$$

$$\dot{\theta}_{LH} \leftrightarrow \dot{\theta}_{RH}$$

$$\dot{\theta}_{LK} \leftrightarrow \dot{\theta}_{RK}$$

$$u_{LH} \leftrightarrow u_{RH}$$

$$u_{LK} \leftrightarrow u_{RK}$$

Then, the left-right symmetry can be exploited to reduce the number of explicitly computed transitions by approximately a factor of two. Indeed, if a transition

$$\mathbf{x} \xrightarrow{\mathbf{u}} F(\mathbf{x}, \mathbf{u})$$

is known, then, provided that the dynamics are invariant under the swap operator  $S$ , the symmetric transition is obtained directly as

$$S(\mathbf{x}) \xrightarrow{S(\mathbf{u})} S(F(\mathbf{x}, \mathbf{u}))$$

where  $S(\mathbf{u})$  denotes the corresponding transformation on the control inputs. For a two-legged actuation vector

$$\mathbf{u} = [u_{LH} \quad u_{RH} \quad u_{LK} \quad u_{RK}]^T$$

the input symmetry is simply defined by the same left–right exchange, namely

$$S(\mathbf{u}) = \begin{bmatrix} u_{RH} & u_{LH} & u_{RK} & u_{LK} \end{bmatrix}^T$$

As a consequence, only one representative half of the symmetric state–input space has to be simulated explicitly. The transitions of the opposite half are then obtained by applying the swap operator to the initial state, the input, and the successor state. In the ideal symmetric case, this reduces the transition generation time by approximately a factor of two, since two symmetric transitions are obtained from a single simulation.

The same idea applies at the controller level. Instead of constructing two separate controllers for the two phases of the gait, a single controller can be synthesized on the canonical half-domain and reused on the symmetric half through

$$\mathcal{C}(S(\mathbf{x})) = S(\mathcal{C}(\mathbf{x})).$$

This reduces both the offline synthesis effort and the amount of controller data that must be stored, while preserving the effective domain covered by the controller.

### 3.3 Distributed Transition Computation Through Job Arrays

The previous section showed that the left–right symmetry of the biped can be exploited to reduce the number of explicit transition computations. However, even after this reduction, abstraction generation remains computationally expensive because the controller synthesis procedure still requires evaluating a very large number of state–input transitions.

The purpose of the present section is therefore not to modify the abstraction method itself, but to reorganise its execution. The key observation is that each state–input transition can be simulated independently from all the others. This makes transition generation naturally suited to distributed computation: the global workload can be split into many independent parts, executed separately, and merged afterwards.

This section is organised as follows. First, the transition-generation problem is rewritten as a finite collection of independent state–input evaluations. Then, a deterministic indexing strategy is introduced to partition this workload exactly across several jobs. The reconstruction procedure used by each job to recover the corresponding physical state and input is then described. Finally, the practical implementation using SLURM job arrays is presented, before discussing the scalability results and the remaining memory limitations.

### 3.3.1 Independent Transition Evaluations

Let  $X$  denote the finite set of abstract states and let  $U$  denote the finite set of discretised control inputs. Constructing the symbolic abstraction requires evaluating the successor of each pair

$$(x_i, u_j) \in X \times U.$$

For each pair, the continuous robot dynamics are simulated over one sampling period. The resulting continuous successor state is then mapped back to the symbolic grid, producing a transition of the form

$$x_i \xrightarrow{u_j} x_k.$$

The abstract state and input sets are obtained by discretising hyperrectangular domains. More precisely, the continuous state domain is assumed to be of the form

$$\mathcal{X} = \prod_{\ell=1}^{n_x} [x_{\min,\ell}, x_{\max,\ell}],$$

and the input domain is

$$\mathcal{U} = \prod_{\ell=1}^{n_u} [u_{\min,\ell}, u_{\max,\ell}].$$

The discretisation steps are allowed to depend on the dimension:

$$h_x = (h_{x,1}, \dots, h_{x,n_x}), \quad h_u = (h_{u,1}, \dots, h_{u,n_u}).$$

Each grid point is represented by an integer multi-index. For the state grid, this integer vector is denoted by

$$k^x = (k_1^x, \dots, k_{n_x}^x),$$

and the corresponding physical state is

$$x_\ell = x_{0,\ell} + k_\ell^x h_{x,\ell}, \quad \ell = 1, \dots, n_x.$$

Similarly, for the input grid, an integer vector

$$k^u = (k_1^u, \dots, k_{n_u}^u)$$

defines the physical input

$$u_\ell = u_{0,\ell} + k_\ell^u h_{u,\ell}, \quad \ell = 1, \dots, n_u.$$

The role of  $k$  is therefore purely discrete: it indicates the position of a grid point along each dimension. The continuous values  $x$  and  $u$  are then obtained by multiplying these integer coordinates by the corresponding dimension-dependent discretisation steps. Since the admissible domains are bounded, the integer indices are restricted to finite ranges

$$k_\ell^x \in \{k_{\min,\ell}^x, \dots, k_{\max,\ell}^x\}, \quad k_\ell^u \in \{k_{\min,\ell}^u, \dots, k_{\max,\ell}^u\}.$$

Thus, the finite symbolic sets  $X$  and  $U$  are defined as

$$X = \{x_0 + k^x \odot h_x : k^x \in K_x\},$$

and

$$U = \{u_0 + k^u \odot h_u : k^u \in K_u\},$$

where  $\odot$  denotes componentwise multiplication and where

$$K_x = \prod_{\ell=1}^{n_x} \{k_{\min,\ell}^x, \dots, k_{\max,\ell}^x\}, \quad K_u = \prod_{\ell=1}^{n_u} \{k_{\min,\ell}^u, \dots, k_{\max,\ell}^u\}.$$

The total number of transition evaluations is therefore

$$N_{\text{trans}} = |X| |U|.$$

The crucial property is that the simulation associated with one pair  $(x_i, u_j)$  does not depend on the result of any other pair. Hence, the transition-generation step is embarrassingly parallel: the complete set  $X \times U$  can be divided into disjoint subsets, each evaluated independently, and the final transition relation is obtained by merging the partial results:

$$\Delta = \Delta_1 \cup \Delta_2 \cup \dots \cup \Delta_P.$$

This idea is illustrated in Figure 3.3. The full state–input workload is partitioned into independent parts, each of which can be evaluated by a separate processor or machine.

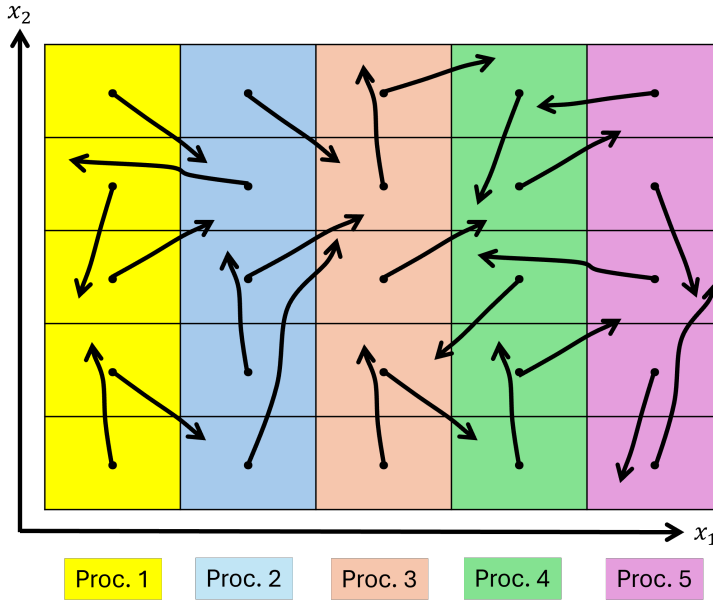


Figure 3.3: Partition of the transition-generation workload into independent state–input evaluations.

### 3.3.2 Deterministic Workload Partitioning

To distribute the transition computations, the Cartesian product  $X \times U$  is first flattened into a single linear workload. Let

$$n_{\text{states}} = |X|, \quad n_{\text{inputs}} = |U|.$$

The total amount of work is

$$\text{total\_work} = n_{\text{states}} n_{\text{inputs}}.$$

The complete workload is therefore represented by the linear index set

$$\{1, \dots, \text{total\_work}\}.$$

Assume that this workload is distributed across  $\text{njobs}$  independent jobs. For a job with index

$$\text{jobid} \in \{1, \dots, \text{njobs}\},$$

the assigned interval is defined by

$$\text{lo} = \left\lfloor \frac{(\text{jobid} - 1) * \text{total\_work}}{\text{njobs}} \right\rfloor + 1,$$

and

$$\text{hi} = \left\lfloor \frac{\text{jobid} * \text{total\_work}}{\text{njobs}} \right\rfloor.$$

Each job therefore processes all indices in the interval

$$[\text{lo}, \text{hi}].$$

These intervals form a partition of the full workload: they are disjoint and their union covers  $\{1, \dots, \text{total\_work}\}$ . Consequently, every state–input pair is evaluated exactly once. No transition is missed, and no transition is computed twice.

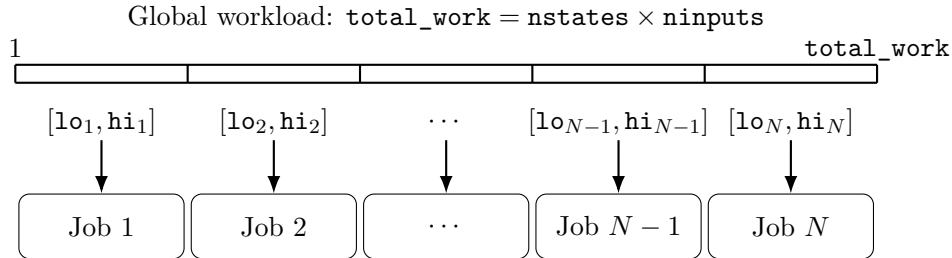


Figure 3.4: Exact partition of the linear workload into independent job intervals.

### 3.3.3 Recovering State–Input Pairs from Linear Indices

The previous subsection defines which linear indices are assigned to each job. However, the simulator does not operate on linear indices: it requires a physical state  $x$  and a physical input  $u$ . Each job must therefore be able to reconstruct the corresponding pair  $(x, u)$  from each assigned index.

For a given linear index `linear_idx`, the input and state indices are recovered by

$$i = \left\lfloor \frac{\text{linear\_idx} - 1}{\text{nstates}} \right\rfloor + 1,$$

and

$$j = ((\text{linear\_idx} - 1) \bmod \text{nstates}) + 1.$$

Here,  $i$  identifies the selected input in the input grid, while  $j$  identifies the selected state in the state grid.

These indices are still one-dimensional labels. Since both  $X$  and  $U$  are multidimensional grids, each label must then be converted into multidimensional grid coordinates. This is done using the function

$$(I_1, \dots, I_N) = \text{lin\_to\_multi}(\text{idx}, \text{dims}),$$

where

$$\text{dims} = (d_1, \dots, d_N)$$

contains the number of grid points along each dimension.

Let  $r = \text{idx} - 1$  be the zero-based version of the index. The coordinates are obtained by successive modulo and integer-division operations:

$$I_1 = (r \bmod d_1) + 1, \quad r_1 = \left\lfloor \frac{r}{d_1} \right\rfloor,$$

$$I_2 = (r_1 \bmod d_2) + 1, \quad r_2 = \left\lfloor \frac{r_1}{d_2} \right\rfloor,$$

and so on. This is equivalent to a mixed-radix representation of the index, where the bases are the grid dimensions  $d_1, \dots, d_N$ . The first coordinate varies fastest, the second coordinate varies over blocks of size  $d_1$ , the third over blocks of size  $d_1 d_2$ , and so forth.

Figure 3.5 illustrates the structure of this decomposition. The modulo operation extracts the current coordinate, while the integer division removes the cycle that has already been accounted for. As the linear index increases, the first coordinate cycles at every step, while the following coordinates vary over increasingly larger blocks.

## Hierarchical variation of the coordinates

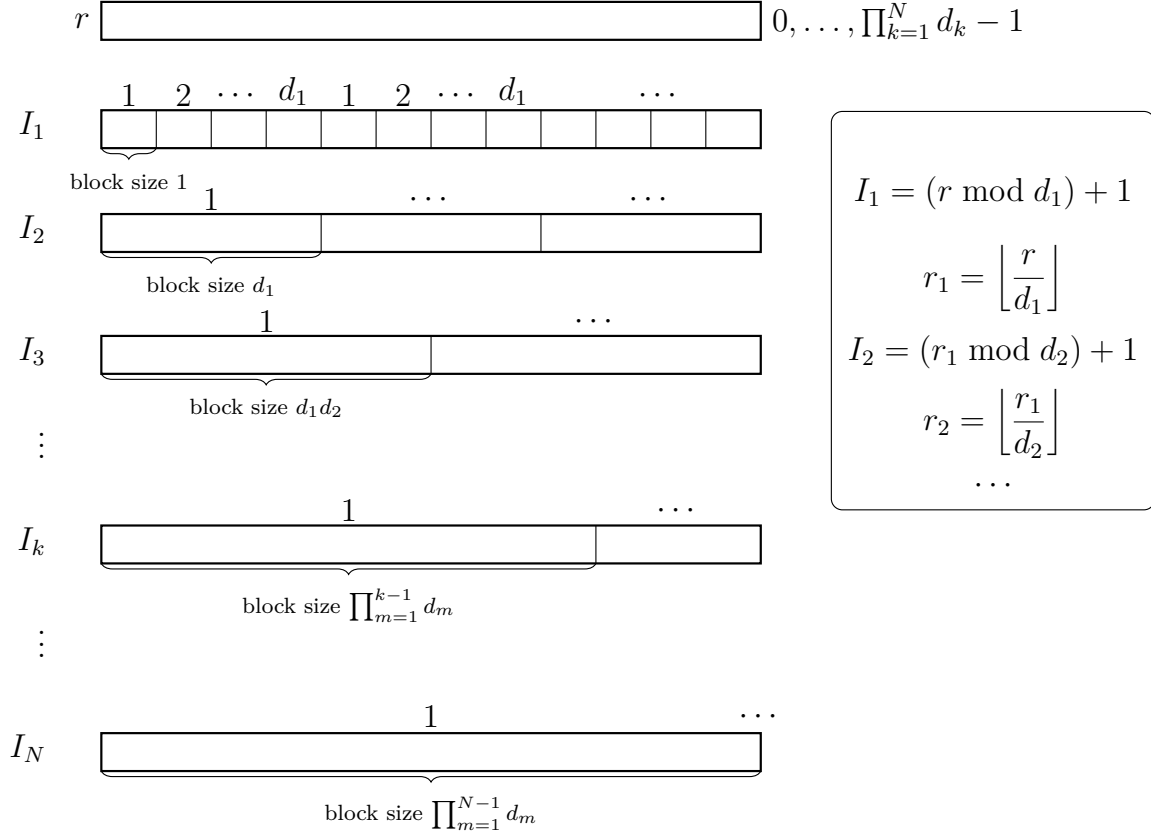


Figure 3.5: Hierarchical decomposition of a global index into coordinate indices.

Once the multidimensional coordinates are known, they are converted into the integer grid indices  $k^x$  and  $k^u$ . If

$$I_i = (I_{i,1}, \dots, I_{i,n_u})$$

is the multidimensional input coordinate, then

$$k_\ell^u = k_{\min,\ell}^u + I_{i,\ell} - 1, \quad \ell = 1, \dots, n_u.$$

The corresponding physical input is then

$$u_\ell = u_{0,\ell} + k_\ell^u h_{u,\ell}, \quad \ell = 1, \dots, n_u.$$

Similarly, if

$$I_j = (I_{j,1}, \dots, I_{j,n_x})$$

is the multidimensional state coordinate, then

$$k_\ell^x = k_{\min,\ell}^x + I_{j,\ell} - 1, \quad \ell = 1, \dots, n_x.$$

The corresponding physical state is then

$$x_\ell = x_{0,\ell} + k_\ell^x h_{x,\ell}, \quad \ell = 1, \dots, n_x.$$

Therefore, each job can independently reconstruct the exact state–input pairs associated with its assigned interval, simulate them, and write the resulting partial transition relation.

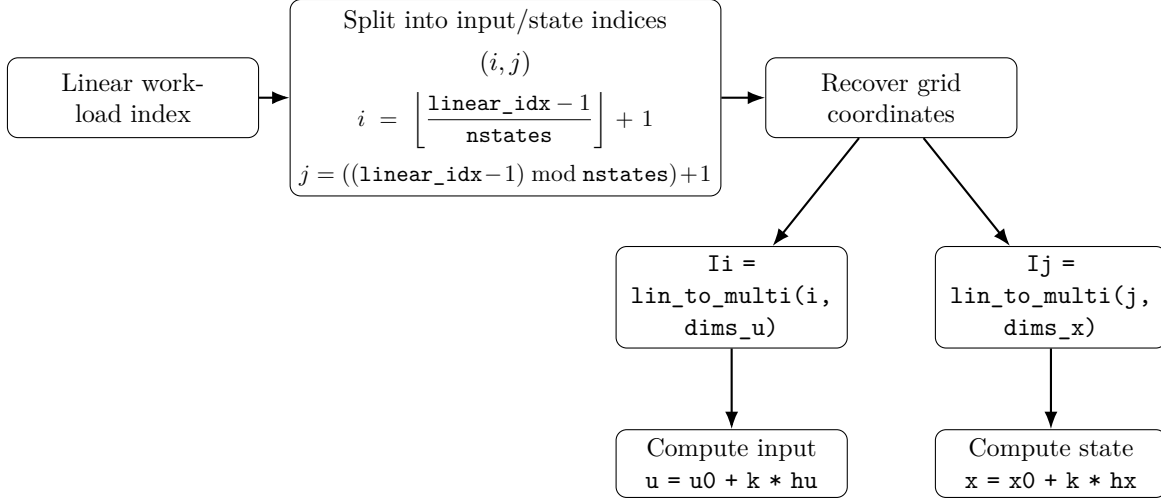


Figure 3.6: Reconstruction of the physical state–input pair from a linear workload index.

### 3.3.4 Implementation with Independent Job Arrays

The previous subsections define a generic distributed algorithm. In this work, this algorithm is implemented using SLURM job arrays on a high-performance computing cluster [36]. Each array task receives a job index, computes the corresponding interval  $[lo, hi]$ , reconstructs the associated state–input pairs, simulates the transitions, and writes a partial transition file. This workflow is shown in Figure 3.7. The only synchronisation step is the final merge of the partial files.

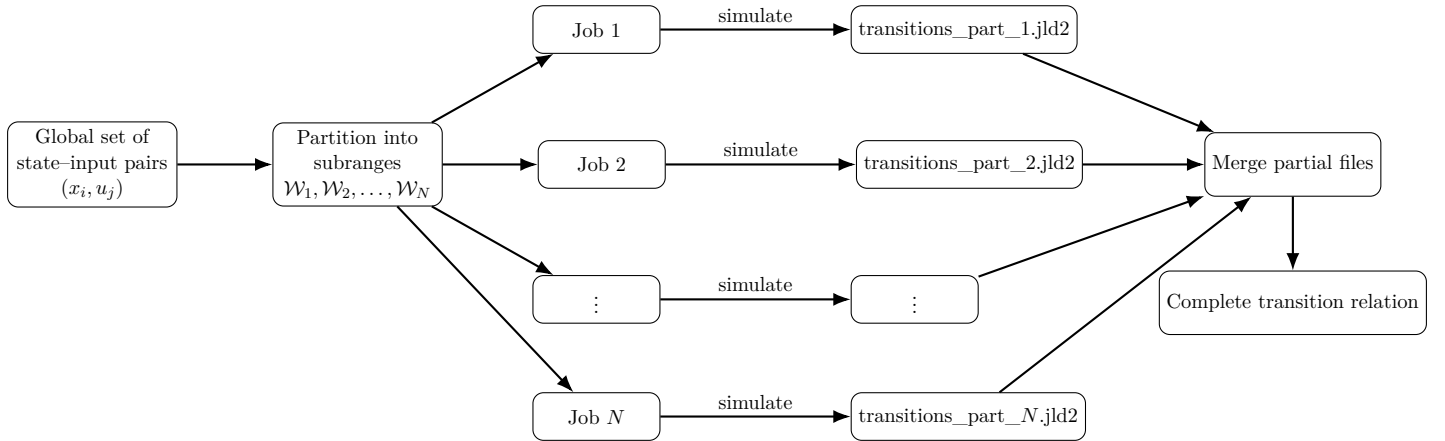


Figure 3.7: Global workflow of the SLURM job-array implementation.

## Multi-processing versus Multi-threading

This implementation differs from a classical multithreaded approach. In a multi-threaded program, several threads share the same memory space and the work distribution can be managed internally by a single execution. In the present implementation, each SLURM job is an independent process with its own memory space. The work distribution must therefore be made explicit through the deterministic interval partition described above.

This multiprocessing strategy has two practical advantages. First, it avoids thread-safety issues associated with shared simulator instances or shared data structures. Second, it makes the computation usable on completely independent machines. Each job only needs the grid definition, the transition function, its own `jobid`, and the total number of jobs `njobs`. From these two integers, it automatically determines its assigned range of linear indices using the deterministic partition rule introduced above. It can then reconstruct the corresponding state-input pairs locally and compute its partial transition relation without requiring any shared memory. This is also well aligned with cluster execution, where processors may be distributed across different nodes that do not necessarily share the same memory space.

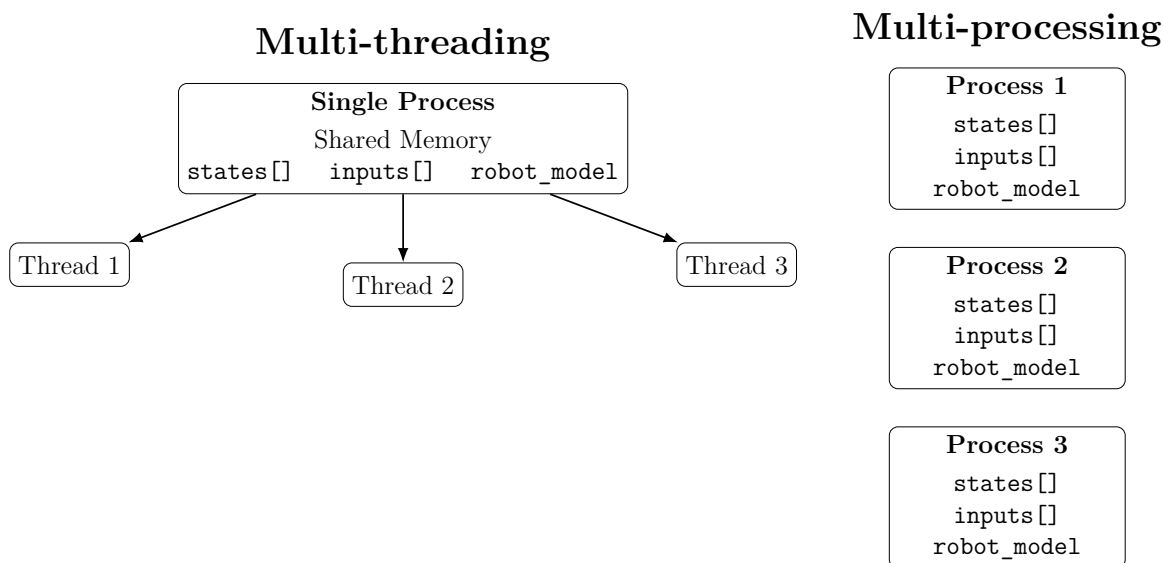


Figure 3.8: Comparison between multi-threading and multi-processing architectures. In multi-threading, several execution threads share the same memory space. In multi-processing, each process owns an independent memory space, improving robustness and isolation at the cost of increased memory usage.

## Many Small Jobs Instead of One Large Job

Rather than launching one large job requesting many processors and a long wall time, the workload is divided into many smaller independent jobs. This is better adapted to a shared cluster environment. Small jobs are easier for the scheduler to place, can often

start earlier, and are less constrained by maximum wall-time limits.

A single large job is also inherently limited by the maximum wall time allowed on the cluster. By contrast, the job-array approach can process abstractions whose total computation time exceeds this limit, because each individual job remains short. The total computation is then obtained by combining the results of many short executions.

This is also a direct advantage of the multiprocessing formulation. Since different jobs do not have access to each other’s memory, the implementation does not require any shared-memory coordination between workers. Each job is simply launched independently and receives its own `jobid`; together with the total number of jobs, this is sufficient to determine the part of the linear workload it has to compute. By contrast, a multithreaded implementation would typically require launching a single large job, because all threads must belong to the same execution and share the same memory space. Such a job would request many processors and a long wall time at once, making it harder for the cluster scheduler to place in the queue. In practice, it may be delayed repeatedly and, in extreme cases, may never start within the available scheduling window.

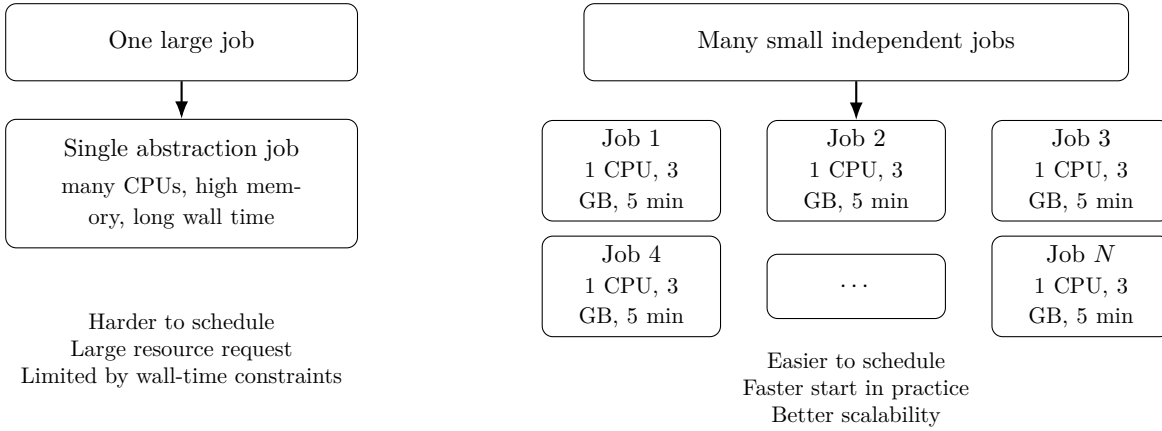


Figure 3.9: Comparison between one large job and many small independent jobs.

## Generality of the Implementation

A major advantage of the proposed implementation is that it is not specific to the bipedal robot considered in this work. The same distributed structure can be applied to many abstraction-based control problems, provided that the problem is defined by a finite set of states, a finite set of inputs, and a transition function that can be evaluated independently for each state–input pair.

In the present implementation, the problem-dependent part is limited to the definition of the state grid, the input grid, and the transition map. For a standard tensor grid, the continuous state and input domains are discretised with possibly dimension-dependent steps:

$$h_x = (h_{x,1}, \dots, h_{x,n_x}), \quad h_u = (h_{u,1}, \dots, h_{u,n_u}).$$

The state and input dimensions can therefore be arbitrary, and each dimension can have its own range and resolution.

The same principle could also be extended to grids with non-uniform discretisation steps inside a given dimension. In that case, instead of recovering the physical value from an affine formula such as

$$x_{0,\ell} + k_\ell h_{x,\ell},$$

each coordinate index would be mapped to a precomputed list of grid values or intervals along that dimension. The global partitioning strategy would remain unchanged: only the local index-to-state reconstruction step would need to be adapted.

Additional constraints can also be included without modifying the distributed architecture. For instance, the predicate-based pruning strategy introduced in Section 3.4 can be incorporated by evaluating a predicate before simulating a transition. If the predicate is false, the corresponding state-input pair is discarded; otherwise, the transition is evaluated normally. Thus, physical constraints, admissibility conditions, or problem-specific exclusions can be added easily.

The distributed execution itself remains very simple. The same problem description is given to all processors: the state grid, the input grid, the transition function, and, if needed, a pruning predicate. Each processor is then given only two additional pieces of information: its own job identifier and the total number of jobs. From these two integers, it automatically computes its assigned interval of linear indices using the deterministic partition rule introduced above. It then reconstructs the corresponding state-input pairs locally, applies the optional pruning predicate, simulates the remaining transitions, and writes its own partial output file.

This makes the implementation simple and scalable. Adding more processors does not require changing the abstraction code or manually distributing the state space. One only increases the total number of jobs; each job receives a different identifier, computes its own part of the linear workload, and runs independently. The limiting factors are then mainly scheduling, file storage, and merging overhead, rather than communication or shared-memory coordination.

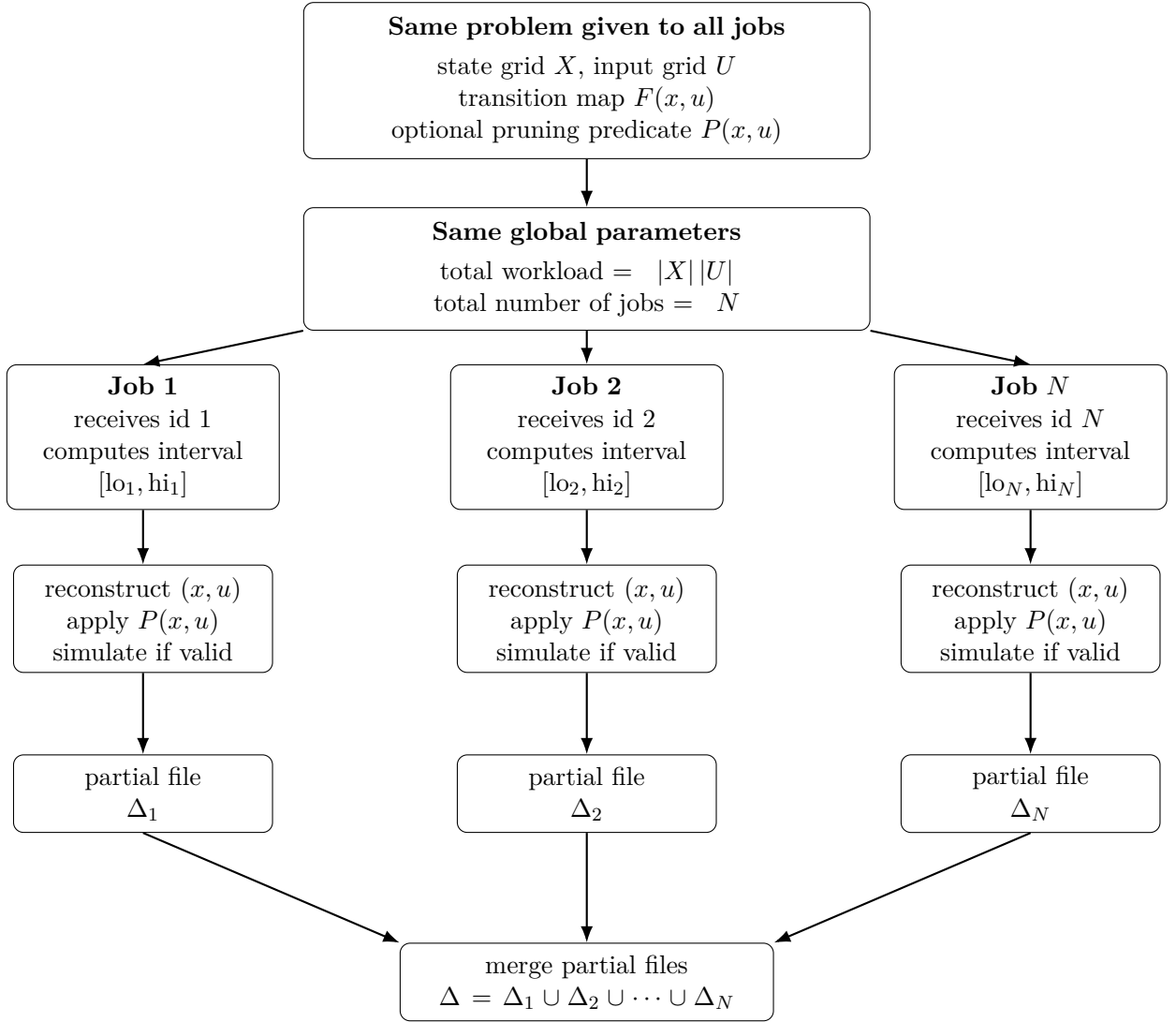


Figure 3.10: Generic structure of the distributed abstraction generator.

Figure 3.10 summarizes the generality of the approach. The same abstraction problem is given to all jobs. The only difference between them is their identifier. Together with the total number of jobs, this identifier is sufficient for each job to determine its own part of the linear workload. Predicate-based pruning can also be applied locally before simulation, without changing the distributed structure.

### 3.3.5 Scalability of Transition Generation

The main benefit of the distributed implementation is that the transition generation time can be reduced almost proportionally to the number of available workers, as long as the workload is sufficiently large and the merge overhead remains negligible.

For an abstraction containing  $N_{\text{trans}}$  transitions, with an average simulation time  $\tau_{\text{trans}}$

per transition, the ideal computation time on  $P$  processors is

$$T(P) = \frac{N_{\text{trans}} \tau_{\text{trans}}}{P}.$$

For example, with

$$N_{\text{trans}} = 10^{10}, \quad \tau_{\text{trans}} = 3 \text{ ms},$$

the single-processor computation time would be approximately

$$T(1) = 10^{10} \times 3 \times 10^{-3} = 3 \times 10^7 \text{ s} \simeq 8333 \text{ h}.$$

This illustrates why distributed computation is necessary: even a moderately fast simulator becomes impractical when the abstraction contains billions of transitions.

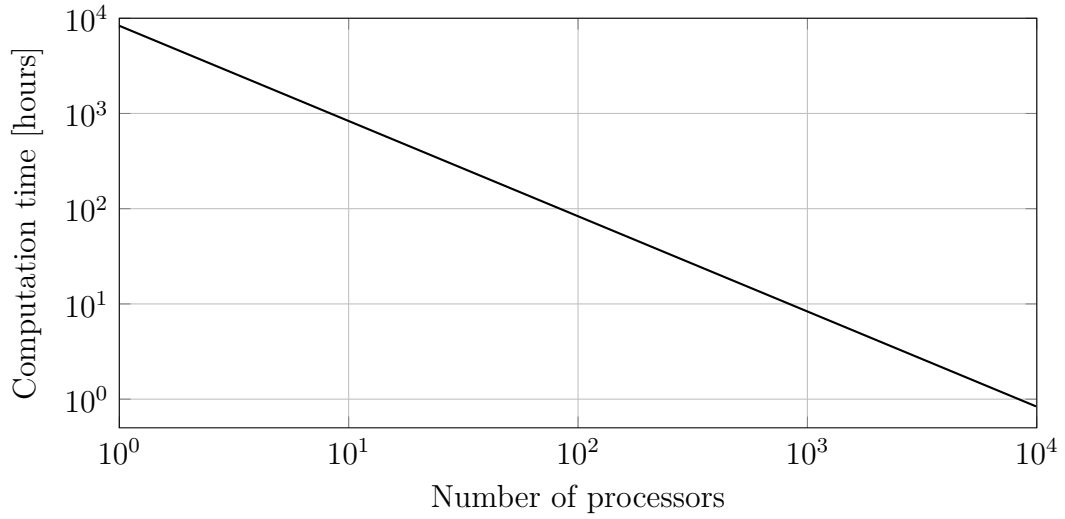


Figure 3.11: Estimated abstraction-generation time for an abstraction containing  $10^{10}$  transitions, assuming 3 ms per transition and ideal parallel scaling.

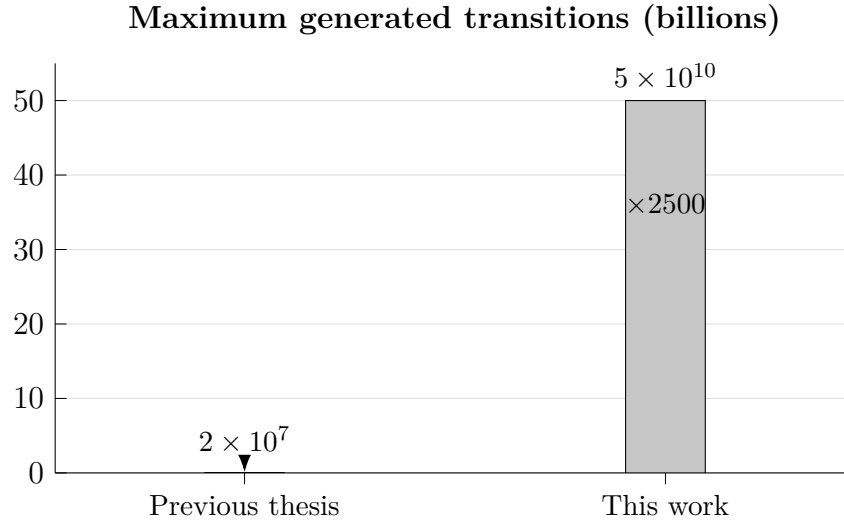


Figure 3.12: Comparison of the largest abstraction sizes generated in the previous thesis and in the present work. Values are expressed in billions of transitions: the previous implementation reached approximately 0.02 billion transitions, whereas the distributed implementation developed here reached approximately 50 billion transitions.

The ideal scaling curve only accounts for transition computation. In practice, the wall-clock time also includes cluster scheduling effects. Figure 3.13 separates the measured runtime into queue time and computation time for an abstraction involving  $10^7$  transitions. The computations were performed on the Lemaitre cluster of the CÉCI high-performance computing infrastructure hosted at UCLouvain/CISM [6].

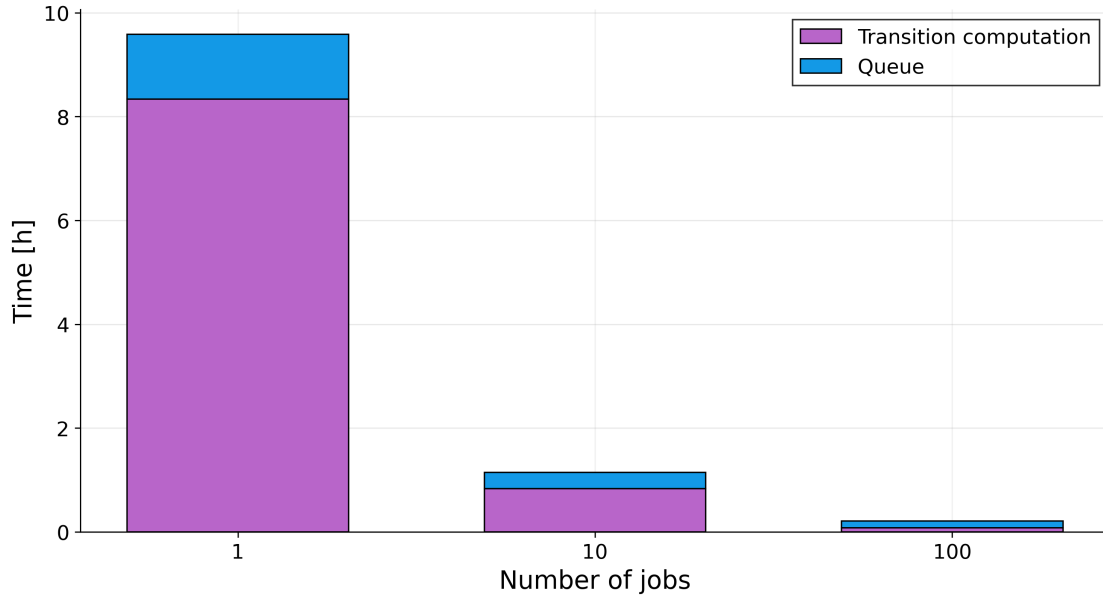


Figure 3.13: Runtime breakdown for the transition-generation step as a function of the number of jobs for an abstraction involving  $10^7$  transitions.

The computation time decreases as the number of jobs increases, confirming that tran-

sition generation is highly parallelizable. This behaviour is expected, because each state–input pair is simulated independently.

The queue time also decreases substantially when the workload is split into many small jobs. This is not an algorithmic effect, but a practical scheduling advantage: small independent jobs are easier for the cluster scheduler to allocate than one large job requesting many processors simultaneously. Thus, the job-array strategy improves scalability in two complementary ways. It reduces the computation time through parallelism, and it improves practical turnaround time by making the workload easier to schedule.

### 3.3.6 Memory Bottleneck for Controller Synthesis

The previous results show that transition generation can be distributed efficiently. However, generating the transitions is only the first step. The resulting transition relation must also be stored and processed during controller synthesis. For very large abstractions, this storage requirement becomes the dominant limitation [12].

In a basic implementation, each transition can be stored as a tuple containing the arrival state, the source state, the input index, and the edge cost. With 64-bit integers for the three indices and a 64-bit floating-point value for the cost, this requires

$$8 + 8 + 8 + 8 = 32$$

bytes per transition. If  $E$  transitions are stored, the raw memory requirement is therefore

$$M_{\text{raw}}(E) = 32E.$$

This linear scaling was confirmed experimentally by storing the raw transition tuples generated for the 8D abstraction. The benchmark reached  $2.325 \times 10^9$  transitions, corresponding to approximately 74.4 GB of raw tuple storage.

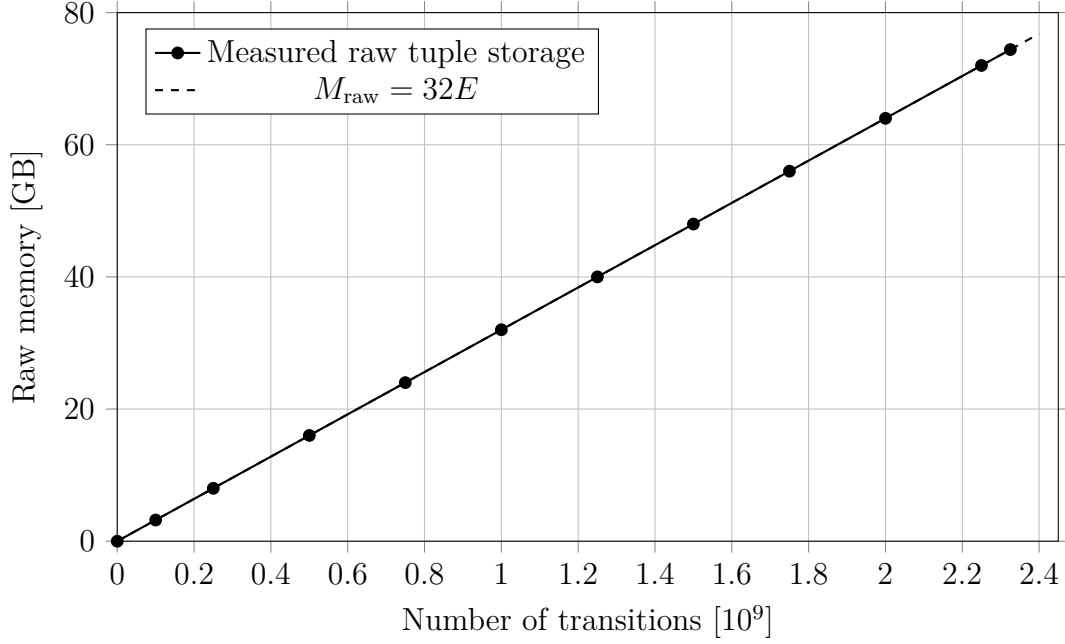


Figure 3.14: Raw memory required to store the transition relation as explicit tuples. The memory grows linearly with the number of transitions, with a slope of approximately 32 bytes per transition.

This shows that, beyond a certain scale, the bottleneck is no longer only the simulation of the transitions. Even if the transitions can be generated by many independent jobs, solving the resulting controller may require storing a graph that exceeds the memory available on a single cluster node.

A more compact representation can reduce this memory pressure. In the backward Dijkstra formulation, transitions are grouped by arrival state. The arrival state does not need to be stored inside every edge, because it is already implicit in the predecessor list being accessed. Each edge can therefore store only an encoded source–input pair and the associated cost:

$$\text{edge} = (\text{encoded}(s, a), c).$$

Using a 64-bit encoded identifier and a 32-bit floating-point cost, the aligned edge representation occupies approximately 16 bytes. This reduces the graph storage by roughly a factor two:

$$M_{\text{opt}}(E) \simeq 16E.$$

With a memory limit of approximately 750 GB, this representation gives the theoretical upper bound

$$E_{\text{max}} = \frac{750 \times 10^9}{16} \simeq 4.69 \times 10^{10}$$

stored transitions for the graph representation alone. This value must be interpreted as an optimistic bound, because the actual solver also requires additional arrays for Dijkstra, compact state mappings, priority queues, and runtime overhead.

This explains the practical limitation observed in the experiments: abstractions with up to approximately  $50 \times 10^9$  transitions could be generated, but they could not be solved on the available cluster nodes because the memory required by the graph and the associated solver data structures exceeded the available RAM.

A possible direction for future work is to replace the in-memory graph solver by an out-of-core implementation. In such an approach, the predecessor graph would be stored on disk in a compact format indexed by arrival state. This could take advantage of the much larger scratch storage available on the cluster. However, it would also introduce a new bottleneck: Dijkstra’s algorithm requires many irregular accesses to predecessor lists, so disk latency and bandwidth could become limiting. Designing such an out-of-core solver is therefore a relevant but non-trivial extension.

### 3.3.7 Conclusion

This section introduced a distributed implementation of the transition-generation step used in abstraction-based control. The abstraction workload is flattened into a linear index set, partitioned deterministically into disjoint intervals, and evaluated by independent jobs. Each job reconstructs its assigned state–input pairs locally, simulates the corresponding transitions, and writes a partial transition file. The complete transition relation is then obtained by merging these files.

The advantage of this approach is that it preserves the exactness of the transition-generation procedure while making it scalable across independent processors or machines. It does not require shared memory, and it is therefore well suited not only to HPC clusters, but also to more general distributed computing environments.

However, this strategy only accelerates the generation of transitions. It does not reduce the size of the abstraction itself. For very large grids, the number of generated transitions may still become too large to store and solve in memory. This motivates the complementary strategy studied in the next section: rather than distributing the full transition set more efficiently, pruning aims to reduce the effective abstraction domain before the transitions are simulated.

## 3.4 Pruning Strategies for Reducing the Effective Abstraction Domain

The distributed implementation presented in the previous section improves the execution of abstraction generation, but it does not change the number of state–input pairs that must be considered. If the discretised domain contains physically irrelevant, unreachable, or uninformative regions, distributing their simulation only makes the computation faster; it does not remove unnecessary work [12].

A complementary approach is therefore to reduce the effective abstraction domain before transition generation. Instead of evaluating all pairs in  $X \times U$ , one restricts the computation to a subset of meaningful state–input pairs. This does not modify the abstraction machinery itself. It acts upstream by excluding regions that are inconsistent with the targeted walking behaviour or unlikely to contribute to the final controller.

### 3.4.1 Principles for Designing Pruning Rules

Although pruning is problem-dependent in its concrete form, its design can be guided by a number of general principles.

1. Pruning should be *goal-oriented*. The purpose is not to approximate the entire physically reachable set of the system, but to focus the abstraction on the subset relevant to the control objective. The admissible state and input sets should therefore reflect the behaviour one aims to capture: periodic walking, recovery motions, balanced postures, energy-efficient manoeuvres, or any other targeted regime.
2. Pruning should be *cheap to evaluate*. Since these predicates are applied before simulation, they should be much less expensive than the transition computation they replace. Simple inequalities, sign tests, ordering conditions, and low-dimensional algebraic relations are therefore especially valuable.
3. Pruning should be *structured*. The most effective rules are not isolated thresholds, but collections of mutually consistent conditions that encode a coherent behavioural interpretation of the state and input. In particular, mode-based pruning is often much more informative than global box-type restrictions.
4. Pruning should be *explicitly conservative with respect to the intended use*. If the abstraction is meant to cover all behaviours in a certain class, the pruning rules must not exclude states or inputs that are genuinely needed for that class. In practice, pruning is therefore a trade-off between computational savings and behavioural coverage. Excessively aggressive pruning may reduce computation time dramatically, but at the cost of producing a controller that is only valid in an overly narrow operating region.
5. Pruning should be *hierarchical whenever possible*. A good implementation first applies a state predicate  $P_X(x)$ , then an input predicate  $P_U(x, u)$ , and only then launches the expensive simulation. This ordering avoids spending effort on input checks for already inadmissible states, and avoids spending effort on simulation for already inadmissible state–input pairs.

### 3.4.2 Generic Implementation Pattern

Formally, let  $X$  denote the discretised state space and  $U$  the discretised input space. A brute-force abstraction would evaluate transitions for every pair  $(x, u) \in X \times U$ . Pruning introduces a Boolean selection rule

$$P : X \times U \rightarrow \{0, 1\},$$

and only pairs satisfying  $P(x, u) = 1$  are retained for simulation. In practice, this selection rule is often decomposed into two stages:

$$P(x, u) = P_X(x) \wedge P_U(x, u),$$

where  $P_X$  is a state-admissibility predicate and  $P_U$  is an input-admissibility predicate conditioned on the current state. This decomposition is particularly useful because it separates two distinct questions: first, whether a state belongs to the region of interest at all; second, whether a given control action is meaningful in that state. From an algorithmic viewpoint, these ideas lead to the following generic structure:

$$\text{for each } x \in X : \begin{cases} \text{discard } x & \text{if } P_X(x) = 0, \\ \text{for each } u \in U : & \\ \quad \text{discard } u & \text{if } P_U(x, u) = 0, \\ \quad \text{otherwise evaluate the transition.} & \end{cases}$$

The practical impact of pruning is therefore twofold: it reduces the number of state cells that must be explored and, within each retained state, it reduces the number of candidate inputs that must be simulated. Since the dominant cost of abstraction generation lies in repeated transition evaluation, even simple pruning rules can yield substantial computational savings when applied systematically.

In the present problem, such predicates were essential to obtain a pruning strategy that is both intuitive and sufficiently selective. Restricting the search space only through global hyper-rectangular bounds like previously implemented in Dionysos would leave many state cells that are mathematically inside the domain but physically irrelevant for walking. For example, configurations in which the hip angles are not approximately antisymmetric may still lie inside the discretisation box while being clearly incompatible with a meaningful gait phase.

In the new version of Dionysos, we added the functions:

- `in_gait_tube(x)` that removes these regions directly by encoding simple walking-structure constraints, thereby discarding large portions of the state grid before any simulation is performed.
- `input_allowed(x,u)` that eliminates wide subsets of the input grid that are inconsistent with the current state.

---

**Algorithm 1** `in_gait_tube(x)` example

---

**Require:** State  $x$

- 1: **if**  $x$  violates the approximate anti-symmetry constraint between the hips **then**
  - 2:     **return** false
  - 3: **end if**
  - 4: **if**  $x$  does not satisfy the desired leg-clearance configuration **then**
  - 5:     **return** false
  - 6: **end if**
  - 7: **return** true
- 

This is what makes predicate-based pruning especially powerful here: it allows large regions to be excluded for transparent mechanical reasons, rather than through coarse geometric cuts alone.

Figure 3.15 illustrates why predicate-based pruning is significantly more expressive than the original hyper-rectangular pruning mechanism. For the biped robot, even a simple condition such as approximate hip-angle antisymmetry defines a narrow oblique band in the state space, which cannot be represented efficiently by axis-aligned boxes alone.

### State pruning in the hip-angle plane

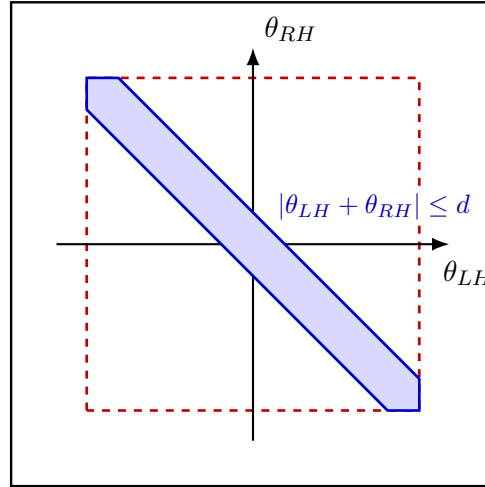


Figure 3.15: Predicate-based state pruning can follow the geometry of the problem. In the  $(\theta_{LH}, \theta_{RH})$  plane, approximate hip-angle antisymmetry defines the band  $|\theta_{LH} + \theta_{RH}| \leq d$ , whereas coarse box-based pruning would require keeping the full enclosing square.

### 3.4.3 Swing-Foot Clearance as a Directional Feasibility Constraint

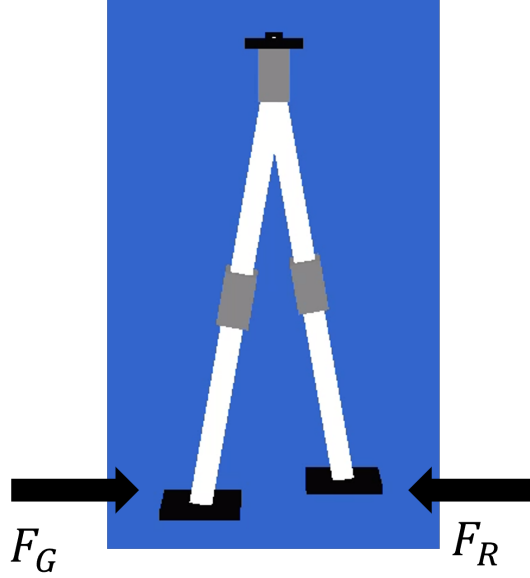


Figure 3.16: Horizontal ground reaction forces applied on each foot during walking.

Since the reduced abstraction only contains joint angles and joint velocities, a posture target alone does not guarantee a genuine forward step. The robot may reach the target posture while keeping both feet close to the ground, leading to slipping or quasi-stationary leg exchange.

The need for forward motion can be understood from Newton's second law:

$$\sum F_x = m\ddot{x}.$$

If the horizontal contact forces applied at the two feet compensate each other, then

$$F_G = F_R \Rightarrow \sum F_x = 0 \Rightarrow \ddot{x} = 0.$$

In that case, no horizontal acceleration is generated. The role of the clearance constraint is therefore not to directly impose the forces, but to prevent persistent double contact configurations where such force compensation can occur. By requiring the swing foot to lift, the abstraction favours motions in which the support is transferred and the robot can progress forward.

Let the reduced state be

$$\mathbf{x} = (\theta_{LH}, \theta_{RH}, \theta_{LK}, \theta_{RK}, \dot{\theta}_{LH}, \dot{\theta}_{RH}, \dot{\theta}_{LK}, \dot{\theta}_{RK}).$$

The effective vertical positions of the left and right feet, measured from the hip, are

$$z_L(\mathbf{x}) = L_{\text{thigh}} \cos(\theta_{LH}) + L_{\text{leg}} \cos(\theta_{LH} + \theta_{LK}),$$

$$z_R(\mathbf{x}) = L_{\text{thigh}} \cos(\theta_{RH}) + L_{\text{leg}} \cos(\theta_{RH} + \theta_{RK}).$$

Assuming that the right foot is the stance foot, the left-foot clearance is defined as

$$h_L(\mathbf{x}) = z_R(\mathbf{x}) - z_L(\mathbf{x}).$$

Thus,  $h_L(\mathbf{x}) > 0$  means that the left foot is higher than the right stance foot, while  $h_L(\mathbf{x}) \simeq 0$  corresponds to persistent double contact.

The clearance predicate is activated only when the swing foot is in the central part of the step. Let

$$\Delta x_{\text{step}} = x_{L,\text{final}} - x_{L,\text{init}}$$

be the planned horizontal displacement of the swing foot, and let

$$x_{L,\text{mid}} = \frac{x_{L,\text{init}} + x_{L,\text{final}}}{2}$$

be the centre of the step. The admissibility predicate is then

$$P_X(\mathbf{x}) := (x_L(\mathbf{x}) \in [x_{L,\text{mid}} - 0.4\Delta x_{\text{step}}, x_{L,\text{mid}} + 0.4\Delta x_{\text{step}}]) \Rightarrow (h_L(\mathbf{x}) \geq h_{\min}),$$

with

$$h_{\min} = 0.03 \text{ m}.$$

Here,  $x_L(\mathbf{x})$  is the horizontal position of the swing foot,  $x_{L,\text{init}}$  and  $x_{L,\text{final}}$  are its initial and final positions,  $\Delta x_{\text{step}}$  is the step length, and  $h_{\min}$  is the required minimum clearance. The factor 0.4 on each side defines a central interval covering 80% of the step. This avoids enforcing clearance near take-off and touchdown, where the swing foot is expected to be close to the ground.



(a) Synthesized controller without the clearance constraint



(b) Synthesized controller under the clearance constraint

Figure 3.17: QR codes providing access to the synthesized walking controllers.

## 3.5 Conclusion

This chapter investigated three complementary strategies for improving the scalability of abstraction-based control for planar bipedal robots. Symmetry exploitation reduces redundant computations, distributed transition generation accelerates abstraction construction, and pruning strategies decrease the number of transitions that must be

evaluated. Combined, these approaches enable significantly larger abstractions while preserving computational tractability.

However, increasing the abstraction size also amplifies the impact of discretization errors, local sensitivity, and hybrid transition effects. Consequently, scalability alone is insufficient: large abstractions must also remain sufficiently accurate and robust for reliable controller synthesis.

The next chapter therefore focuses on robustness and sensitivity analysis, with particular emphasis on Lipschitz-based error propagation and adaptive refinement strategies for improving abstraction fidelity while maintaining tractable computation times.

# Chapter 4

## Sensitivity and Error Propagation in Abstraction-Based Control

### 4.1 Introduction

The previous chapter focused on improving the computational scalability of abstraction generation for planar bipedal robots. However, enlarging the abstraction domain also amplifies the impact of discretization errors and local sensitivity effects. In nonlinear hybrid robotic systems, small perturbations may propagate non-uniformly across the state space, particularly near contact transitions where local trajectory divergence can increase significantly [26].

As a consequence, abstraction quality depends not only on the discretization resolution, but also on how local errors propagate through the dynamics. Understanding these mechanisms is therefore essential for designing abstractions that remain sufficiently accurate and robust for controller synthesis.

The objective of this chapter is to analyze the origin and propagation of approximation errors in abstraction-based control for planar bipedal robots. Particular emphasis is placed on local sensitivity amplification, Lipschitz-based error propagation, Jacobian interpretation, and empirical sensitivity estimation for hybrid robotic dynamics.

### 4.2 Sources of Approximation Errors in Symbolic Abstractions

The principal approximation mechanisms associated with symbolic abstractions were already introduced in Section 2.3. The present section revisits these errors in the specific context of abstraction-based control with cell center simulation, with particular emphasis on how local sensitivity amplification influences abstraction robustness and

transition conservatism.

Symbolic abstractions approximate the continuous dynamics by replacing each continuous state with a representative point, typically the center of the corresponding cell. In our implementation, for each cell center  $x_c$  and each admissible input  $u$ , the system is simulated over one sampling period. The resulting state  $F(x_c, u)$  is then projected onto the abstraction grid through the quantizer  $Q$ , producing the symbolic successor  $Q(F(x_c, u))$ .

When the synthesized controller is applied to the concrete system, however, the actual state  $x_k$  is generally not exactly equal to the cell center  $x_c$ . The transition followed by the real system is therefore  $F(x_k, u)$ , whereas the abstraction represents it by  $Q(F(x_c, u))$ . The resulting transition error can be decomposed into two contributions, as illustrated in Figure 4.1.

As these errors are independant, the triangle inequality provides an interesting bound on the error of a given transition :

$$\|F(x_k, u) - Q(F(x_c, u))\| \leq \underbrace{\|F(x_k, u) - F(x_c, u)\|}_{\text{Local error propagation}} + \underbrace{\|F(x_c, u) - Q(F(x_c, u))\|}_{\text{Successor quantization error}}. \quad (4.1)$$

### Local error propagation

The first term measures how an initial mismatch inside the same cell is amplified by the dynamics. If  $F$  is locally Lipschitz on a cell  $C_i$ , with local Lipschitz constant  $L_i$ , then for  $x_k, x_c \in C_i$ ,

$$\|F(x_k, u) - F(x_c, u)\| \leq L_i \|x_k - x_c\|.$$

Let  $\eta_i$  denote the radius of cell  $C_i$ . Since both  $x_k$  and  $x_c$  belong to the same cell,

$$\|x_k - x_c\| \leq \eta_i,$$

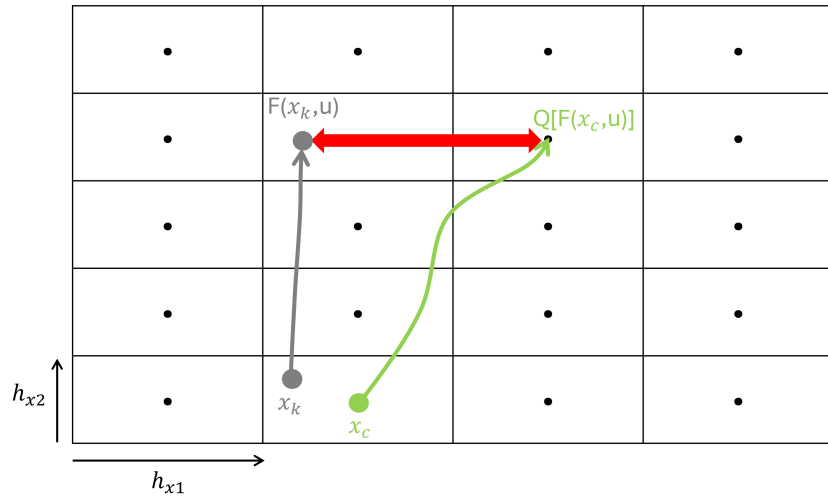
and therefore

$$\|F(x_k, u) - F(x_c, u)\| \leq L_i \eta_i. \quad (4.2)$$

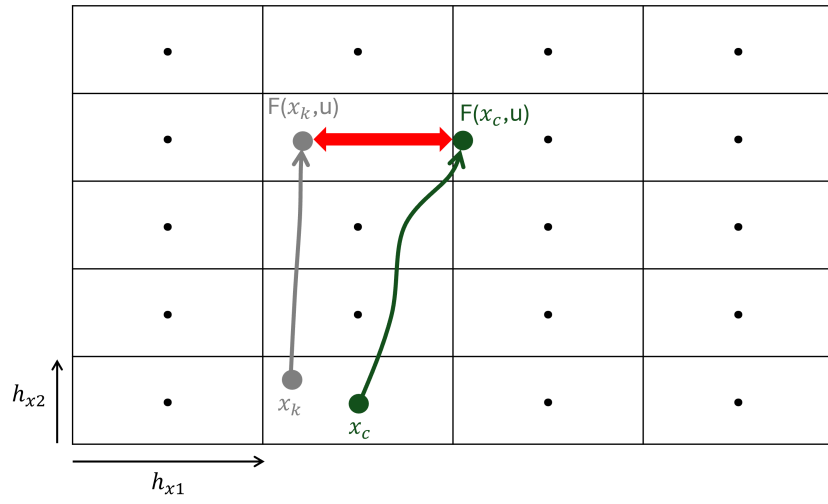
Hence, the local propagated error is governed by the product

$$e_i^{\text{prop}} \lesssim L_i \eta_i.$$

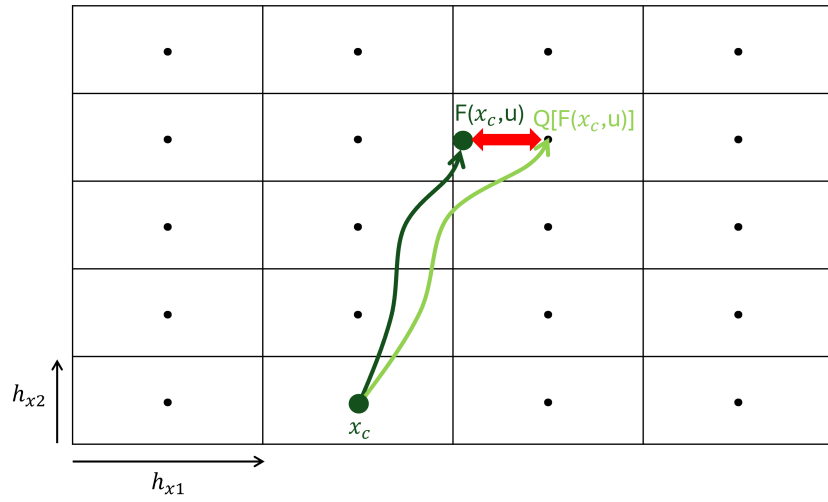
This relation is important because it separates two effects: the size of the cell, represented by  $\eta_i$ , and the local sensitivity of the dynamics, represented by  $L_i$ . If  $L_i$  is large, even a small initial displacement inside the cell can lead to a large successor error. Conversely, if  $L_i$  is small, the same cell size may remain acceptable.



(a) Total transition error



(b) Local error propagation



(c) Successor quantization error

Figure 4.1: Decomposition of the total transition error into local error propagation and successor quantization error.

## Successor quantization error

The second term corresponds to the error introduced when the continuous successor of the cell center is projected back onto the abstraction grid:

$$\|F(x_c, u) - Q(F(x_c, u))\|.$$

Unlike the local propagation error, this term does not depend on the sensitivity of the dynamics to the initial condition. It only depends on the resolution of the abstraction grid around the successor state. If  $F(x_c, u)$  belongs to a successor cell  $C_j$ , and if  $\eta_j$  denotes the radius of that cell, then the quantization error satisfies

$$\|F(x_c, u) - Q(F(x_c, u))\| \leq \eta_j.$$

For a uniform grid, or when the cells have comparable sizes, this term is therefore of the order

$$e^{\text{quant}} \sim \eta.$$

This means that the successor quantization error is essentially controlled by the grid resolution. Reducing this error generally requires reducing the size of the cells, and therefore increasing the total number of symbolic states.

## Comparison of the two error mechanisms

The two terms have different origins and different scalings. The local propagation error scales as

$$e_i^{\text{prop}} \sim L_i \eta_i,$$

whereas the successor quantization error scales as

$$e^{\text{quant}} \sim \eta.$$

The comparison between these two quantities is therefore mainly governed by the local amplification factor  $L_i$ . If  $L_i < 1$ , the dynamics is locally contractive, and the propagation of the initial error remains smaller than the quantization error. If  $L_i \approx 1$ , both errors are of the same order. However, if  $L_i \gg 1$ , even a small displacement inside the cell can be strongly amplified, and the local propagation error becomes the dominant contribution.

This distinction is important because the two errors do not suggest the same refinement strategy. If the successor quantization error dominates, the most direct way to reduce the abstraction error is to refine the grid uniformly. This is the standard approach used in uniform symbolic abstractions. In that case, all cells are reduced in size because the error is mainly controlled by the geometric resolution of the grid.

By contrast, if the local propagation error dominates, the relevant quantity is not the cell size alone, but the product

$$L_i \eta_i.$$

In this case, two cells with the same size may have very different propagated errors if their local Lipschitz constants are different. It then becomes more efficient to refine the cells where the dynamics is highly sensitive, rather than refining the whole state space uniformly.

This motivates an adaptive refinement strategy. Suppose that the state space is partitioned into  $N$  cells  $C_i$ , each with radius  $\eta_i$  and local Lipschitz constant  $L_i$ . The local propagated error satisfies

$$e_i^{\text{prop}} \leq L_i \eta_i.$$

If one aims to minimize the worst propagated error under a fixed cell budget, the idealized refinement problem can be written as

$$\begin{aligned} \min_{\eta_i > 0} \quad & \max_i L_i \eta_i \\ \text{subject to} \quad & \text{a fixed cell budget.} \end{aligned} \tag{4.3}$$

At the optimum, the active propagated errors should be balanced:

$$L_i \eta_i^* = \varepsilon. \tag{4.4}$$

This shows that cells with larger local Lipschitz constants should receive smaller radii, while less sensitive cells can remain coarser. In other words, the product  $L_i \eta_i$  provides a principled criterion for redistributing the available resolution toward dynamically sensitive regions.

## 4.3 Lipschitz-Based Error Propagation

The previous section showed that the local propagation error inside a cell is controlled by a product of the form  $L_i \eta_i$ , where  $\eta_i$  measures the size of the cell and  $L_i$  measures the local sensitivity of the dynamics. This section explains how this sensitivity is defined and why it cannot be computed exhaustively in practice.

Lipschitz continuity is commonly used in nonlinear control and abstraction-based verification because it provides a way to bound how local perturbations are amplified by the dynamics [21, 25, 24].

### 4.3.1 Weighted norm and local Lipschitz bound

Let  $C_i \subset \mathbb{R}^8$  be a cell of the state space, and let

$$\mathcal{U} = \{u_1, \dots, u_m\}$$

be the finite set of admissible inputs. For a given input  $u$ , the one-step transition map is denoted by

$$F(\cdot, u) : \mathbb{R}^8 \rightarrow \mathbb{R}^8.$$

Since the state contains both angular positions and angular velocities, a weighted norm is used:

$$\|z\|_W = \|Wz\|_2,$$

with

$$W = \text{diag}(1, 1, 1, 1, T, T, T, T), \quad T = T_{\text{step}}.$$

This weighting makes position and velocity components comparable in the metric used to measure perturbations.

For a given cell  $C_i$  and input  $u$ , a local Lipschitz constant  $L_i(u)$  satisfies

$$\forall x, y \in C_i, \quad \|F(x, u) - F(y, u)\|_W \leq L_i(u) \|x - y\|_W.$$

Since the controller has not yet been synthesized when the abstraction is built, the input that will be selected in each cell is not known in advance. We therefore consider a conservative cell-wise constant valid for all admissible inputs:

$$L_i = \max_{u \in \mathcal{U}} L_i(u).$$

With this definition, for every  $u \in \mathcal{U}$  and every  $x, y \in C_i$ ,

$$\|F(x, u) - F(y, u)\|_W \leq L_i \|x - y\|_W.$$

If  $F(\cdot, u)$  is differentiable on  $C_i$ , then a sufficient Lipschitz constant is obtained by bounding the weighted norm of the Jacobian over the cell:

$$L_i(u) = \sup_{x \in C_i} \|W J_x F(x, u) W^{-1}\|_2,$$

where

$$J_x F(x, u) = \frac{\partial F}{\partial x}(x, u) \in \mathbb{R}^{8 \times 8}.$$

The corresponding cell-wise Lipschitz constant is therefore

$$L_i = \max_{u \in \mathcal{U}} \sup_{x \in C_i} \|W J_x F(x, u) W^{-1}\|_2.$$

For clarity, we denote the pointwise local amplification factor by

$$\ell(x, u) = \|W J_x F(x, u) W^{-1}\|_2.$$

Thus, the theoretical cell-wise constant can be written compactly as

$$L_i = \max_{u \in \mathcal{U}} \sup_{x \in C_i} \ell(x, u).$$

This quantity is a worst-case measure: it describes the largest local amplification that can occur anywhere in the cell and for any admissible input.

### 4.3.2 Finite-difference estimation of local amplification

The previous definition is theoretical. Computing  $L_i$  exactly would require bounding  $\ell(x, u)$  over the whole continuous cell  $C_i$ , and for every admissible input. This is not directly available for the black-box simulation model considered in this work.

A possible way to estimate the local amplification around a given state-input pair  $(x, u)$  is to approximate the Jacobian of  $F(\cdot, u)$  by centered finite differences. For the  $r$ -th state component,

$$\frac{\partial F}{\partial x_r}(x, u) \approx \frac{F(x + \delta_r e_r, u) - F(x - \delta_r e_r, u)}{2\delta_r},$$

where  $e_r$  is the  $r$ -th canonical basis vector and

$$\delta_r = \varepsilon_{\text{frac}} s_r.$$

The scale  $s_r$  is chosen according to the physical unit of the component:

$$s_r = \begin{cases} 10^\circ, & \text{for angular position components,} \\ 100^\circ/\text{s}, & \text{for angular velocity components.} \end{cases}$$

This gives a numerical Jacobian  $\hat{J}_\varepsilon(x, u)$ , and therefore a pointwise estimate of the local amplification:

$$\hat{\ell}_\varepsilon(x, u) = \left\| W \hat{J}_\varepsilon(x, u) W^{-1} \right\|_2.$$

The perturbation factor  $\varepsilon_{\text{frac}}$  must be chosen in an intermediate range. If it is too small, the finite-difference quotient becomes dominated by numerical noise. If it is too large, it no longer represents the local derivative, but rather a larger-scale secant slope affected by nonlinearities and possible hybrid events.

Figure 4.2 shows this effect for one local diagnostic evaluation. A stable plateau appears around

$$\varepsilon_{\text{frac}} \in [10^{-3}, 10^{-2}].$$

This suggests that, at this sampled state-input pair, the estimated local amplification is approximately independent of the perturbation size in this interval.

With the chosen component scalings, the upper end of this stable interval,  $\varepsilon_{\text{frac}} \approx 10^{-2}$ , corresponds to perturbation radii of approximately

$$r_q = 0.1^\circ, \quad r_{\dot{q}} = 1^\circ/\text{s}.$$

At this scale, the finite-difference estimate can therefore be interpreted as a reliable local Lipschitz diagnostic around the sampled state-input pair. However, this remains a local estimate; it is not a certified bound over the whole cell  $C_i$ .

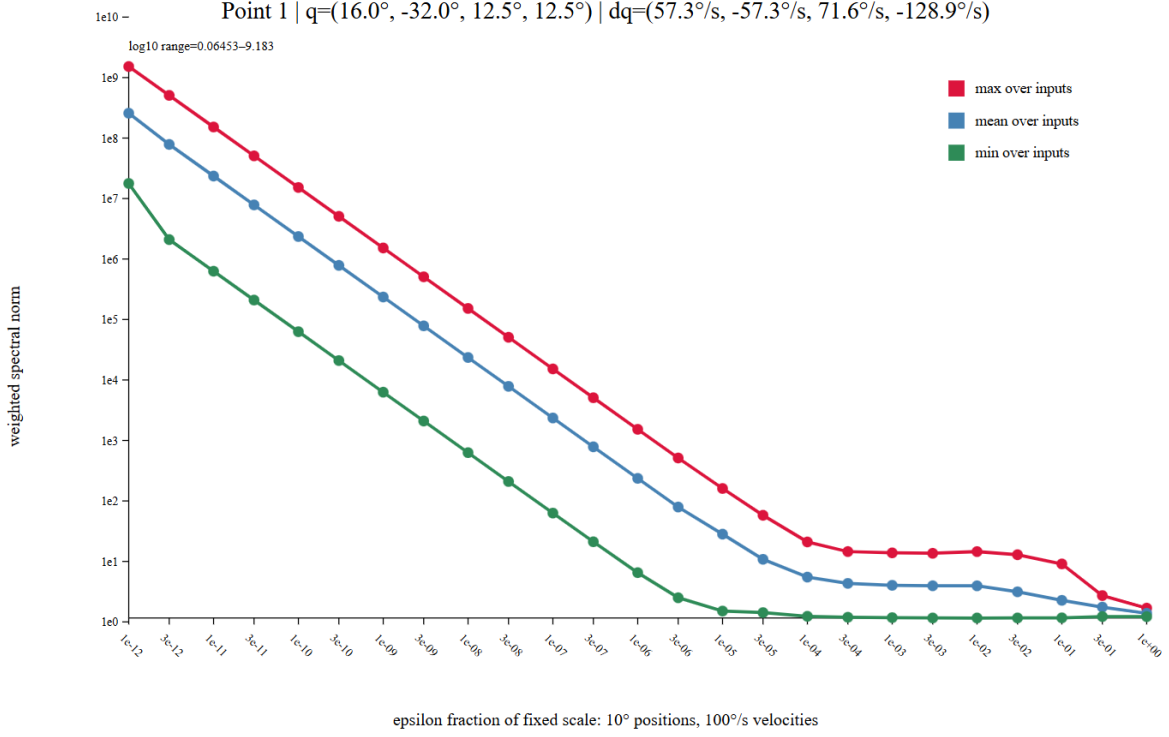


Figure 4.2: Estimated weighted spectral norm  $\widehat{\ell}_\varepsilon(x, u)$  as a function of the finite-difference perturbation scale  $\varepsilon_{\text{frac}}$  for one local diagnostic evaluation.

### 4.3.3 Why exhaustive Lipschitz estimation is infeasible

The finite-difference procedure above only estimates a local amplification factor around one sampled state-input pair. Extending this idea to the full abstraction would require estimating such quantities for many sampled states in each cell, for all cells, and for every admissible input [28, 25, 14].

This is computationally impossible at the scale of the considered abstraction. In an 8-dimensional state space, one centered finite-difference Jacobian already requires 16 perturbed simulations for a single state-input pair. This cost would have to be multiplied by the number of sampled states, the number of inputs, and the number of cells.

The magnitude of the problem can be estimated by considering cell sizes consistent with the local radii for which the finite-difference estimate is meaningful:

$$r_q = 0.1^\circ, \quad r_{\dot{q}} = 1^\circ/\text{s}.$$

Over the considered state-space bounds, this would lead to approximately

$$N_{\text{cells}} \approx 8.89 \times 10^{18}$$

cells. With 16 admissible inputs, this already corresponds to

$$N_{\text{trans}} \approx 1.42 \times 10^{20}$$

nominal transitions, before accounting for the additional simulations required by finite differences.

Assuming a transition time of 3 ms, the nominal computation time would be

$$4.26 \times 10^{17} \text{ s} \approx 1.35 \times 10^{10} \text{ years.}$$

Therefore, computing or even systematically estimating a Lipschitz constant  $L_i$  for every cell of the abstraction is out of reach. The role of finite-difference analysis is only diagnostic: it shows how local perturbations can be amplified by the dynamics, and it explains why exhaustive Lipschitz-based abstraction design is computationally infeasible.

## 4.4 Empirical Sensitivity Estimation

The previous section showed that a cell-wise Lipschitz constant  $L_i$  provides a useful way to describe how local errors are amplified by the dynamics. However, it also showed that computing, or even systematically estimating, such a constant over every cell and every admissible input is computationally out of reach for the considered abstraction.

For this reason, the objective of the present section is not to obtain certified Lipschitz bounds. Instead, we introduce an empirical sensitivity indicator that can be computed directly from simulated transitions. This indicator measures how strongly two nearby states inside the same cell may separate after one time step, and therefore provides a finite-scale approximation of the local amplification effects discussed above.

Let  $C_i$  be a cell of the abstraction, and let

$$\mathcal{X}_i^{\text{samp}} = \{x_i^{(1)}, \dots, x_i^{(N_s)}\} \subset C_i$$

be a finite set of sampled states inside this cell. The number of sampled states  $N_s$  is a user-chosen parameter. In practice, it should be large enough to distinguish cells with different sensitivity levels reliably, while remaining compatible with the available transition budget. Similarly, we denote by

$$N_u = |\mathcal{U}|$$

the number of admissible inputs.

For each sampled state  $x_i^{(a)}$  and each input  $u \in \mathcal{U}$ , the simulator is used to compute the one-step successor  $F(x_i^{(a)}, u)$ . This requires

$$N_s N_u$$

simulated transitions per cell.

The empirical sensitivity estimate associated with cell  $C_i$  is then defined as

$$\hat{L}_i = \max_{u \in \mathcal{U}} \max_{1 \leq a < b \leq N_s} \frac{\|F(x_i^{(a)}, u) - F(x_i^{(b)}, u)\|_W}{\|x_i^{(a)} - x_i^{(b)}\|_W}.$$

The condition  $a < b$  means that each unordered pair of sampled states is tested only once. Therefore, for each admissible input, the number of finite-scale slopes evaluated inside the cell is

$$\binom{N_s}{2} = \frac{N_s(N_s - 1)}{2}.$$

Across all admissible inputs, the total number of evaluated slopes is thus

$$N_u \binom{N_s}{2}.$$

These pairwise slopes do not require additional simulations once the  $N_s N_u$  sampled transitions have been computed; they are obtained by comparing the simulated successors.

However,  $\hat{L}_i$  remains an empirical quantity. Since it only tests a finite number of sampled pairs, it cannot certify the worst-case Lipschitz constant over the whole continuous cell. It should instead be interpreted as a diagnostic indicator of local sensitivity at the scale of the cell. Large values of  $\hat{L}_i$  indicate regions where small differences in the initial state can lead to significantly different successors, and therefore where the propagated abstraction error  $L_i \eta_i$  may become dominant.

## Limitations of the Maximum Sensitivity Indicator

To assess whether this maximum is a reliable criterion for comparing cells, we evaluate it on ten cells selected uniformly over the considered state space. The experiment uses a position grid step

$$h_q = 2^\circ.$$

Since the velocity components are weighted by  $T_{\text{step}}$ , this corresponds to a velocity step

$$h_{\dot{q}} = \frac{h_q}{T_{\text{step}}} = 20^\circ/\text{s}.$$

The centers of the ten cells are reported in Table 4.1. Position components are expressed in degrees, while velocity components are expressed in degrees per second.

Table 4.1: Centers of the ten cells used for the empirical sensitivity experiment.

| Cell | Position center ( $^{\circ}$ ) |               |               |               | Velocity center ( $^{\circ}/s$ ) |                     |                     |                     |
|------|--------------------------------|---------------|---------------|---------------|----------------------------------|---------------------|---------------------|---------------------|
|      | $\theta_{LH}$                  | $\theta_{RH}$ | $\theta_{LK}$ | $\theta_{RK}$ | $\dot{\theta}_{LH}$              | $\dot{\theta}_{RH}$ | $\dot{\theta}_{LK}$ | $\dot{\theta}_{RK}$ |
| 1    | 16                             | -32           | 12.5          | 12.5          | 57.3                             | -57.3               | 71.6                | -128.9              |
| 2    | -48                            | -40           | 39.5          | 30.5          | -34.4                            | 103.1               | 43.0                | 71.6                |
| 3    | 8                              | -16           | 21.5          | 75.5          | 11.5                             | 80.2                | 100.3               | -43.0               |
| 4    | -24                            | -8            | 66.5          | 48.5          | 80.2                             | 34.4                | -100.3              | 100.3               |
| 5    | -56                            | 0             | 3.5           | 57.5          | 103.1                            | -80.2               | -128.9              | -71.6               |
| 6    | -32                            | -56           | -5.5          | 21.5          | -57.3                            | -11.5               | 14.3                | 14.3                |
| 7    | -16                            | 8             | 75.5          | 66.5          | -11.5                            | -34.4               | 128.9               | 43.0                |
| 8    | 0                              | 16            | 48.5          | -5.5          | -80.2                            | 11.5                | -14.3               | -100.3              |
| 9    | -8                             | -48           | 57.5          | 39.5          | -103.1                           | 57.3                | -43.0               | 128.9               |
| 10   | -40                            | -24           | 30.5          | 3.5           | 34.4                             | -103.1              | -71.6               | -14.3               |

Using these ten representative cells, Figure 4.3 shows the evolution of  $\hat{L}_i$  as the number of sampled states per cell increases.

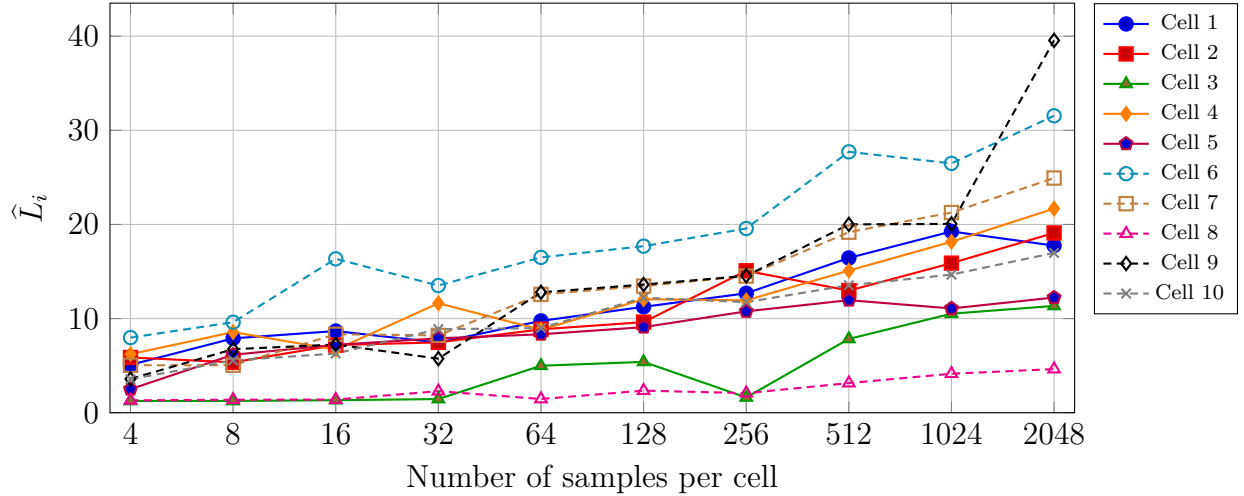


Figure 4.3: Evolution of the empirical maximum sampled slope as a function of the number of sampled states per cell. The experiment is performed on ten cells uniformly distributed over the state space, using a position grid step  $h_q = 2^{\circ}$ , corresponding to  $20^{\circ}/s$  for velocity components.

The figure shows that  $\hat{L}_i$  does not converge smoothly as the number of sampled states increases. Instead, its value often exhibits abrupt jumps and a persistent upward drift. This behaviour is expected because  $\hat{L}_i$  is determined by a single pair of sampled states. Increasing the number of samples therefore increases the probability of observing an isolated extreme pair, rather than stabilizing the estimate.

This instability also affects the relative comparison between cells. To illustrate this, we

consider the normalized ratios

$$\rho_i^{\max}(N_s) = \frac{\hat{L}_i(N_s)}{\hat{L}_1(N_s)}.$$

If  $\hat{L}_i$  were a reliable criterion for classifying cells, these ratios should converge as  $N_s$  increases. In other words, the relative ordering between cells should remain approximately stable when the sampling effort grows.

Figure 4.4 shows the evolution of these ratios for a few representative cells. The same graphical styles as in Figure 4.3 are used, so that each ratio keeps the style of the corresponding cell.

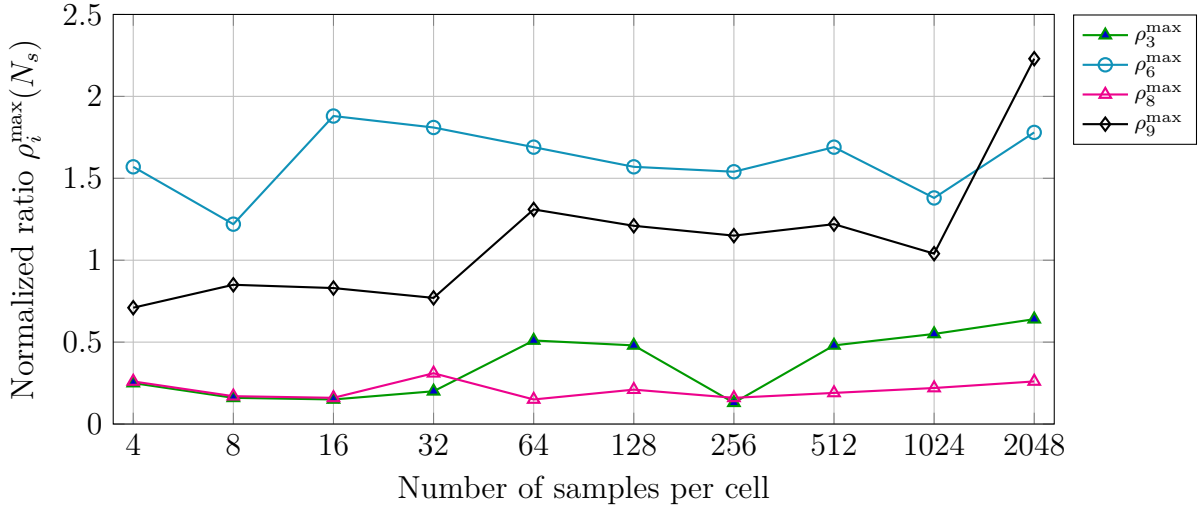


Figure 4.4: Evolution of the normalized ratios  $\rho_i^{\max}(N_s) = \hat{L}_i(N_s)/\hat{L}_1(N_s)$  for selected cells as the number of sampled states per cell increases.

The figure shows that these ratios do not converge smoothly as  $N_s$  increases. Instead, they exhibit significant fluctuations, and some of them change markedly when more samples are added. In particular, the ratio associated with cell 9 remains below 1 for small sample counts, but rises above 2 at  $N_s = 2048$ . This means that cell 9 appears less sensitive than cell 1 at first, and then becomes more than twice as sensitive when the sample set is enlarged.

This confirms that the relative ordering between cells is not preserved when using  $\hat{L}_i$ . The reason is that  $\hat{L}_i$  is defined by a maximum, and is therefore dominated by isolated outliers. Increasing the number of samples mainly increases the probability of encountering a more extreme pair of states, rather than stabilizing the comparison between cells.

Consequently, although  $\hat{L}_i$  is meaningful as a sampled worst-case diagnostic, it is not reliable for ranking or classifying cells according to their sensitivity. A more robust statistic is therefore required when the objective is to characterize the typical local sensitivity of each cell.

## A More Robust Indicator Based on the 90th Percentile

Since the maximum sampled slope is dominated by isolated outliers, we now consider a high-percentile statistic instead. Let  $\mathcal{R}_i(N_s)$  denote the set of all sampled slopes computed in cell  $C_i$ :

$$\mathcal{R}_i(N_s) = \left\{ \frac{\|F(x_i^{(a)}, u) - F(x_i^{(b)}, u)\|_W}{\|x_i^{(a)} - x_i^{(b)}\|_W} : u \in \mathcal{U}, 1 \leq a < b \leq N_s \right\}.$$

The 90th-percentile sensitivity indicator is then defined as

$$L_{90,i}(N_s) = Q_{0.90}(\mathcal{R}_i(N_s)),$$

where  $Q_{0.90}$  denotes the empirical 90th percentile. This statistic is still focused on the upper part of the sampled slope distribution, but it is not determined by a single extreme pair of sampled states. It therefore provides a compromise between sensitivity to strongly amplified transitions and robustness with respect to isolated outliers.

Figure 4.5 shows the evolution of  $L_{90,i}$  for the same ten cells.

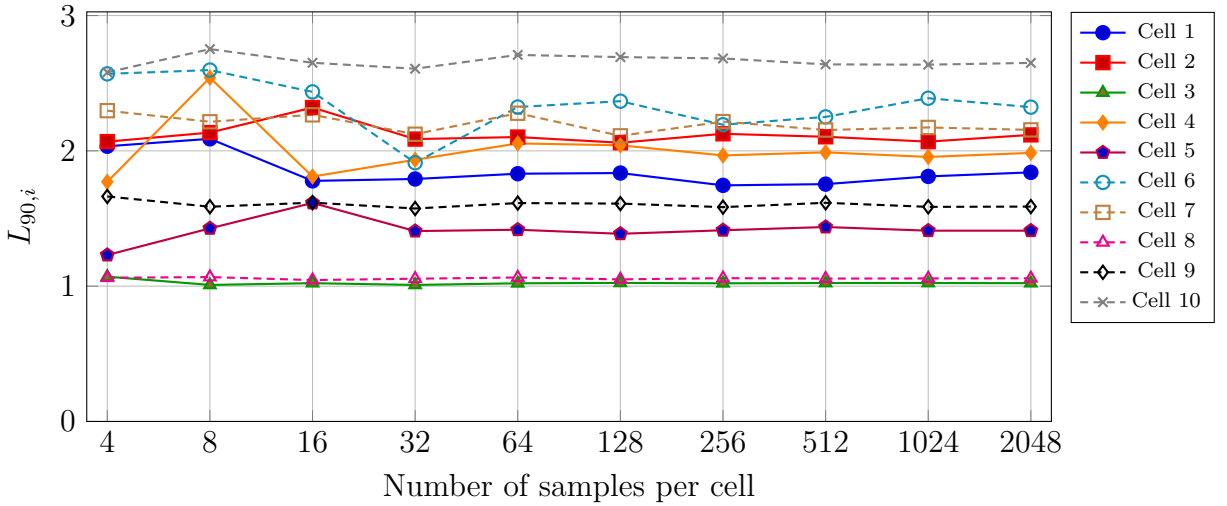


Figure 4.5: Evolution of the empirical 90th-percentile sampled slope as a function of the number of sampled states per cell for the ten representative cells.

In contrast with the maximum sampled slope, the curves in Figure 4.5 are much more stable as the number of samples increases. Although small variations remain, the values do not exhibit the same abrupt upward jumps as  $\hat{L}_i$ . This confirms that replacing the maximum by a high percentile strongly reduces the influence of isolated outliers.

The stability of the relative comparison can again be assessed through normalized ratios. We define

$$\rho_i^{90}(N_s) = \frac{L_{90,i}(N_s)}{L_{90,1}(N_s)}.$$

If  $L_{90,i}$  is a reliable indicator for cell classification, these ratios should remain approximately stable as the number of sampled states increases.

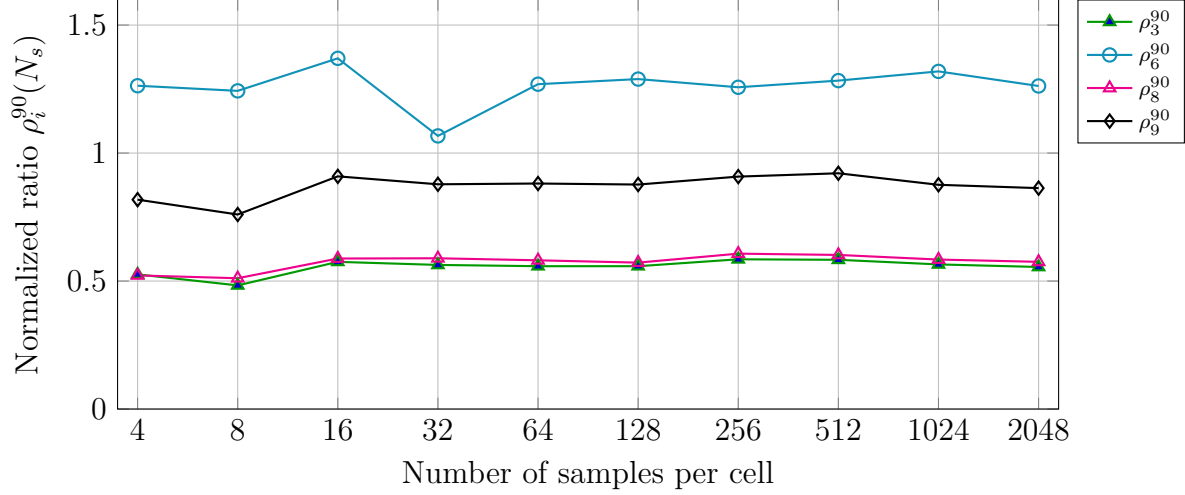


Figure 4.6: Evolution of the normalized ratios  $\rho_i^{90}(N_s) = L_{90,i}(N_s)/L_{90,1}(N_s)$  for selected cells as the number of sampled states per cell increases.

Figure 4.6 shows that the ratios based on  $L_{90,i}$  are much more stable than those based on  $\hat{L}_i$ . The relative position of the selected cells is largely preserved when the sampling effort increases. For example, cells 3 and 8 remain consistently less sensitive than cell 1, while cell 6 remains consistently more sensitive. Unlike the maximum-based ratios, there is no abrupt reversal caused by the sudden appearance of a single extreme pair.

This suggests that  $L_{90,i}$  is more appropriate when the goal is to classify cells according to their empirical sensitivity. It does not provide a certified worst-case Lipschitz bound, but it captures the upper range of the local amplification distribution in a way that is much less dominated by outliers. Therefore,  $L_{90,i}$  offers a more reliable practical criterion for ranking cells and identifying regions where local error propagation may be significant.

### Scale Dependence of the Empirical Lipschitz Estimate

The previous results showed that  $L_{90,i}$  is less sensitive to isolated outliers than the maximum empirical estimate  $\hat{L}_i$ . However, both quantities remain finite-scale estimates. They are not intrinsic local Lipschitz constants independent of the scale at which the cell is sampled.

This point is important when such quantities are used to adapt the grid. The empirical estimate  $\hat{L}_i$  compares a transition error with an intra-cell displacement. If the discretization step is changed, the estimator no longer probes the dynamics at the same scale. Therefore, a decrease of  $\hat{L}_i$  does not necessarily mean that the dynamics become less sensitive.

To illustrate this effect, the same ten representative cells were evaluated with three different position discretization steps:

$$h_q = 0.2^\circ, \quad h_q = 2^\circ, \quad h_q = 10^\circ.$$

The velocity discretization step is chosen consistently with the weighted norm, as in the previous sections.

Figure 4.7 first shows the evolution of the maximum corner transition error. As expected, the error increases when  $h_q$  increases, because points sampled inside the same cell may be farther apart. However, this increase is not linear, the error tends to saturate for large discretization steps. Once the successor discrepancy is already large, it becomes difficult to make it much worse.

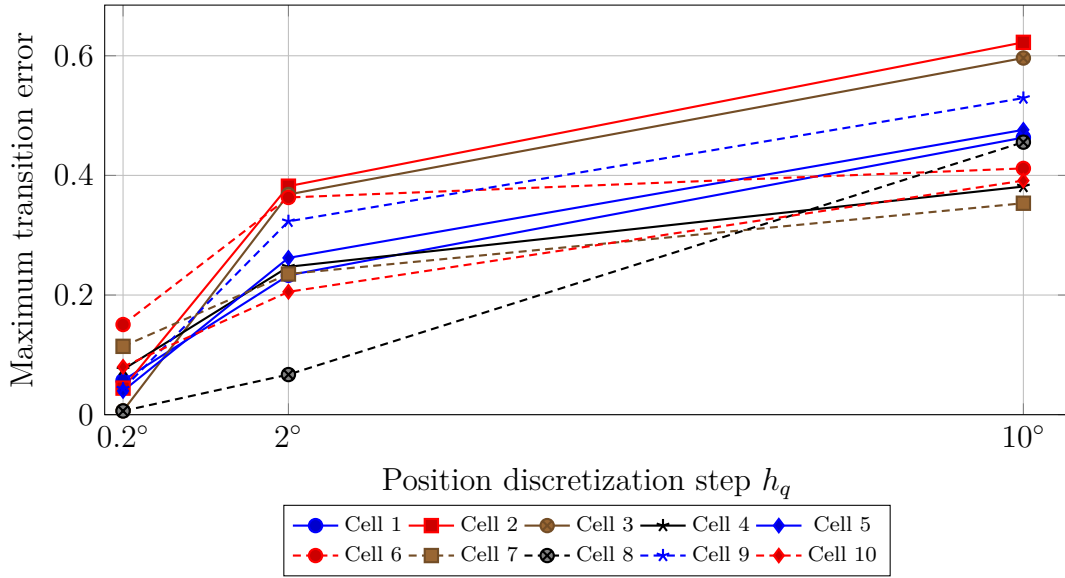


Figure 4.7: Evolution of the maximum corner transition error for the ten representative cells as the position discretization step  $h_q$  increases. The error increases with the cell size, but several curves tend to level off for large steps.

This saturation directly affects the empirical Lipschitz estimate. Since  $\hat{L}_i$  is a finite-scale ratio, its denominator increases with the cell size. If the transition error grows more slowly than the discretization step, or if it saturates, the quotient decreases.

This is exactly what is observed in Figure 4.8. The empirical estimate  $\hat{L}_i$  generally decreases when  $h_q$  increases. This decrease should not be interpreted as an improvement of the abstraction. It is mainly a consequence of measuring a finite-scale slope after the transition error has already saturated.

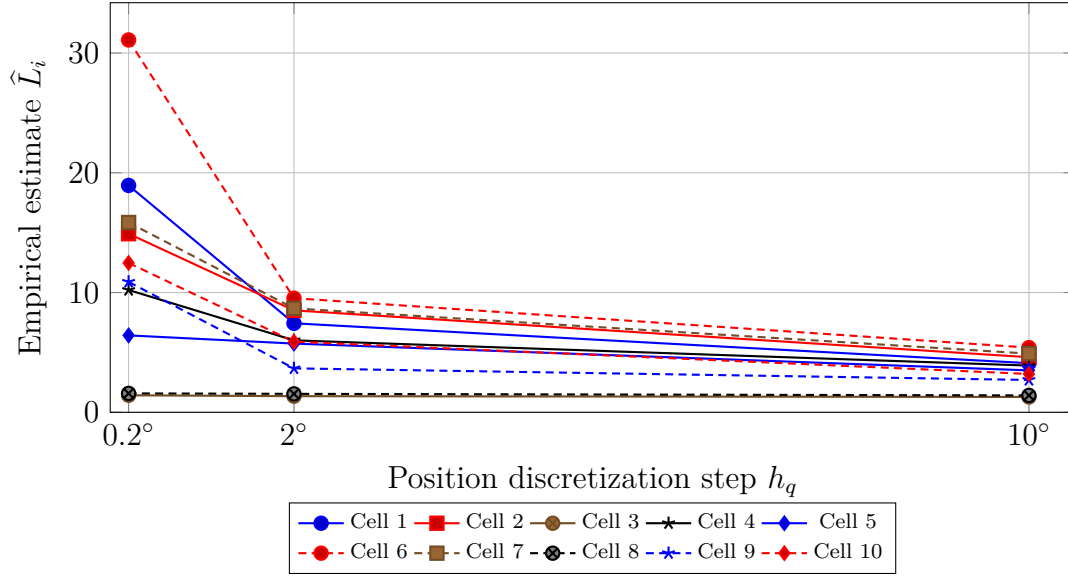


Figure 4.8: Evolution of the empirical finite-scale Lipschitz estimate  $\hat{L}_i$  for the same ten cells. The decrease of  $\hat{L}_i$  with  $h_q$  does not mean that larger cells are more robust. It reflects the saturation of the transition error relative to the increasing intra-cell displacement.

These results show that  $\hat{L}_i$  and  $L_{90,i}$  are useful diagnostics when the sampling scale is fixed, but they should not be used naively to compare cells of different sizes. In particular, if an adaptive algorithm increases a discretization step, the empirical slope may decrease simply because it is now measured at a larger scale.

This creates a feedback bias:

$$\hat{L}_i \text{ decreases} \implies \eta_i \text{ is increased} \implies \hat{L}_i \text{ is evaluated at an even larger scale.}$$

The cell may then continue to grow, not because the dynamics are locally less sensitive, but because the finite-scale estimator itself decreases with the sampling scale.

## 4.5 Dimension-Wise Sensitivity

The previous analysis provides a scalar sensitivity indicator for each cell. This is useful to decide which cells are more sensitive than others, but it does not indicate which state variables are responsible for this sensitivity. For adaptive refinement, this information is important: once a sensitive cell has been identified, one still has to decide along which dimension the cell should be refined [18].

To obtain such information without running additional simulations, we reuse the same sampled transitions as before. For each sampled pair  $(x_i^{(a)}, x_i^{(b)})$ , input  $u$ , and cell  $C_i$ ,

we use the finite-scale slope already introduced in the previous section:

$$R_{i,u}^{a,b} = \frac{\|F(x_i^{(a)}, u) - F(x_i^{(b)}, u)\|_W}{\|x_i^{(a)} - x_i^{(b)}\|_W}. \quad (4.5)$$

This quantity measures the amplification of the distance between two sampled states after one transition.

The denominator is computed with the weighted norm introduced earlier. For the  $r$ -th state component, let

$$w_r = \begin{cases} 1, & r = 1, \dots, 4, \\ T_{\text{step}}, & r = 5, \dots, 8, \end{cases}$$

so that

$$\|x_i^{(a)} - x_i^{(b)}\|_W^2 = \sum_{r=1}^8 w_r^2 (x_{i,r}^{(a)} - x_{i,r}^{(b)})^2.$$

To identify which dimensions are involved in this amplification, one could directly decompose the weighted distance component by component. However, this would be biased if the cell has different widths in different dimensions. A larger displacement along one coordinate could simply result from a larger cell width, rather than from a stronger contribution of that coordinate to the sampled amplification.

Let  $h_{i,r}$  denote the width of cell  $C_i$  along dimension  $r$ . If  $X_{i,r}^{(a)}$  and  $X_{i,r}^{(b)}$  are two independent uniformly sampled coordinates in this interval, then

$$\mathbb{E} \left[ (X_{i,r}^{(a)} - X_{i,r}^{(b)})^2 \right] = \frac{h_{i,r}^2}{6}.$$

The expected weighted squared displacement along dimension  $r$  is therefore

$$\mathbb{E} \left[ w_r^2 (X_{i,r}^{(a)} - X_{i,r}^{(b)})^2 \right] = w_r^2 \frac{h_{i,r}^2}{6}.$$

We therefore normalize the squared displacement in each dimension by its expected value under uniform sampling in the cell:

$$\delta_{i,r}^{a,b} = \frac{w_r^2 (x_{i,r}^{(a)} - x_{i,r}^{(b)})^2}{w_r^2 h_{i,r}^2 / 6} = 6 \frac{(x_{i,r}^{(a)} - x_{i,r}^{(b)})^2}{h_{i,r}^2}.$$

This normalization expresses the displacement along each dimension relative to what would be expected from random sampling in the same cell. It therefore avoids assigning a large contribution to a dimension only because the cell is wider along that coordinate.

The relative contribution of dimension  $r$  to the sampled pair is then defined as

$$\alpha_{i,r}^{a,b} = \frac{\delta_{i,r}^{a,b}}{\sum_{s=1}^8 \delta_{i,s}^{a,b}}. \quad (4.6)$$

By construction,

$$\sum_{r=1}^8 \alpha_{i,r}^{a,b} = 1.$$

The coefficient  $\alpha_{i,r}^{a,b}$  therefore measures how much of the normalized intra-cell displacement between the two sampled states lies along dimension  $r$ .

In this chapter, this normalization is used only as a diagnostic at a fixed sampling scale. The previous subsection showed that finite-scale slopes should not be compared blindly across different cell sizes, because the empirical slope may decrease when the transition error saturates while the intra-cell distance continues to grow. For this reason, the grid-optimization procedure of the next chapter does not use the slopes directly as update indicators. Instead, it uses the propagated transition errors themselves. This avoids the division by the sampled intra-cell distance and therefore removes the feedback mechanism by which a coarser dimension could appear artificially less sensitive.

Since the objective is to understand the directions involved in large amplifications, we focus on the upper part of the sampled slope distribution. Let

$$\mathcal{H}_i = \{(u, a, b) : R_{i,u}^{a,b} \geq L_{90,i}\}$$

be the set of sampled pairs whose slope belongs to the largest 10% of the distribution in cell  $C_i$ . A dimension-wise sensitivity score can then be defined as

$$S_{i,r}^{90} = \frac{1}{|\mathcal{H}_i|} \sum_{(u,a,b) \in \mathcal{H}_i} \alpha_{i,r}^{a,b} R_{i,u}^{a,b}.$$

This quantity assigns more importance to dimensions that are both strongly represented in the normalized displacement and associated with large successor separations.

For comparison between dimensions, the scores are normalized:

$$\bar{S}_{i,r}^{90} = \frac{S_{i,r}^{90}}{\sum_{s=1}^8 S_{i,s}^{90}}.$$

The normalized score satisfies

$$\sum_{r=1}^8 \bar{S}_{i,r}^{90} = 1,$$

and can be interpreted as the relative contribution of each state component to the high-sensitivity behaviour observed in the cell.

In the present eight-dimensional state representation, the components correspond to

$$x = [\theta_{LH}, \theta_{RH}, \theta_{LK}, \theta_{RK}, \dot{\theta}_{LH}, \dot{\theta}_{RH}, \dot{\theta}_{LK}, \dot{\theta}_{RK}].$$

The dimension-wise scores therefore indicate whether the local amplification is mainly associated with hip angles, knee angles, hip velocities, or knee velocities.

This analysis does not provide a causal derivative with respect to each individual coordinate. It remains a finite-sample diagnostic based on pairwise state comparisons.

However, it has the advantage of requiring no additional simulations beyond those already used to compute  $L_{90,i}$ . It therefore provides a practical way to guide anisotropic refinement: sensitive cells can be split preferentially along the dimensions with the largest normalized dimension-wise scores.

The detailed dimension-wise sensitivity results for the ten representative cells are reported in Appendix A. For the fixed sampling scale used in this diagnostic experiment, the results indicate that velocity components generally contribute less than position components to the high-sensitivity sampled slopes. Since the ten cells were selected uniformly in the considered state space, this suggests that the same tendency may also hold more globally over the abstraction domain.

However, this conclusion must be interpreted at the tested discretization scale. As shown above, finite-scale slope estimates depend on the scale at which the cell is sampled. Consequently, the dimension-wise scores are useful as diagnostic indicators at a fixed resolution, because they show which state variables are involved in the largest finite-scale amplifications observed at that scale. However, they should not be used directly to compare grids of different sizes. As discussed above, finite-scale slopes may decrease when the transition error saturates while the sampled intra-cell distance increases. For this reason, the next chapter uses sampled propagated errors directly, rather than slope-based scores, to guide the global grid optimization.

# Chapter 5

## Global Sensitivity-Based Grid Optimization

### 5.1 Motivation

Chapter 4 showed that abstraction errors are not only determined by the size of the grid cells, but also by the way local perturbations are amplified by the one-step transition map. In particular, the local propagation term introduced in (4.2) suggests that the relevant quantity is not the cell radius alone, but the product between a local sensitivity measure and the cell size.

This observation naturally suggests an adaptive discretization strategy. Regions where the dynamics strongly amplifies intra-cell perturbations should be refined, while less sensitive regions could remain coarser. In the terminology of Chapter 4, this would amount to using the sampled slopes defined in (4.5) and the dimension-wise contributions introduced in (4.6) to decide where additional resolution is most useful.

However, the objective of the present chapter is not to build a fully adaptive cell-wise refinement method. Instead, the goal is to adapt the discretization in a simpler and more scalable way. The main idea is to keep a Cartesian abstraction, but to optimize its grid steps globally across the state dimensions. This gives an anisotropic grid: the abstraction remains structured, but some variables are discretized more finely than others.

The contribution of this chapter is therefore a global grid optimization procedure. It reuses the sensitivity indicators introduced in Chapter 4, but changes the level at which they are aggregated. Instead of estimating one indicator per cell, the method estimates global dimension-wise sensitivity scores over the abstraction domain and uses them to redistribute a fixed transition budget across the state variables.

## 5.2 Why Not a Fully Adaptive Cell-Wise Partition?

A direct continuation of the previous chapter would be to refine the cells with the largest estimated local errors. Such a method cannot be reduced to placing additional samples in selected regions: the state space must remain covered by a finite abstraction at all times. A typical way to achieve this is to represent the domain as a variable-resolution partition,

$$\mathcal{X} = \bigcup_{i \in I} C_i, \quad C_i \cap C_j = \emptyset \quad \text{for } i \neq j.$$

Each state still belongs to one cell, but the size of the cells may vary across the domain. This idea is central in variable-resolution discretization methods, such as the approach proposed by [18], where selected regions are progressively refined while the full domain remains covered.

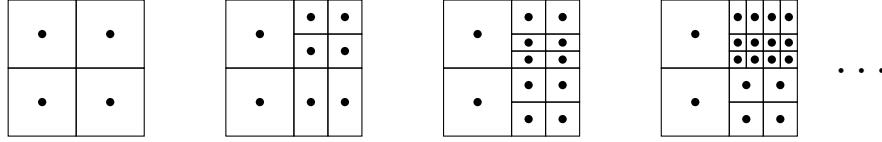


Figure 5.1: Iterative adaptive refinement of a full-domain partition. The state space remains fully covered, but the local resolution becomes non-uniform.

This type of refinement is conceptually well aligned with the error-propagation analysis of Chapter 4. If accurate cell-wise sensitivity estimates were available, one could refine the cells with the largest local error indicators and keep less sensitive cells coarser.

In practice, however, this approach raises three difficulties.

First, reliable cell-wise estimation requires many samples. For a cell  $C_i$ , one must draw a sufficiently rich set of sampled states  $\mathcal{X}_i^{\text{samp}} \subset C_i$  and evaluate the transition map for the admissible inputs. Even if the pairwise slopes are computed afterwards from the same simulated successors, the quality of the estimate still depends on the number and distribution of the sampled points inside the cell. This is especially important when the objective is not only to estimate the magnitude of the sensitivity, but also to identify the relative contribution of each state dimension. With too few samples, the resulting cell-wise indicators may be dominated by the particular directions that happened to be sampled.

Second, the refinement process is inherently iterative. The cells refined at a given iteration depend on sensitivity indicators, error estimates, or value information computed on the current partition. After refinement, the partition is modified, the transition relation must be updated, and the workload has to be redistributed. This introduces global synchronization steps that are difficult to combine with the massively parallel transition-generation strategy discussed earlier.

Third, the symbolic graph is no longer induced by a simple Cartesian-product grid. The cells may remain axis-aligned rectangles, but the symbolic state set can no longer

be indexed as

$$X_1 \times X_2 \times \cdots \times X_{n_x}$$

using one global resolution per dimension. The tensor-product structure is lost. Consequently, locating the cell containing a continuous state, enumerating neighbors, storing transitions, and distributing the graph over independent jobs become more involved. This does not make variable-resolution abstractions impossible, but it makes them significantly harder to use in the large-scale setting considered in this work.

For these reasons, the fully adaptive cell-wise approach is left as a possible direction for future work. The method developed in this chapter keeps the Cartesian structure of the abstraction and modifies only the global dimension-wise grid steps.

### 5.3 Global Anisotropic Reformulation

The previous sections motivated the use of the dimension-wise information obtained from intra-cell perturbations, while also showing why a fully cell-wise adaptive partition would be difficult to exploit at the scale considered here. The remaining objective is therefore more restricted: keep a Cartesian abstraction, but redistribute the available resolution among the state coordinates.

In contrast with the finite-scale slope indicators used for diagnosis in Chapter 4, the update criterion used here is based directly on propagated successor discrepancies. This avoids comparing ratios computed at different intra-cell scales. For a sampled pair of states in the same cell, the quantity of interest is therefore

$$E_m = \left\| F(x_m^{(a)}, u_m) - F(x_m^{(b)}, u_m) \right\|_W, \quad x_m^{(a)}, x_m^{(b)} \in C_{i_m}.$$

This error measures how much two continuous states represented by the same symbolic state may separate after one transition under the same input.

The grid is parameterized by the integer number of cells in each state dimension,

$$N_x = (N_{x,1}, \dots, N_{x,n_x}).$$

For each dimension  $r$ , the prescribed interval  $[\underline{x}_r, \bar{x}_r]$  remains fully covered, and the corresponding grid width is recomputed as

$$h_{x,r} = \frac{\bar{x}_r - \underline{x}_r}{N_{x,r}}.$$

Thus, changing  $N_x$  changes the anisotropy of the grid, but not the portion of the state space represented by the abstraction.

The input grid is kept fixed. The number of symbolic transitions is therefore

$$N_{\text{trans}}(N_x, h_u) = \left( \prod_{r=1}^{n_x} N_{x,r} \right) |U(h_u)|.$$

The purpose of the update is to modify  $N_x$  while keeping

$$N_{\text{trans}}(N_x, h_u) \approx N_{\text{target}}.$$

The method should therefore be interpreted as a redistribution of a fixed transition budget, not as a reduction of the total abstraction size.

For each sampled pair, the normalized dimension-wise contribution  $\alpha_{m,r}$  is computed as in (4.6), using the current full-range cell widths  $h_{x,r}$ . The score is then computed only on the upper tail of the sampled propagated errors. Let

$$E_p^{\text{glob}} = Q_p(\{E_1, \dots, E_M\})$$

be the empirical  $p$ -quantile, and define

$$\mathcal{H}_E^{\text{glob}} = \{m : E_m \geq E_p^{\text{glob}}\}.$$

The global error contribution of dimension  $r$  is

$$S_r^{\text{err}} = \frac{1}{|\mathcal{H}_E^{\text{glob}}|} \sum_{m \in \mathcal{H}_E^{\text{glob}}} \alpha_{m,r} E_m.$$

It is normalized as

$$\bar{S}_r^{\text{err}} = \frac{S_r^{\text{err}}}{\sum_{s=1}^{n_x} S_s^{\text{err}}}, \quad \sum_{r=1}^{n_x} \bar{S}_r^{\text{err}} = 1.$$

If  $\bar{S}_r^{\text{err}} > 1/n_x$ , dimension  $r$  contributes more than average to the largest propagated discrepancies and should receive more resolution. Conversely, if  $\bar{S}_r^{\text{err}} < 1/n_x$ , some resolution can be reallocated from that dimension to more influential ones.

## 5.4 Grid Update Under a Fixed Budget

The normalized scores indicate how the largest sampled propagated errors are distributed among the state dimensions. The ideal situation would be that no dimension dominates this upper-error tail. Since the scores are normalized, this corresponds to the balanced value

$$\tau = \frac{1}{n_x}.$$

A dimension such that  $\bar{S}_r^{\text{err}} > \tau$  contributes more than average to the high-error samples. The grid should therefore allocate more cells to this coordinate. Conversely, if  $\bar{S}_r^{\text{err}} < \tau$ , the dimension contributes less than average and some of its resolution can be transferred to other coordinates.

This idea is implemented through a multiplicative update of the number of cells:

$$\widetilde{N}_{x,r} = N_{x,r} \left( \frac{\bar{S}_r^{\text{err}}}{\tau} \right)^\gamma.$$

The ratio  $\overline{S}_r^{\text{err}}/\tau$  compares the observed contribution of dimension  $r$  with its balanced contribution. If this ratio is larger than one, the number of cells in that dimension is increased. If it is smaller than one, the number of cells is decreased. The exponent  $\gamma \in (0, 1]$  acts as a relaxation parameter: for  $\gamma = 1$ , the update directly follows the score ratio, whereas smaller values of  $\gamma$  make the update more conservative.

Because the scores are estimated from random samples, the update should not react too strongly to a single noisy estimate. For this reason, the multiplicative factor is clipped:

$$\widetilde{N}_{x,r} = N_{x,r} \text{clip} \left( \left( \frac{\overline{S}_r^{\text{err}}}{\tau} \right)^\gamma, c_{\min}, c_{\max} \right).$$

The constants  $c_{\min}$  and  $c_{\max}$  bound the relative change of each dimension during one iteration. This prevents the algorithm from making a dimension suddenly much finer or much coarser because of sampling fluctuations.

The real-valued vector  $\widetilde{N}_x$  obtained after the score update does not necessarily satisfy the transition budget. Since the input grid is fixed, the target number of symbolic states is

$$N_{\text{states}}^{\text{target}} = \frac{N_{\text{target}}}{|U(h_u)|}.$$

The candidate grid is therefore rescaled by a common factor

$$s = \left( \frac{N_{\text{states}}^{\text{target}}}{\prod_{r=1}^{n_x} \widetilde{N}_{x,r}} \right)^{1/n_x}.$$

The projected integer counts are then obtained as

$$N_{x,r}^+ = \text{round} \left( s \widetilde{N}_{x,r} \right)$$

Before rounding, this rescaling gives

$$\prod_{r=1}^{n_x} s \widetilde{N}_{x,r} = N_{\text{states}}^{\text{target}},$$

so the total number of transitions is brought back close to  $N_{\text{target}}$ . The remaining difference comes only from the integer rounding of the grid counts.

## 5.5 Algorithm

Algorithm 2 summarizes the implemented procedure.

---

**Algorithm 2** Global error-based full-range grid optimization

---

**Require:** Transition map  $F$ , state bounds  $\mathcal{X}$ , fixed input grid  $U(h_u)$

**Require:** Initial counts  $N_x^{(0)}$ , target budget  $N_{\text{target}}$

**Require:** Quantile level  $p$ , number of random pairs  $M$ , maximum number of iterations

$K_{\max}$

- 1:  $k \leftarrow 0$
- 2: **while**  $k < K_{\max}$  **do**
- 3:   Build the full-range Cartesian grid induced by  $N_x^{(k)}$
- 4:   **for**  $m = 1, \dots, M$  **do**
- 5:     Draw a random cell  $C_{i_m}$ , an input  $u_m$ , and two states  $x_m^{(a)}, x_m^{(b)} \in C_{i_m}$
- 6:     Simulate  $F(x_m^{(a)}, u_m)$  and  $F(x_m^{(b)}, u_m)$
- 7:     Compute  $E_m$  and  $\alpha_m$
- 8:   **end for**
- 9:   Select the high-error set  $\mathcal{H}_E^{\text{glob}}$
- 10:   Compute the normalized scores  $\bar{S}_r^{\text{err}}$
- 11:   Compute a candidate multiplicative update of  $N_x^{(k)}$
- 12:   Project the counts to satisfy the integer constraints and transition budget
- 13:   **if** the score imbalance and the grid update are sufficiently small **then**
- 14:     **break**
- 15:   **end if**
- 16:   Store the projected grid as  $N_x^{(k+1)}$
- 17:    $k \leftarrow k + 1$
- 18: **end while**
- 19: **return**  $N_x^{(k)}$

---

## 5.6 Results

This section reports the result of the full-range anisotropic grid optimization. The input grid is kept fixed throughout the experiment:

$$u \in [-2, 2]^4 \text{ V}, \quad h_u = [2, 2, 2, 2] \text{ V}.$$

This gives

$$N_u = (3, 3, 3, 3), \quad |U(h_u)| = 81.$$

The target number of symbolic transitions is

$$N_{\text{target}} = 2 \times 10^9.$$

Table 5.1 compares the initial and final discretizations.

| State variable      | Range               | $N_{x,r}^{\text{init}}$ | $h_{x,r}^{\text{init}}$ | $N_{x,r}^{\text{final}}$ | $h_{x,r}^{\text{final}}$ |
|---------------------|---------------------|-------------------------|-------------------------|--------------------------|--------------------------|
| $\theta_{LH}$       | $[-60, 20]^\circ$   | 11                      | $7.27^\circ$            | 17                       | $4.71^\circ$             |
| $\theta_{RH}$       | $[-60, 20]^\circ$   | 11                      | $7.27^\circ$            | 18                       | $4.44^\circ$             |
| $\theta_{LK}$       | $[-10, 80]^\circ$   | 13                      | $6.92^\circ$            | 21                       | $4.29^\circ$             |
| $\theta_{RK}$       | $[-10, 80]^\circ$   | 13                      | $6.92^\circ$            | 20                       | $4.50^\circ$             |
| $\dot{\theta}_{LH}$ | $[-2, 2]$ rad/s     | 5                       | 0.80 rad/s              | 3                        | 1.33 rad/s               |
| $\dot{\theta}_{RH}$ | $[-2, 2]$ rad/s     | 5                       | 0.80 rad/s              | 4                        | 1.00 rad/s               |
| $\dot{\theta}_{LK}$ | $[-2.5, 2.5]$ rad/s | 7                       | 0.714 rad/s             | 4                        | 1.25 rad/s               |
| $\dot{\theta}_{RK}$ | $[-2.5, 2.5]$ rad/s | 7                       | 0.714 rad/s             | 4                        | 1.25 rad/s               |

Table 5.1: State ranges and discretization steps before and after the full-range anisotropic optimization. The angular steps are reported in degrees for readability, while the velocity steps are reported in rad/s.

### 5.6.1 Preservation of the Transition Budget

A first point to verify is that the projection step keeps the abstraction size close to the prescribed transition budget. Figure 5.2 shows the number of symbolic transitions during the optimization.

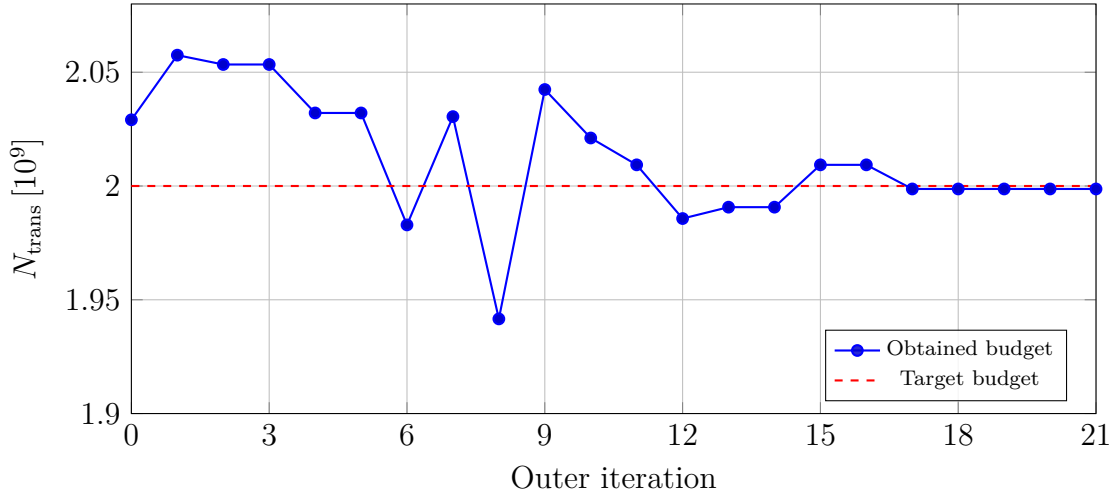


Figure 5.2: Evolution of the number of symbolic transitions during the full-range anisotropic optimization. The projection step keeps the transition count close to the target value  $N_{\text{target}} = 2 \times 10^9$ .

The transition count remains close to the target throughout the run. It stays approximately between  $1.94 \times 10^9$  and  $2.06 \times 10^9$ , and the final grid has  $1.999 \times 10^9$  transitions. The method therefore reallocates resolution across the state dimensions without significantly changing the total size of the abstraction.

### 5.6.2 Convergence of the Error-Score Estimate

The update direction is based on a Monte Carlo estimate of the normalized dimension-wise scores  $\overline{S}_r^{\text{err}}$ . It is therefore useful to check how this estimate depends on the number of sampled pairs.

The sampling procedure is global in the sense that the sampled cell is drawn randomly over the full state partition. However, each error sample remains local to one symbolic state: after drawing a cell  $C_i$  and an input  $u$ , two states  $x^{(a)}$  and  $x^{(b)}$  are drawn inside the same cell,

$$C_i \sim \text{Uniform}(\mathcal{X}_h), \quad u \sim \text{Uniform}(U_h), \quad x^{(a)}, x^{(b)} \sim \text{Uniform}(C_i).$$

The sampled error is then

$$E = \left\| F(x^{(a)}, u) - F(x^{(b)}, u) \right\|_W.$$

Thus, the samples cover the full state domain statistically, but the measured quantity is an intra-cell propagation error. This is the relevant quantity for a finite abstraction, since both  $x^{(a)}$  and  $x^{(b)}$  would be represented by the same symbolic state.

Each sample requires two calls to the transition map, one for  $F(x^{(a)}, u)$  and one for  $F(x^{(b)}, u)$ . Therefore,  $N$  sampled pairs correspond to  $2N$  transition-map evaluations.

For the final grid

$$N_x = (17, 18, 21, 20, 3, 4, 4, 4),$$

the normalized scores were recomputed cumulatively for

$$N = 4, 8, 16, \dots, 16384$$

sampled pairs. Figure 5.3 shows the evolution of the eight components of  $\overline{S}^{\text{err}}$ .

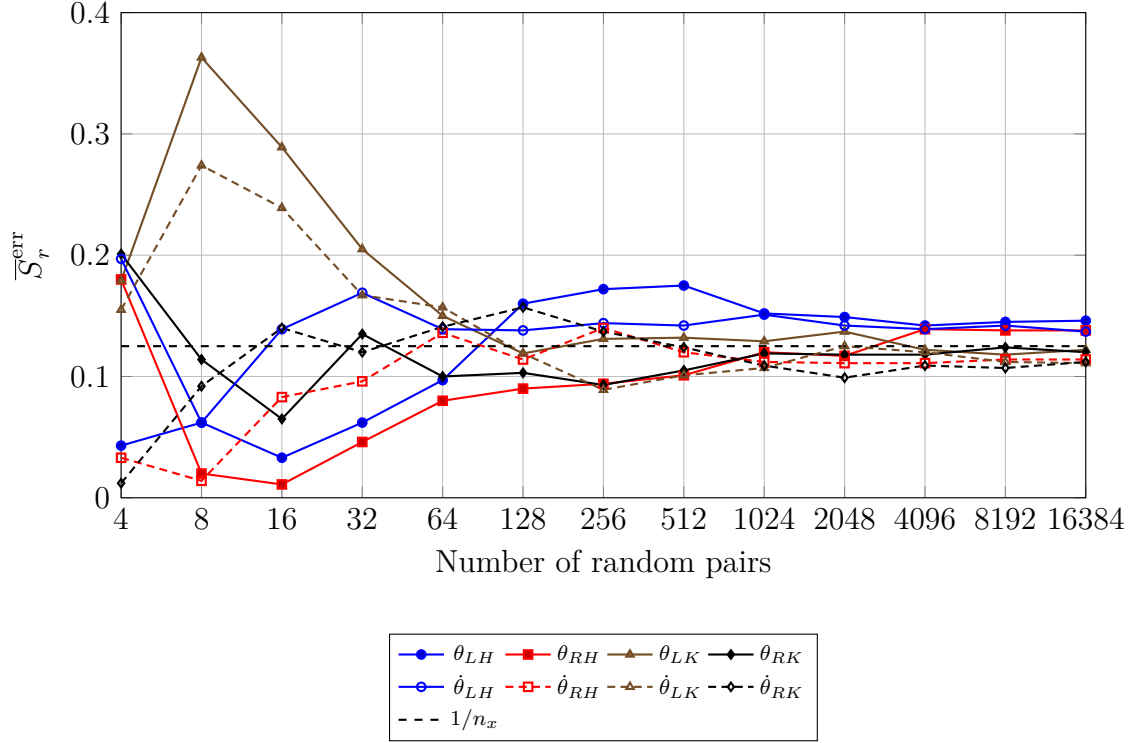


Figure 5.3: Evolution of the normalized dimension-wise error scores as a function of the number of sampled pairs. The sampling is global over the full state domain, but each pair is drawn inside a single cell. The horizontal dashed line corresponds to the balanced value  $1/n_x = 0.125$ .

For very small sample sizes, the score estimate is highly variable. This is expected because the score is computed only on the upper tail of the sampled errors. With only a few samples, this tail contains very little information, so one or two samples can dominate the dimension-wise score. This explains the large fluctuations observed for  $N = 4, 8, 16$ .

As the number of samples increases, the estimate becomes much more stable. From approximately 4096 sampled pairs onward, the different components remain in a relatively narrow range around the balanced value  $1/n_x = 0.125$ . At  $N = 16384$ , the score is

$$\bar{S}^{\text{err}} = (0.146, 0.138, 0.122, 0.120, 0.137, 0.114, 0.111, 0.112).$$

The corresponding imbalance is

$$\left\| \bar{S}^{\text{err}} - \frac{1}{n_x} \mathbf{1} \right\|_{\infty} = 0.0210.$$

The evolution therefore indicates that the dimension-wise score is sufficiently stable to be used as a heuristic update direction, provided that enough random pairs are sampled.

Figure 5.4 shows the same phenomenon through the scalar imbalance measure.

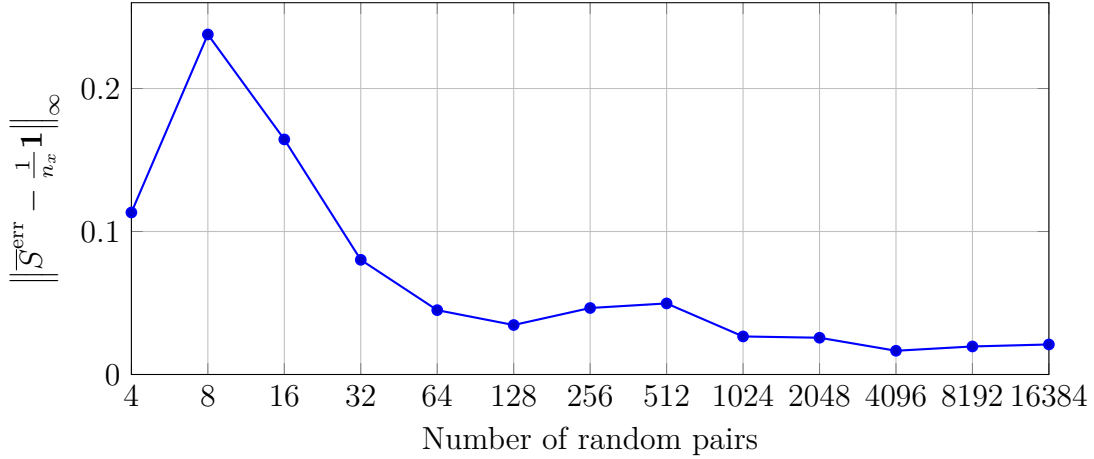


Figure 5.4: Convergence of the score imbalance as a function of the number of sampled pairs. Each pair requires two calls to the transition map, so the last point corresponds to 32768 transition-map evaluations.

The imbalance decreases rapidly when moving from a few samples to a few thousand samples. After that point, the remaining variations are relatively small and mostly reflect the Monte Carlo nature of the estimator. These results justify using several thousand random pairs when estimating the global dimension-wise error score.

### 5.6.3 Independent Initial–Final Validation

Figure 5.5 compares the validation error distribution before and after the optimization.

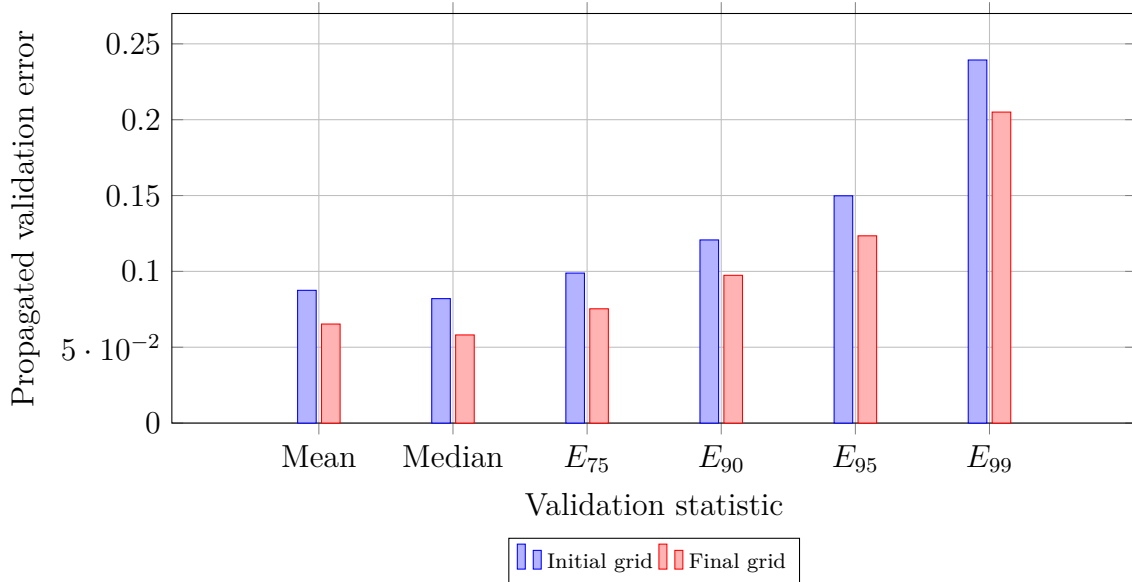


Figure 5.5: Independent validation of the propagated discrepancy before and after the anisotropic grid optimization. The comparison is performed on 4800 common validation samples.

The validation errors decrease over the whole distribution. The mean propagated error decreases from 0.0875 to 0.0653, while the median decreases from 0.0820 to 0.0581. The upper part of the distribution also improves:  $E_{90}$  decreases from 0.1207 to 0.0974, and  $E_{95}$  decreases from 0.1498 to 0.1234.

The relative reductions are summarized in Figure 5.6. They show that the improvement is not limited to the mean error, but also affects the upper quantiles.

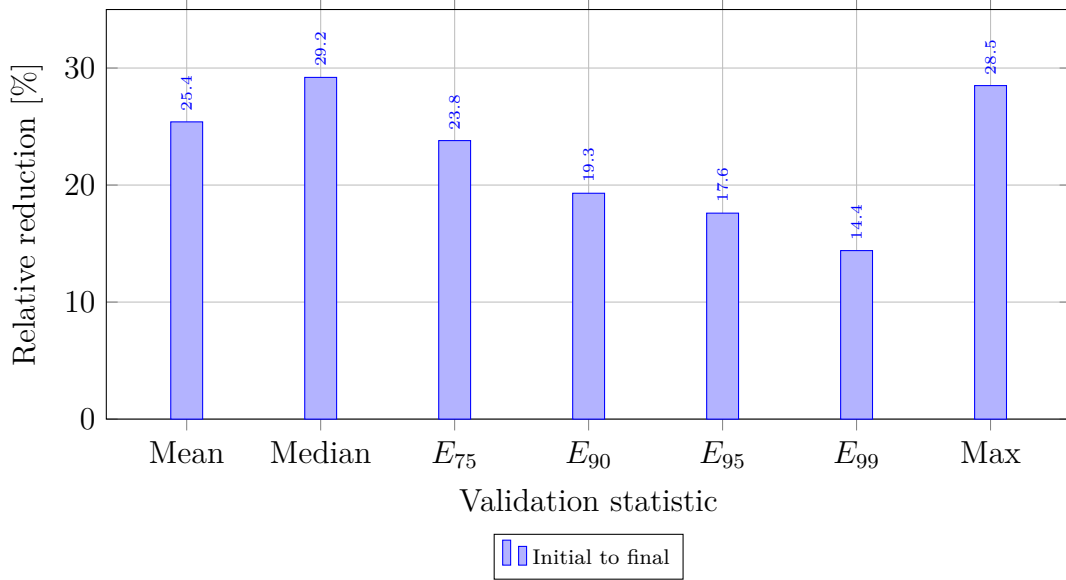


Figure 5.6: Relative reduction of the validation error statistics between the initial and final grids.

The corresponding numerical values are also reported in Table 5.2.

| Grid            | $E_{\text{mean}}$ | $E_{50}$ | $E_{75}$ | $E_{90}$ | $E_{95}$ | $E_{99}$ | $E_{\text{max}}$ |
|-----------------|-------------------|----------|----------|----------|----------|----------|------------------|
| Initial         | 0.0875            | 0.0820   | 0.0989   | 0.1207   | 0.1498   | 0.2393   | 0.5448           |
| Final           | 0.0653            | 0.0581   | 0.0753   | 0.0974   | 0.1234   | 0.2050   | 0.3894           |
| Relative change | -25.4%            | -29.2%   | -23.8%   | -19.3%   | -17.6%   | -14.4%   | -28.5%           |

Table 5.2: Independent initial-final validation of the propagated discrepancy. The validation is performed on 4800 common samples for the initial and final grids.

The maximum observed validation error also decreases, from 0.5448 to 0.3894. This value is more sensitive to isolated samples than the quantiles, so the most relevant indicators are the mean and the high quantiles. Overall, the validation confirms that the optimized anisotropic grid improves the propagated-error behaviour beyond the random samples used to compute the refinement score.

## 5.7 Interpretation, Generality, and Limitations

The full-range global anisotropic optimization keeps the Cartesian-product structure of the abstraction while redistributing a fixed transition budget among the state variables. The method optimizes the integer counts  $N_x$ , and the grid widths are recomputed from the fixed state ranges after each update. The state domain therefore remains fully covered throughout the optimization.

In the experiment, the initial grid

$$N_x^{\text{init}} = (11, 11, 13, 13, 5, 5, 7, 7)$$

is replaced by

$$N_x^{\text{final}} = (17, 18, 21, 20, 3, 4, 4, 4).$$

The number of transitions remains close to the target:

$$N_{\text{trans}}^{\text{final}} = 1.999 \times 10^9.$$

The optimized grid therefore does not reduce the abstraction budget. Instead, it real-locates this budget toward the dimensions where the transition map is more sensitive.

The final grid refines the position coordinates and coarsens the velocity coordinates. This anisotropy has a natural interpretation in terms of one-step error propagation. Each sampled error compares two states inside the same cell propagated with the same input:

$$E_m = \left\| F(x_m^{(a)}, u_m) - F(x_m^{(b)}, u_m) \right\|_W.$$

The input therefore does not create a difference between the two trajectories: it applies the same actuation to both. If the two states mainly differ in velocity, the actuated dynamics can partially reduce this discrepancy during the integration interval, because the input acts directly on the accelerations. The velocity mismatch affects the future motion, but it is a quantity on which the dynamics can act over the transition.

A position discrepancy is different. It is already a configuration difference between the two robot postures. It changes the geometry of the robot, the gravity terms, the contact-related quantities, and the way the velocities are integrated over the sampling time. At first order,

$$q^+ \approx q + T_{\text{step}} \dot{q}.$$

Even if the velocity discrepancy is partially reduced during the step, the initial configuration offset remains present in the successor state. The input can influence the future velocity evolution, but it cannot instantaneously remove the initial difference in configuration.

This explains why reducing the position cell widths is effective for decreasing the propagated discrepancies. Coarse position cells may group together states with noticeably different robot geometries, and these differences persist after one transition. Coarse velocity cells are still an approximation, but their effect can be partly absorbed by the dynamics over the sampling interval. Therefore, at a fixed transition budget, it is

reasonable for the optimizer to allocate more resolution to the joint positions and less to the joint velocities.

This redistribution is important not only for robustness, but also for the quality of the symbolic graph itself. In an abstraction-based controller, the solution is computed on a finite transition graph. If a cell contains states whose successors are very different, then the corresponding symbolic transition is a poor representative of the continuous dynamics. This can make the graph more conservative and can also degrade the optimality of the synthesized solution, because the cost-to-go and the selected actions are computed on an imprecise discrete model.

Reducing the propagated intra-cell errors therefore improves the effective resolution of the graph. More of the fixed budget is allocated to directions in which the transition map produces varied successors, while dimensions along which the successors remain globally closer can be represented more coarsely. In this sense, the optimization does not simply make the abstraction more robust to local perturbations. It also improves the ability of the graph to distinguish dynamically different transitions. This is essential because the quality and optimality of the symbolic controller depend directly on the resolution of the transition graph.

A key advantage of the proposed criterion is that it does not require an analytical separation of the different sources of abstraction error. Since the full transition map  $F$  is evaluated, the measured error includes all mechanisms that affect the successor state: nonlinear dynamics, couplings between coordinates, actuation effects, contact-related effects, and the interaction between positions and velocities. The method therefore does not assume that the dimensions act independently.

The dimension-wise scores should instead be interpreted statistically. They identify which state coordinates are most frequently associated with the largest sampled propagated discrepancies. In this sense, the method uses probabilistic correlations between intra-cell variations and successor errors to detect which dimensions are most sensitive for the abstraction. This makes the approach quite general: as long as a transition map can be evaluated and random pairs can be sampled inside the cells, the same procedure can be applied to other abstraction-based control problems without deriving problem-specific sensitivity formulas.

The independent validation confirms the practical benefit of this redistribution. On 4800 common validation samples, the mean propagated discrepancy decreases from 0.0875 to 0.0653, and the 95th percentile decreases from 0.1498 to 0.1234. The result therefore supports the use of global anisotropic discretization as a scalable compromise between a uniform Cartesian grid and a fully adaptive variable-resolution abstraction.

The method remains heuristic. The scores  $\bar{S}_r^{\text{err}}$  are Monte Carlo estimates and do not prove that the final grid is globally optimal. They indicate which dimensions are most represented in the largest sampled propagated discrepancies, but they are not certified error bounds.

Another limitation is the dependence on the weighted norm  $\|\cdot\|_W$ . Velocity errors are

weighted according to their effect over one sampling period, whereas position errors are measured directly as configuration errors. A different norm could therefore lead to a different anisotropic allocation.

Finally, the optimization acts on integer counts. The update is consequently non-smooth: several iterations can leave the grid unchanged, while a small change in the score can later cross a rounding threshold and modify one or more dimensions at once. This is a practical consequence of keeping a finite Cartesian abstraction with a prescribed transition budget.

# Chapter 6

## Further Investigations: Robust and Physically Informed Synthesis

The previous chapters focused on the construction of scalable and accurate symbolic abstractions. Chapter 2 introduced the abstraction-based control framework and showed how controller synthesis can be interpreted as a graph problem. Chapter 3 then addressed the computational cost of generating large transition graphs. Chapter 4 studied how discretization errors propagate through the one-step transition map, while Chapter 5 used this information to redistribute a fixed transition budget among the state variables.

A natural next step is to use the same error information not only to design the grid, but also to synthesize better controllers on the resulting graph. Indeed, the symbolic controller is only optimal with respect to the graph on which it is computed. If the graph represents the continuous dynamics poorly, then the resulting policy may be sensitive to discretization errors or may choose transitions that are optimal only for the cell centers. The perspectives presented in this chapter therefore aim to make controller synthesis more aware of transition uncertainty and of the physical balance mechanism of the robot.

### 6.1 From Error Propagation to Transition Costs

The local error-propagation analysis of Chapter 4 suggests that the one-step abstraction error can be decomposed into two main terms. If  $e_k$  denotes the error at step  $k$ , then locally one may write

$$e_{k+1} \leq L_k e_k + \eta_{k+1},$$

where  $L_k$  is a local amplification factor associated with the transition, and  $\eta_{k+1}$  is the radius of the successor cell. The term  $L_k e_k$  corresponds to the propagation of the current error through the dynamics, whereas  $\eta_{k+1}$  corresponds to the new quantization error introduced when the successor is represented by its symbolic cell.

In a center-based abstraction, the controller is re-evaluated at each sampling time from the current symbolic cell. It is therefore natural to approximate the current error by the radius of the current cell,

$$e_k \simeq \eta_k.$$

This gives the local transition-error indicator

$$c_{\text{err}}(s_k, a_k, s_{k+1}) = L_k \eta_k + \eta_{k+1}.$$

For a path

$$s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} \dots \xrightarrow{a_{N-1}} s_N,$$

a conservative accumulated score is then

$$J_{\text{err}} = \sum_{k=0}^{N-1} (L_k \eta_k + \eta_{k+1}).$$

In principle, if reliable estimates of  $L_k$  were available for all relevant state–input pairs, controller synthesis could be formulated as a weighted shortest-path problem. Instead of minimizing only the number of abstract transitions, one could use Dijkstra’s algorithm to minimize a cumulative transition-error score.

In practice, however, obtaining accurate amplification estimates over the whole abstraction is not realistic with the current implementation. The abstraction contains a very large number of state–input pairs, and estimating a local amplification factor for each of them would require many additional simulations. A more realistic solution would be to estimate these quantities from the transitions already computed in the graph. Since the source points are the cell centers, their concrete coordinates can be reconstructed from the grid indices. The missing information is therefore not the concrete source state, but the continuous successor

$$y = F(x_c, u)$$

before quantization into a symbolic successor cell.

If these concrete successors were stored, one could compare the successors of nearby cell centers under the same or similar inputs. This would provide an empirical estimate of how much neighboring source states separate after one transition, and therefore an estimate of the local amplification associated with that region of the graph.

The main difficulty is memory. In the current graph representation, it is sufficient to store the symbolic successor information used for controller synthesis. Storing the continuous successor  $F(x_c, u)$  for every computed state–input pair would require keeping an additional vector of dimension  $n_x$  for each transition. For large abstractions, this could easily double or triple the memory required to store the graph, which is already one of the main bottlenecks of large-scale abstraction generation.

A relevant direction for future work is therefore to design a memory-efficient way of estimating transition amplification from the generated graph. This would require determining which successor information should be stored, whether it can be compressed

or recomputed locally, and whether the resulting estimates are accurate enough to improve controller synthesis. If this can be achieved, the transition graph would no longer encode only reachability, but also local information about the reliability or uncertainty of each transition.

## 6.2 Why the Quantization Term Matters

The bound

$$e_{k+1} \leq L_k \eta_k + \eta_{k+1}$$

also shows that reducing only the amplification term is not sufficient. When  $L_k$  is close to one, or smaller than one, the propagated error  $L_k \eta_k$  is of the same order as the cell radius. In that case, the successor quantization term  $\eta_{k+1}$  can be just as important as the propagated error.

This observation is consistent with the empirical amplification estimates of Chapter 4, but from a different statistical viewpoint. In Chapter 4, the analysis mainly focused on high quantiles of the sampled amplifications, because these quantiles are relevant to identify the transitions and cells that may produce large worst-case discrepancies. However, if one considers the empirical expectation of the same sampled amplifications, the values are much smaller.

For the diagnostic experiment performed on ten representative cells with a position step of  $2^\circ$ , the empirical expectation of the amplification was often close to one, as shown in Table 6.1.

| Cell | Empirical expectation of amplification |
|------|----------------------------------------|
| 1    | 1.021                                  |
| 2    | 1.162                                  |
| 3    | 0.750                                  |
| 4    | 1.114                                  |
| 5    | 0.967                                  |
| 6    | 1.176                                  |
| 7    | 1.217                                  |
| 8    | 0.775                                  |
| 9    | 1.014                                  |
| 10   | 1.702                                  |

Table 6.1: Empirical expectation of the sampled amplification for ten representative cells using a position step of  $2^\circ$ . Most values are close to one, which indicates that the successor quantization error can be as important as the propagated error term.

The average value over these ten cells is approximately 1.09, and the median is approximately 1.07. This suggests that, in many cells, the local amplification does not

dominate the error budget by several orders of magnitude. Therefore, even if a transition is not strongly amplifying, re-quantizing the successor into a symbolic cell may still introduce a significant part of the total abstraction error.

This observation motivates a robust transition-aware synthesis strategy. When the estimated amplification satisfies  $L_k \leq 1$ , the current error is not locally amplified. In this case, it becomes particularly important to choose successors that are well centered in their target cells, because the next quantization error may dominate the local bound. Conversely, when  $L_k > 1$ , the propagated error term becomes more important and should be penalized more directly.

### 6.3 Physical Cost: Hip-Torque-Aware Synthesis

The previous cost focuses on symbolic robustness and transition uncertainty. A complementary direction is to incorporate physical criteria into the graph cost. This is particularly relevant for the considered biped because some of the model reductions used in previous abstraction-based studies modify the actuation structure of the robot.

In particular, previous works fixed the global orientation of the robot in order to reduce the dimension of the abstraction. This simplification removes one floating-base degree of freedom. As a consequence, the reduced model no longer has the same underactuated structure as the original robot: the remaining four generalized coordinates are directly controlled by four motor inputs. The reduced model is therefore fully actuated.

This creates a discrepancy with the physical balance mechanism of a biped in single support. In the original underactuated model, the robot can rotate around the stance foot, and the motion is strongly constrained by the pendulum-like dynamics of the center of mass. By fixing the global orientation, this underactuated rotational behaviour is removed from the model. Previous works therefore compensated for this loss by forcing the motion to satisfy an inverted-pendulum-like relation.

This interpretation can be seen from the sagittal moment balance. Let  $p_x$  denote the support point,  $x_c$  and  $z_c$  the center of mass coordinates, and  $\tau_{\text{hip}}$  the hip torque contribution. Using the Euler equation :

$$\tau_{\text{hip}} - mg(p_x - x_c) = mz_c\ddot{x}_c.$$

Equivalently,

$$p_x = x_c - \frac{z_c}{g}\ddot{x}_c + \frac{\tau_{\text{hip}}}{mg}.$$

If the hip torque contribution is set to zero, this reduces to the usual inverted-pendulum or ZMP-like relation

$$p_x = x_c - \frac{z_c}{g}\ddot{x}_c.$$

Thus, imposing this simplified relation on the fully actuated reduced model amounts to forcing the motion to behave as if no corrective hip torque were available. In other

words, the inverted-pendulum constraint artificially reintroduces an underactuated balance behaviour that was lost when the global orientation was fixed.

This corresponds to a passive inverted-pendulum-like evolution around the stance foot. Large values of  $\tau_{\text{hip}}$ , on the other hand, indicate that the fully actuated reduced model is using the hip to generate motions that would not naturally arise from the underactuated balance dynamics.

This motivates a hip-torque-aware graph cost. Instead of enforcing the inverted-pendulum relation as a hard constraint, one can synthesize the symbolic controller with a cost that penalizes the hip torque required by each transition. For example, for a transition  $(s, a, t)$ , one may define

$$c_{\tau}(s, a, t) = \int_0^{T_{\text{step}}} \tau_{\text{hip}}(t)^2 dt,$$

The controller can then be synthesized using Dijkstra’s algorithm with the objective

$$J_{\tau} = \sum_{k=0}^{N-1} c_{\tau}(s_k, a_k, s_{k+1}).$$

This formulation makes explicit what was previously imposed implicitly through the inverted-pendulum relation. The abstraction can still be generated from the reduced fully actuated model, but the graph search is biased toward trajectories that require little hip correction. Such trajectories are closer to the underactuated balance behaviour of an inverted pendulum around the stance foot.

Preliminary simulations using this idea showed that minimizing the hip-torque cost with Dijkstra’s algorithm produces a gait qualitatively similar to the motions obtained in previous theses. This is consistent with the interpretation above: previous works enforced an inverted-pendulum-like equation to compensate for the fully actuated reduced model, whereas the weighted graph formulation achieves a similar effect by penalizing the hip torque directly during controller synthesis. A video illustrating the resulting Dijkstra-based resolution and the corresponding gait can be accessed through the QR code in Figure 6.1.



Figure 6.1: Dijkstra-based synthesized controller using the hip-torque-aware cost.

## 6.4 Combined Robust and Physical Synthesis

The most promising future direction is to combine the robustness and physical criteria in a single weighted graph formulation. A possible transition cost is

$$c(s, a, t) = c_{\text{step}} + \lambda_{\text{err}} \left( \hat{L}(s, a) \eta_s + \eta_t \right) + \lambda_{\tau} c_{\tau}(s, a, t).$$

The first term penalizes long paths, the second term penalizes transition-error propagation and the last term favors motions that require, for example, smaller hip torques or energy consumption.

This formulation follows naturally from the previous chapters. The graph is generated using the scalable methods of Chapter 3; the error terms come from the sensitivity and discretization analysis of Chapter 4; the grid can be chosen using the anisotropic strategy of Chapter 5; and the hip-torque term connects the symbolic controller to the physical constraints observed on the experimental robot.

Such a controller would not only seek a target-reaching path. It would seek a path that is short, robust to discretization, well represented by the graph, and physically compatible with the balance mechanism of the platform.

## 6.5 Discussion

The central idea of these further investigations is that the abstraction should not be viewed only as a reachability graph. It is also a finite approximation of the continuous dynamics, and the quality of the synthesized controller depends directly on the quality of this approximation.

Reducing transition errors is therefore important for two reasons. First, it improves robustness: two continuous states represented by the same symbolic state should not diverge too much after one transition. Second, it improves the effective resolution of the graph: transition budget should be allocated to directions and regions where the dynamics produces genuinely different successors, while coarser representations are acceptable where successors remain globally close.

If accurate estimates of  $L$  and successor-centering defects were available over the full abstraction, Dijkstra’s algorithm could be used to synthesize controllers that explicitly minimize the accumulated transition-error bound

$$\sum_{k=0}^{N-1} \left( \hat{L}_k \eta_k + \eta_{k+1} \right),$$

possibly together with physical costs such as the required hip torque. This would extend the abstraction-based framework from pure reachability synthesis toward robust, resolution-aware, and physically meaningful locomotion synthesis.

The approach remains heuristic because the quantity  $\hat{L}(s, a)$  is estimated from simulations and does not provide certified continuous-time guarantees. However, it offers a practical way to exploit the information already computed during abstraction generation and to make the final policy more consistent with both the symbolic approximation error and the mechanics of the robot.

# Chapter 7

## Conclusion

This thesis studied how abstraction-based control can be made more scalable and more robust for bipedal locomotion. Symbolic abstractions are attractive because they transform nonlinear control problems into finite graph problems, but their size grows rapidly with the state and input dimensions. This limitation is especially important for bipedal robots, whose dynamics are nonlinear, hybrid, and sensitive to contact changes.

The objective was not to recover full continuous-state formal guarantees, but to develop practical tools for simulation-based abstractions. The thesis focused on the trade-off between scalability, abstraction accuracy, and robustness to discretization errors.

First, scalable abstraction generation was improved through three complementary ideas: exploiting left-right symmetry, distributing transition computations across independent jobs, and pruning physically irrelevant regions before simulation. These strategies reduce the cost of generating large abstractions, while also showing that memory usage and controller synthesis become major bottlenecks once transition computation is sufficiently parallelized.

Second, discretization-error propagation was analyzed. Since the abstractions used in this work are center-based, a transition computed from a cell center does not necessarily represent all concrete states inside the cell. A Lipschitz-based interpretation was used to study local error amplification, and finite-difference sampling provided empirical sensitivity indicators. The results showed that percentile-based indicators are more robust than maximum sampled amplifications and that, at the tested scale, joint-position variables contribute more strongly to large propagated errors than velocity variables.

Third, these sensitivity indicators were used to design a global anisotropic grid-optimization method. The method preserves the Cartesian-product structure of the abstraction while redistributing a fixed transition budget across the state variables. The optimized grid allocated more resolution to the most sensitive dimensions and reduced the sampled propagated discrepancies in an independent validation. This does not prove global optimality, but it shows that empirical sensitivity information can guide a more meaningful use of a fixed abstraction budget.

Finally, the thesis discussed how robustness and physical information could be incorporated directly into controller synthesis. Transition costs based on error amplification, successor quantization, and hip-torque requirements were proposed to favor symbolic policies that are both more reliable and more physically meaningful. These ideas remain future extensions rather than fully validated closed-loop controllers.

Several limitations remain. The abstractions are simulation-based and center-based, so they do not provide certified guarantees for all continuous states inside each cell. The sensitivity and grid-optimization methods rely on finite random samples and should be interpreted as empirical diagnostics rather than worst-case bounds. Physical deployment was also not achieved and would require reliable state estimation, contact monitoring, actuator-aware low-level control, and experimental validation.

A promising future direction is to combine the scalability of the present simulation-based abstraction with the formal guarantees of robust abstractions. Constructing a conservative abstraction over the full high-dimensional state space remains computationally prohibitive. However, the nominal symbolic controller computed from the scalable abstraction identifies a structured set of relevant transitions, trajectories, and reachable regions. Instead of robustifying the whole domain uniformly, one could build a local robust abstraction around the closed-loop trajectory, or around the subset of symbolic states effectively used by the controller.

This would provide an intermediate path between purely empirical simulation-based control and fully conservative abstraction-based synthesis. A global abstraction would first identify a feasible symbolic policy, while a local robust analysis would then certify only the transitions that are relevant for execution. Such a strategy could help recover formal guarantees without facing the full curse of dimensionality associated with a uniform robust abstraction of the complete state space.

Other future work includes addressing the memory bottleneck in controller synthesis through sparse graph representations, on-demand transition generation, or local synthesis around relevant regions of the abstraction. The robust and physically informed transition costs proposed in this thesis should also be implemented and evaluated in closed-loop simulation. Finally, deployment on the physical robot would require connecting the symbolic controller to a reliable low-level control and estimation pipeline.

In summary, this thesis shows that abstraction-based control for bipedal locomotion can be pushed further by combining scalable computation, symmetry exploitation, pruning, empirical sensitivity analysis, and global anisotropic grid design. While these methods do not yet recover the full guarantees of conservative symbolic control, they provide practical tools for improving the scalability-robustness trade-off in high-dimensional robotic systems.

# Appendix A

## Dimension-Wise Sensitivity Results

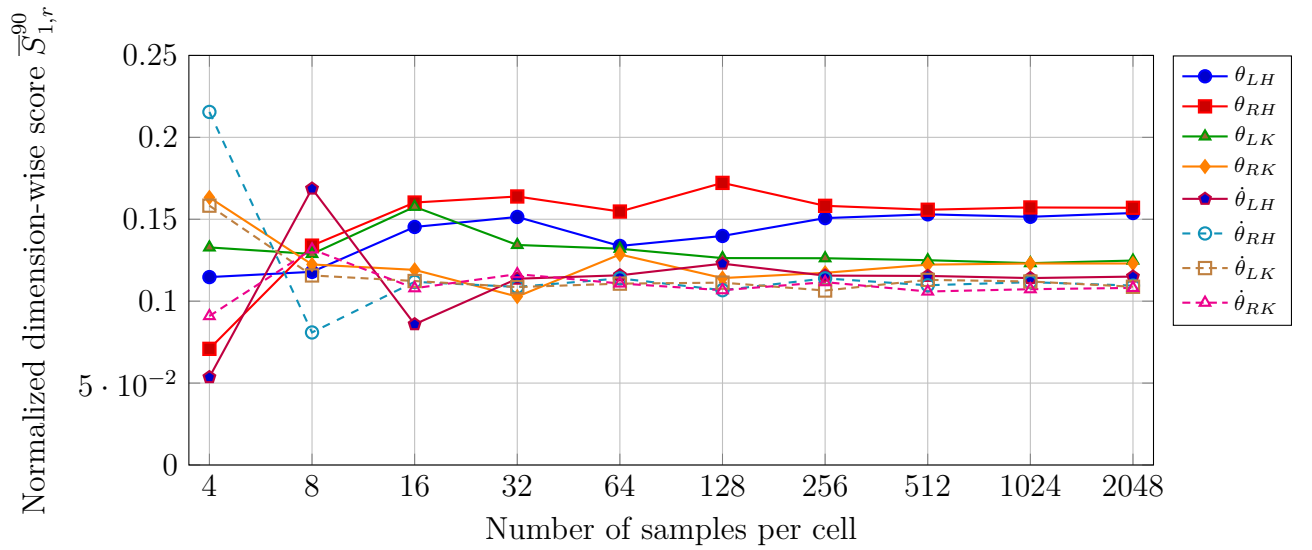


Figure A.1: Evolution of the normalized dimension-wise sensitivity scores  $\bar{S}_{1,r}^{90}$  as a function of the number of sampled states per cell for cell  $C_1$ .

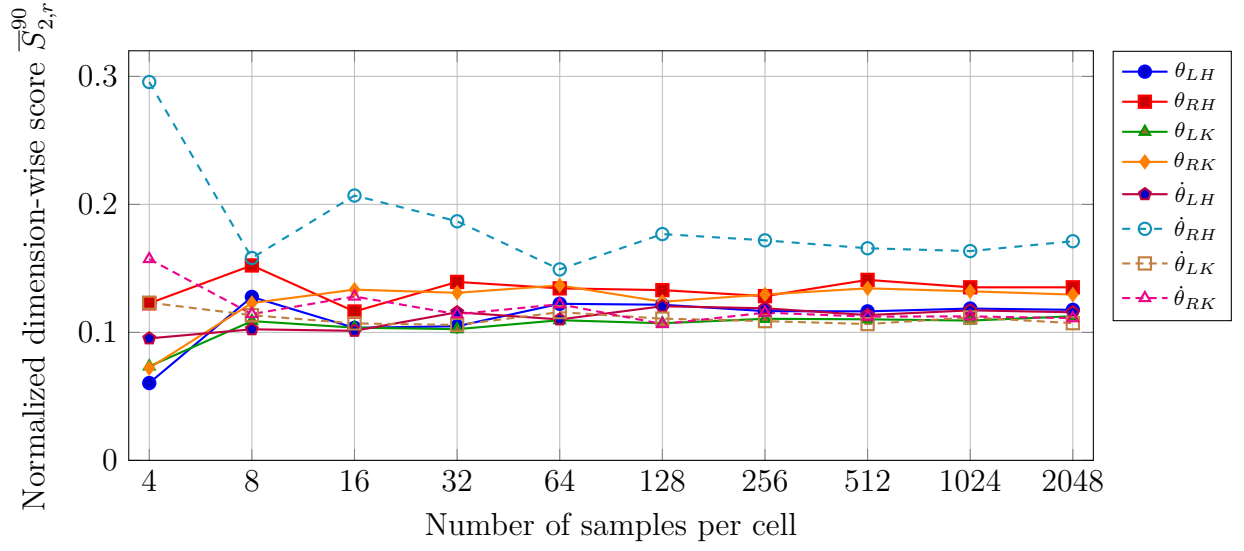


Figure A.2: Evolution of the normalized dimension-wise sensitivity scores  $\bar{S}_{2,r}^{90}$  as a function of the number of sampled states per cell for cell  $C_2$ .

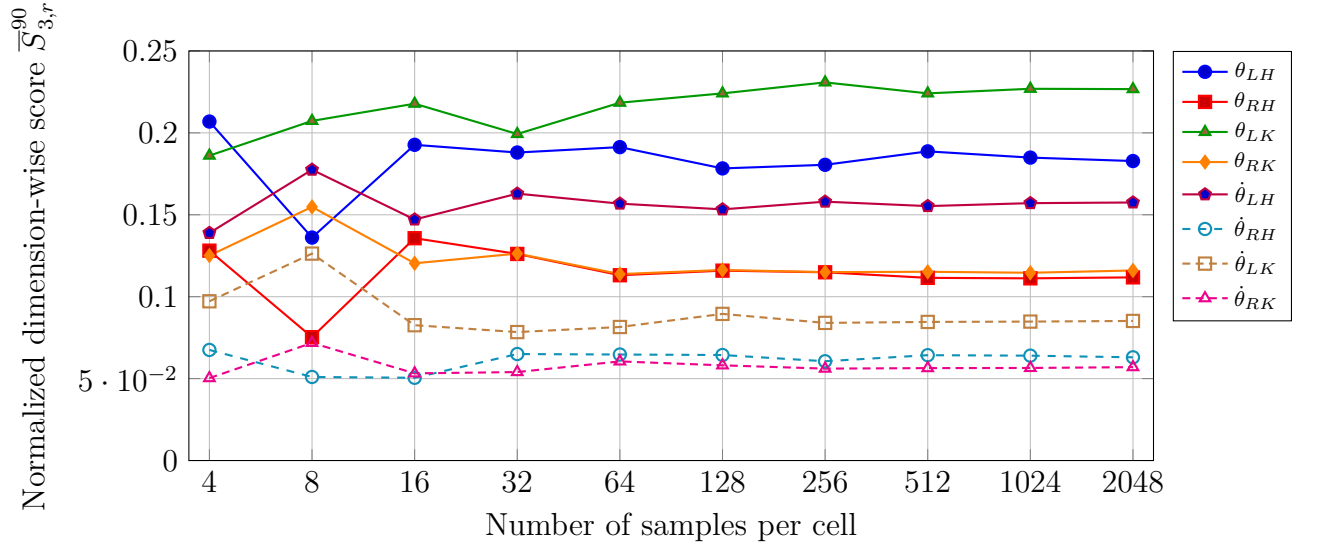


Figure A.3: Evolution of the normalized dimension-wise sensitivity scores  $\bar{S}_{3,r}^{90}$  as a function of the number of sampled states per cell for cell  $C_3$ .

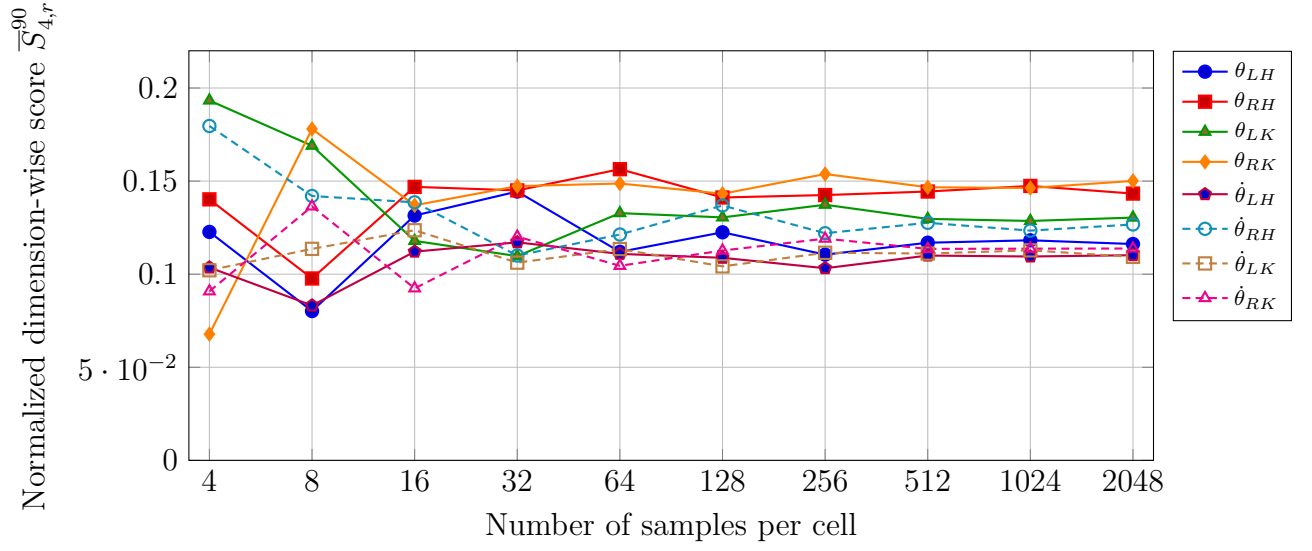


Figure A.4: Evolution of the normalized dimension-wise sensitivity scores  $\bar{S}_{4,r}^{90}$  as a function of the number of sampled states per cell for cell  $C_4$ .

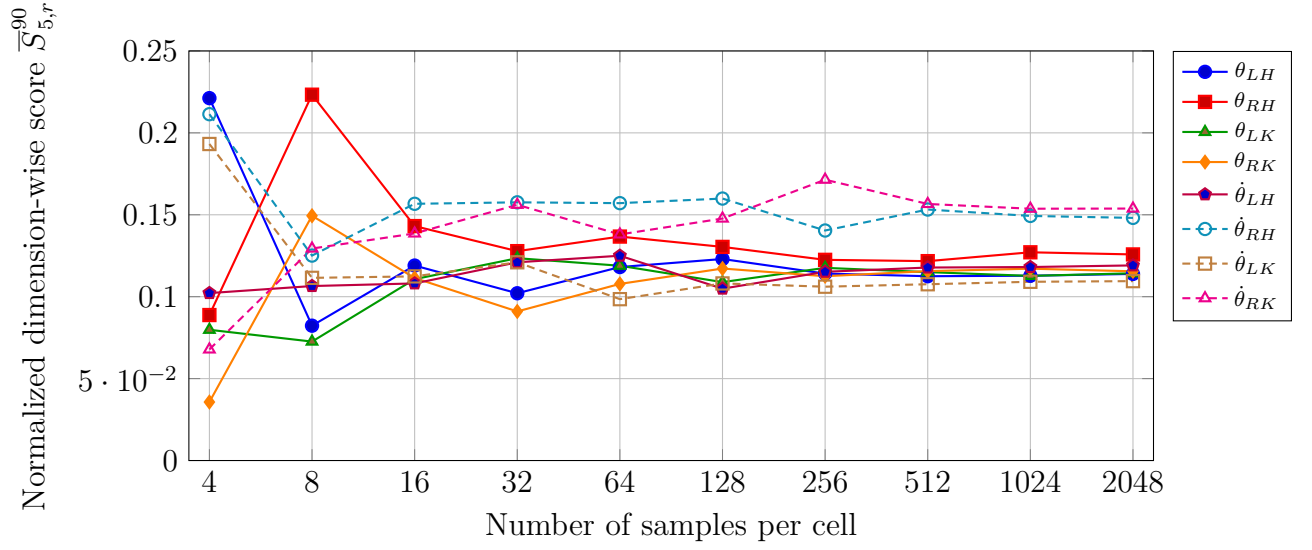


Figure A.5: Evolution of the normalized dimension-wise sensitivity scores  $\bar{S}_{5,r}^{90}$  as a function of the number of sampled states per cell for cell  $C_5$ .

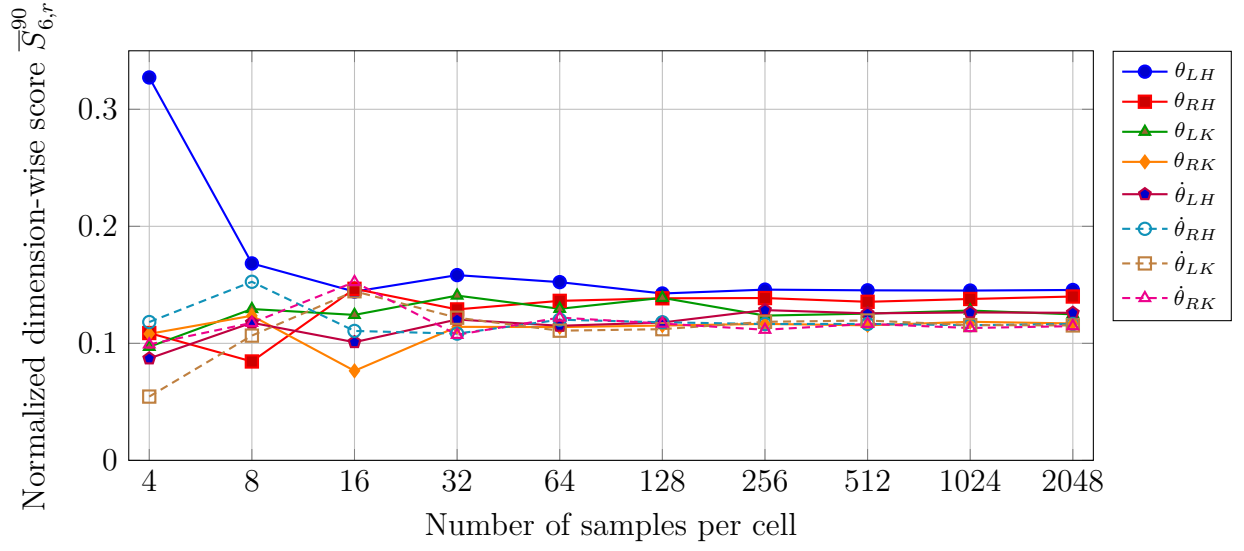


Figure A.6: Evolution of the normalized dimension-wise sensitivity scores  $\bar{S}_{6,r}^{90}$  as a function of the number of sampled states per cell for cell  $C_6$ .

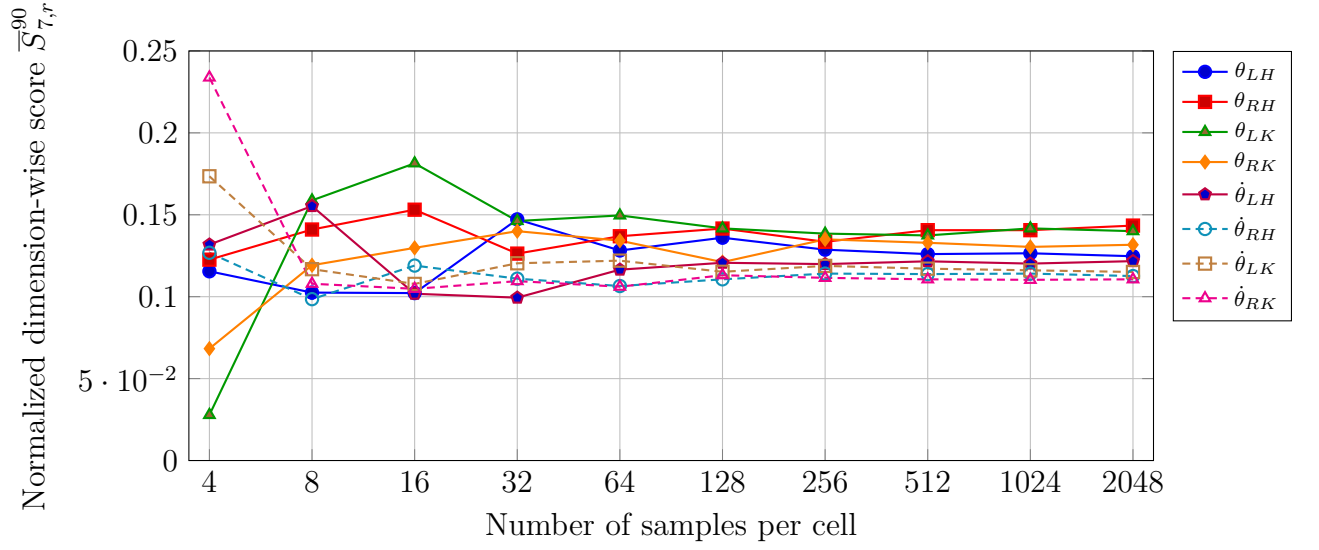


Figure A.7: Evolution of the normalized dimension-wise sensitivity scores  $\bar{S}_{7,r}^{90}$  as a function of the number of sampled states per cell for cell  $C_7$ .

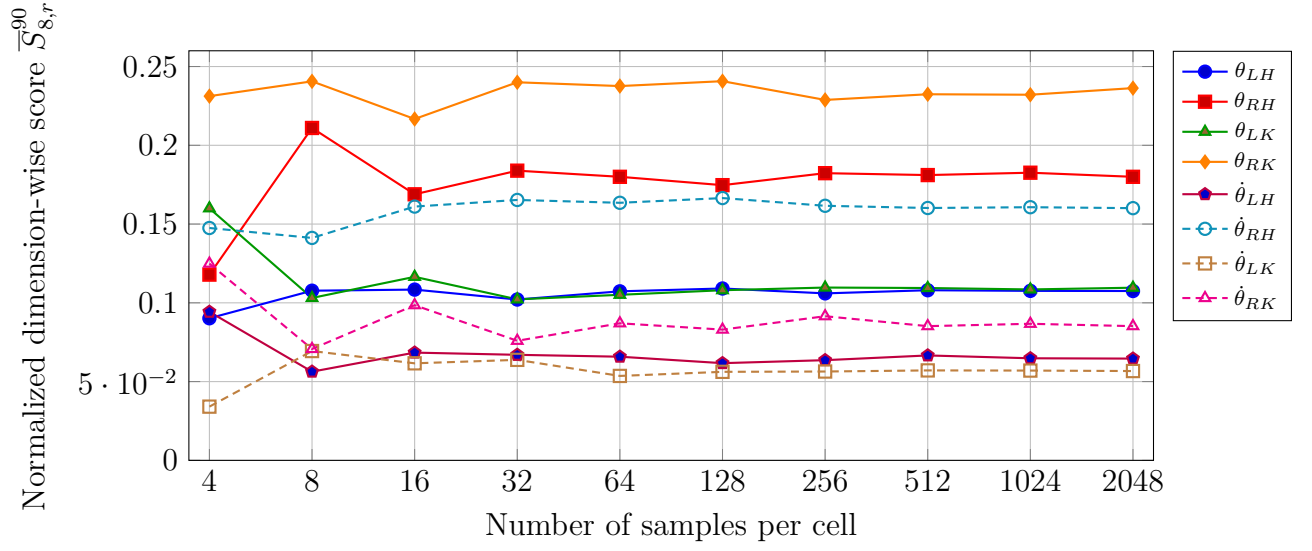


Figure A.8: Evolution of the normalized dimension-wise sensitivity scores  $\bar{S}_{8,r}^{90}$  as a function of the number of sampled states per cell for cell  $C_8$ .

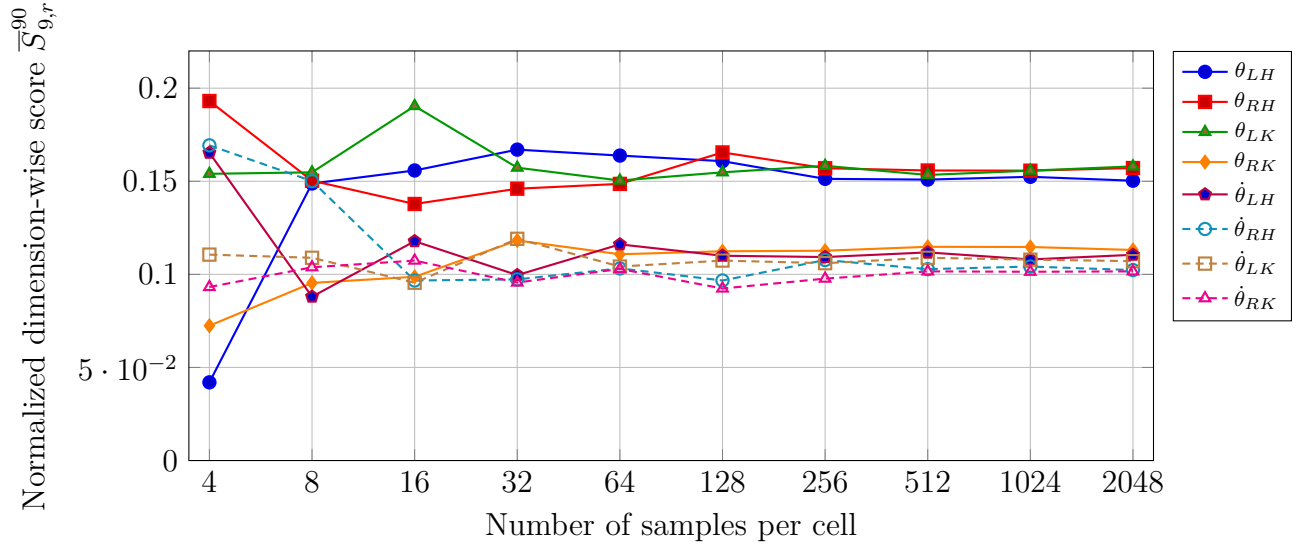


Figure A.9: Evolution of the normalized dimension-wise sensitivity scores  $\bar{S}_{9,r}^{90}$  as a function of the number of sampled states per cell for cell  $C_9$ .

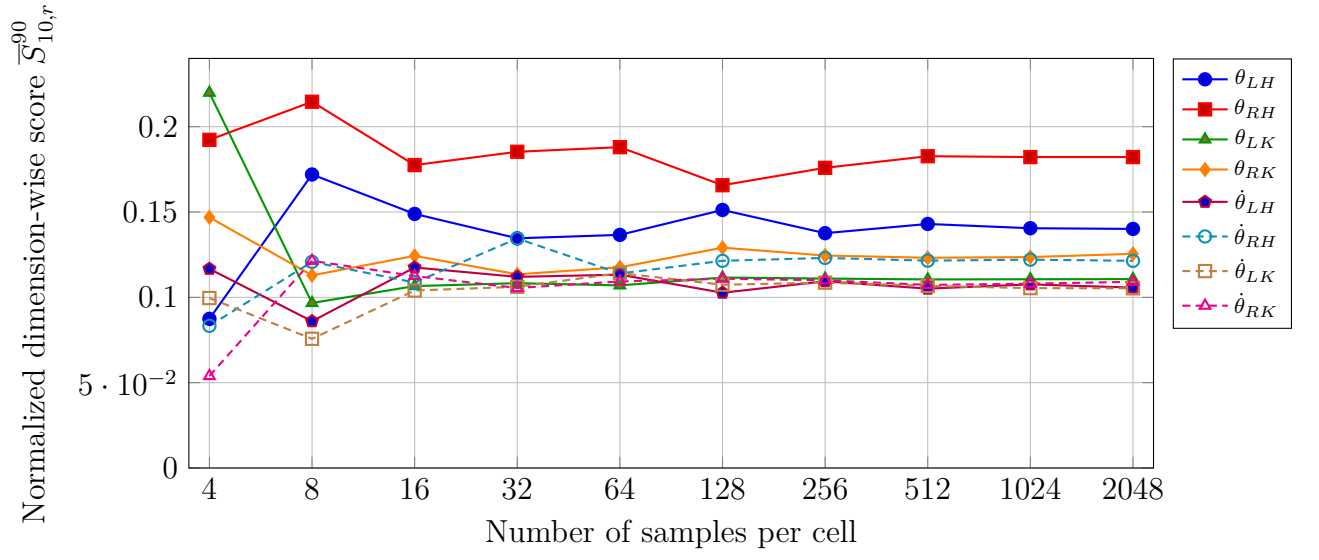


Figure A.10: Evolution of the normalized dimension-wise sensitivity scores  $\bar{S}_{10,r}^{90}$  as a function of the number of sampled states per cell for cell  $C_{10}$ .

# Bibliography

- [1] A. Abate and S. Soudjani. Adaptive and sequential gridding procedures for the abstraction and verification of stochastic processes. *SIAM Journal on Applied Dynamical Systems*, 12(4):1740–1773, 2013.
- [2] Eric Ambrose, Wen-Loong Ma, Christian Hubicki, and Aaron D. Ames. Toward benchmarking locomotion economy across design configurations on the modular robot: Amber-3m. In *2017 IEEE Conference on Control Technology and Applications (CCTA)*, 2017. doi:[10.1109/CCTA.2017.8062633](https://doi.org/10.1109/CCTA.2017.8062633).
- [3] Cédric Amerijckx and Maxime Morabito. Implementation of abstraction-based control on a 5-link planar robot. Master’s thesis, Université catholique de Louvain, 2025. Accessed: 2026-02-20.
- [4] Julien Calbert. *Abstraction Relations over Dynamical Systems: Theory and Algorithms*. Ph.d. dissertation, Université catholique de Louvain (UCLouvain), Louvain-la-Neuve, Belgium, 2024.
- [5] Christine Chevallereau, Gabriel Abba, Yannick Aoustin, Franck Plestan, Eric R. Westervelt, Carlos Canudas-de-Wit, and Jessy W. Grizzle. RABBIT: A testbed for advanced control theory. *IEEE Control Systems Magazine*, 23(5):57–79, October 2003. doi:[10.1109/MCS.2003.1234651](https://doi.org/10.1109/MCS.2003.1234651).
- [6] CÉCI. Lemaitre4 cluster. <https://www.ceci-hpc.be/clusters/lemaitre4/>. Accessed: 2026-05-28.
- [7] Briec De Poucques. Model predictive control approach to enhance stable walking for planar bipedal robots. Master’s thesis, Université catholique de Louvain, 2025. Accessed: 2026-02-20.
- [8] Edsger W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959. doi:[10.1007/BF01386390](https://doi.org/10.1007/BF01386390).
- [9] Magnus B. Egerstedt and Roger W. Brockett. Feedback can reduce the specification complexity of motor programs. *IEEE Transactions on Automatic Control*, 48(2):213–223, 2003. doi:[10.1109/TAC.2002.808466](https://doi.org/10.1109/TAC.2002.808466).
- [10] A. Girard and G. Gössler. Safety synthesis for incrementally stable switched systems using discretization-free multi-resolution abstractions. *Acta Informatica*, 57(3):417–438, 2020.

- [11] Jessy W. Grizzle and Christine Chevallereau. Virtual constraints and hybrid zero dynamics for realizing underactuated bipedal locomotion, 2017. [arXiv:1706.01127](#), [doi:10.48550/arXiv.1706.01127](#).
- [12] Felix Gruber, Eric S. Kim, and Murat Arcak. Sparsity-aware finite abstraction. In *Proceedings of the 56th IEEE Conference on Decision and Control*, pages 2366–2371, 2017. [doi:10.1109/CDC.2017.8263995](#).
- [13] Shuuji Kajita, Hirohisa Hirukawa, Kensuke Harada, and Kazuhito Yokoi. *Introduction to Humanoid Robotics*, volume 101 of *Springer Tracts in Advanced Robotics*. Springer, Berlin, Heidelberg, 2014. [doi:10.1007/978-3-642-54536-8](#).
- [14] Mahmoud Khaled, Eric S. Kim, Murat Arcak, and Majid Zamani. Synthesis of symbolic controllers: A parallelized and sparsity-aware approach. In *Tools and Algorithms for the Construction and Analysis of Systems*, volume 11428 of *Lecture Notes in Computer Science*, pages 265–281. Springer, 2019. [doi:10.1007/978-3-030-17465-1\\_15](#).
- [15] Shishir Kolathaya, Ryan Sinnet, Wenlong Ma, and Aaron Ames. Human-inspired walking in AMBER 1.0 and AMBER 2.0. Presented at Dynamic Walking 2013, June 2013. URL: [https://www.cmu.edu/dynamic-walking/files/abstracts/Kolathaya\\_2013\\_DW.pdf](https://www.cmu.edu/dynamic-walking/files/abstracts/Kolathaya_2013_DW.pdf).
- [16] Wen-Loong Ma, Shishir Kolathaya, Eric R. Ambrose, Christian M. Hubicki, and Aaron D. Ames. Bipedal robotic running with durus-2d: Bridging the gap between theory and experiment. In *Proceedings of the 20th International Conference on Hybrid Systems: Computation and Control*, pages 265–274, 2017. [doi:10.1145/3049797.3049823](#).
- [17] Tadeusz Mikolajczyk, Emilia Mikołajewska, Hayder F. N. Al-Shuka, Tomasz Malinowski, Adam Kłodowski, Danil Yurievich Pimenov, Tomasz Paczkowski, Fuwen Hu, Khaled Giasin, Dariusz Mikołajewski, and Marek Macko. Recent advances in bipedal walking robots: Review of gait, drive, sensors and control systems. *Sensors*, 22(12):4440, 2022. [doi:10.3390/s22124440](#).
- [18] Rémi Munos and Andrew Moore. Variable resolution discretization in optimal control. *Machine Learning*, 49:291–323, 2002. [doi:10.1023/A:1017992615625](#).
- [19] Pandaily Tech News. Behind honor’s robot marathon: 7 chinese suppliers powering the “lightning” robot, May 2026. [Online]. Available: <https://pandaily.com>. URL: <https://pandaily.com/honor-robot-marathon-7-chinese-suppliers>.
- [20] George J. Pappas. Bisimilar linear systems. *Automatica*, 39(12):2035–2047, December 2003. [doi:10.1016/j.automatica.2003.07.003](#).
- [21] Giordano Pola, Antoine Girard, and Paulo Tabuada. Approximately bisimilar symbolic models for nonlinear control systems. *Automatica*, 44(10):2508–2516, 2008. [doi:10.1016/j.automatica.2008.02.021](#).

- [22] Jenna Reher and Aaron D. Ames. Dynamic walking: Toward agile and efficient bipedal robots. *Annual Review of Control, Robotics, and Autonomous Systems*, 4:535–572, May 2021. doi:[10.1146/annurev-control-071020-045021](https://doi.org/10.1146/annurev-control-071020-045021).
- [23] Gunther Reissig. Computing abstractions of nonlinear systems. *IEEE Trans. Autom. Control.*, 56(11):2583–2598, 2011. doi:[10.1109/TAC.2011.2118950](https://doi.org/10.1109/TAC.2011.2118950).
- [24] Gunther Reissig, Alexander Weber, and Matthias Rungger. Feedback refinement relations for the synthesis of symbolic controllers. *IEEE Transactions on Automatic Control*, 62(4):1781–1796, 2017. doi:[10.1109/TAC.2016.2593947](https://doi.org/10.1109/TAC.2016.2593947).
- [25] Matthias Rungger and Majid Zamani. SCOTS: A tool for the synthesis of symbolic controllers. In *Proceedings of the 19th ACM International Conference on Hybrid Systems: Computation and Control*, HSCC ’16, pages 99–104, Vienna, Austria, 2016. Association for Computing Machinery. doi:[10.1145/2883817.2883834](https://doi.org/10.1145/2883817.2883834).
- [26] Alessandro Saccon, Nathan van de Wouw, and Henk Nijmeijer. Sensitivity analysis of hybrid systems with state jumps with application to trajectory tracking. In *Proceedings of the 53rd IEEE Conference on Decision and Control*, pages 3065–3070, 2014. doi:[10.1109/CDC.2014.7039861](https://doi.org/10.1109/CDC.2014.7039861).
- [27] M. W. Spong, S. Hutchinson, and M. Vidyasagar. *Robot Modeling and Control*. John Wiley & Sons, Hoboken, NJ, USA, 2006.
- [28] Paulo Tabuada. *Verification and Control of Hybrid Systems: A Symbolic Approach*. Springer Science & Business Media, New York, NY, USA, 2009.
- [29] Russ Tedrake. Underactuated robotics: Algorithms for walking, running, swimming, flying, and manipulation. Course notes for MIT 6.832, 2024. Downloaded on May 31, 2026 from <https://underactuated.csail.mit.edu/>.
- [30] Miomir Vukobratovic and Branislav Borovac. Zero-moment point — thirty five years of its life. *International Journal of Humanoid Robotics*, 1(1):157–173, 2004. doi:[10.1142/s0219843604000083](https://doi.org/10.1142/s0219843604000083).
- [31] Alexander Weber, Matthias Rungger, and Gunther Reissig. Optimized state space grids for abstractions. *IEEE Transactions on Automatic Control*, 62(11):5816–5821, November 2017. doi:[10.1109/TAC.2016.2642794](https://doi.org/10.1109/TAC.2016.2642794).
- [32] E. R. Westervelt, J. W. Grizzle, C. Chevallereau, J. H. Choi, and B. Morris. *Feedback Control of Dynamic Bipedal Robot Locomotion*. CRC Press, Boca Raton, FL, USA, 2007.
- [33] P.-B. Wieber. Trajectory free linear model predictive control for stable walking in the presence of strong perturbations. In *Proceedings of the 2006 6th IEEE-RAS International Conference on Humanoid Robots*, pages 137–142, Genova, Italy, 2006. doi:[10.1109/ICHR.2006.321375](https://doi.org/10.1109/ICHR.2006.321375).
- [34] Xinhua Sports. Team monkey king’s autonomous humanoid robot “lightning” wins 2026 beijing e-town humanoid robot half marathon, April 2026. [Online]. URL: <https://www.news.cn>.

- [35] T. Yang, E. R. Westervelt, J. P. Schmiedeler, and R. A. Bockbrader. Design and control of a planar bipedal robot ernie with parallel knee compliance. *Autonomous Robots*, 25:317–330, 2008.
- [36] Andy B. Yoo, Morris A. Jette, and Mark Grondona. SLURM: Simple linux utility for resource management. In *Job Scheduling Strategies for Parallel Processing*, volume 2862 of *Lecture Notes in Computer Science*, pages 44–60. Springer, 2003. [doi:10.1007/10968987\\_3](https://doi.org/10.1007/10968987_3).
- [37] M. Zamani, G. Pola, M. Mazo, and P. Tabuada. Symbolic models for nonlinear control systems without stability assumptions. *IEEE Transactions on Automatic Control*, 57(7):1804–1809, 2012.

UNIVERSITÉ CATHOLIQUE DE LOUVAIN  
École polytechnique de Louvain

Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve, Belgique | [www.uclouvain.be/epl](http://www.uclouvain.be/epl)