Policy Decomposition: A Framework for Discovery of Control Hierarchies for Efficient Policy Optimization with Suboptimality Estimates

Ashwin Rajendra Khadke

CMU-RI-TR-24-38

July 12, 2024

School of Computer Science
The Robotics Institute
Carnegie Mellon University
Pittsburgh, PA

Thesis Committee:
Hartmut Geyer, *Chair*
Christopher Atkeson
Zachary Manchester
Alex Gorodetsky, *(Univeristy of Michigan)*
Nikolai Matni, *(Univeristy of Pennsylvania)*

Keywords: Optimal control and reinforcement learning, Hierarchical control, Evolutionary methods

To my parents, for being my loudest cheerleaders, and to Prasad Kaka, for encouraging me to keep asking questions

Abstract

Optimal Control is a popular formulation for designing controllers for dynamic robotic systems. Under the formulation, the desired long-term behavior of the system is encoded via a cost function and the policy, i.e. a mapping from the state of the system to control commands, to achieve the desired behavior is derived by solving an optimization problem. A fundamental challenge in scaling up policy optimization to complex systems is that the computational requirement scales exponentially with the dimensionality of the state-space. Owing to this *curse of dimensionality* simplifying hierarchies are employed to reduce the computational burden. Very often, these hierarchies are hand-designed based on intuitions about the system's dynamics, and do not account for their effect on the system's closed-loop behavior under the resulting policies. The systematic design of hierarchies to simplify controller synthesis is a critical and active area of research and is the focus of this work.

This thesis introduces Policy Decomposition, a framework that alleviates the curse of dimensionality by algorithmically reducing a complex policy optimization problem into a hierarchy of simpler subproblems that are much more tractable to solve. Two standout features of this framework are its ability to 1) automatically propose control hierarchies and 2) estimate *a priori* how the control performance under policies resulting from different hierarchies compares with the optimal policy. Additionally, we develop search methods based on Genetic Algorithm and Monte Carlo tree search to automatically discover promising hierarchies. Therefore, those that dramatically reduce the required computation in policy optimization while sacrificing minimally on control performance can be readily identified. The framework is agnostic to the choice of policy representations and optimization algorithms.

We demonstrate the generality of the Policy Decomposition framework by applying it towards finding hierarchies for several robotic systems, including the control of a simplified biped, and a quadcopter. Furthermore, we present results using Policy Iteration with look-up table based policy representations as well as more modern methods such as Proximal Policy Optimization with neural network policies. The discovered hierarchies either outperform heuristically constructed ones in closed-loop performance or provide dramatic reductions in required compute with marginally suboptimal control performance.

Acknowledgments

The ideas presented in this thesis have evolved through insights and stimulating criticisms from several individuals. The work to fully explore the consequences of these ideas would not have come to fruition without the advice, support, and constant motivation from many more. Therefore, I begin this thesis by expressing my sincere gratitude for them.

I am immensely grateful to my advisor, Hartmut Geyer, for his guidance in refining my research ideas, his selfless support for opportunities that aided my professional development, and his unwavering encouragement during the difficult stages of my thesis. Hartmut motivated me to not deter from difficult research problems, gave me time and space to explore ideas that at times did not pan out, and encouraged (at times even partook in) my extra curricular interests to maintain a balanced lifestyle. I will miss his cool temperament and zeal for scientific adventure.

Next, I would like to thank my committee members. Chris Atkeson, for always asking the right questions and helping direct my efforts toward the most pressing problems. Zac Manchester, for his practical advice on optimization algorithms and navigating academic research. Nikolai Matni, for lending his expertise in controls. The Robotics Institute at Carnegie Mellon University is a remarkably supportive environment for graduate students, and I am incredibly grateful for their unconditional financial support for my thesis. Additionally, this work would not have been possible without the computational resources provided through allocation CIS220132 from the Advanced Cyberinfrastructure Coordination Ecosystem: Services and Support (ACCESS) program, which is supported by National Science Foundation grants #2138259, #2138286, #2138307, #2137603, and #2138296.

I was lucky to be a part of a very supportive research group during my PhD. I would like to thank Akshara Rai for her mentorship during my first year. I am grateful to William Martin for patiently answering all my questions regarding the ATRIAS robot. I am thankful to Timothy Kyung for his presence through many milestones during my PhD. I am also grateful to Nitish Thatte and Jennifer Padgett for their support, encouragement, and grounded advice in navigating graduate school.

I was fortunate to find remarkable friends during my stay in Pittsburgh.

Through 8 years of graduate school, Arpit Agarwal has been a colleague, a friend, and at times even a parent. His passion for robotics research is infectious and inspiring. Arpit supported me during my lows and cheered me on in my successes. Life in Pittsburgh has been all the more memorable with him, and I could not be more grateful for his companionship. I am also grateful to Kartik Iyer and Fengying Yang for their care and support. I would like to thank Muhammad Suhail for his affable company, Ramkumar Natarajan and Anahita Mohseni-Kabir for being my partners in several misadventures, and Simin Liu for motivating me to do my best.

Hiren and Radhika Muzumdar became my family away from home. Their presence made me never feel homesick. I could visit them or call them up any time and have a discussion about anything, just like I could with my parents. They are a huge reason why Pittsburgh feels like a second home to me.

Finally, I would like to thank my parents, my sister Savari, and my grandmother for their unconditional love and support. My parents taught me that the only sensible way to live is to pursue what you love. I owe them all my accomplishments.

Contents

When this dissertation is viewed as a PDF, the page header is a link to this Table of Contents.

List of Figures

List of Tables

Chapter 1

Introduction

Optimal Control is a popular formulation for designing controllers for dynamical systems that has found applications in varied domains [47, 67, 88] especially robotics [27]. The solution to an optimal control problem is a policy that minimizes a specified cost function. The optimal policy is a mapping from the state of the system to control commands that regulate the system to exhibit the desired behavior encoded by the cost function. Although several policy optimization methods exist [1, 8, 11, 73, 79] the complexity of synthesizing the *global* optimal policy scales exponentially with the dimensionality of the state-space [9]. As a consequence, several simplifying control architectures are adopted. The systematic design of such architectures is a critical research area [3] and their automated discovery is an active line of research [23, 61]. This thesis introduces Policy Decomposition, a framework to alleviate this curse of dimensionality by algorithmically reducing a complex policy optimization problem into a hierarchy of simpler subproblems that are much more tractable. A key feature of this framework is its ability to automatically propose control hierarchies/architectures and assess *a priori* how well the closed-loop behavior under resulting hierarchical policies matches with that under the optimal policy. Therefore, policy optimization for complex systems can be readily simplified while sacrificing minimally on closed-loop performance.

Several approaches have been devised to reduce the computational burden in optimizing control policies for complex systems, broadly based on three ideas: layered control [61], decentralized control [83], and novel function approximators for the policy [82]. **Layered control** [61] refers to a two or three layer hierarchical structure where a reduced

order/lower-dimensional model of the system is used to reason about its long-term behavior (either for online local policy computation in a model-predictive fashion [31, 70, 93] or for offline global policy optimization [37, 99]). For robotic systems, the reduced order models are typically hand-designed [29, 33, 45, 87]. Systematic approaches towards model reduction by attempting to match the open loop-dynamics [76] or based on controllability measures [55, 56] exist. These are more commonly applied towards controller design for large linear systems [4] and continuum systems [57], but their adoption for robotic control problems is limited [36, 69]. Reduction methods which explicitly attempt to match the optimal closed-loop performance have been proposed for networked *linear systems* [32, 100] and only recently for robotic systems [22, 30].

Decentralized control methods impose a structure on the controller whereby policies for different subspaces of the control inputs are functions of only some subspace of the system state. In some cases, structures arise naturally, such as in case of large networked systems where controllers have locality constraints on account of communication delays [96]. In other cases suitable structures have to be determined [52, 94], and the policies are then optimized in a decentralized fashion leading to reduction in computation [38, 48, 83]. Extensive literature exists on structure selection for linear systems based on interaction measures [19, 63], passivity measures [7], and controllability or observability measures [77, 98]. Works addressing control structure selection for nonlinear systems do exist [48]. However, all of these approaches are oblivious to the optimal control objective and as such may lead to substantially suboptimal closed-loop behavior. In contrast, Policy Decomposition explicitly reasons about the suboptimality of the resulting policies in determining suitable hierarchical control structures.

The Policy Decomposition framework is agnostic to the choice of policy representation and optimization algorithm. Although no particular choice of **function approximator** for representing control policies inherently resolves the curse of dimensionality, practical trade-offs arise. The simplest policy representations are look-up tables over the state-space that store the desired values for control inputs over a grid of states [10]. Dynamic programming based methods such as Policy Iteration provide strong convergence guarantees to the optimal solution when policies are represented as look-up tables [74]. However, the computational and memory complexity is exponential in the dimensionality of the state-space. Tensor-train decompositions [35] provide a compact alternative to look-up tables with polynomial complexity in the rank of the tensors they represent. Another choice

of function approximator for the optimal policy are state and input trajectories derived by optimizing for the objective of the control problem [90]. Trajectory-based dynamic programming is more memory efficient in practice but the number of trajectories required to converge to the optimal solution still grows exponentially with the dimensionality of the state-space [6]. A very popular and memory efficient choice for policy representation are neural networks [11]. Algorithms that compute neural network policies have been developed with [1, 2] and without [46, 65, 79, 80] knowing the dynamics model for the underlying system. Although, neural networks have been successful in capturing policies for control problems with very high-dimensional state-spaces [58], the obtained solutions are at best local optima and convergence guarantees are missing [12, 43].

1.1 Contributions

We introduce Policy Decomposition, a framework to reduce complex policy optimization problems into a hierarchy of simpler and much more tractable sub-problems. The framework algorithmically constructs control hierarchies by decoupling and cascading the process of computing policies for different subspaces of the control inputs. To identify promising hierarchies, Policy Decomposition solves relaxed versions of the policy optimization problem to assess *a priori* how well the closed-loop behavior under different hierarchical policies matches with that under the optimal policy. Policy Decomposition faces a combinatorial challenge whereby the number of possible hierarchies turn out to be substantial even for moderately complex systems, and we develop search algorithms to efficiently discover promising ones. We have investigated a range of optimal control problems, including the balance control of a simplified biped model, the swing-up control of planar manipulators, and flight control of quadcopter. For these problems, we have rediscovered some known control simplifications and also identified some new ones. The specific contributions of this work are

- A framework to automatically propose decomposition strategies for the optimal control of a dynamical system along with a suboptimality measure and two estimates of it to assess the quality of control policies resulting from different reductions (Chapter 2; [49]).

- Genetic Algorithm and Monte Carlo Tree Search based formulation for efficiently discovering hierarchies that optimally trade-off reduction in computation and closed-loop performance (Chapter 3; [50]).
- A method to derive system representations conducive to hierarchical policy optimization (Chapter 4; [51]).
- Experiments investigating the hierarchical policy optimization for a range of robotic systems of varying complexity for regulation (Chapters 2 to 4) and trajectory tracking control (Chapter 5)

Chapter 2

Policy Decomposition

We motivate the idea of Policy Decomposition with an illustrative example. Consider designing a control policy to swing up a pole on a cart while moving the cart to a goal position (Figure 2.1). The dynamics of this cart-pole system are given by

$$\begin{aligned}\ddot{x} =& \frac{F - \frac{\tau}{l}\cos\theta + m_p l\dot{\theta}^2 \sin\theta + \frac{m_p g}{2}\sin 2\theta}{m_c + m_p \sin^2\theta} \\ \ddot{\theta} =& \frac{\frac{\tau}{l^2}\left(\frac{m_c}{m_p} + 1\right) - \frac{F}{l}\cos\theta - \frac{m_p \dot{\theta}^2}{2}\sin 2\theta - \frac{g}{l}(m_c + m_p)\sin\theta}{m_c + m_p \sin^2\theta}\end{aligned} \tag{2.1}$$

where x and $\dot{x}$ are the horizontal position and velocity of the cart, θ and $\dot{\theta}$ are the angle and angular velocity of the pole, and the cart force F and pole torque τ are the two control inputs driving the system. Parameters m_p and m_c are the masses of the pole and cart, respectively, l is the pole length, and g is the gravitational acceleration (Figure 2.1(a)). Optimal policies π_F^* and π_τ^* are obtained by jointly solving for the two over a 4-dimensional state-space. However, note that the pole dynamics are independent of the position and velocity of the cart (Equation (2.1)). Therefore, a cascaded control optimization, where a policy $\pi_\tau(\theta, \dot{\theta})$ is first computed disregarding the cart and then a policy $\pi_F(x, \dot{x}, \theta, \dot{\theta})$ is derived keeping π_τ fixed, produces close to optimal closed-loop behavior (Figure 2.1(c)) while offering reduction in computation. Additionally, if the cart has significantly higher inertia relative to the pole ($m_c \gg m_p$), solving for $\pi_\tau(\theta, \dot{\theta})$ and $\pi_F(x, \dot{x})$ in a decoupled fashion would further reduce the computation. Several hierarchies can be constructed

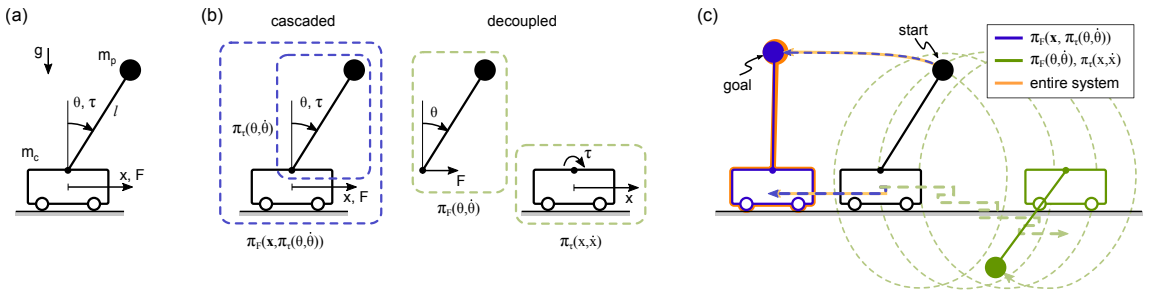


Figure 2.1: Policy Decomposition for cart-pole system shown in (**a**). (**b**) Cascaded and decoupled examples of policy decomposition. (**c**) Resulting closed-loop behavior (blue and green) in comparison to optimal control of entire system (red).

from just a combination of cascading and decoupling, however it isn't always intuitive to predict how closely the resulting policies approximate the optimal ones. For example, the decoupled strategy shown in figure 2.1(b) produces an unstable policy (Figure 2.1(c)).

We introduce Policy Decomposition, a framework to reduce the required computation in policy optimization for complex systems. It achieves this by constructing a hierarchy of cascaded and/or decoupled lower-dimensional optimizations for subsystems of the full-system that are much more tractable to solve. A key feature of Policy Decomposition is that it estimates the suboptimality of the policies resulting from a hierarchy a priori, and thus hierarchies that offer dramatic reduction in computation while sacrificing minimally on control performance can be algorithmically discovered. In section 2.1, we formally introduce the Policy Decomposition framework, specifically the approach to construct hierarchies using a combination of the cascading and decoupling strategies. We also present an intuitive abstraction that encodes the recipe to compute policies under such hierarchies. Subsequently, in section 2.2 we introduce a metric to quantify the suboptimality of a hierarchical policy and present methods to estimate it. In section 2.3, we identify hierarchies that offer near optimal control performance with significantly lower policy computation times for several example systems.

2.1 Core Idea

To formally develop the idea of policy decomposition, we consider the general dynamical system

$$\dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u}) \tag{2.2}$$

with state $\boldsymbol{x}$ and input $\boldsymbol{u}$. The optimal control for this system is defined as the control policy $\pi^*_{\boldsymbol{u}}(\boldsymbol{x})$ that minimizes

$$J = \int_0^\infty e^{-\lambda t} c(\boldsymbol{x}(t), \boldsymbol{u}(t))\, dt. \tag{2.3}$$

This objective function describes the discounted sum of some costs $c(\boldsymbol{x}, \boldsymbol{u})$ accrued over time with the discount factor λ characterizing the trade-off between immediate and future costs. We assume a quadratic cost function,

$$c(\boldsymbol{x}, \boldsymbol{u}) = (\boldsymbol{x} - \boldsymbol{x}^d)^T \boldsymbol{Q} (\boldsymbol{x} - \boldsymbol{x}^d) + (\boldsymbol{u} - \boldsymbol{u}^d)^T \boldsymbol{R} (\boldsymbol{u} - \boldsymbol{u}^d) \tag{2.4}$$

where $\boldsymbol{x}^d$ and $\boldsymbol{u}^d$ define the goal state and input. For the sake of simplicity and without loss of generality we illustrate the main ideas behind Policy Decomposition considering optimal regulation problems where $\boldsymbol{x}^d$ is fixed and $\boldsymbol{u}^d$ is the input that stabilizes the system at this state. We discuss how the framework can be readily extended to trajectory tracking problems in chapter 5.

Instead of jointly optimizing policies for all control inputs to a system $\pi^*_{\boldsymbol{u}}(\boldsymbol{x})$, Policy Decomposition computes lower-dimensional sub-policies for individual subsets of inputs. These sub-policies are a function of only a subset of state variables and are derived by solving lower-dimensional optimal control problems in either a cascaded or decoupled fashion leading to reduction in policy computation times. These sub-policies together form the hierarchical policy $\pi^\delta_{\boldsymbol{u}}(\boldsymbol{x})$ for the entire system (Figure 2.2).

A sub-policy $\pi_{\boldsymbol{u}_i}(\boldsymbol{x}_i)$ is the solution to the optimal control problem characterized by the subsystem dynamics and cost,

$$\begin{aligned} \dot{\boldsymbol{x}}_i &= \boldsymbol{f}_i(\boldsymbol{x}_i, \boldsymbol{u}_i \mid \bar{\boldsymbol{x}}_i = \bar{\boldsymbol{x}}_i^d, \bar{\boldsymbol{u}}_i = \pi_{\bar{\boldsymbol{u}}_i}(\boldsymbol{x}_i)), \\ c_i(\boldsymbol{x}_i, \boldsymbol{u}_i) &= c(\boldsymbol{x}_i, \boldsymbol{u}_i \mid \bar{\boldsymbol{x}}_i = \bar{\boldsymbol{x}}_i^d, \bar{\boldsymbol{u}}_i = \pi_{\bar{\boldsymbol{u}}_i}(\boldsymbol{x}_i)) \end{aligned}$$

where $\boldsymbol{x}_i$ and $\boldsymbol{u}_i$ are subsets of $\boldsymbol{x}$ and $\boldsymbol{u}$, $\boldsymbol{f}_i$ only contains the dynamics associated with $\boldsymbol{x}_i$, and the complement state $\bar{\boldsymbol{x}}_i = \boldsymbol{x} \setminus \boldsymbol{x}_i$ is assumed to be constant. The complement input $\bar{\boldsymbol{u}}_i = \boldsymbol{u} \setminus \boldsymbol{u}_i = \boldsymbol{u}_i^{\text{cas}} \cup \boldsymbol{u}_i^{\text{dec}}$ comprises inputs ($\boldsymbol{u}_i^{\text{dec}}$) that are decoupled from, and inputs ($\boldsymbol{u}_i^{\text{cas}}$) that are in cascade with $\boldsymbol{u}_i$. The decoupled inputs are set to zero and sub-policies for the cascaded inputs are used as is while computing $\pi_{\boldsymbol{u}_i}(\boldsymbol{x}_i)$. In general, $\pi_{\bar{\boldsymbol{u}}_i}(\boldsymbol{x}_i) = [0, \ldots, 0, \pi_{\boldsymbol{u}_j}(\boldsymbol{x}_j), 0, \ldots, 0]$ where $\boldsymbol{u}_j \subseteq \boldsymbol{u}_i^{\text{cas}}$ are inputs that appear lower

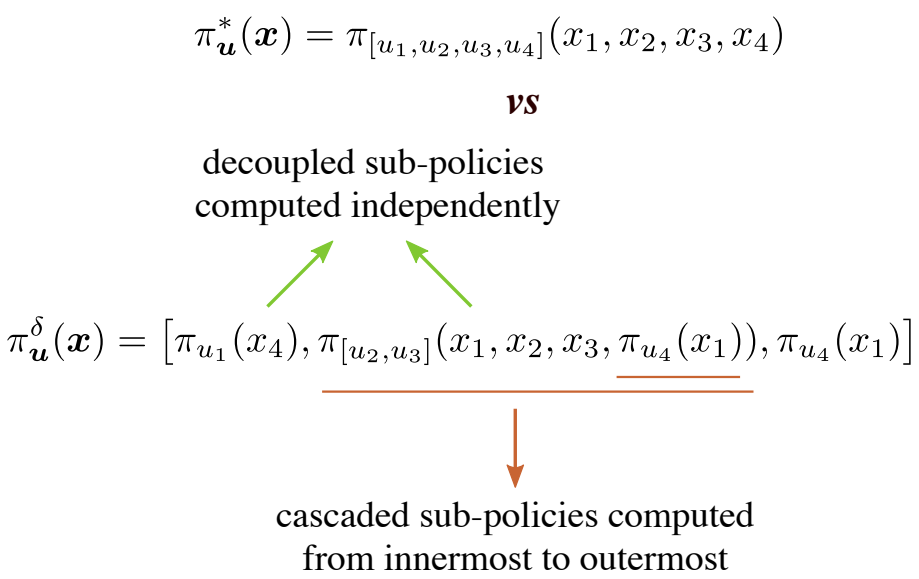


Figure 2.2: Idea of Policy Decomposition shown for a fictive system. Instead of jointly optimizing policies for all inputs over the full state-space (**top**), the policies are approximated with decoupled and cascaded sub-policies for smaller sub-systems that are faster to compute (**bottom**).

in the cascade to $\boldsymbol{u}_i$, and $\boldsymbol{x}_j \subseteq \boldsymbol{x}_i$. The set of inputs $\boldsymbol{u}_i^{\text{dec}}$ are decoupled from $\boldsymbol{u}_i$ in the hierarchy, and their absence in the sub-policy calculation is captured by setting them to 0. Note, (i) $\pi_{\bar{\boldsymbol{u}}_i}(\boldsymbol{x}_i)$ can contain multiple sub-policies, (ii) these sub-policies can themselves be computed in a cascaded or decoupled fashion, and (iii) they have to be known before computing $\pi_{\boldsymbol{u}_i}(\boldsymbol{x}_i)$.

We introduce an intuitive abstraction for hierarchies generated by Policy Decomposition. A hierarchy δ can be represented using an input-tree $T^{\delta} = (\mathcal{V}, \mathcal{E})$ where all nodes except the root node are tuples of disjoint subsets of inputs and state variables $\boldsymbol{v}_i = (\boldsymbol{u}_i, \boldsymbol{x}_{\boldsymbol{u}_i})\ \forall \boldsymbol{v}_i \in \mathcal{V} \setminus \{\boldsymbol{v}_{\text{root}}\}$. Policy computation for inputs that belong to different branches is decoupled. Inputs that lie on the same branch are in a cascade where policies for inputs lower in the branch (leaf node being the lowest) influence the policies for inputs higher-up. A sub-tree rooted at node $\boldsymbol{v}_i$ characterizes a subsystem with control inputs $\boldsymbol{u}_i$ and state $\boldsymbol{x}_i = \boldsymbol{x}_{\boldsymbol{u}_i} \cup (\cup \boldsymbol{x}_{\boldsymbol{u}_j})$, where $\boldsymbol{u}_j \subseteq \boldsymbol{u}_i^{\text{cas}}$ and $\boldsymbol{x}_{\boldsymbol{u}_j}$ are inputs and state variables corresponding to the other nodes in the sub-tree. Note, $\boldsymbol{x}_{\boldsymbol{u}_i}$ can be an empty set if it does not belong to a leaf-node. Figure 2.3 depicts the input-tree for the hierarchy in figure 2.2, and the resulting subsystems. Policies are computed in a child-first order, starting from leaf nodes followed by their parents and so on. An input-tree prescribes a recipe to compute policies for different subspaces of the input-space ($\boldsymbol{u}_i$) as functions of subspaces

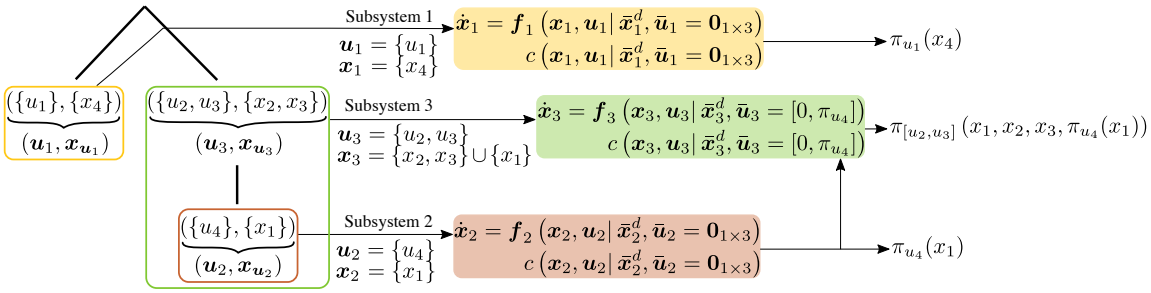


Figure 2.3: Input-tree (**left**) for the decomposition shown in figure 2.2 and the resulting subsystems (**mid**) are depicted. Policies for u_1 and u_4, i.e. inputs at the leaf nodes, are obtained first by independently solving optimal control problems for subsystems 1 and 2 respectively. Policies for u_2 and u_3 are computed jointly by solving the optimal control problem for subsystem 3. Resulting policies for different inputs (**right**) are functions of different state variables.

of the state-space ($\boldsymbol{x}_i$). Here, we restrict $\boldsymbol{u}_i$ and $\boldsymbol{x}_i$ to be subsets of the inputs and state variables respectively. In chapter 4, we present a method to identify linear subspaces of the input and state spaces that offer hierarchies with better control performance.

2.2 Quantifying the Suboptimality of a Hierarchy

To assess the quality of the closed-loop behavior for a policy derived using a hierarchy δ, we introduce the value error: the average difference between value functions V^{δ} and V^{*} of policies obtained with and without the hierarchy,

$$\text{err}^{\delta} = \frac{\int_{\mathcal{S}} \left(V^{\delta}(\boldsymbol{x}) - V^{*}(\boldsymbol{x}) \right) \, d\boldsymbol{x}}{\int_{\mathcal{S}} V^{*}(\boldsymbol{x}) \, d\boldsymbol{x}} \tag{2.5}$$

where $\int_{\mathcal{S}}$ denotes the integral over state space $\mathcal{S}$. The value function for a policy maps states to the cumulative cost accrued by following the policy after initializing the system from the said states, and is the ultimate measure of a policy's closed-loop performance. As defined, err^{δ} directly quantifies the suboptimality of the policies obtained from hierarchy δ. But, computing the value error requires knowing V^{δ} and V^{*} which can only be obtained by solving the original and decomposed optimal control problems. To estimate the value error for a hierarchy a priori, we present two approaches based on Linear Quadratic Regulator (LQR) [72] and Control-limited Differential Dynamic Programming (DDP) [89].

2.2.1 The LQR Estimate

The first approach relies on the system obtained by linearizing the dynamics (Equation (2.2)) about the goal state and input,

$$\dot{\boldsymbol{x}} = \boldsymbol{A}(\boldsymbol{x} - \boldsymbol{x}^d) + \boldsymbol{B}(\boldsymbol{u} - \boldsymbol{u}^d) \tag{2.6}$$

$$\boldsymbol{A} = \left.\frac{\partial \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u})}{\partial \boldsymbol{x}}\right|_{(\boldsymbol{x}^d, \boldsymbol{u}^d)}, \quad \boldsymbol{B} = \left.\frac{\partial \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u})}{\partial \boldsymbol{u}}\right|_{(\boldsymbol{x}^d, \boldsymbol{u}^d)}$$

Because the costs are quadratic (Equation (2.4)), the optimal controller of this linearized system is a Linear Quadratic Regulator (LQR). The optimal policy, $\pi^*_{\boldsymbol{u}_{\text{lqr}}}(\boldsymbol{x}) = -\boldsymbol{K}^*(\boldsymbol{x} - \boldsymbol{x}^d)$, and the corresponding value function, $V^*_{\text{lqr}}(\boldsymbol{x}) = \boldsymbol{x}^T \boldsymbol{P}^* \boldsymbol{x}$, can be computed by solving the Algebraic Riccati Equation [72]. The value error estimate for decomposition δ then becomes

$$\text{err}^{\delta}_{\text{lqr}} = \frac{\int_{\mathcal{S}} \left(V^{\delta}_{\text{lqr}}(\boldsymbol{x}) - V^*_{\text{lqr}}(\boldsymbol{x})\right) d\boldsymbol{x}}{\int_{\mathcal{S}} V^*_{\text{lqr}}(\boldsymbol{x})\, d\boldsymbol{x}} \tag{2.7}$$

where $V^{\delta}_{\text{lqr}}(\boldsymbol{x})$ is the value function for the equivalent decomposition of the linear system. Computing $V^{\delta}_{\text{lqr}}(\boldsymbol{x})$ entails first constructing the hierarchical policy for the linear system, and then solving for its value function. The hierarchical policy is obtained by deriving sub-policies for the linear subsystems generated by hierarchy δ, in a child-first order as per the corresponding input-tree T^{δ}, and then assembling them into a policy for the full linear system. Sub-policies $\pi_{\boldsymbol{u}_i}(\boldsymbol{x}_i)$ are LQR solutions to the corresponding linear subsystems,

$$\dot{\boldsymbol{x}}_i = \boldsymbol{A}_i(\boldsymbol{x}_i - \boldsymbol{x}^d_i) + \boldsymbol{B}_i(\boldsymbol{u}_i - \boldsymbol{u}^d_i) \tag{2.8}$$

$$\boldsymbol{A}_i = \left.\frac{\partial \boldsymbol{f}_i}{\partial \boldsymbol{x}_i}\right|_{(\boldsymbol{x}^d, \boldsymbol{u}^d)} + \left.\frac{\partial \boldsymbol{f}_i}{\partial \bar{\boldsymbol{u}}_i}\right|_{(\boldsymbol{x}^d, \boldsymbol{u}^d)} \left.\frac{\partial \pi_{\bar{\boldsymbol{u}}_i}}{\partial \boldsymbol{x}_i}\right|_{\boldsymbol{x}^d_i},$$

$$\boldsymbol{B}_i = \left.\frac{\partial \boldsymbol{f}_i}{\partial \boldsymbol{u}_i}\right|_{(\boldsymbol{x}^d, \boldsymbol{u}^d)}$$

where $\pi_{\bar{\boldsymbol{u}}_i}(\boldsymbol{x}_i) = [0, \ldots, 0, \boldsymbol{u}^d_j - \boldsymbol{K}_j(\boldsymbol{x}_j - \boldsymbol{x}^d_j), 0, \ldots, 0]$. $\boldsymbol{K}_j$ is the LQR solution for a subsystem characterized by $\boldsymbol{u}_j \subseteq \boldsymbol{u}^{\text{cas}}_i$ and $\boldsymbol{x}_j \subseteq \boldsymbol{x}_i$. Inputs $\boldsymbol{u}^{\text{dec}}_i$ are set to 0.

LQR problem for the subsystem defined by $\boldsymbol{u}_i$ and $\boldsymbol{x}_i$ is then characterized by $\boldsymbol{A}_i$, $\boldsymbol{B}_i$

and cost

$$c_i(\boldsymbol{x}_i, \boldsymbol{u}_i) = \left(\boldsymbol{u}_i - \boldsymbol{u}_i^d\right)^T \boldsymbol{R}_i \left(\boldsymbol{u}_i - \boldsymbol{u}_i^d\right) + (\boldsymbol{x}_i - \boldsymbol{x}_i^d)^T \left(\boldsymbol{Q}_i + \begin{bmatrix} 0 & \cdots & 0 \\ \vdots & \boldsymbol{K}_j & \vdots \\ 0 & \cdots & 0 \end{bmatrix}^T \bar{\boldsymbol{R}}_i \begin{bmatrix} 0 & \cdots & 0 \\ \vdots & \boldsymbol{K}_j & \vdots \\ 0 & \cdots & 0 \end{bmatrix}\right)(\boldsymbol{x}_i - \boldsymbol{x}_i^d)$$

where $\boldsymbol{Q}_i$ is the appropriate sub-matrix of the original cost function matrix $\boldsymbol{Q}$, and $\boldsymbol{R}_i$ and $\bar{\boldsymbol{R}}_i$ are sub-matrices of $\boldsymbol{R}$ corresponding to $\boldsymbol{u}_i$ and $\bar{\boldsymbol{u}}_i$ respectively.

In effect, the hierarchical policy for the linear system is a linear controller,

$$\pi^{\delta}_{\boldsymbol{u}_{\text{lqr}}}(\boldsymbol{x}) = \boldsymbol{u}^d - \boldsymbol{K}^{\delta}(\boldsymbol{x} - \boldsymbol{x}^d)$$

whose gain $\boldsymbol{K}^{\delta}$ is a block matrix composed of subsystem LQR gains $\boldsymbol{K}_i$. For the hierarchy in figure 2.2 the gain is

$$\boldsymbol{K}^{\delta_{\text{example}}} = \begin{bmatrix} & \boldsymbol{0}_{1\times 3} & \overbrace{\boldsymbol{K}_1}^{1\times 1} \\ \overbrace{\boldsymbol{K}_3}^{2\times 3} & & \boldsymbol{0}_{2\times 1} \\ \overbrace{\boldsymbol{K}_2}^{1\times 1} & \boldsymbol{0}_{1\times 3} & \end{bmatrix}$$

With the gain $\boldsymbol{K}^{\delta}$ defined, V^{δ}_{lqr} resolves to

$$V^{\delta}_{\text{lqr}}(\boldsymbol{x}) = (\boldsymbol{x} - \boldsymbol{x}^d)^T \boldsymbol{P}^{\delta}(\boldsymbol{x} - \boldsymbol{x}^d) \tag{2.9}$$

where $\boldsymbol{P}^{\delta}$ is the solution of the Lyapunov equation,

$$\left(\boldsymbol{A} - \boldsymbol{B}\boldsymbol{K}^{\delta} - \frac{\lambda \boldsymbol{I}}{2}\right)^T \boldsymbol{P}^{\delta} + \boldsymbol{P}^{\delta}\left(\boldsymbol{A} - \boldsymbol{B}\boldsymbol{K}^{\delta} - \frac{\lambda \boldsymbol{I}}{2}\right) + \boldsymbol{Q} + \boldsymbol{K}^{\delta T}\boldsymbol{R}\boldsymbol{K}^{\delta} = \boldsymbol{0} \tag{2.10}$$

2.2.2 The Unscented Estimate

The second approach uses trajectory optimization methods to estimate the value error (Equation (2.5))). These methods [62, 89, 91, 92] compute state and input trajectories for a dynamical system that optimize the objective of the optimal control problem. An initial guess $\boldsymbol{X}^0(t)$, $\boldsymbol{U}^0(t)$ is iteratively improved to produce a locally optimal solution $\boldsymbol{X}(t)$, $\boldsymbol{U}(t)$ whose value function,

$$V^*_{\text{unscented}}(\boldsymbol{x}) = \int_0^{t_{\max}} e^{-\lambda t} c\left(\boldsymbol{X}(t), \boldsymbol{U}(t)\right) dt \tag{2.11}$$

approximates $V^*(\boldsymbol{x})$ for the system under consideration at the point $\boldsymbol{x} = \boldsymbol{X}(0)$ in the state space. We use this approximation to estimate the value error. Specifically, we introduce the suboptimality estimate

$$\text{err}^{\delta}_{\text{unscented}} \frac{\sum_{s=1}^{k} \left(V^{\delta}_{\text{unscented}}(\boldsymbol{x}^s) - V^*_{\text{unscented}}(\boldsymbol{x}^s)\right)}{\sum_{s=1}^{k} V^*_{\text{unscented}}(\boldsymbol{x}^s)} \tag{2.12}$$

which averages the value errors obtained from trajectories of the original and hierarchical optimal control problems for k initial points centered on the goal state $\boldsymbol{x}^d$ (Figure 2.4(a)). We use Control-Limited DDP (DDP) [89] to compute $\text{err}^{\delta}_{\text{unscented}}$ but a different trajectory optimization algorithm can just as easily be applied. DDP uses quadratic approximations of the system dynamics, but to curb computational costs, we consider only linear ones.

DDP starts from an initial input trajectory $\boldsymbol{U}^0(t)$, rolls it out with the system dynamics to get $\boldsymbol{X}^0(t)$, then iteratively updates the input trajectory $\boldsymbol{U}^+(t) = \boldsymbol{U}^-(t) - \boldsymbol{K}(t)\left(\boldsymbol{x} - \boldsymbol{X}^-(t)\right)$ and subsequently the state trajectory $\boldsymbol{X}^+(t)$. Additionally, DDP produces a local linear approximation to the optimal control policy in the vicinity of the converged trajectory. While $V^*_{\text{unscented}}(\boldsymbol{x}^s)$ can be obtained right away, obtaining $V^{\delta}_{\text{unscented}}(\boldsymbol{x}^s)$ requires further explanation.

A policy decomposition δ with r subsystems creates r optimal control problems, whose individual DDP solutions need to be combined for computing the approximate value function $V^{\delta}_{\text{unscented}}(\boldsymbol{x}^s)$. We achieve this with the following procedure. First, starting from the initial sub-states $\{\boldsymbol{x}^s_i | s \in 1, \cdots, k\}$ we compute for each subsystem i locally optimal solutions characterized by $\boldsymbol{X}^s_i(t)$, $\tilde{\boldsymbol{X}}^s_i(t)$, $\boldsymbol{U}^s_i(t)$ and $\boldsymbol{K}^s_i(t)$. $\boldsymbol{X}^s_i(t)$ are the final DDP state trajectories for the subsystem i originating from state $\boldsymbol{x}^s_i$. $\tilde{\boldsymbol{X}}^s_i(t)$, $\boldsymbol{U}^s_i(t)$ and $\boldsymbol{K}^s_i(t)$ are

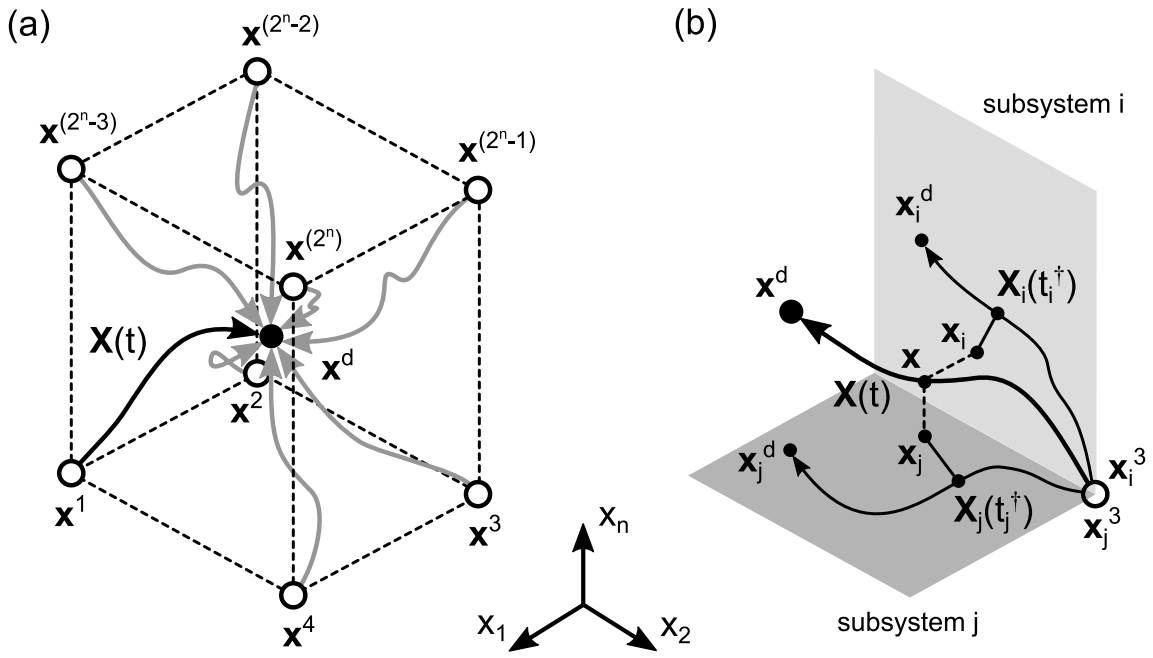


Figure 2.4: Trajectory optimization based approximation of value function. (**a**) Trajectories $\boldsymbol{X}(t)$ from $k = 2^n$ initial points $\boldsymbol{x}^s$ located at edges of hyper-cube that defines boundary of explored state-space. (**b**) Nearest neighbors $\boldsymbol{X}_i(t_i^\dagger)$ on subsystem solutions $\boldsymbol{X}_i(t)$ for current state $\boldsymbol{x}$ along solution $\boldsymbol{X}(t)$.

the control reference trajectory, control inputs and local linear gains respectively that produce the subsystem trajectory $\boldsymbol{X}_i^s(t)$. Next, we define the subsystem control policy as the nearest neighbor policy [6],

$$\pi_{\boldsymbol{u}_i\,\mathrm{DDP}}(\boldsymbol{x}_i) = \min\left(\max\left(\boldsymbol{U}_i^{s^\dagger}(t^\dagger) - \boldsymbol{K}_i^{s^\dagger}(t^\dagger)\left(\boldsymbol{x}_i - \tilde{\boldsymbol{X}}_i^{s^\dagger}(t^\dagger)\right), \boldsymbol{u}_{i\,\min}\right), \boldsymbol{u}_{i\,\max}\right) \tag{2.13}$$

where $\boldsymbol{u}_{i\,\min}$ and $\boldsymbol{u}_{i\,\max}$ are the bounds on the control inputs $\boldsymbol{u}_i$. $s^\dagger$ and $t^\dagger$ respectively mark the trajectory ID and time at which $\boldsymbol{X}_i^s(t)$ is closest to the subsystem state $\boldsymbol{x}_i$ (Figure 2.4(b)),

$$s^\dagger, t^\dagger = \arg\min_{s,t} \left\| \boldsymbol{X}_i^s(t) - \boldsymbol{x}_i \right\|_2 . \tag{2.14}$$

Lastly, we run the policy $\pi^\delta_{\boldsymbol{u}_{\mathrm{DDP}}}(\boldsymbol{x}) = \left(\pi_{\boldsymbol{u}_1\,\mathrm{DDP}}(\boldsymbol{x}_1), \ldots, \pi_{\boldsymbol{u}_r\,\mathrm{DDP}}(\boldsymbol{x}_r)\right)$ on the complete system (Equation (2.2))) initialized at $\boldsymbol{x}^s$ and compute $V^\delta_{\mathrm{ddp}}(\boldsymbol{x}^s)$ from the resulting trajectory $\boldsymbol{X}(t)$,

$$V^\delta_{\mathrm{unscented}}(\boldsymbol{x}^s) = \int_0^{t_{\max}} e^{-\lambda t} c\left(\boldsymbol{X}(t), \pi^\delta_{\boldsymbol{u}_{\mathrm{DDP}}}(\boldsymbol{X}(t))\right) dt. \tag{2.15}$$

Note that $\boldsymbol{X}(t)$ will differ from the collected trajectories of the individual DDP solu-

tions, $(\boldsymbol{X}_1(t), \ldots, \boldsymbol{X}_r(t))$, as the latter ignore at least some of the input couplings that influence the behavior of the complete system. When optimizing for subsystem trajectories the linearized dynamics are derived similar to the procedure in section 2.2.1, except the linearizations are computed at each sub-state in the reference trajectory. Furthermore, LQR gains corresponding to subsystems lower in cascade that feature in the dynamics linearization, are replaced with the appropriate DDP feedback gains based on the nearest neighbor policy (Equation (2.13)). To generate the initial input sequence for a subsystem i, we use the LQR controller gain $\boldsymbol{K}_i$ (described in section 2.2.1), along with nearest neighbor policies for subsystems lower in the cascade, to roll-out trajectories and generate $\boldsymbol{U}_i^0(t)$.

2.3 Results

We evaluate several hierarchies for the swing-up control of the cart-pole (Figure 2.1), 2 and 3 link planar manipulators (Figure 2.5(a) and (b)), and the balance control of a simplified biped (Figure 2.5(c)). We compute the true value errors by solving for the optimal and

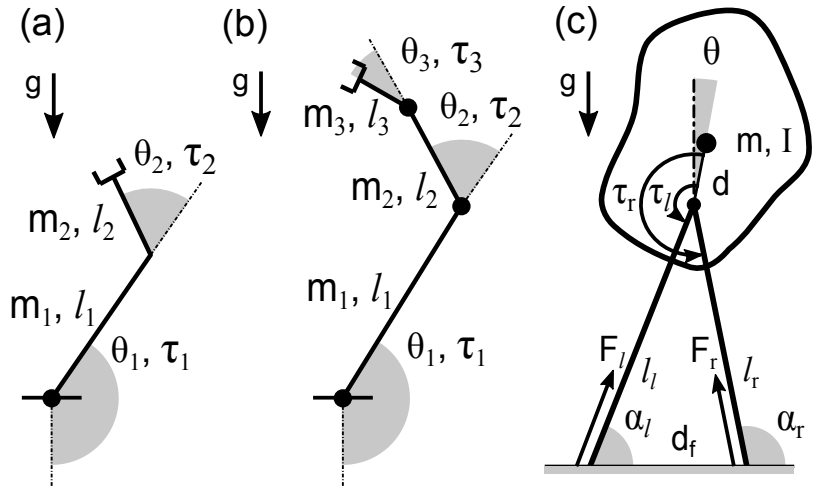


Figure 2.5: 2 and 3 link planar manipulators are shown in (**a**) and (**b**) respectively, and a simplified model of a biped in stance is depicted in (**c**). Policies to swing-up the manipulators into an upright position, and to balance the biped midway between the two foot-holds are derived using several hierarchies. Appendix C describes the system dynamics and the cost functions used.

hierarchical policies using Policy Iteration [10]. The policies and value functions are represented as look-up tables which store values corresponding to a uniform grid over the state-space. When computing hierarchical policies the individual sub-policies are look-up tables over appropriate subspaces of the state-space. The choice of algorithm and policy

representation is motivated by guaranteed convergence to the optimal solution (within discretization resolution). Details pertaining to the system's dynamics, cost functions, and other hyper-parameters can be found in appendix C.

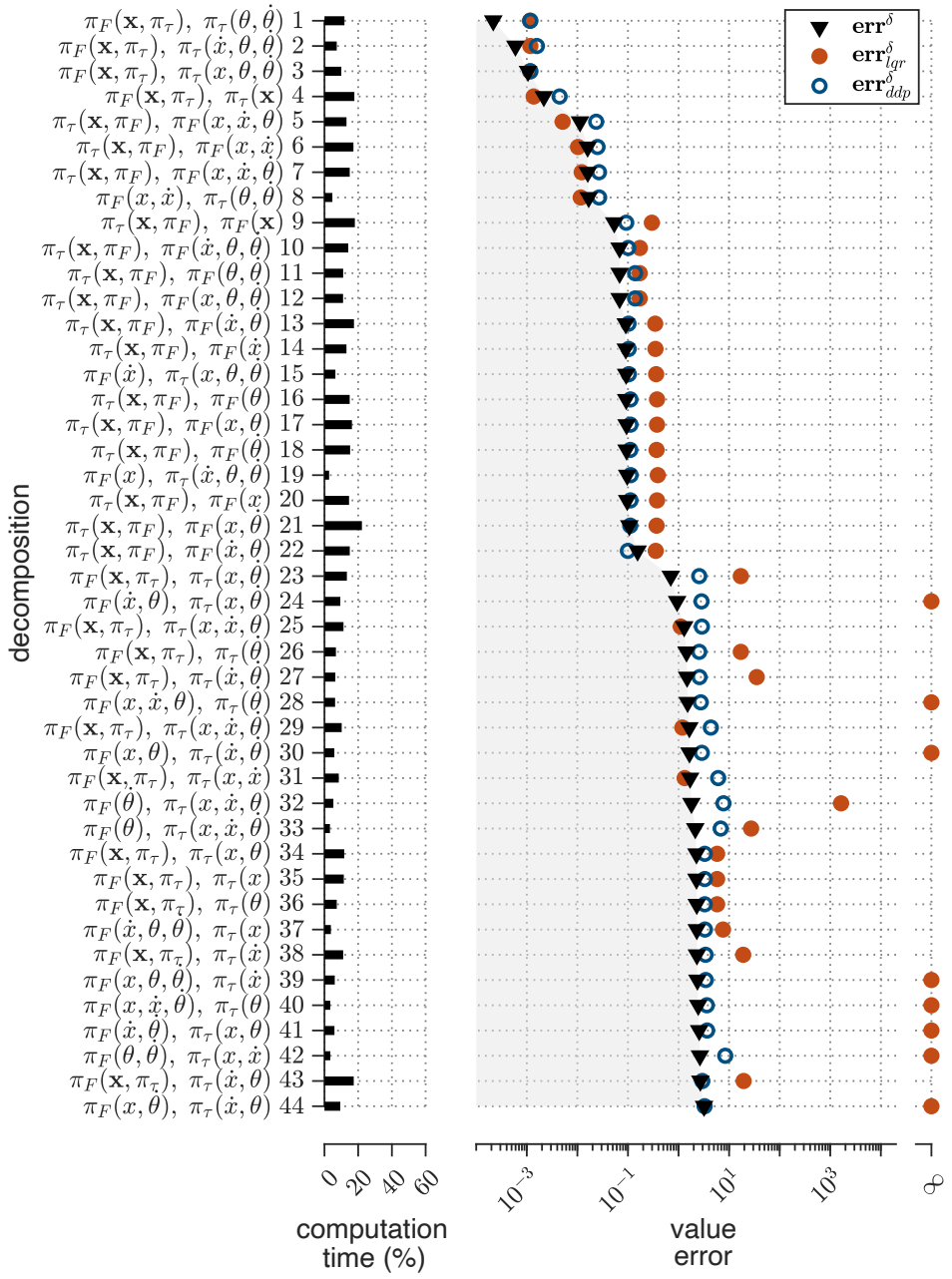


Figure 2.6: Cart-pole hierarchies listed in order of true value error. The computation times (relative to optimal control) and value errors (triangles) are shown together with their LQR and DDP estimates (filled and open circles). LQR estimates are set to infinity for uncontrollable systems after linearization.

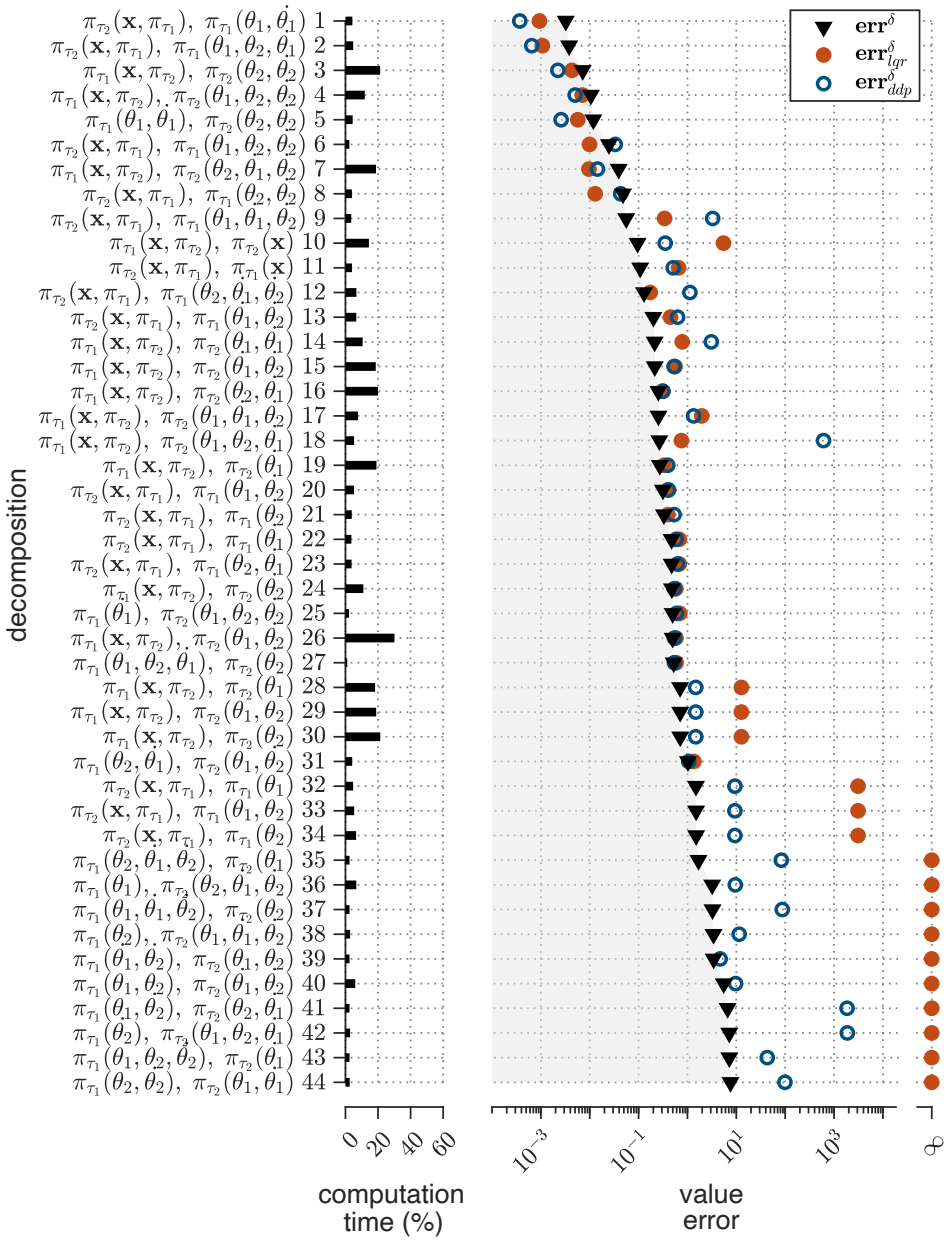


Figure 2.7: Hierarchies for 2 link planar manipulator listed in order of true value error. The computation times (relative to optimal control) and value errors (triangles) are shown together with their LQR and DDP estimates (filled and open circles). LQR estimates are set to infinity for uncontrollable systems after linearization.

The cart-pole and the 2 link manipulator, with two control inputs and four state variables, have 44 possible hierarchies. Comparison of the true and estimated value errors as well as the required time to solve for hierarchical policies (as percentage of the time required to compute the optimal policies) is reported in figure 2.6 and figure 2.7. The 3

link manipulator and the biped have 10,512 and 396,716 possible hierarchies, respectively and computing DDP estimates for every hierarchy is not feasible. We thus identify the Pareto optimal set of hierarchies (figure 2.8 and figure 2.9) based on the LQR estimates and the expected reduction in computation time for policy optimization (see appendix A.2.1 for details), and evaluate their true value errors and DDP estimates.

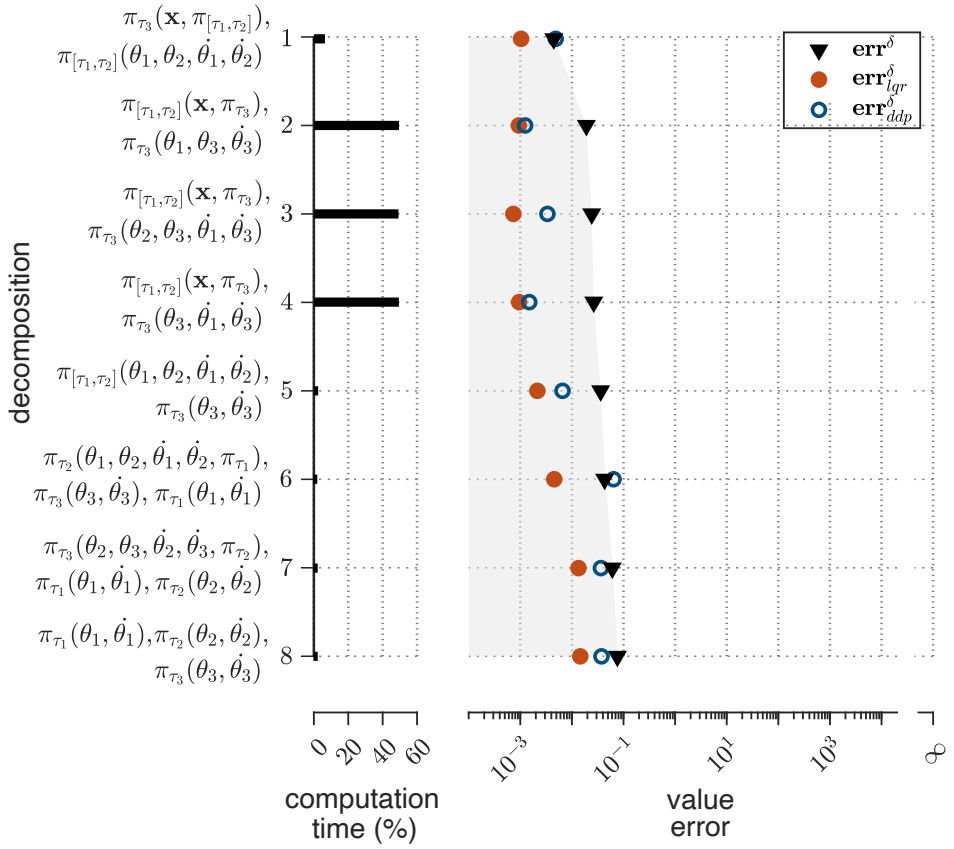


Figure 2.8: Hierarchies for 3 link planar manipulator listed in order of true value error. The computation times (relative to optimal control) and value errors (triangles) are shown together with their LQR and DDP estimates (filled and open circles). LQR estimates are set to infinity for uncontrollable systems after linearization.

In general, the two estimates trend similarly and in line with the true value error. For the cart-pole, both estimates rank the cascaded hierarchy described before (Figure 2.1(b)) as the one with least value error, and it reduces policy computation times by a factor of 66. Importantly, the LQR and DDP estimates identify similar best policies for all problems (top four hierarchies in figures 2.6 to 2.9), and the corresponding policies have a true value error of less than 1% and offer about an order of magnitude improvement in computation time. For the biped problem, these hierarchies include a popular heuristic strategy for

controller design [37, 60] (first design an independent policy for leg forces to regulate the center of mass behavior and then compute a policy for the hip torques to balance the torso, hierarchy #2 in figure 2.9). The Pareto optimal sets further include hierarchies with much more dramatic improvements of 500 to 1000 times less computation time if value errors of 10% are acceptable (hierarchies #5-8 in figure 2.8 and #5 in figure 2.9). While the DDP estimate provides a closer estimate of the true value error in almost all of the problems, it performs significantly worse than the LQR estimate in the biped problem. The poor estimates are a consequence of diverging trajectories from DDP resulting in highly inaccurate value function estimates for the hierarchical policy. More recent trajectory optimization methods with better convergence properties such as [59] could help to correct this issue.

Policy Decomposition, is an approximate method for solving policy optimization problems that reduces search for one high-dimensional control policy to a search for a collection of lower-dimensional sub-policies that are faster to compute yet preserve closed-loop performance when combined. We also introduced the value error, a measure of a hierarchy's suboptimality, and derived two estimates of it using LQR and Control-Limited DDP. These estimates allow us to assess a decomposition's closed-loop performance without computing the policy. A measure to predict the closed-loop performance of control hierarchies is a useful tool. Proposed measures to help select simplified controller structures typically make use of open-loop transfer functions [5] and Gramians [24] or are based on an assessment of controller robustness [101]. These measures are agnostic to the objective of the optimal control problem and need not correlate well with closed-loop behavior. For linear systems, measures that account for the control objective have been proposed, including sum of output covariances of the resulting LQG control [39] and value function bounds for LQR controllers obtained through nested$-\epsilon$ decompositions [83]. In contrast, the value error is a general measure for a hierarchy's suboptimality in nonlinear systems.

Although, we primarily discussed regulation control problems where the objective is to drive the system to a fixed desired state, Policy Decomposition readily extends to trajectory tracking problems where the objective is to track a time-indexed sequence of desired states. In this setting, the control policies and the corresponding value functions are now also a function of a time. The hierarchical policies are a collection of policies for subspaces of the inputs as functions of subspaces of the state-space and *time*. We discuss this in more

detail in chapter 5. Additionally, the combinatorics of Policy Decomposition challenges its practical utility. The number of possible hierarchies grows prohibitively with system complexity and even for moderately complex systems discussed in this chapter, evaluating the suboptimality estimates for all possible hierarchies quickly becomes infeasible. We tackle this problem next.

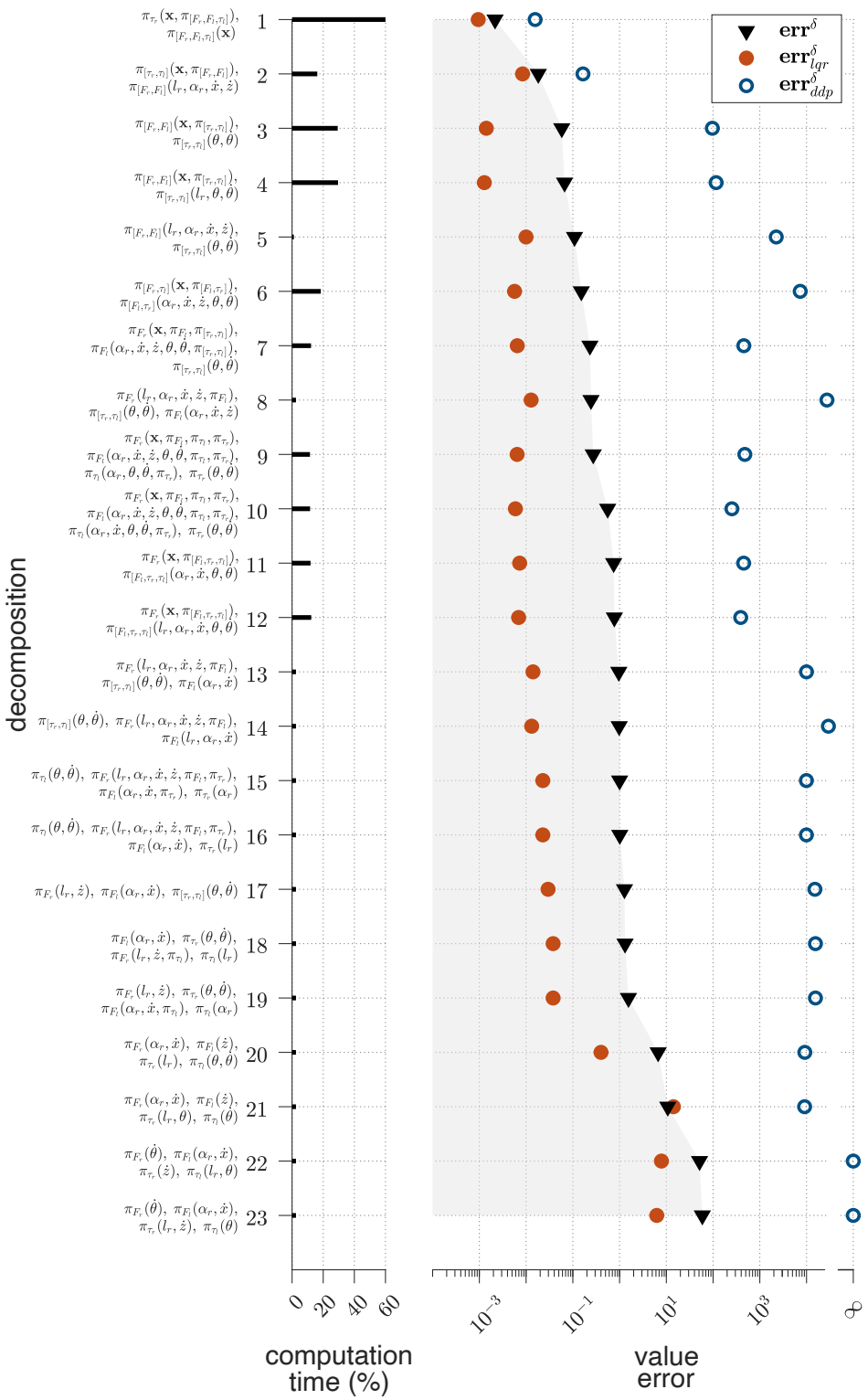


Figure 2.9: Biped hierarchies listed in order of true value error. The computation times (relative to optimal control) and value errors (triangles) are shown together with their LQR and DDP estimates (filled and open circles). LQR estimates are set to infinity for uncontrollable systems after linearization.

Chapter 3

Search for Hierarchies within the Policy Decomposition Framework

Although Policy Decomposition provides a systematic approach to construct hierarchies that lessen the curse of dimensionality in solving a policy optimization problem, the number of possible hierarchies grows combinatorially with system complexity. Exhaustively

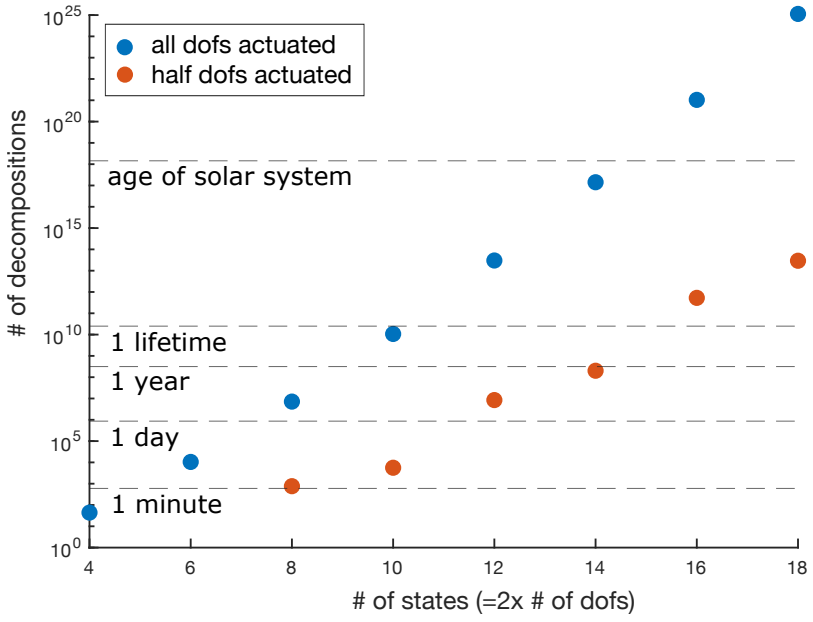


Figure 3.1: Number of possible decompositions as the number of state variables and control inputs increase. Appendix A.1 details the procedure to count the number of decompositions. Assuming it takes **0.1sec/value error estimate** computation, the number of decompositions that can be evaluated in some fixed times are marked with dotted lines.

evaluating the suboptimality estimates for all possible hierarchies becomes impractical (Figure 3.1). This necessitates the development of search algorithms that efficiently search over the combinatorial space of possibilities. Moreover, the search for promising hierarchies is a bi-objective optimization problem, in which the suboptimality of the resulting policies and the reduction in the required computation to obtain them have to be suitably traded off. We investigate two strategies, first to perform a single objective search by combining the two criteria into a single fitness function (Section 3.1), and second to find a Pareto-optimal set of candidates based on the two criteria (Section 3.2).

Among the several existing methods [15, 34, 95] to address combinatorial search problems, population based evolutionary algorithms [95] have been remarkably effective over others in several domains [21, 84, 85]. Genetic Algorithm (GA) is a popular method in this category [41, 64]. GA evolves a randomly generated initial population of candidates through mutation, selection and crossover to find promising candidates based on the fitness function. GA is a metaheuristic [13] which makes minimal assumptions on the underlying fitness landscape and thus can be applied to problems in varied domains. But, designing problem specific sampling strategies and operators for population evolution are critical to its success. In section 3.1.1, we describe our GA based search for hierarchies.

A top-down alternative to GA is to start from the original system and decompose it step-by-step. The search for promising hierarchies can then be posed as a sequential decision making problem for which a number of algorithms exist, especially in the context of computer games [18]. Among these, Monte-Carlo Tree Search (MCTS) [20] methods have been shown to be highly effective for games that have a large number of possible moves in each step, a similar challenge that we encounter when generating hierarchies. Similar to GA, we tailor MCTS to find promising hierarchies (Section 3.1.2).

3.1 Single Objective Search

We introduce the following fitness function to assess the quality of a hierarchy.

$$\begin{aligned} F(\delta) &= F_{\text{err}}(\delta) \times F_{\text{comp}}(\delta) \\ \text{where } F_{\text{err}}(\delta) &= (1 - \exp(-\text{err}_{\text{lqr}}^{\delta})), \\ F_{\text{comp}}(\delta) &= \frac{\#\text{ estimated FLOP for } \pi^{\delta}}{\#\text{ estimated FLOP for } \pi^{*}} \end{aligned} \tag{3.1}$$

Algorithm 1 Genetic Algorithm ($\text{System}_{\text{info}}$, $\text{time}_{\max}$, $\text{iter}_{\max}$)

$\text{time}_{\text{current}} \leftarrow 0$
$\text{iter} \leftarrow 0$
$\text{HashTable} \leftarrow \{ : \}$ ▷ initialize an associative array
while $\text{time}_{\text{current}} < \text{time}_{\max}$ **do**
  if $\text{iter} == 0$ **then**
    $\text{P} \leftarrow \text{UniformSample}(\text{System}_{\text{info}},\ \text{N}_\text{P})$ ▷ initialize population
  end if
  $\text{F}_\text{P} \leftarrow \left\{\text{Fitness}\left(\text{System}_{\text{info}},\ \delta,\ \text{HashTable}\right) \mid \delta \in \text{P}\right\}$ ▷ evaluate fitness
  $\text{P}_{\text{elite}} \leftarrow \text{Selection}(\text{P},\ \text{F}_\text{P},\ r_{\text{elite}})$
  $\text{P}_{\text{mutate}} \leftarrow \text{Mutate}(\text{P},\ \text{F}_\text{P},\ 1 - (r_{\text{elite}}))$
  $\text{P} \leftarrow \text{P}_{\text{elite}} \cup \text{P}_{\text{mutate}}$ ▷ union with repitition
  $\text{iter} \leftarrow \text{Modulo}(\text{iter} + 1, \text{iter}_{\max})$
  $\text{time}_{\text{current}} \leftarrow \text{ElapsedTime}()$
end while

where $F_{\text{err}}(\delta)$ and $F_{\text{comp}}(\delta)$ quantify the suboptimality of a hierarchy and potential reduction in policy computation time respectively. $F_{\text{err}}(\delta)$, is the value error estimate (Section 2.2.1, equation (2.7)) scaled to the range $[0, 1]$. We use the LQR based estimate for assessing the suboptimality because it can be computed in minimal time. $F_{\text{comp}}(\delta)$ is the ratio of estimates of floating point operations required to compute policies with and without decomposition δ. $F_{\text{comp}}(\delta)$ depends on the policy representation used as well as the algorithm of choice for policy optimization. We use look-up tables corresponding to a uniform grid over the state-space to represent the control policies and Policy Iteration [10] to compute them. And in appendix A.2.1, we describe how $F_{\text{comp}}(\delta)$ is computed for this choice of algorithm and policy representation.

Note, $F_{\text{err}}(\delta), F_{\text{comp}}(\delta) \in [0, 1]$ and thus $F(\delta) \in [0, 1]$ where lower values indicate better hierarchies. Lower $F_{\text{err}}(\delta)$ indicates lower suboptimality i.e. better control performance and lower $F_{\text{comp}}(\delta)$ indicates lower required compute to obtain the policies.

3.1.1 Genetic Algorithm

GA iteratively evolves a randomly generated initial population of candidates, through selection, crossovers and mutations, to find promising ones (Algorithm 1). In this case, the candidates are input-trees. The objective of the search is to minimize the fitness (Equation (3.1)). We generate the initial population by uniformly sampling from the set

of all possible input-trees for a system (Section 3.1.1.1). In each iteration, r_{elite} fraction of the next generation of the population is composed of candidates with the lowest fitness (the elite) from the current population. The remaining candidates for the next generation are obtained by mutating promising parents from the current population. In section 3.1.1.2, we introduce the mutation operators for input-trees. In standard GA, a fraction of the next generation is obtained by crossing over pairs of candidates from the current population. We also developed crossover operators for input-trees and investigated incorporating them in the search. But, we found that they slowed down the search with no significant improvement in the best fitness hierarchy discovered and therefore exclude them (Appendix A.3). Moreover, we add a memory component to GA using a hash table, that alleviates the need to recompute fitness values for previously seen candidates (Section 3.1.1.3).

3.1.1.1 Uniform Sampling of Input-Trees

The strategy to sample input-trees is tightly linked to the process of constructing them, which entails partitioning the set of control inputs into groups, arranging the groups in a tree, and then assigning the state variables to nodes of the tree. We use the idea of probability-proportional-to-size sampling [40] in each step of the construction process to uniformly draw input-trees. Specifically, the probabilities of choosing a partition and tree structure are scaled proportional to the number of valid input-trees resulting from said partition and structure. Finally, an assignment of state variables to different nodes of the input-tree consistent with the tree structure is uniformly sampled.

For a system with m inputs and n state variables we first sample $r \in \{2, \cdots, m\}$, i.e. the number of input groups in an input-tree, with probability proportional to the number of possible input-trees with r input groups (each entry in the outermost summation in Eq. A.1). Next, we generate all partitions of the inputs into r groups and pick one uniformly at random. Subsequently, we sample the number of leaf-nodes $k \in \{1, \cdots, r\}$, with probabilities proportional to the number of input-trees with r input groups and k leaf-nodes (each entry in the inner summation in Eq. A.1). To sample the tree structure we use Prufer Codes [14]. Any tree with N labelled nodes can be uniquely represented using a $(N-2)$ character long sequence of its node labels, such sequences are called Prufer Codes. Furthermore, labels corresponding to leaf-nodes are always absent from the

encoding. We uniformly sample a sequence of numbers of length $r-1$ with exactly $r-k$ distinct entries from $\{1,\cdots,r+1\}$ to generate an input-tree with k leaf-nodes. Finally, for assigning state variables to different nodes of the input-tree we uniformly sample a label for every variable from $\{1,\cdots,r\}$. Variables with the label i are assigned to node i. We re-sample labels if a state assignment is invalid i.e. no variables are assigned to leaf-nodes.

In figure 3.2, the KL-divergence between the uniform distribution and the resulting sample distribution from our scheme for systems of varying complexity is noted. The KL-divergence tends to 0 with increasing samples, establishing the uniformity of our sampling scheme.

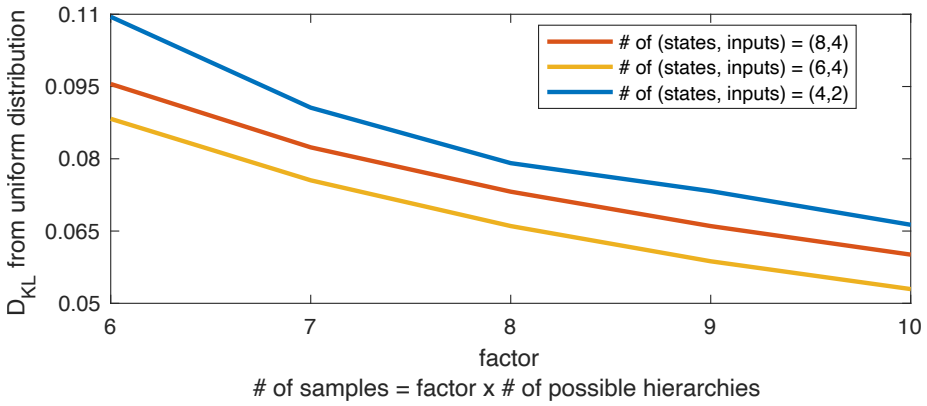


Figure 3.2: D_{KL} between the uniform distribution and sampling distribution of hierarchies for systems of varying complexity

3.1.1.2 Mutations

We introduce the following operations for mutating an input-tree into another valid input-tree:

(i) Swap state variables between nodes

(ii) Move a single state variable from one node to another

(iii) Move a sub-tree

(iv) Couple two nodes

(v) Decouple a node into two nodes

Fig. 3.3 depicts the above operations applied to an input-tree. In every GA iteration, a candidate selected for mutation is modified using only one operator. Operators (i), (ii) and (iii) each have a 25% chance of being applied to a candidate. If neither of these three

operators are applied then two distinct inputs are randomly selected and depending on whether they are decoupled or coupled operators (iv) or (v) are applied respectively.

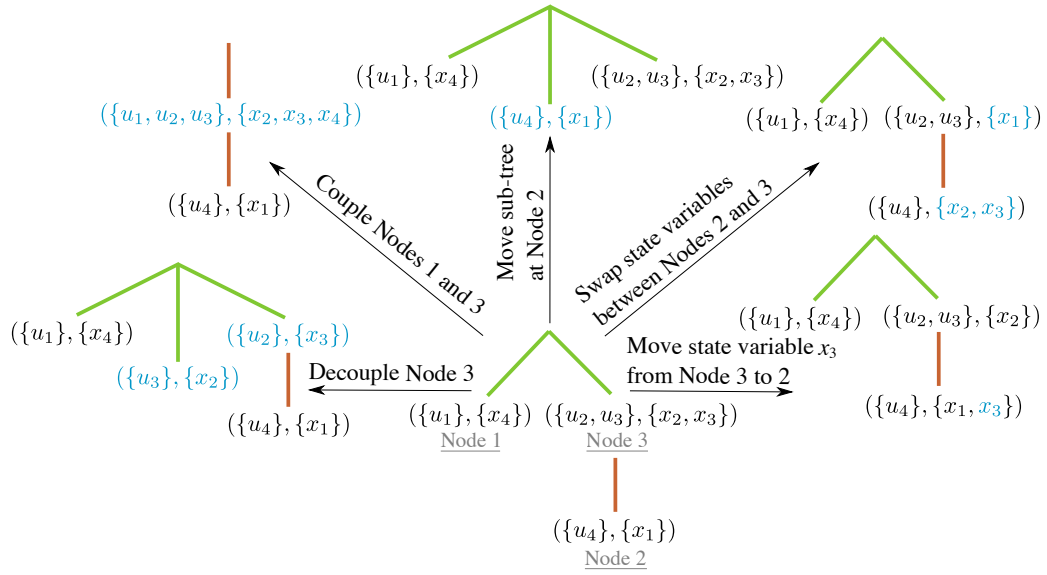


Figure 3.3: Mutation operators for input-trees described in Sec. 3.1.1.2.

3.1.1.3 Hashing Input-Trees

To avoid re-evaluating the fitness for previously seen candidates, we maintain a hash table [25]. This hash table maps a unique identifier, or a *key*, for an input-tree to its computed fitness values. For a system with m inputs, we use a binary connectivity matrix ($C \equiv m \times m$) to encode the graph. If input u_j belongs to node $\boldsymbol{v}_j$ in the input-tree, then the j^{th} row in C has entries 1 for all *other* inputs that belong to $\boldsymbol{v}_j$ as well as for inputs that belong to the parent node of $\boldsymbol{v}_j$. Additionally, We define the binary state-dependence matrix ($S \equiv m \times n$) to encode the influence of the n state variables on the m inputs. The j^{th} row in S corresponds to input u_j and has entries 1 for all state variables that belong to the node $\boldsymbol{v}_j$. For the input-tree in Fig. 3.3 the connectivity and state-dependence matrices are

$$C = \begin{array}{c} \\ u_1 \\ u_2 \\ u_3 \\ u_4 \end{array} \begin{array}{c} \begin{array}{cccc} u_1 & u_2 & u_3 & u_4 \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \end{bmatrix} \end{array}, \quad S = \begin{array}{c} \\ u_1 \\ u_2 \\ u_3 \\ u_4 \end{array} \begin{array}{c} \begin{array}{cccc} x_1 & x_2 & x_3 & x_4 \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

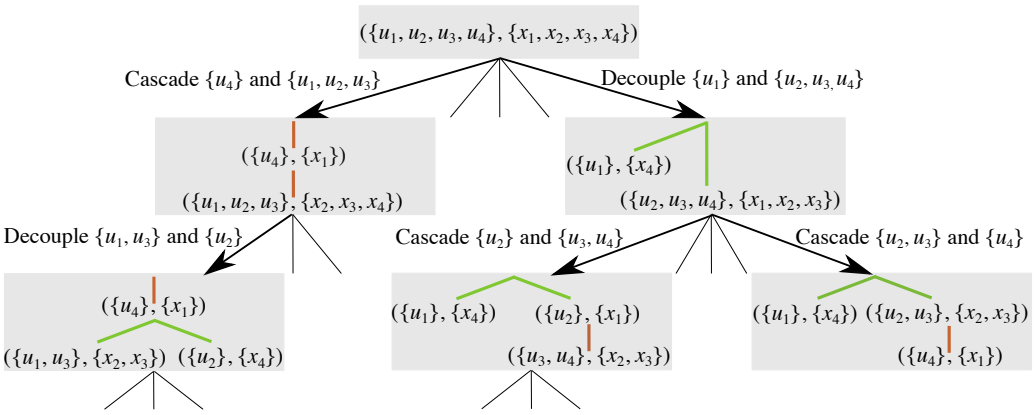


Figure 3.4: Example of a search tree generated from MCTS rollouts for a fictive system with four inputs and four state variables.

The matrices C and S uniquely encode an input-tree.

3.1.2 Monte-Carlo Tree Search

MCTS takes a top-down approach, and creates a search tree of input-trees with the original system at its root. It builds the search tree through continuous rollouts. Each rollout is a sequence of node expansions starting from the root till a terminal node is reached. After every rollout, a backup operation is performed to assign/update a value to/for nodes in the explored branch of the search tree. These values guide subsequent rollouts toward promising parts of the search space. The pseudo-code can be found in algorithm 2. In a rollout, expanding a node for hierarchy δ entails evaluating the fitness $F(\delta)$ (Equation (3.1)), enumerating possible children to the node, and then selecting which child to expand next. The children are input-trees created by splitting the subsystem corresponding to a leaf in T^{δ} into two cascaded or decoupled subsystems (Figure 3.4). If no valid children are possible we terminate the rollout. To decide which child to expand next, we use the UCT strategy [53],

$$T^{\delta_{\text{expand}}} = \underset{T^{\delta'} \in \text{children}(T^{\delta})}{\text{argmin}} \; Q(T^{\delta'}) - \sqrt{\frac{2 \ln(N_{T^{\delta}} + 1)}{N_{T^{\delta'}}}} \tag{3.2}$$

where $Q(T^{\delta'})$ is the minimum fitness in the subtree rooted at $T^{\delta'}$ and $N_{T^{\delta'}}$ denotes the number of times $T^{\delta'}$ was visited. We break ties randomly. $Q(T^{\delta'})$ is initialized to $F(\delta')$ and at the end of each rollout, the values for the nodes visited during the rollout are updated

Algorithm 2 Monte Carlo Tree Search ($\text{System}_{\text{info}}$, $\text{time}_{\max}$, $\text{iter}_{\max}$)

```
time_current ← 0
HashTable ← { : }                                   ▷ initialize an associative array
SearchTree.initialize(T^{δ_0})
while time_current < time_max do
    node_current ← SearchTree.get_root()
    while not is_terminal(node_current) do                                  ▷ Rollout
        if node_current.N == 0 then
            node_current.Q ← Fitness (System_info, node_current.δ, HashTable)
            node_current.enumerate_children()
        end if
        node_current.N ← (node_current.N + 1)
        node_current ← node_current.select_child_with_UCT()
    end while
    while not is_root(node_current) do                                      ▷ Backup
        node_parent ← node_current.get_parent()
        node_parent.Q ← min(node_parent.Q, node_current.Q)
        node_current ← node_parent
    end while
    time_current ← ElapsedTime()
end while
```

$$Q(T^{\delta}) = \min_{T^{\delta'} \in \left(\{T^{\delta}\} \cup \text{children}(T^{\delta})\right)} Q(T^{\delta'})$$

starting from the terminal node and going towards the root of the search tree. Moreover, to prevent redundant rollouts, we remove a node from consideration in the UCT strategy (Equation (3.2)) if all possible nodes reachable through it have been expanded. The Q-value of the root is the fitness value of the best hierarchy identified in the search. Similar to GA, we maintain an associative array to avoid re-evaluating the fitness for previously visited input-trees (Section 3.1.1.3).

3.1.3 Results

We evaluate the efficacy of our search methods in finding promising hierarchies in a fixed time budget (Section 3.1.3.1) and performance of the resulting policies (Section 3.1.3.2) for three distinct robotic control problems: balancing a simplified biped (Figure 3.5(a)),

swinging up a planar manipulator (Figure 3.5(b)), and hover control of a quadcopter (Figure 3.5(c)).

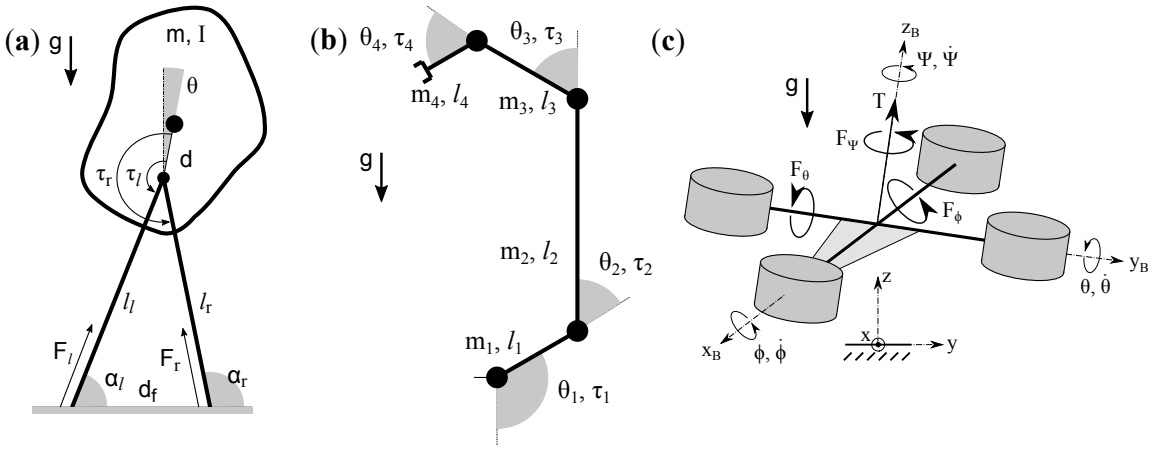


Figure 3.5: (**a**) Balance control of a simplified planar biped. (**b**) Swing-up control of a 4-link planar manipulator. (**c**) Hover control of a quadcopter.

3.1.3.1 Search Performance

We compare GA and MCTS based search against random sampling (Section 3.1.1.1). Each algorithm is run 5 times with a fixed time budget for every system; results are summarized in Table 3.1. Overall, GA consistently outperforms MCTS and random sampling in terms of the lowest fitness hierarchy found as well as the number of unique hierarchies discovered. For the biped and manipulator, GA identified hierarchies with markedly lower value error estimates than the other methods.

The GA search performed well in absolute terms too. For the biped and manipulator problems, brute force evaluation of all possible hierarchies (396,716 and 7,147,628) took 2,433 and 81,318 seconds respectively. (With over $120{\times}10^6$ possibilities, this was not possible for the quadcopter problem.) In contrast, GA evaluated only a fraction of the possible hierarchies (in a fraction of the time, see Table 3.1), but consistently found the one with the *absolute* lowest fitness.

3.1.3.2 Evaluation of Identified Control Hierarchies

To evaluate the identified control hierarchies, we first use Policy Iteration [10] with a look-up table representation to obtain the hierarchical controllers and note the required

Table 3.1: Summary of search results. The number of unique hierarchies discovered, and the fitness ($F(\delta)$) and LQR suboptimality estimates ($\text{err}^{\delta}_{\text{lqr}}$) for the lowest fitness hierarchy found are reported, averaged across 5 runs. Each algorithm is allotted a fixed time budget in each run; 150, 600 and 1200 seconds for the biped, manipulator and quadcopter, respectively.

		GA	MCTS	RANDOM
Biped	$\boldsymbol{F(\delta)}$ $(\times 10^{-8})$	$\mathbf{4.89 \pm 0}$	6.36 ± 0.12	13.8 ± 12.5
	$\mathbf{err}^{\delta}_{\text{lqr}}$ $(\times 10^{-2})$	$\mathbf{2.95 \pm 0}$	3.78 ± 0	8.8 ± 8.4
	$\boldsymbol{\#\delta}$ **found**	18604 ± 1037	11716 ± 171	$\mathbf{19684 \pm 447}$
Manipulator	$\boldsymbol{F(\delta)}$ $(\times 10^{-12})$	$\mathbf{7.76 \pm 0}$	1756 ± 334	2627 ± 791
	$\mathbf{err}^{\delta}_{\text{lqr}}$	$\mathbf{0.0327 \pm 0}$	4.17 ± 9.29	4.61 ± 8.41
	$\boldsymbol{\#\delta}$ **found**	$\mathbf{58465 \pm 2549}$	18962 ± 399	37499 ± 1341
Quadcopter	$\boldsymbol{F(\delta)}$ $(\times 10^{-20})$	$\mathbf{0.29 \pm 0.18}$	3.9 ± 4.7	833 ± 742
	$\mathbf{err}^{\delta}_{\text{lqr}}$ $(\times 10^{-16})$	0.66 ± 0.38	2.33 ± 1.3	$\mathbf{0.31 \pm 0.22}$
	$\boldsymbol{\#\delta}$ **found**	$\mathbf{185797 \pm 5290}$	33882 ± 1404	70776 ± 1296

computation time. Second, to assess their closed-loop performance, we additionally solve for the optimal policy (π^*) in the biped problem or, in the computationally intractable manipulator and quadcopter problems, use popular reinforcement learning algorithms, Advantage Actor Critic (A2C) [65] and Proximal Policy Optimization (PPO) [80], to obtain reference policies. In either case, we then simulate 100 closed-loop trajectories from the computed policies to evaluate the value error (Equation (2.5)) for the biped problem or the reference value errors

$$\text{err}^{\delta}_{\text{A2C/PPO}} = \frac{\sum (V^{\delta} - V^{\text{A2C/PPO}})}{\sum V^{\text{A2C/PPO}}}$$

for the manipulator and quadcopter problems, where V^{δ}, V^{A2C}, and V^{PPO} are the discounted costs of the simulated closed-loop trajectories using the hierarchical, A2C and PPO policies, respectively (Table 3.2). In table 3.2, the negative $\text{err}^{\delta}_{\text{A2C/PPO}}$ indicate that the look-up table hierarchical policies computed using PI offer better closed-loop performance than the neural network policies obtained with A2C and PPO. This is not entirely surprising as neural network policies are at best locally optimal [43] but they scale much better, in terms of memory footprint, to more complex systems in comparison to look-up tables. PI with look-up tables provides guaranteed convergence to the optimal solution (within discretization resolution) and hence our pick to compute and compare policies. We omit a comparison of the required computation time as the methods to obtain the

Table 3.2: Comparison of policy computation + search times and value errors of the hierarchical policies. Value errors are obtained by simulating 100 closed-loop trajectories with the hierarchical and the optimal policy.

	Biped		Manipulator			Quadcopter		
	time (sec)	$\mathbf{err}^{\delta}$	**time** (sec)	$\mathbf{err}^{\delta}_{\text{A2C}}$	$\mathbf{err}^{\delta}_{\text{PPO}}$	**time** (sec)	$\mathbf{err}^{\delta}_{\text{A2C}}$	$\mathbf{err}^{\delta}_{\text{PPO}}$
π^*	12288	-	-	-	-	-	-	-
GA	**93.4 150**	0.28	**104.5 600**	**−0.025**	**−0.098**	30420 + 1200	**−0.114**	**−0.26**
MCTS	103.2+150	0.33	465.9 + 600	−0.0235	−0.096	64061 + 1200	0.22	0.02
Random	109.1+150	0.34	465.2 + 600	−0.016	−0.09	-	-	-
Baseline	1936.6	**0.01**	-	-	-	**6502**	−0.038	−0.20

policies are vastly different.

3.1.3.3 Balance control of simplified biped

As a baseline we use a popular hierarchy for the balance control of a bipedal system which regulates the behavior of the center of mass (l_r, α_r, $\dot{x}$, $\dot{z}$) using leg forces (F_r and F_l), and then a controller for the hip torques (τ_r and τ_l) is designed in cascade to balance the torso (θ, $\dot{\theta}$) [37, 60] (compare figure 3.5 for notations). In contrast, the hierarchies discovered by GA, MCTS and random sampling further decompose control. All three discover controllers that decouple the torso from the center of mass control and further decouple the latter into separate fore-aft and height regulation (Figure 3.6). This aggressive decoupling reduces the computation cost by an order of magnitude when compared to the baseline hierarchy but it comes at the cost of clearly worse performance (biped columns, table 3.2).

3.1.3.4 Swing-up control for planar manipulator

The 4 link manipulator shown in figure 3.5(b) is similar in design to the 2 and 3 link ones presented earlier (Figure 2.5(a) and (b)). (See appendix C for details on the dynamics parameters, the cost function to design policies for swing-up control, and other details for policy optimization.) GA discovers the hierarchy with the best overall fitness (Table 3.2), a fully decoupled manipulator control (Figure 3.7(a)). The best hierarchy that MCTS discovers is similar but retains a coupled control for the first two joints (Figure 3.7(b)). Both hierarchies have comparable reference value errors, but the GA hierarchy is computed four times faster than the MCTS hierarchy. Moreover, these errors are negative, which means that the hierarchical policies offer better closed-loop performance than the

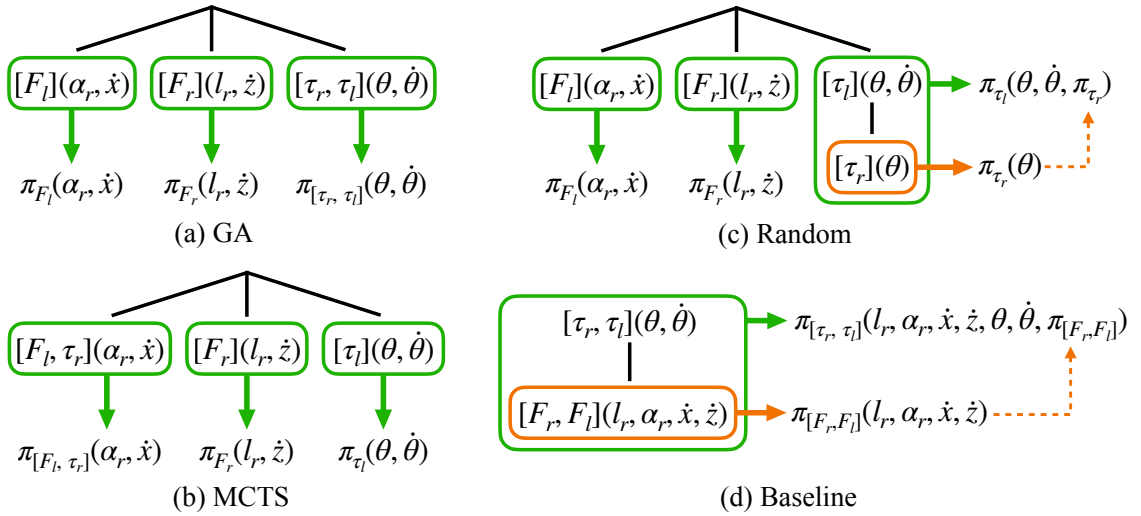


Figure 3.6: Hierarchies for the simplified biped shown in figure 3.5(a). Hierarchies discovered by GA, MCTS and random sampling decouple the fore-aft control, the height regulation and the torso balance. The baseline hierarchy is based on several reported works in the literature [37, 60].

policies obtained with the popular A2C and PPO methods. This observation holds even for the hierarchy discovered by random sampling, although it is coupled and cascaded (Figure 3.7(c)) and does not perform as well as the other two hierarchies.

3.1.3.5 Hover control of a quadcopter

The quadcopter depicted in figure 3.5(c) is described with its center of mass height (z) and velocity ($\dot{x}$, $\dot{y}$, $\dot{z}$), orientation expressed as roll (ϕ), pitch (θ) and yaw (ψ), and angular velocity expressed as roll, pitch and yaw rates ($\dot{\phi}$, $\dot{\theta}$ and $\dot{\psi}$). The control inputs are defined as the net thrust T and the differential thrusts for roll, pitch and yaw, F_{roll}, F_{pitch} and F_{yaw}, respectively. We search for hierarchical policies that stabilize attitude and bring the quadcopter to a stop at a height of 1m. (See appendix C for dynamics parameters, the cost function used to design policies, and hyper-parameters for policy optimization.)

The best hierarchies discovered by GA and MCTS are reported in figure 3.8. These hierarchies decouple the yaw control from the rest, and the one identified by GA further cascades the roll and pitch controller with the thrust controller. Note that random sampling did not yield a hierarchy with significant decoupling or cascading; hence, computing its policy remained beyond our computational budget and we do not report on it further. A widely recognized Integral Backstepping Control for the quadcopter is presented in [17]. We extract a suitable hierarchy for policy optimization based on the reported cascaded

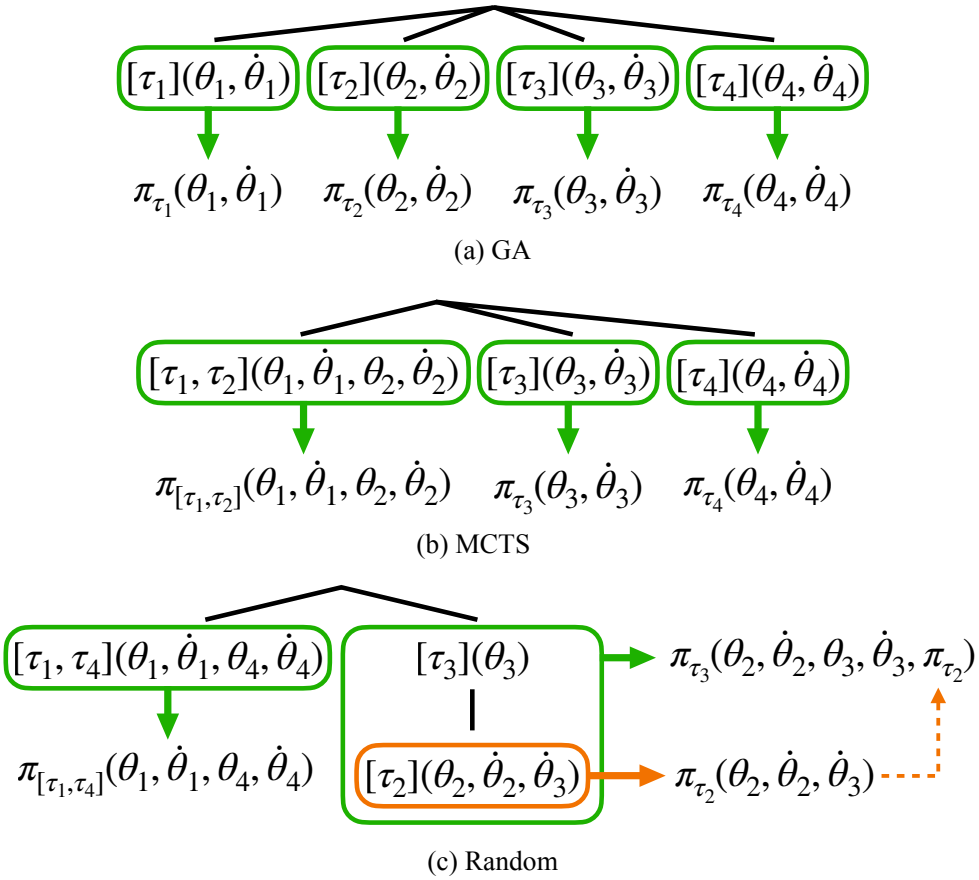


Figure 3.7: Hierarchies discovered by GA, MCTS and random sampling for the manipulator shown in figure 3.5(b).

control structure as a baseline and compare its performance with the algorithmically identified ones. This baseline hierarchy is depicted in figure 3.8(c) [1].

As for the previous problems, the hierarchy discovered with GA offers the best closed-loop performance (Table 3.2). It provides significant improvements in performance over the baseline, highlighting the utility of the Policy Decomposition search framework. Furthermore, it also outperforms the neural network based policies derived using A2C and PPO. For the hierarchy discovered with MCTS, only 78 of the 100 simulated trajectories converge, resulting in a very high value error estimate (even though the trajectories that do converge have substantially lower costs than those derived from other hierarchical policies).

[1]This controller structure cannot be represented using an input-tree and does not correspond to a valid hierarchy under the current framework. A directed acyclic graph can be used to represent such a structure and inclusion of such hierarchies in the Policy Decomposition framework is left for future work.

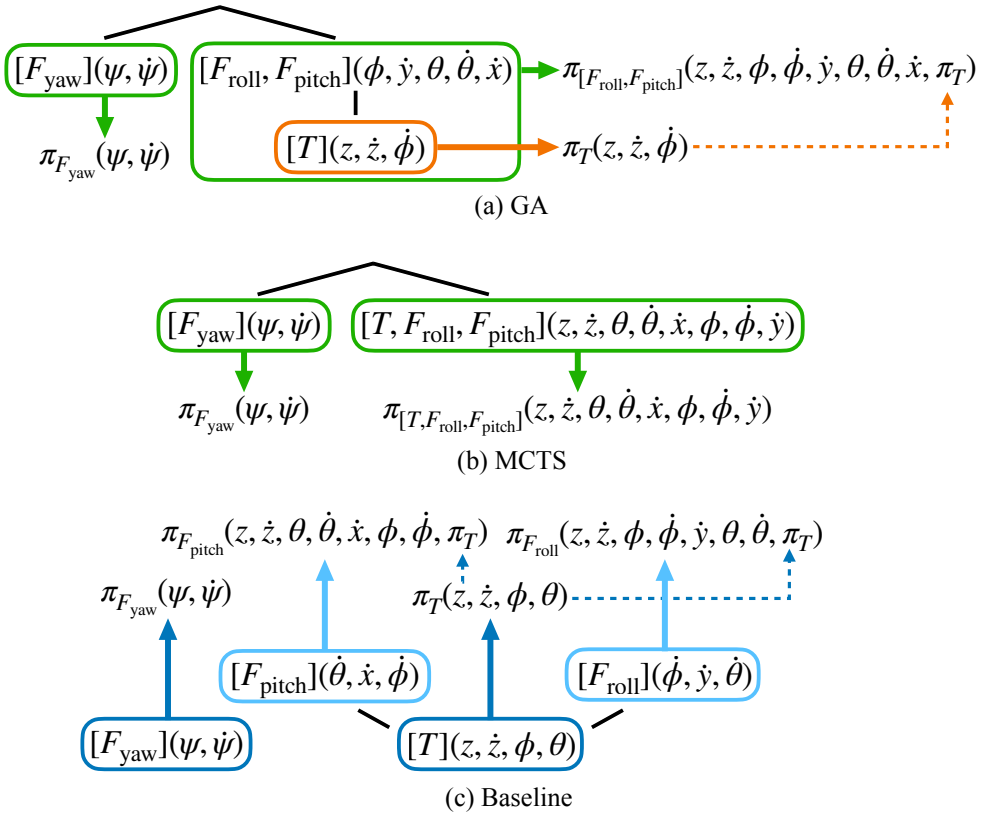


Figure 3.8: Hierarchies discovered by GA and MCTS for the quadcopter shown in figure 3.5(c), as well as a baseline hierarchy based on [17].

3.2 Pareto Search

The search for promising hierarchies is a bi-objective optimization with suboptimality and computation time as the two competing goals. The fitness function (Equation (3.1)) reduces this optimization to a single score and can thereby introduce bias toward one of the two goals. For example, in the biped problem (Section 3.1.3.3), cascaded hierarchies exist which offer lower suboptimality than the decoupled ones discovered by GA and MCTS. But, these cascaded hierarchies do not feature as the top candidates based on fitness score, because their lower suboptimality cannot compensate for the orders of magnitude reduction in computation time offered by the decoupled hierarchies. Contrarily, in the quadcopter problem (Section 3.1.3.5), the suboptimality criterion dominates the fitness. To avoid such bias, we investigate finding a Pareto front of hierarchies using a variant of GA (NSGA-II [26]) with the adaptations introduced in section 3.1.1, and $F_{\text{err}}(\delta)$ and

$F_{\text{comp}}(\delta)$ (Equation (3.1)) as measures of the two competing criteria.

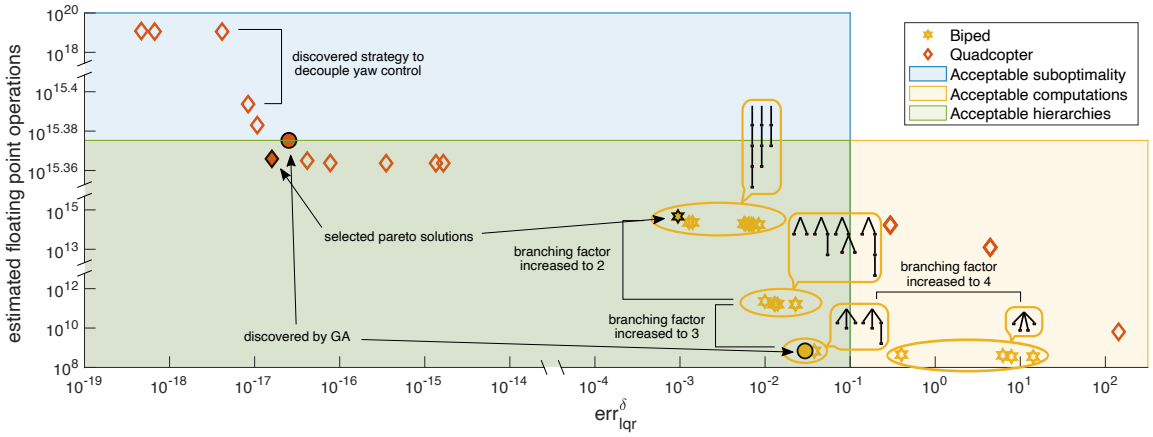


Figure 3.9: Pareto fronts of hierarchies for the biped and the quadcopter. Hierarchies towards the top-left exhibit low suboptimality but require more floating point operations to compute control policies whereas those towards the bottom-right are highly suboptimal but offer dramatic reduction in computation. Discernible structural changes in the hierarchies for the biped and the quadcopter are highlighted.

Figure 3.9 depicts the Pareto front of hierarchies for the biped and quadcopter. We allowed a time budget of 150 seconds and 1200 seconds in finding hierarchies for the biped and the quadcopter respectively. Hierarchies that require more computation for policy optimization but offer low value errors appear in the top left corner of figure 3.9, and those that require minimal computation but have large errors feature in the bottom right. For the biped problem, the hierarchies in the Pareto front fall into four groups (yellow stars) based on the number of branches in the underlying input-trees. Similarly, the quadcopter hierarchies (red diamonds) fall into three groups, one for hierarchies with a single branch in the underlying input-trees (top left), one for hierarchies with the yaw control decoupled (middle), and one for highly decoupled and suboptimal hierarchies (towards bottom right). Additionally, for the purpose of demonstration, assumed ranges of acceptable suboptimality (blue area) and computation time (yellow area) are shown with their overlap defining acceptable hierarchies (green area). Two of these acceptable hierarchies corresponding to the lowest possible suboptimality for the biped and the quadcopter are highlighted (bold red diamond and yellow star, respectively) in figure 3.9. The input-trees for these highlighted hierarchies are shown in figure 3.10, and the required time for policy optimization as well as the value error estimates for the resulting policies are reported in table 3.3. For the quadcopter, the highlighted hierarchy reduces policy computation time by more than half in comparison to the hierarchy with the lowest fitness

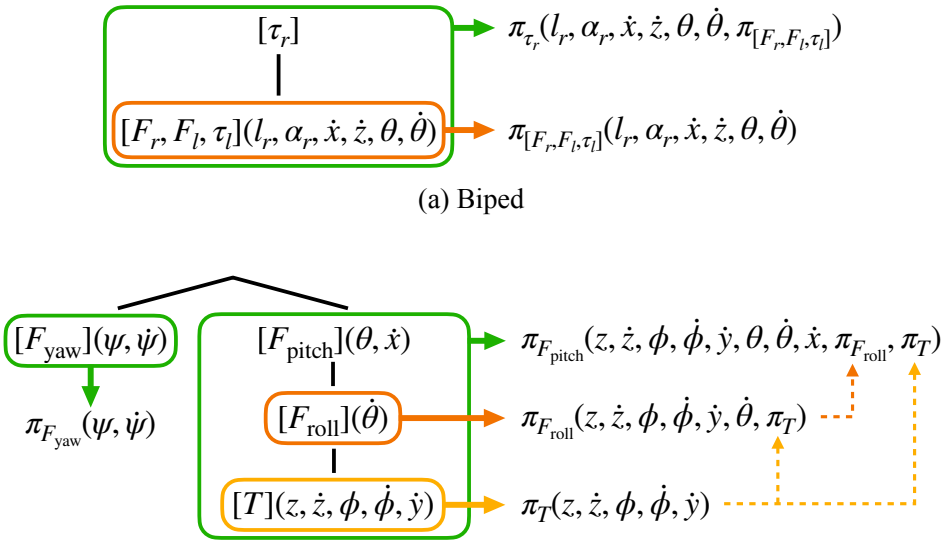


Figure 3.10: Hierarchies for the biped (Figure 3.5(a)) and the quadcopter (Figure 3.5(c)) selected from the Pareto fronts depicted in figure 3.9.

found previously by GA (compare quadcopter columns in table 3.2 and table 3.3) with only marginally worse suboptimality. In the case of the biped, the highlighted hierarchy requires substantially more computation time but provides a dramatic improvement in performance compared to the lowest fitness hierarchy found previously by GA (compare biped columns in table 3.2 and table 3.3).

Table 3.3: Policy computation + search times and value error estimates for the Pareto optimal hierarchies for the biped and quadcopter depicted in figure 3.10.

Biped		Quadcopter		
time (sec)	$\mathbf{err}^{\delta}$	**time** (sec)	$\mathbf{err}^{\delta}_{\text{A2C}}$	$\mathbf{err}^{\delta}_{\text{PPO}}$
8220 + 150	0.0005	12725 + 1200	−0.085	−0.236

Deriving the Pareto front for the two competing search criteria of suboptimality and computation time not only provides choices of hierarchies that satisfy design constraints but also delivers more fundamental insights into what changes in the control structure affect the closed-loop performance of a given dynamical system. These analysis benefits are obtained with negligible additional search effort, as in the examples above, the total time allotted for search with the NSGA-II algorithm equalled the time alloted for the plain GA search (150 and 1200 seconds for the biped and quadcopter problems respectively).

3.3 Discussion

As an alternative to the discrete search methods described earlier, we explored a formulation to pose the search for hierarchies as a continuous optimization problem. Specifically, we investigated if the cascading and decoupling operations used to generate different hierarchies under the Policy Decomposition framework could be emulated by constructing suitable *masks* to the original system dynamics. The search for promising hierarchies could then be posed as an optimization over the masking variables with the fitness function (Equation (3.1)) as the optimization objective. To formulate the search as a continuous optimization problem that can be solved using gradient descent, we require 1) an encoding scheme that uses continuous variables to represent hierarchies, and 2) a differentiable process to compute the corresponding suboptimality estimate. In section 3.3.1 we present an encoding scheme for decentralized hierarchies using binary variables (which are then relaxed to be continuous variables in the range $[0, 1]$), and discuss how gradients of the LQR suboptimality estimate can be computed with respect to the encoding. The resulting optimization is not a tight convex relaxation and thus the converged solution to the optimization need (does) not in fact correspond to a valid hierarchy. We then apply randomized rounding to find the "closest" valid solution.

The encoding scheme in section 3.3.1 only captures decentralized hierarchies. Encompassing hierarchies that include cascaded subsystems in such a scheme is difficult since cascading reduces the original optimization problem into subproblems that need to be solved in a particular order. Furthermore, even for decentralized hierarchies, this approach for continuous search does not find better hierarchies than the discrete methods we have presented earlier in this chapter. We include the formulation here only for completeness.

3.3.1 Hierarchy Search as a Continuous Optimization

An encoding scheme for decentralized hierarchies is as follows

$$
\begin{aligned}
&\dot{\boldsymbol{x}} = \tilde{\boldsymbol{A}}(\boldsymbol{x} - \boldsymbol{x}^d) + \tilde{\boldsymbol{B}}(\boldsymbol{u} - \boldsymbol{u}^d) \\
\text{where}\quad &\tilde{\boldsymbol{A}} = \boldsymbol{A} \odot \boldsymbol{X},\ \tilde{\boldsymbol{B}} = \boldsymbol{B} \odot \boldsymbol{S} \\
&\boldsymbol{A} = \left.\frac{\partial \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u})}{\partial \boldsymbol{x}}\right|_{(\boldsymbol{x}^d, \boldsymbol{u}^d)},\ \boldsymbol{B} = \left.\frac{\partial \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u})}{\partial \boldsymbol{u}}\right|_{(\boldsymbol{x}^d, \boldsymbol{u}^d)} \\
&\boldsymbol{X}_{i,j} = 1 \ \text{ if } x_j \text{ affects } x_i\text{'s dynamics} \\
&\boldsymbol{S}_{i,k} = 1 \ \text{ if } u_k \text{ affects } x_i\text{'s dynamics} \\
&\boldsymbol{C}_{k,l} = 1 \ \text{ if policies for } u_k \text{ and } u_l \text{ jointly optimized}
\end{aligned}
\tag{3.3}
$$

For a system with n state variables and m control inputs, the binary masking matrices $\boldsymbol{X} \equiv n \times n$ and $\boldsymbol{S} \equiv m \times n$ encode the cross-dependence of different state-variables and control inputs on the dynamics and the coupling matrix $\boldsymbol{C} \equiv m \times m$ encodes policies for which inputs are jointly optimized. Figure 3.11 depicts the resulting masks for a decentralized hierarchy of fictive system with four state variables and control inputs. To

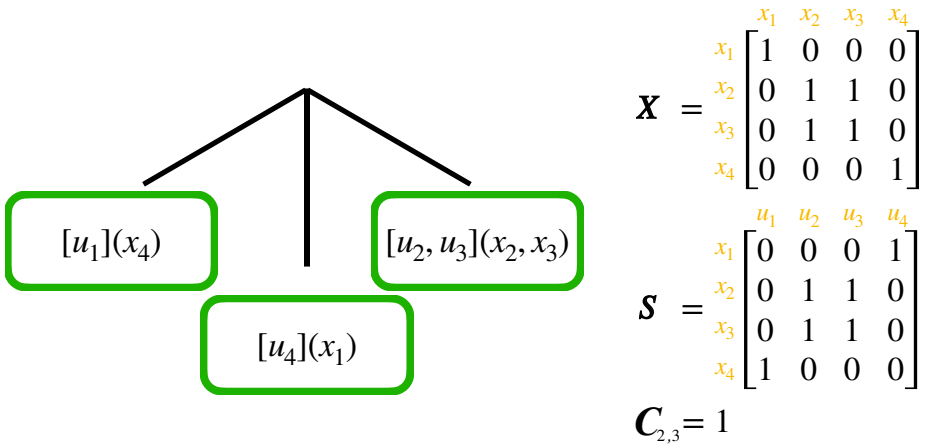


Figure 3.11: Encoding for a hierarchy of a fictive system with four state variables and control inputs under the scheme described in equation (3.3)

.

ensure a valid decentralized hierarchy, the following constraints are imposed

$$\begin{aligned}
\sum_{k=1}^{m} \boldsymbol{S}_{i,k} &\geq 1 \quad \text{each state variable } x_i \text{ must be controlled by at least one input} \\
|\boldsymbol{S}_{i,j} - \boldsymbol{S}_{i,k}| &\leq 1 - \boldsymbol{C}_{j,k} \quad \text{coupled inputs } u_j \text{ and } u_k \text{ must control the same state variables} \\
\boldsymbol{S}_{i,j} + \boldsymbol{S}_{i,k} &\leq 1 + \boldsymbol{C}_{j,k} \quad \text{decoupled inputs } u_j \text{ and } u_k \text{ must not control the same state variables} \\
\boldsymbol{X}_{i,i} &= 1 \quad \text{each state variable } x_i \text{ affects its dynamics} \\
\boldsymbol{X}_{i,j} &\geq \boldsymbol{S}_{i,k} + \boldsymbol{S}_{j,l} + \boldsymbol{C}_{k,l} \quad \text{if } x_i \text{ and } x_j \text{ are controlled by coupled inputs } u_k \text{ and } u_l \text{ respectively then their dynamics are coupled} \\
\boldsymbol{C}_{i,j} + \boldsymbol{C}_{j,k} &\leq 1 + \boldsymbol{C}_{i,k} \text{ and } \boldsymbol{C}_{i,j} = \boldsymbol{C}_{j,i} \quad \text{coupling must satisfy transitivity and reflectivity}
\end{aligned} \tag{3.4}$$

Then to compute the LQR suboptimality estimate (Equation (2.7)) over a unit ball around $\boldsymbol{x}^d$ we have from equation (2.10)

$$\begin{aligned}
&\int_{\mathcal{S}=\{\boldsymbol{x} \mid \|\boldsymbol{x}-\boldsymbol{x}^d\| \leq 1\}} V_{\text{lqr}}^{\delta}(\boldsymbol{x}) = \text{trace}(\boldsymbol{P}^{\delta}) \\
\text{s.t. } &\left(\boldsymbol{A} - \boldsymbol{B}\tilde{\boldsymbol{K}} - \frac{\lambda \boldsymbol{I}}{2}\right)^T \boldsymbol{P}^{\delta} + \boldsymbol{P}^{\delta}\left(\boldsymbol{A} - \boldsymbol{B}\tilde{\boldsymbol{K}} - \frac{\lambda \boldsymbol{I}}{2}\right) + \boldsymbol{Q} + \tilde{\boldsymbol{K}}^T \boldsymbol{R} \tilde{\boldsymbol{K}} = \boldsymbol{0}
\end{aligned} \tag{3.5}$$

where $\tilde{\boldsymbol{K}}$ is the LQR controller for the masked system obtained by solving the riccatti equation

$$\begin{aligned}
\tilde{\boldsymbol{K}} &= \boldsymbol{R}^{-1} \tilde{\boldsymbol{B}}^T \tilde{\boldsymbol{P}} \\
\text{s.t. } \tilde{\boldsymbol{A}}^T \tilde{\boldsymbol{P}} + \tilde{\boldsymbol{P}} \tilde{\boldsymbol{A}} + \boldsymbol{Q} + \tilde{\boldsymbol{P}} \tilde{\boldsymbol{B}} \boldsymbol{R}^{-1} \tilde{\boldsymbol{B}} \tilde{\boldsymbol{P}} &= \boldsymbol{0}
\end{aligned} \tag{3.6}$$

The gradients for the masking matrices $\boldsymbol{X}$ and $\boldsymbol{S}$ are obtained using chain rule

$$\begin{aligned}
\frac{\partial \text{trace}(\boldsymbol{P}^{\delta})}{\partial \boldsymbol{X}_{i_j}} &= \underbrace{\frac{\partial \text{trace}(\boldsymbol{P}^{\delta})}{\partial \tilde{\boldsymbol{K}}}}_{\text{refer [42]}} \underbrace{\frac{\partial \tilde{\boldsymbol{K}}}{\partial \tilde{\boldsymbol{A}}}}_{\text{refer [42]}} \frac{\partial \tilde{\boldsymbol{A}}}{\partial \boldsymbol{X}_{i,j}} \\
\frac{\partial \text{trace}(\boldsymbol{P}^{\delta})}{\partial \boldsymbol{S}_{i,k}} &= \underbrace{\frac{\partial \text{trace}(\boldsymbol{P}^{\delta})}{\partial \tilde{\boldsymbol{K}}}}_{\text{refer [42]}} \underbrace{\frac{\partial \tilde{\boldsymbol{K}}}{\partial \tilde{\boldsymbol{B}}}}_{\text{refer [42]}} \frac{\partial \tilde{\boldsymbol{B}}}{\partial \boldsymbol{S}_{i,k}}
\end{aligned} \tag{3.7}$$

3.3.2 Conclusions

We addressed the combinatorial challenge of applying Policy Decomposition to complex systems, finding non-trivial hierarchies using Genetic Algorithm and Monte-Carlo Tree Search. By applying to three very distinct control problems we demonstrated the generality of the framework. Optimal control problems with more than 7 dimensional state have remained intractable when representing policies as look-up tables over the state-space, and we showcase near optimal solutions for the control of a quadcopter system with a 10 dimensional state. Although, look-up tables and Policy Iteration are a primitive choice of policy representation and algorithm, they provide strong convergence guarantees to the optimal solution [10]. Look-up tables do not scale well in terms of memory footprint as well as required computation for policy optimization of further higher-dimensional problems. In chapter 5 we showcase that Policy Decomposition can be applied even towards training neural network policies which scale much better to more complex problems.

Under the current framework, we assume a fixed system representation, i.e. the state-variables and control inputs describing a system, and derive hierarchies accordingly. This assumption is restrictive. For example, in case of the quadcopter, policy computation for the rotor forces F_i cannot be readily reduced to a hierarchy that offers notable reduction in computation while still providing desirable closed-loop performance. However, for the linearly transformed inputs (net thrust T, and the roll, pitch and yaw differential thrusts F_ϕ, F_θ and F_ψ), we demonstrated that hierarchies that offer dramatic reduction in computation while sacrificing minimally on closed-loop performance can be found. Next, in chapter 4 we address the problem of discovering system representations that lead to promising hierarchies.

Chapter 4

Aligning System Representations for Policy Decompositions

The Policy Decomposition framework simplifies the process of computing optimal control policies by decomposing (decoupling and/or cascading) the policy computation for different control inputs. However, the choice of system representation, namely state variables and control inputs describing a system, directly influences the quality of the resulting hierarchies. Consider the problem of obtaining an optimal controller for the quadcopter to hover in place (Figure 4.1). Policy Decomposition is unable to find a strategy for decomposing the policy optimization for motor thrusts F_i that offers significant reduction in computation while producing stable closed-loop behavior. In contrast, suitable hierarchies are found if one uses the linearly transformed inputs, net thrust T and the roll, pitch and yaw differential thrusts (F_ϕ, F_θ and F_ψ respectively). For the quadcopter, we intuitively posit a system representation that lead to hierarchies with lower suboptimality. Such representations are difficult to hand design for a general system.

Methods that find a change of coordinates for the state and input spaces to facilitate hierarchical controller design do exist. These works can be divided into two categories based on the type of hierarchical controllers they design: decentralized or cascaded. For decentralized controller design [5], different subspaces of inputs are paired with subspaces of the outputs[1] and controllers for the resulting subsystems are derived in a decoupled fashion. These pairings are constructed based on measures of interaction between the

[1]for fully observable systems outputs are the same as the system state

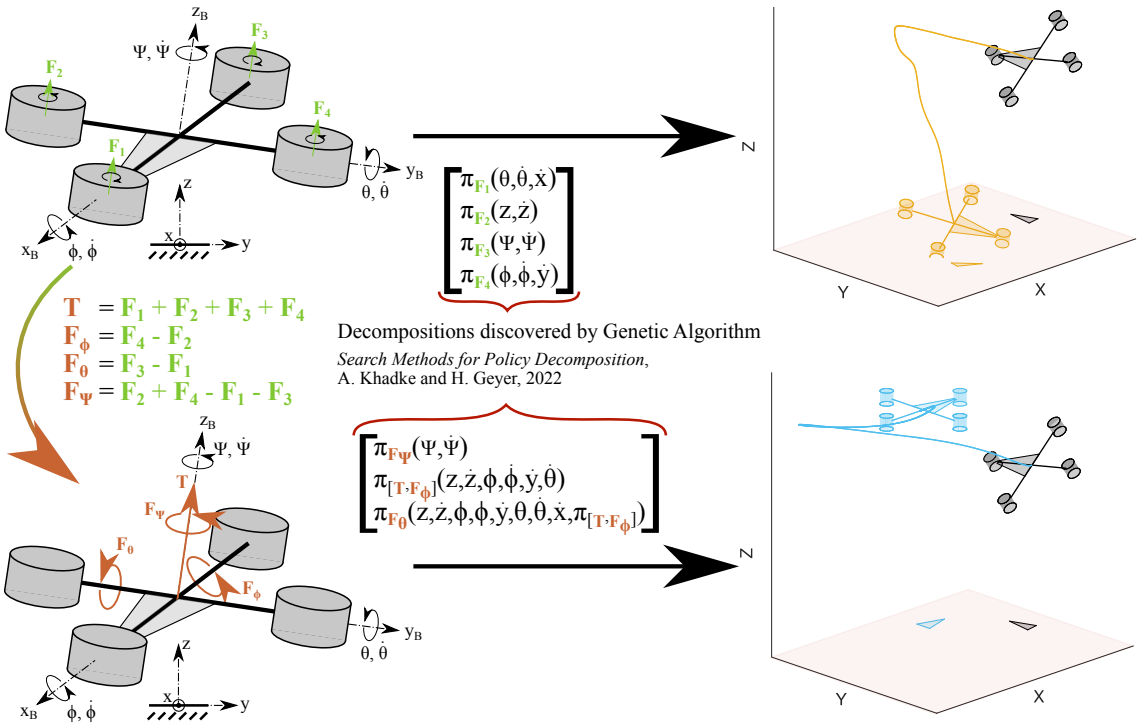


Figure 4.1: Best hierarchies for the two representations of the quadcopter system found using Genetic Algorithm.

inputs and outputs, primarily based on controllability [19, 24, 38, 48, 52, 63, 77, 98]. For cascaded design, either underlying kinetic symmetries are exploited [71] or feedback is employed to transform the system into a normal form [44] which naturally lends itself to a cascaded controller design. All of these works are agnostic to the objective of the optimal control problem, and as such the transformed system representations can lead to highly suboptimal controllers. Here, we present an approach that accounts for the control objective to transform a given system representation to one that generates hierarchies with reduced suboptimality. Furthermore, our approach is not specific to any particular type of hierarchy.

4.1 Sparsity Inducing System Representations

We pose the search for a system representation $(\boldsymbol{y}, \boldsymbol{v})$ as the search for state and input mappings from a known representation $(\boldsymbol{x}, \boldsymbol{u})$. Here, we only consider linear and invertible

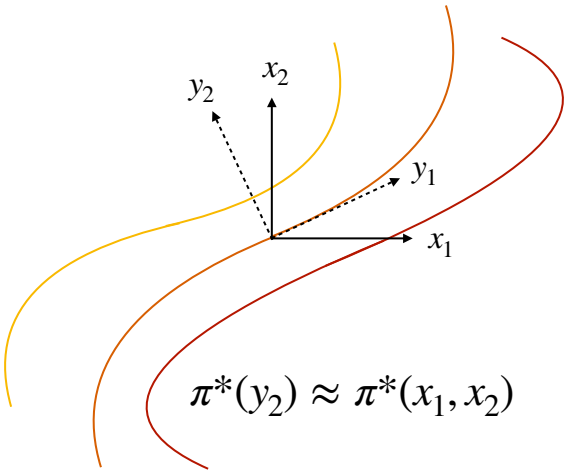


Figure 4.2: Illustrating the idea of maximal variation co-ordinates for an optimal policy. Contour plot of an optimal policy over a two dimensional state-space is shown. Expressing the policy in the maximal variation co-ordinates allows ignoring dependencies along sub-spaces with minimal variation.

mappings,

$$\boldsymbol{y} = \boldsymbol{T}_y\ (\boldsymbol{x} - \boldsymbol{x}^d), \quad \boldsymbol{v} = \boldsymbol{T}_v\ (\boldsymbol{u} - \boldsymbol{u}^d)$$

The dynamics of the system in equation (2.2) can then be expressed in representation $(\boldsymbol{y}, \boldsymbol{v})$ as,

$$\dot{\boldsymbol{y}} = \boldsymbol{T}_y \boldsymbol{f}(\boldsymbol{T}_y^{-1}\boldsymbol{y} + \boldsymbol{x}^d, \boldsymbol{T}_v^{-1}\boldsymbol{v} + \boldsymbol{u}^d)$$

Let $\delta_{(\boldsymbol{y},\boldsymbol{v})}$ denote the hierarchies of the system when expressed in the $(\boldsymbol{y}, \boldsymbol{v})$ representation. We desire mappings that minimize the value-error of the least suboptimal hierarchy,

$$\underset{\boldsymbol{T}_y, \boldsymbol{T}_v}{\operatorname{argmin}} \left(\min_{\delta_{(\boldsymbol{y},\boldsymbol{v})}} \operatorname{err}^{\delta_{(\boldsymbol{y},\boldsymbol{v})}} \right) \tag{4.1}$$

However, the above objective is not differentiable even for a linear system, and using gradient-free methods to search for mappings $\boldsymbol{T}_y$ and $\boldsymbol{T}_v$ along with the hierarchy can be too computationally expensive.

An alternative heuristic is to find a system representation that *aligns* with the maximum variation in the optimal policy. Every hierarchy imposes certain information constraints on the resulting policy π^δ at the cost of optimality, whereby the dependence of (variation in) the policy for different input subspaces on (with respect to) certain subspaces of the state-space is ignored. Intuitively, aligning with the directions of maximal variation in the optimal policy would lead to minimal dependencies being ignored when constructing

hierarchical policies (Figure 4.2). The mappings $\boldsymbol{T}_y$ and $\boldsymbol{T}_v$, that transform the known system representation to one aligned with the maximal variation, Jacobian of the optimal policy with respect to the state variables. However, this requires knowing the optimal policy. Subsequently, we obtain $\boldsymbol{T}_y$ and $\boldsymbol{T}_v$ based on an approximation of the optimal policy and its Jacobian near the goal state $\boldsymbol{x}^d$. We linearize the system dynamics,

$$\dot{\boldsymbol{y}} = \boldsymbol{T}_y \boldsymbol{A} \boldsymbol{T}_y^{-1} \boldsymbol{y} + \boldsymbol{T}_y \boldsymbol{B} \boldsymbol{T}_v^{-1} \boldsymbol{v}$$

$$\boldsymbol{A} = \left.\frac{\partial \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u})}{\partial \boldsymbol{x}}\right|_{(\boldsymbol{x}^d, \boldsymbol{u}^d)}, \quad \boldsymbol{B} = \left.\frac{\partial \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u})}{\partial \boldsymbol{u}}\right|_{(\boldsymbol{x}^d, \boldsymbol{u}^d)}$$

Since our cost function is quadratic, the approximation to the optimal policy, $\pi_{\boldsymbol{v}}^{*\text{approx}}(\boldsymbol{y})$, is the solution to the LQR problem

$$\begin{gathered}
\left(\boldsymbol{A} - \frac{\lambda}{2}\boldsymbol{I}\right)^T \boldsymbol{P} + \boldsymbol{P}\left(\boldsymbol{A} - \frac{\lambda}{2}\boldsymbol{I}\right) + \boldsymbol{Q} - \boldsymbol{P}\boldsymbol{B}\boldsymbol{R}^{-1}\boldsymbol{B}^T\boldsymbol{P} = \boldsymbol{0} \\
\pi_{\boldsymbol{v}}^{*\text{approx}}(\boldsymbol{y}) = -\boldsymbol{\Theta}\boldsymbol{y}, \quad \frac{\partial \pi_{\boldsymbol{v}}^{*\text{approx}}}{\partial \boldsymbol{y}} = -\boldsymbol{\Theta} \\
\text{where, } \boldsymbol{\Theta} = \boldsymbol{T}_v \boldsymbol{K}^* \boldsymbol{T}_y^{-1} \text{ and } \boldsymbol{K}^* = \boldsymbol{R}^{-1}\boldsymbol{B}^T\boldsymbol{P}
\end{gathered} \tag{4.2}$$

We obtain a sparse Jacobian to the optimal policy using the singular value decomposition. Let $\boldsymbol{K}^* = \boldsymbol{U}_{\boldsymbol{K}^*}\boldsymbol{S}_{\boldsymbol{K}^*}\boldsymbol{V}_{\boldsymbol{K}^*}^T$ where $\boldsymbol{U}_{\boldsymbol{K}^*} \in \mathbb{R}^{m\times m}$ and $\boldsymbol{V}_{\boldsymbol{K}^*} \in \mathbb{R}^{n\times n}$ are orthogonal, then $\boldsymbol{T}_y = \boldsymbol{V}_{\boldsymbol{K}^*}^T$ and $\boldsymbol{T}_v = \boldsymbol{U}_{\boldsymbol{K}^*}^T$ result in a completely diagonal $\boldsymbol{\Theta} = \boldsymbol{S}_{\boldsymbol{K}^*}$. Note that the singular value decomposition is not unique for rectangular matrices. We encounter wide $\boldsymbol{K}^*$s as all our systems have fewer control inputs than state variables ($m < n$) and therefore we have some degree of freedom in choosing $\boldsymbol{U}_{\boldsymbol{K}^*}$ and $\boldsymbol{V}_{\boldsymbol{K}^*}$. Appendix B.1 presents a regularization strategy to resolve this non-uniqueness.

4.2 Empirical Analysis with Linear Systems

We verify our intuition that system representations which induce sparsity in the Jacobian of the optimal policy indeed lead to hierarchies with reduced value-errors. Towards this end, we sample several optimal control problems with linear system dynamics, enumerate all possible hierarchies of the original and transformed systems, and compare the value-errors. We use two sampling strategies to randomly generate the underlying linear

dynamical system and cost function that define the optimal control problem

- **strategy I** : We directly sample the linear system and the cost function. $\boldsymbol{A}$, $\boldsymbol{B}$, $\boldsymbol{Q}_s$ and $\boldsymbol{R}_s$ are sampled independently and uniformly over $[0,1]^{n\times n}$, $[0,1]^{n\times m}$, $[0,1]^{n\times n}$ and $[0,1]^{m\times m}$ respectively. $\boldsymbol{A}$ and $\boldsymbol{B}$ are the dynamics matrices, and the cost matrices are $\boldsymbol{Q} = \boldsymbol{Q}_s\boldsymbol{Q}_s^T$ and $\boldsymbol{R} = \boldsymbol{R}_s\boldsymbol{R}_s^T$. The discount factor λ is set to 0.
- **strategy II** : We sample a square optimal gain matrix $\boldsymbol{K}^*$ with equal singular values. Repeating singular values introduces additional degrees of freedom in the singular value decomposition. This sampling procedure is meant to test our regularization strategy (Appendix B.1) for resolving the non-uniqueness. Dynamics matrices $\boldsymbol{A}$ and $\boldsymbol{B}$, and cost matrices $\boldsymbol{Q}$ and $\boldsymbol{R}$ are chosen such that $\boldsymbol{K}^*$ is the solution to the LQR problem equation (4.2). The discount factor λ is set to 0.

For each of these, we compute mappings $\boldsymbol{T}_y$ and $\boldsymbol{T}_v$ using our approach (Section 4.1) as well as using balanced realization [66], and evaluate the value-error for every possible hierarchy of the original and transformed systems. In table 4.1, we report the number of instances when value-error corresponding to the least suboptimal hierarchy of the transformed system is smaller than that for the original one. For more than 90% of the sampled linear systems, transforming to a representation using our approach leads to hierarchies with reduced suboptimality.

Table 4.1: Number of linear systems that exhibit hierarchies with lower value-errors when transformed with the singular value decomposition based mappings introduced in section 4.1, and when transformed with balanced realization [66]. For system sizes indicated with †, evaluating all possible hierarchies is too time consuming and we thus use Genetic Algorithm (Section 3.1.1) to find hierarchies of the original and transformed systems that minimize the value error.

strategy	**system**	**# systems with lower value-error when transformed**	
	(# inputs, states)	ours	balanced realization
I	(2,4)	97 / 100	56 / 100
	(2,6)	100 / 100	48 / 100
	(2,10)	99 / 100	45 / 100
	(2,15)	100 / 100	44 / 100
	(3,3)	100 / 100	60 / 100
	(3,4)	100 / 100	50 / 100
	(3,10)†	100 / 100	64 / 100
II	(2,2)	92 / 100	27 / 100
	(3,3)	99 / 100	49 / 100
	(4,4)	99 / 100	73 / 100
	(8,8)†	95 / 100	66 / 100

4.3 Hierarchies for Biped, Manipulator and Quadcopter

We revisit the design of hierarchical policies for balancing the simplified biped, swing-up control of 4 degree of freedom manipulator and hover control of the quadocpter figure 4.3. The original system representations $(\boldsymbol{x}, \boldsymbol{u})$ used to describe these systems are,

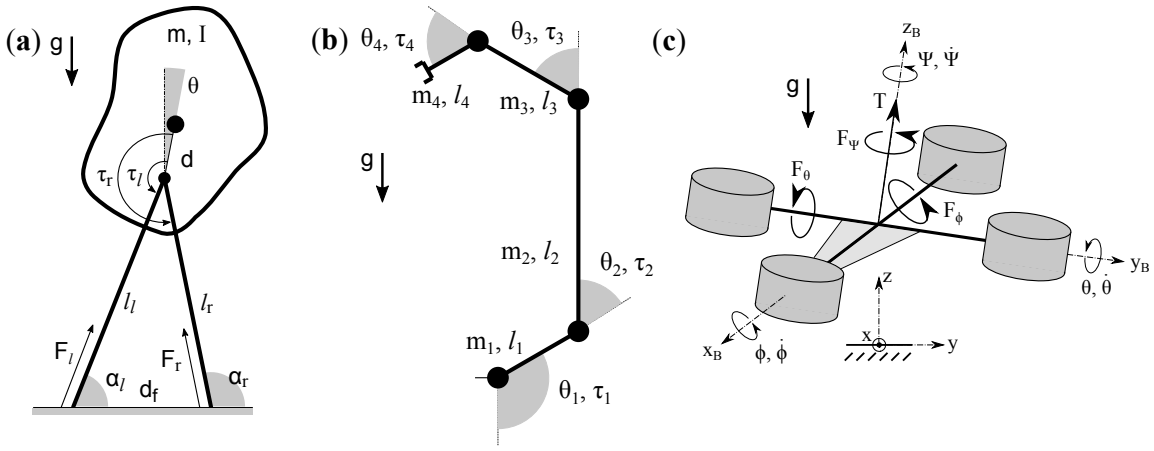
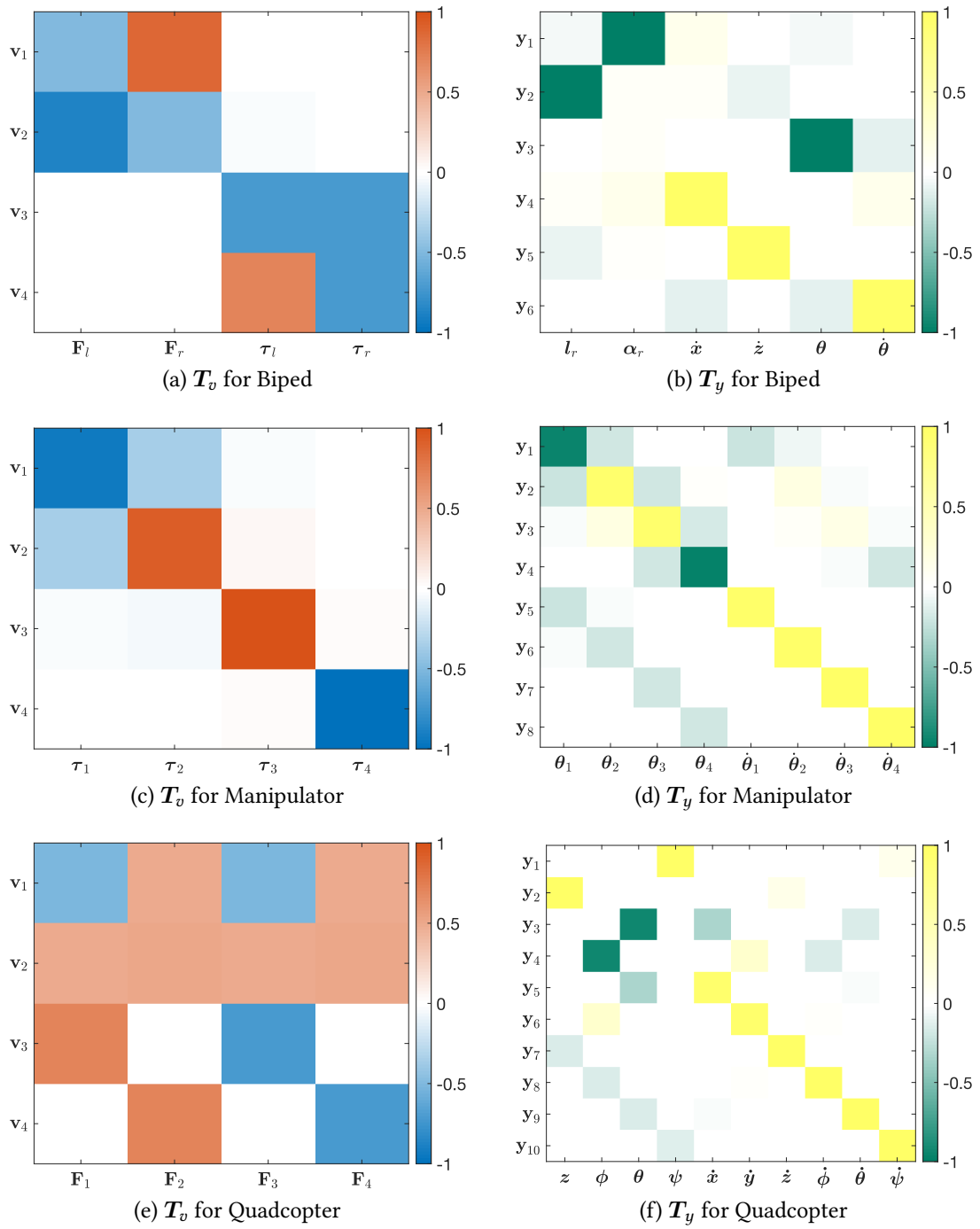


Figure 4.3: (**a**) Balance control of a simplified planar biped. (**b**) Swing-up control of a 4-link planar manipulator. (**c**) Hover control of a quadcopter.

- **Biped** : The simplified biped shown in figure 4.3(a) has four inputs (leg forces $F_{l/r}$ and hip torques $\tau_{l/r}$) and is described by six state variables, the leg length l_r, leg angle α_r, and torso angle θ; velocities, $\dot{x}$, $\dot{z}$, and $\dot{\theta}$. Therefore, $(\boldsymbol{x}, \boldsymbol{u}) = ([l_r, \alpha_r, \dot{x}, \dot{z}, \theta, \dot{\theta}], [F_l, F_r, \tau_l, \tau_r])$.
- **Manipulator** : The planar manipulator shown in figure 4.3(b) has four inputs (joint torques τ_i) and is completely described by the joint positions θ_i and joint velocities $\dot{\theta}_i$. Therefore, $(\boldsymbol{x}, \boldsymbol{u}) = ([\theta_1, \theta_2, \theta_3, \theta_4, \dot{\theta}_1, \dot{\theta}_2, \dot{\theta}_3, \dot{\theta}_4], [\tau_1, \tau_2, \tau_3, \tau_4])$.
- **Quadcopter** : The quadcopter shown in figure 4.3(c) has four inputs (motor thrusts F_i) and is described by ten state-variables, its centre-of-mass height z, attitude expressed in roll (ϕ), pitch (θ) and yaw (ψ), and velocities $\dot{x}$, $\dot{y}$, $\dot{z}$, $\dot{\phi}$, $\dot{\theta}$ and $\dot{\psi}$. Therefore, $(\boldsymbol{x}, \boldsymbol{u}) = ([z, \phi, \theta, \psi, \dot{x}, \dot{y}, \dot{z}, \dot{\phi}, \dot{\theta}, \dot{\psi}], [F_1, F_2, F_3, F_4])$.

We consider the linearized system dynamics at the goal state only to derive the linear mappings $\boldsymbol{T}_y$ and $\boldsymbol{T}_v$ using the approach in section 4.1; these are shown in figure 4.4. For the biped, the inputs are split into two groups in the transformed space, one consisting



(a) $\boldsymbol{T}_v$ for Biped

(b) $\boldsymbol{T}_y$ for Biped

(c) $\boldsymbol{T}_v$ for Manipulator

(d) $\boldsymbol{T}_y$ for Manipulator

(e) $\boldsymbol{T}_v$ for Quadcopter

(f) $\boldsymbol{T}_y$ for Quadcopter

Figure 4.4: Singular value decomposition based mappings $\boldsymbol{T}_y$ and $\boldsymbol{T}_v$ for the systems shown in Fig. 3.5.

of the leg-forces F_l and F_r, and the other the hip torques τ_l and τ_r (Figure 4.4(a)). The transformed state variables y_1, y_2, y_4 and y_5 roughly describe the positions and velocities

of the center-of-mass in the sagittal plane, whereas y_3 and y_6 correspond to the torso orientation and angular velocity respectively (Figure 4.4(b)). For the manipulator, v_3 and v_4 correspond to the torques applied at joints 3 and 4 respectively, whereas v_1 and v_2 are a combination of torques at joints 1 and 2 (Figure 4.4(c)). the state variable mapping is difficult to interpret (Figure 4.4(d)). The mappings for the quadcopter are quite intuitive. Transformed inputs v_2, v_1, v_3 and v_4 are the scaled net thrust and differential yaw, pitch and roll thrusts respectively (Figure 4.4(e)). Furthermore, transformed state variables can be split into four sets $\{y_1, y_{10}\}$, $\{y_2, y_7\}$, $\{y_3, y_5, y_9\}$ and $\{y_4, y_6, y_8\}$ corresponding to the system's yaw, altitude, pitch and roll descriptors (Figure 4.4(f)).

We search for promising hierarchies in the original and transformed system representations using our GA based search (Section 3.1.1). We compute the hierarchical policies using Policy Iteration [10] where the policies are look-up tables over the appropriate subspaces of the state-space. Additionally, as baselines to compare performance against, we compute policies using Proximal Policy Optimization (PPO) [80], and Advantage Actor Critic (A2C) [65]. For PPO and A2C, we use the SB3 implementation [75], and the policies are feedforward neural networks with rectified linear unit activations. These neural network policies have two hidden layers of sizes $[128, 128]$, $[512, 512]$ and $[256, 256]$ for the biped, the manipulator and the quadcopter respectively. The PPO and A2C policies are trained for 20 million steps. At the end of training, the policy parameters (neural network weights) corresponding to the best performing policy are used for comparison.

To compare the closed-loop performance, we compute 50 trajectories for each system starting from different initial states with the different policies and note the number of trajectories that converge to the goal state (Table 4.2, column 5). Note that, for the quadcopter, decomposing policy computation in the original system representation leads to unstable closed-loop behavior (none of the trajectories converge to the goal) whereas the policy computation in the transformed representation is readily decomposable and results in a stable closed-loop system (all trajectories converge to the goal). Furthermore, we estimate the value function for the different policies using costs of the trajectories that converge to the goal state. We use the normalized value function error,

$$\left(\boldsymbol{V}^{\boldsymbol{\delta}_{(\boldsymbol{y},\boldsymbol{v})}} - \boldsymbol{V}\right) / \boldsymbol{V}^{\boldsymbol{\delta}_{(\boldsymbol{y},\boldsymbol{v})}} \tag{4.3}$$

to quantify the relative quality of a policy, with value function estimate $\boldsymbol{V}$, in comparison

to the decomposition policy derived from the transformed system representation, whose value function estimate we term $\boldsymbol{V}^{\boldsymbol{\delta}_{(\boldsymbol{y},\boldsymbol{v})}}$. More negative the error, relatively more optimal the decomposition policy. As can be seen in table 4.2, columns 2, 3 and 4, for all three systems the normalized value function error is negative, indicating that the policy derived by decomposing the transformed system representation provides lower closed-loop trajectory costs, and thus is more optimal, than the hierarchical policy obtained from the original representation as well as the policies computed using A2C and PPO.

Table 4.2: Fifty trajectories of the closed-loop system under the policies obtained from decompositions δ and $\delta_{(\boldsymbol{y},\boldsymbol{v})}$, and state-of-the-art reinforcement learning algorithms are computed. Value function estimates are derived from trajectories that converge to the goal state.

system	**normalized value function error**			**#trajectories converged**
	δ	A2C	PPO	δ / $\delta_{(y,v)}$ / A2C / PPO
Biped	-0.29 ± 0.49	-0.39 ± 0.61	-0.08 ± 0.35	43 / 39 / 42 / 50
Manipulator	-0.3 ± 0.58	-1 ± 1.16	-0.39 ± 0.33	50 / 50 / 38 / 49
Quadcopter	-	-1.14 ± 1.34	-1.47 ± 1.25	0 / 50 / 46 / 50

Here, we presented a heuristic to automatically discover system representations that are more suited for policy decomposition. Our approach is based on the intuition that sparsity in the variation of the optimal policy is likely to lead to hierarchies with lower suboptimalities (or value-errors). Building on this intuition we construct linear state and input mappings from a known system representation to one that induces sparsity in the Jacobian of the optimal policy. For more than 90% of systems with linear dynamics, our strategy produces representations that lead to hierarchies with lower value-errors. For the balancing control of a simplified biped, the swing-up control of a planar manipulator and hover control of a quadcopter, hierarchies discovered with the transformed system representation result in reduced trajectory costs. Furthermore, the hierarchical policies, represented as lookup tables over the state-space, produce trajectories with substantially lower costs compared to neural network policies derived from A2C and PPO. Although, representing policies as lookup tables provides convergence guarantees [78], it becomes difficult with increasing dimensionality. And although neural network policies converge to at best locally optimal solutions [43], they scale well to more complex systems [81]. In chapter 5, we discuss how the Policy Decomposition framework can be applied towards training neural network policies for optimal control.

Chapter 5

Extension to Optimal Trajectory Tracking Control

In chapter 2, we developed the theory for the Policy Decomposition framework applied to optimal regulation problems where the control objective is to drive the system to a fixed state. Here, we present how the theory can be readily extended to address optimal trajectory tracking problems, where the objective is to track a time-indexed sequence of desired states. Regulation problems are in fact a special case where the trajectory is a sequence of the same fixed state. In case of trajectory tracking control, the optimal policy is not just a function of the state of the system but also time. A natural extension then is to have hierarchical policies with individual sub-policies being functions of subspaces of the state of the system and *time* (Figure 5.1). Additionally, we have so far used look-up tables to represent policies and Policy Iteration [10] to compute them, which do not scale well to complex problems. In this chapter, we discuss how Policy Decomposition can be applied towards training neural network policies using actor-critic methods [80], which have been demonstrably effective in high-dimensional state-spaces [81].

This chapter is organized similar to chapter 2 to highlight only the key modifications to the theory of Policy Decomposition for trajectory tracking control. In section 5.1, we formally state the problem. In section 5.2 we discuss how the suboptimality estimates can be derived for this setting. In section 5.3, we discuss how neural network policies can be trained in a hierarchical fashion in-line with the Policy Decomposition framework for optimal trajectory tracking control.

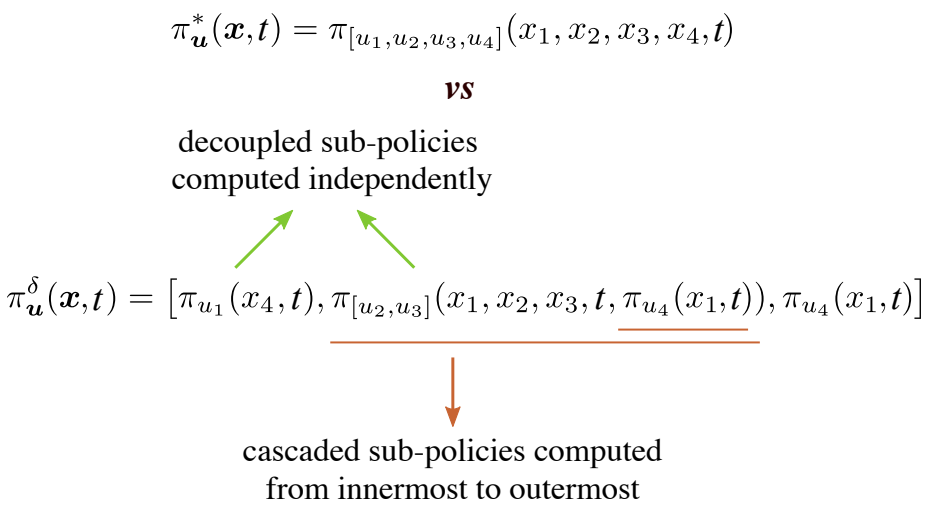


Figure 5.1: Policy Decomposition of a fictive system for trajectory tracking control. The optimal policy and the hierarchical policies are now functions of the system state and time.

5.1 Core Idea

We consider the finite horizon optimal trajectory tracking problem where an optimal policy $\pi^*_{\boldsymbol{u}}(\boldsymbol{x},t)$ that minimizes

$$\begin{aligned} J &= \int_0^{t_f} e^{-\lambda t} c(\boldsymbol{x},\boldsymbol{u},t)\, dt. \\ \text{where } \; c(\boldsymbol{x},\boldsymbol{u},t) &= \left(\boldsymbol{x}-\boldsymbol{x}^d(t)\right)^T \boldsymbol{Q}(t) \left(\boldsymbol{x}-\boldsymbol{x}^d(t)\right) \\ &\quad + \left(\boldsymbol{u}-\boldsymbol{u}^d(t)\right)^T \boldsymbol{R}(t) \left(\boldsymbol{u}-\boldsymbol{u}^d(t)\right) \end{aligned} \tag{5.1}$$

is to be computed. Here $\boldsymbol{x}^d(t)$ and $\boldsymbol{u}^d(t)$ are the desired state and input trajectories to track, $\boldsymbol{Q}$ and $\boldsymbol{R}$ are the weighting matrices, and λ trades-off between the current and future tracking costs.

Policy Decomposition computes sub-policies for subsets of inputs as functions of some subset of the system's state variables; except now the sub-policies are also a function of time. These sub-policies together form the hierarchical policy $\pi^{\delta}_{\boldsymbol{u}}(\boldsymbol{x},t)$ for the entire system. The sub-policies are computed by optimizing the behavior of subsystems, with a

time-varying control objective

$$\dot{\boldsymbol{x}}_i = \boldsymbol{f}_i(\boldsymbol{x}_i, \boldsymbol{u}_i, t \mid \bar{\boldsymbol{x}}_i = \bar{\boldsymbol{x}}_i^d(t), \bar{\boldsymbol{u}}_i = \pi_{\bar{\boldsymbol{u}}_i}(\boldsymbol{x}_i, t)),$$
$$c_i(\boldsymbol{x}_i, \boldsymbol{u}_i, t) = c(\boldsymbol{x}_i, \boldsymbol{u}_i, t \mid \bar{\boldsymbol{x}}_i = \bar{\boldsymbol{x}}_i^d(t), \bar{\boldsymbol{u}}_i = \pi_{\bar{\boldsymbol{u}}_i}(\boldsymbol{x}_i, t))$$

Input-trees, as described in section 2.1, can be used as is to encode a hierarchy.

5.2 Quantifying the Suboptimality of a Hierarchy

Since the optimal and the hierarchical polices are now time-varying, so are the corresponding value functions. Therefore, the value-error to quantify the suboptimality of the hierarchical policies is modified to

$$\text{err}^{\delta} = \frac{\int_0^{t_f} \int_{\mathcal{S}} \left(V^{\delta}(\boldsymbol{x}, t) - V^*(\boldsymbol{x}, t)\right) \, d\boldsymbol{x} \, dt}{\int_0^{t_f} \int_{\mathcal{S}} V^*(\boldsymbol{x}, t) \, d\boldsymbol{x} \, dt} \tag{5.2}$$

5.2.1 The Time-varying LQR Estimate

For the LQR estimate, we earlier adopted a time-invariant infinite horizon formulation (Section 2.2.1). For trajectory tracking control we consider the finite horizon time-varying formulation. Similar to the time-invariant case, we construct linearized dynamics of the system

$$\boldsymbol{A}(t) = \left.\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}}\right|_{\left(\boldsymbol{x}^d(t), \boldsymbol{u}^d(t)\right)}, \quad \boldsymbol{B}(t) = \left.\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{u}}\right|_{\left(\boldsymbol{x}^d(t), \boldsymbol{u}^d(t)\right)}$$

The time-varying LQR estimate of the optimal value function then becomes

$$\begin{aligned}
V^*_{\text{lqr}}(\boldsymbol{x}, t) &= \left(\boldsymbol{x} - \boldsymbol{x}^d(t)\right)^T \boldsymbol{P}(t) \left(\boldsymbol{x} - \boldsymbol{x}^d(t)\right) \\
\pi^*_{\boldsymbol{u}_{\text{lqr}}}(\boldsymbol{x}, t) &= \boldsymbol{u}^d(t) - \boldsymbol{K}^*(t) \left(\boldsymbol{x} - \boldsymbol{x}^d(t)\right), \\
\text{where} \quad \boldsymbol{K}^*(t) &= \boldsymbol{R}^{-1} \boldsymbol{B}(t)^T \boldsymbol{P}(t) \\
-\dot{\boldsymbol{P}}(t) &= \boldsymbol{Q} + \left(\boldsymbol{A}(t) - \frac{\lambda}{2}\boldsymbol{I}\right)^T \boldsymbol{P}(t) + \boldsymbol{P}(t) \left(\boldsymbol{A}(t) - \frac{\lambda}{2}\boldsymbol{I}\right) \\
&\quad - \boldsymbol{P}(t) \boldsymbol{B}(t) \boldsymbol{R}^{-1} \boldsymbol{B}^T(t) \boldsymbol{P}(t), \text{ s.t. } \boldsymbol{P}(t_f) = \boldsymbol{Q}(t_f)
\end{aligned} \tag{5.3}$$

To obtain $V^{\delta}_{\text{lqr}}(\boldsymbol{x}, t)$, we first compute the hierarchical policy for the linearized system. Similar to the time-invariant case, the hierarchical policy is

$$\pi^{\delta}_{\boldsymbol{u}_{\text{lqr}}}(\boldsymbol{x}, t) = \boldsymbol{u}^d(t) - \boldsymbol{K}^{\delta}(t)\left(\boldsymbol{x} - \boldsymbol{x}^d(t)\right)$$

where $\boldsymbol{K}^{\delta}(t)$ is a time-varying block matrix composed of linear sub-policies derived from time-varying LQR solutions for the corresponding linear subsystems

$$\begin{aligned}
\boldsymbol{A}_i(t) &= \left.\frac{\partial \boldsymbol{f}_i}{\partial \boldsymbol{x}_i}\right|_{\left(\boldsymbol{x}^d(t), \boldsymbol{u}^d(t)\right)} + \left.\frac{\partial \boldsymbol{f}_i}{\partial \bar{\boldsymbol{u}}_i}\right|_{\left(\boldsymbol{x}^d(t), \boldsymbol{u}^d(t)\right)} \left.\frac{\partial \pi_{\bar{\boldsymbol{u}}_i}}{\partial \boldsymbol{x}_i}\right|_{\boldsymbol{x}_i^d(t)}, \\
\boldsymbol{B}_i(t) &= \left.\frac{\partial \boldsymbol{f}_i}{\partial \boldsymbol{u}_i}\right|_{\left(\boldsymbol{x}^d(t), \boldsymbol{u}^d(t)\right)}
\end{aligned}$$

where $\pi_{\bar{\boldsymbol{u}}_i}(\boldsymbol{x}_i, t) = [0, \ldots, 0, \boldsymbol{u}_j^d(t) - \boldsymbol{K}_j(t)(\boldsymbol{x}_j - \boldsymbol{x}_j^d(t)), 0, \ldots, 0]$. $\boldsymbol{K}_j(t)$ is the LQR solution for the subsystems in cascade with the i^{th} subsystem. The time-varying LQR solution for the i^{th} subsystem is then characterized by $\boldsymbol{A}_i(t)$, $\boldsymbol{B}_i(t)$ and cost

$$\begin{aligned}
c_i(\boldsymbol{x}_i, \boldsymbol{u}_i, t) = &\left(\boldsymbol{u}_i - \boldsymbol{u}_i^d(t)\right)^T \boldsymbol{R}_i(t) \left(\boldsymbol{u}_i - \boldsymbol{u}_i^d(t)\right) \\
&+ \left(\boldsymbol{x}_i - \boldsymbol{x}_i^d(t)\right)^T \left(\boldsymbol{Q}_i(t) + \begin{bmatrix} 0 & \cdots & 0 \\ \vdots & \boldsymbol{K}_j(t) & \vdots \\ 0 & \cdots & 0 \end{bmatrix}^T \bar{\boldsymbol{R}}_i(t) \begin{bmatrix} 0 & \cdots & 0 \\ \vdots & \boldsymbol{K}_j(t) & \vdots \\ 0 & \cdots & 0 \end{bmatrix}\right)\left(\boldsymbol{x}_i - \boldsymbol{x}_i^d(t)\right)
\end{aligned}$$

where $\boldsymbol{Q}_i(t)$ is the appropriate sub-matrix of the original cost function matrix $\boldsymbol{Q}(t)$, and $\boldsymbol{R}_i(t)$ and $\bar{\boldsymbol{R}}_i(t)$ are sub-matrices of $\boldsymbol{R}(t)$ corresponding to $\boldsymbol{u}_i$ and $\bar{\boldsymbol{u}}_i$ respectively.

The value function estimate for the hierarchical policy then resolves to

$$\begin{aligned}
V^{\delta}_{\text{lqr}}(\boldsymbol{x}, t) &= \left(\boldsymbol{x} - \boldsymbol{x}^d(t)\right)^T \boldsymbol{P}^{\delta}(t) \left(\boldsymbol{x} - \boldsymbol{x}^d(t)\right) \\
-\dot{\boldsymbol{P}}^{\delta} = \boldsymbol{Q} + \boldsymbol{K}^{\delta}(t)^T \boldsymbol{R} \boldsymbol{K}^{\delta}(t) &+ \left(\boldsymbol{A}(t) - \boldsymbol{B}(t)\boldsymbol{K}^{\delta}(t) - \frac{\lambda}{2}\boldsymbol{I}\right)^T \boldsymbol{P}^{\delta}(t) \\
&+ \boldsymbol{P}^{\delta}(t)\left(\boldsymbol{A}(t) - \boldsymbol{B}(t)\boldsymbol{K}^{\delta}(t) - \frac{\lambda}{2}\boldsymbol{I}\right), \text{ where } \boldsymbol{P}^{\delta}(t_f) = \boldsymbol{Q}
\end{aligned} \tag{5.4}$$

Algorithm 3 OnPolicyActorCritic($\text{System}_{\text{info}}$, $\text{step}_{\max}$, rollout_length, $\text{epoch}_{\max}$, $\text{batch}_{\max}$)

step $\leftarrow 0$
buffer $\leftarrow [\ ]$
while step $<$ $\text{step}_{\max}$ **do**
    $\boldsymbol{a} \leftarrow \boldsymbol{\pi}_{\boldsymbol{\theta}}(\boldsymbol{s})$ ▷ forward pass actor
    $\boldsymbol{s}', \boldsymbol{r} \leftarrow \text{simulate}(\text{System}_{\text{info}}, \boldsymbol{a})$ ▷ step
    $\boldsymbol{v}' \leftarrow \boldsymbol{V}_{\boldsymbol{\phi}}(\boldsymbol{s}')$ ▷ forward pass critic
    buffer.append($\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{r}, \boldsymbol{s}', \boldsymbol{v}'$)
    step $\leftarrow$ step + 1
    if Modulo(step, rollout_length) == 0 **then**
        for epoch, batch = 1, 1 to $\text{epoch}_{\max}$, $\text{batch}_{\max}$ **do**
            loss, gradient $\leftarrow$ compute_loss(buffer, batch) ▷ backward pass
            $\boldsymbol{\pi}_{\boldsymbol{\theta}}, \boldsymbol{V}_{\boldsymbol{\phi}} \leftarrow$ update($\boldsymbol{\pi}_{\boldsymbol{\theta}}, \boldsymbol{V}_{\boldsymbol{\phi}}$, gradient)
        end for
        buffer $\leftarrow [\ ]$
    end if
end while

5.2.2 The Time-varying Unscented Estimate

The approach to derive the time-varying unscented estimate is very similar to the time-invariant case described in section 2.2.2. The approximations to the optimal and hierarchical policies are still based on a nearest neighbor look-up with trajectories of the full system and appropriate subsystems respectively (Equation (2.13)). However, the nearest neighbor strategy is now time conditioned

$$s^{\dagger}(t) = \underset{s}{\operatorname{argmin}} \left\| \boldsymbol{X}_i^s(t^{\dagger}) - \boldsymbol{x}_i \right\|_2, \text{ where } t^{\dagger} = \underset{t'}{\operatorname{argmin}} |t' - t| \tag{5.5}$$

and t' are the time instances along the discrete-time trajectories $\boldsymbol{X}_i^s$. The estimates of the value function of the optimal and hierarchical policies are then obtained by rolling out trajectories with these approximated policies.

5.3 Decomposing Neural Policy Optimization

In chapter 3, we analysed the trade-offs between closed-loop performance and reduction in computation in decomposing the policy optimization when the policies were represented

as look-up tables over the state-space and computed using Policy Iteration [50]. Guarantees of convergence to the optimal solution (within discretization resolution) exist, when using look-up tables to represent policies [78]. But, these representations do not scale well in terms of memory footprint and required computation to higher-dimensional problems, making the advantages of Policy Decomposition very apparent. On the other hand, neural networks have been demonstrably effective as policy representations for complex policy optimization problems [46, 80] but are known to at best converge to locally optimal solutions [43]. Here, we apply Policy Decomposition towards training neural network policies and investigate whether the advantages translate.

Algorithm 3 outlines on-policy actor-critic methods [54], a class of algorithms to simultaneously train policies (actor) and value functions (critic) for an optimal control problem. These methods iterate between two phases, 1) monte carlo rollouts with the existing policy to collect interaction data and 2) updating the policy and value function using temporal differences learning [81]. The majority of the computation happens during the forward passes to the actor and critic (Lines 4 and 6: $\mathcal{O}(|\boldsymbol{\theta}| + |\boldsymbol{\phi}|)$), simulating the dynamics of the underlying system (Line 5), and computing gradients (Line 11: $\mathcal{O}(|\boldsymbol{\theta}|+|\boldsymbol{\phi}|)$) to update the policy and value function. Table 5.1 depicts how the time required to train policies scales with increasing size. As the network size grows, the majority of the compute

#hidden units	**time** (sec / 20 million steps)			**steps to converge** (million)
	forward	gradient	simulation	
[256, 256, 256]	49.4	71.5		24.47
[512, 512, 512]	50.8	133.2		18.48
[1024, 1024, 1024]	78.9	438.3	476.1 ± 8.76	35.95
[2048, 2048, 2048]	182.3	2028		24.97
[4096, 4096, 4096]	864.5	8779		25.97

Table 5.1: Breakdown of policy computation times for quadcopter control using Proximal Policy Optimization [80]. The cumulative time required for forward pass (policy and value function inference), and computation of gradients for neural network weights when training networks of different sizes is reported. The time required for simulating system dynamics is also reported for comparison.

effort is spent in the forward pass and the gradient computation which scale linearly with the number of weights. This motivates the use of smaller neural networks to represent sub-policies. We scale the number of hidden units proportional to the dimension of the

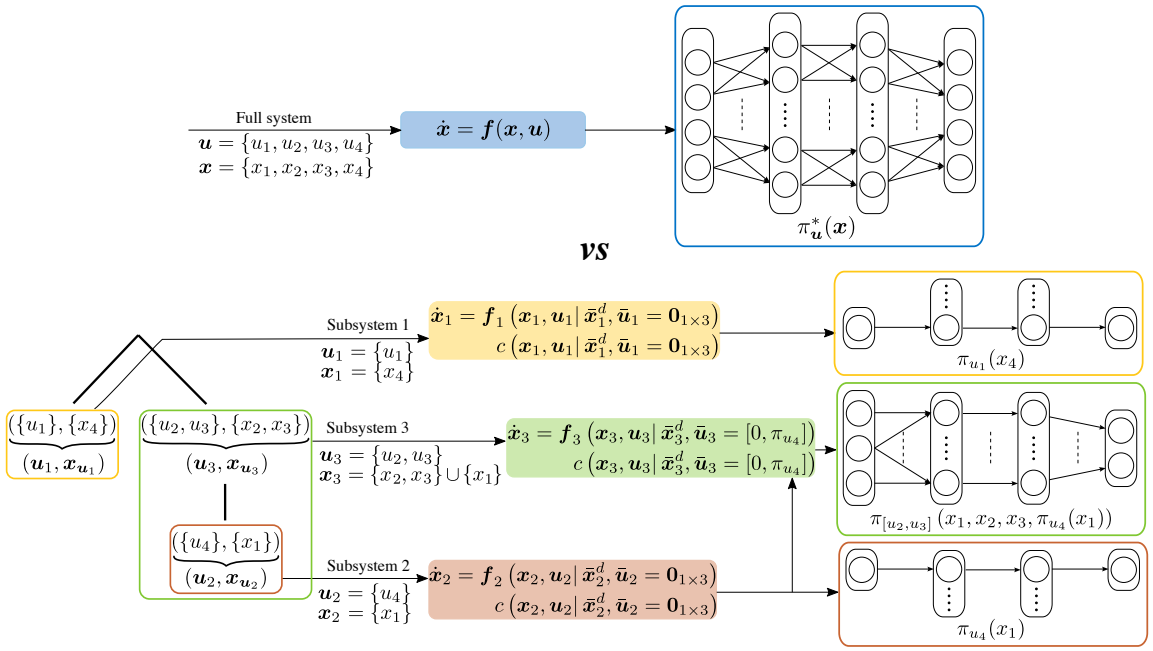


Figure 5.2: The idea of Policy Decomposition extended towards training neural network policies. The optimal policy is represented using a fully-connected neural network, and the sub-policies are represented using networks with reduced hidden units proportional to the dimensionality of the input and state-space for the subsystem.

input and state subspaces of a subsystem, in the neural network sub-policies (Figure 5.2).

$$\underbrace{[h_1, \cdots, h_N]}_{\#\text{hidden units in the full policy}} \rightarrow \underbrace{[h_1, \cdots, h_N] \times \frac{\dim(\boldsymbol{x}_i) \times \dim(\boldsymbol{u}_i)}{\dim(\boldsymbol{x}) \times \dim(\boldsymbol{u})}}_{\#\text{hidden units in the } i^{\text{th}} \text{ sub-policy}} \tag{5.6}$$

The estimates of reduction in policy computation with this scheme are derived in appendix A.2.2.

5.4 Results

We design policies for the quadcopter to track trajectories shown in figure 5.5. In figure 5.6, hierarchies discovered by GA for the optimally tracking the corresponding trajectories are depicted. For constant hover control, the yaw control is decoupled from the rest of the system and the pitch control is optimized in cascade with thrust and roll (Figure 5.6(a)). Tracking the sinusoid trajectory requires a more tightly coupled controller structure, the thrust, roll and pitch policies are jointly optimized first followed by the yaw policy as a

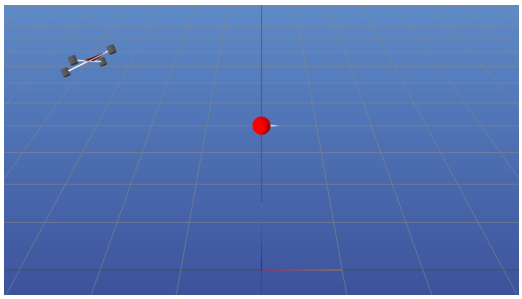


Figure 5.3: Constant hover

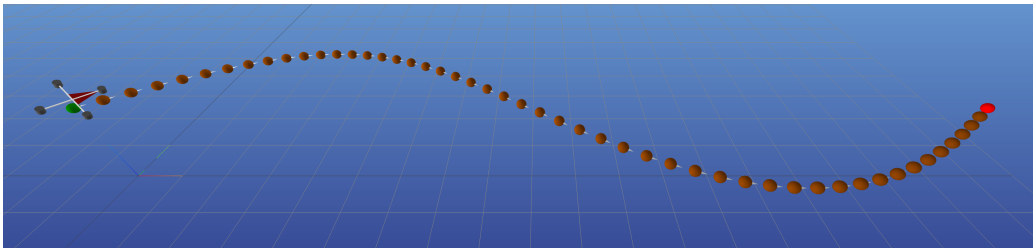
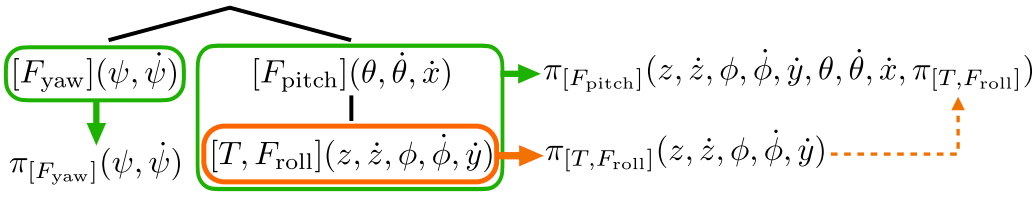
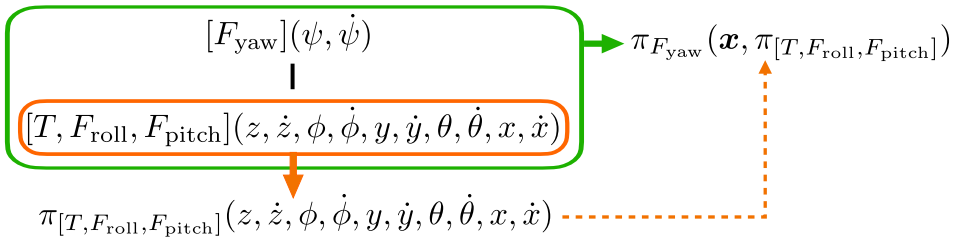


Figure 5.4: Sinusoid

Figure 5.5: Candidate trajectories for tracking control of the quadcopter



(a) Constant hover



(b) Sinusoid

Figure 5.6: Hierarchies discovered by GA for optimal trajectory tracking for the quadcopter for the corresponding trajectories depicted in figure 5.5

function of the full state (Figure 5.6(b)).

Using PPO, we train neural networks of different widths to approximate the optimal policy and proportionally scaled smaller networks for the hierarchical policies. We test and track the performance of the policies during the training process by rolling out trajectories

and computing their costs. These are depicted in Figures 5.7 and 5.8. We observe that the hierarchical policies require more environment interactions to train in comparison to the full policy. However, for larger neural network sizes, i.e. in the regime where neural network inference and gradient estimation require more computation than simulating the environment, the hierarchical policies, which employ smaller networks, require less wall-clock time to train.

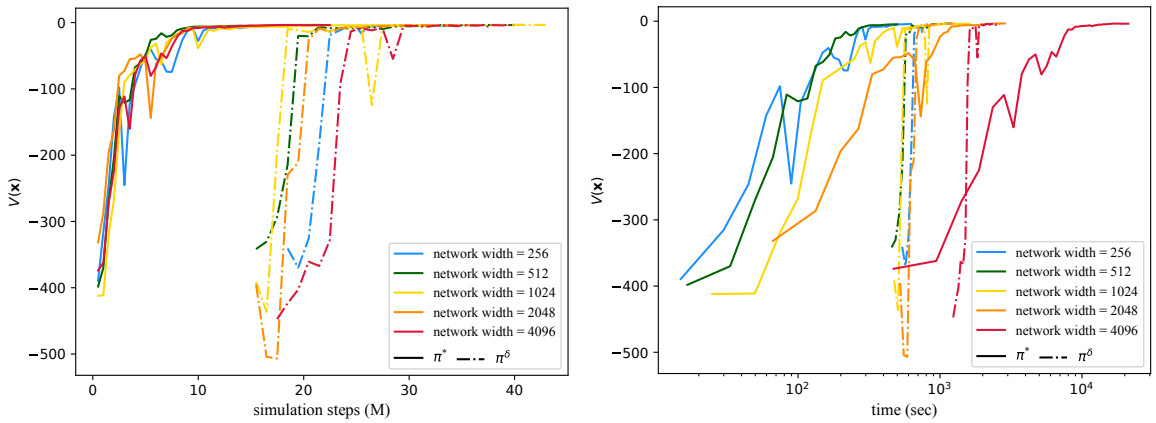


Figure 5.7: Performance trend of the jointly optimized and hierarchical policies for **constant hover** control of the quadcopter. The policies are optimized using PPO and trends for neural network policies of different widths is shown.

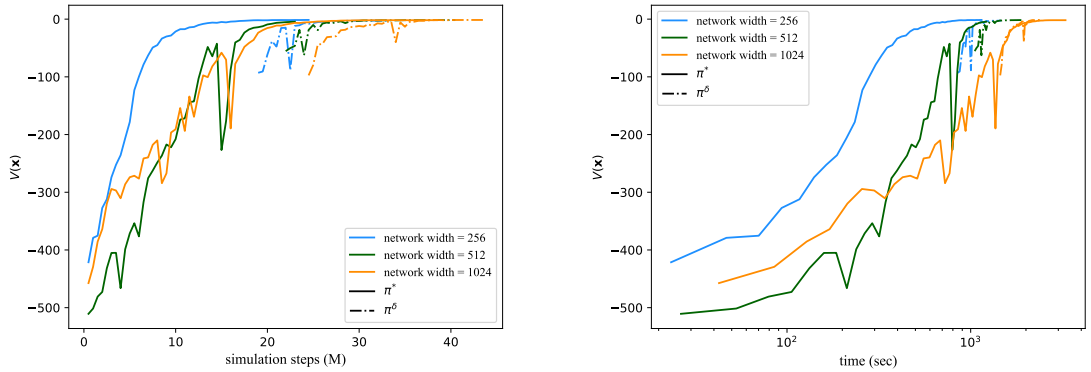


Figure 5.8: Performance trend of the jointly optimized and hierarchical policies for tracking a **sinusoid** trajectory with the quadcopter. The policies are optimized using PPO and trends for neural network policies of different widths is shown.

Chapter 6

Conclusions and Future Work

We introduced a novel framework for approximately solving optimal control problems of complex dynamical systems. Policy Decomposition stands out from other hierarchical control methods by automatically proposing candidate hierarchies with minimal assumptions about the underlying system and by providing *a priori* estimates of suboptimality for the resulting hierarchical policies. Furthermore, we demonstrated how Genetic Algorithm and Monte Carlo tree search can be adapted to efficiently discover hierarchies that can sharply reduce the computational cost without giving up much on closed-loop performance. Additionally, we investigated how the choice of system representation affects the suitability for hierarchical policy optimization and presented an approach to construct representations more amenable to hierarchical control.

We applied Policy Decomposition to identify control hierarchies for a range of robotic systems, including planar manipulators of varying complexity, a simplified biped and a quadcopter, highlighting the generality of the framework. The discovered hierarchies either outperform heuristically constructed ones in closed-loop performance or provide dramatic reductions in required compute but marginally worse control. Furthermore, Policy Decomposition is agnostic to the choice of policy representation and optimization algorithm. We showcase hierarchical policy optimization using Policy Iteration with look-up table based policies as well as using more modern methods such as Proximal Policy Optimization with neural network policies. The computational resources for this work were provided by the Advanced Cyberinfrastructure Coordination Ecosystem: Services and Support (ACCESS) program [16].

Two future research directions would further broaden the utility of the policy decomposition framework. First, to identify system representations more amenable to hierarchical control. In chapter 4, we discussed the tight coupling between system representations and resulting hierarchies and posited that representations that induce sparsity in the optimal policy are more conducive to decomposition. A more through investigation and a more general approach would further broaden the class of control problems that can be tackled with this framework. Second, we assumed a noiseless system, and perfect knowledge of the dynamics. These assumptions almost never hold for realistic systems. Almost every actuator is noisy with some uncertainty in the applied output, and dynamics parameters for the system are usually only known only up to a certain level of accuracy. Incorporating these uncertainties in the Policy Decomposition framework, specifically, in evaluating different hierarchies would provide more realistic estimates of control performance.

Appendix A

Search Methods

A.1 Hierarchy Count

Any hierarchy for a system can be represented using a unique input-tree, and therefore the number of possible hierarchies equals the number of valid input-trees. For a system with n state variables and m control inputs, a valid input-tree has at most m nodes (excluding the root) and can be constructed as follows

(I) Group the m inputs into $r \in \{2, \cdots, m\}$ non-empty subsets.

(II) Arrange the r subsets of inputs into input-trees such that $k \in \{1, \cdots, r\}$ of these are leaf-nodes[1].

(III) Distribute the n state variables into r groups (one for each input subset) such that the k groups corresponding to the leaf nodes are non-empty.

We first enumerate $\mathbf{I}(m, r)$, $\mathbf{II}(r, k)$ and $\mathbf{III}(n, r, k)$. Subsequently, the number of possible hierarchies is

$$\sum_{r=2}^{m} \mathbf{I}(m, r) \sum_{k=1}^{r} \mathbf{II}(r, k)\mathbf{III}(n, r, k) \tag{A.1}$$

(I) m inputs can be divided into r non-empty subsets in

$$\mathbf{I}(m, r) = \frac{\Delta(m, r)}{r!} \tag{A.2}$$

[1]Leaf-nodes are nodes that do not have children.

ways, where $\Delta(m, r)$ is as follows

$$\Delta(m,r) = r^m - \binom{r}{1}(r-1)^m + \binom{r}{2}(r-2)^m + \cdots + (-1)^{r-1}\binom{r}{r-1}(1)^m \tag{A.3}$$

r^m denotes the number of possibilities of assigning m inputs to r distinct subsets but some of the subsets can be empty. Therefore we subtract the cases where at least one subset is empty i.e. $\binom{r}{1}(r-1)^m$. But we end up subtracting the cases where at least two subsets are empty twice, therefore we add these back i.e. $\binom{r}{2}(r-2)^m$. Continuing with this reasoning we arrive at Eq. (A.3). In Eq. A.2, division by $r!$ is to account for repetitions from treating the subsets as distinct [86].

(II) We first select k of the r input subsets to assign to the k leaf-nodes of the input-tree. There are $\binom{r}{k}$ possible ways to make this selection. To enumerate the different possible structures of an input-tree we make use of Prüfer codes [14]. Every tree consisting of N labelled nodes can be represented using a unique $(N-2)$ character long sequence of these labels. The sequence can have repetitions. Furthermore, this mapping is a bijection, i.e. every sequence of size $(N-2)$ corresponds to a unique valid tree structure. Additionally, the **only missing characters** in a Prüfer code **are labels of the leaf-nodes**. An input-tree for r input subsets consists of $N = (r+1)$ nodes and has an encoding of length $(r-1)$. If k of the input subsets are leaf-nodes then the encodings have either $(r-k)$ or $(r-k+1)$ distinct characters, depending on whether the root node is or is not a leaf-node.

   (a) If the root node **is** also a leaf-node, the encoding has to be generated by only using labels of the $(r-k)$ remaining input subsets, each of which must appear at least once. This is equivalent to assigning the $(r-1)$ spots in the sequence to $(r-k)$ non-empty groups $\big(\Delta(r-1, r-k)$, from Eq. (A.3)$\big)$.

   (b) If the root node **is not** a leaf-node, the $(r-1)$ length encoding must consist of all the remaining $(r-k+1)$ labels, $\Delta(r-1, r-k+1)$ possibilities.

Thus number of possible input-trees with r input subsets and k leaf-nodes are[2]

$$\mathbf{II}(r,k) = \binom{r}{k}\Big(\Delta\big(r-1, r-k\big) + \Delta\big(r-1, r-k+1\big)\Big) \tag{A.4}$$

[2]When $k = 1$, $\mathbf{II}(r,k) = \binom{r}{k}\Delta(r-1, r-k)$ whereas when $k = r$, $\mathbf{II}(r,k) = \binom{r}{k}\Delta(r-1, r-k+1)$

(III) A valid state assignment for an input-tree consisting of r input subsets and k leaf-nodes can be generated by selecting $i \in \{0, \cdots, n-k\}$ of the n variables, assigning them to the $r-k$ non-leaf-nodes, and distributing the remaining $(n-i)$ variables into k non-empty groups corresponding to the k leaf-nodes. Possible number of valid state-assignments are

$$\mathbf{III}(n, r, k) = \sum_{i=0}^{n-k} \binom{n}{i} \Delta(n-i, k)(r-k)^i \tag{A.5}$$

A.2 Estimates of reduction in computation from Policy Decomposition

A.2.1 Policy Iteration

Here, we calculate the potential reduction in floating point operations offered by a hierarchy when policies are represented as look-up tables and are optimized using Policy Iteration (PI) [10]. A look-up table based policy stores control values corresponding to a uniform grid over the state-space. For a hierarchical policy, the individual subpolicies are look-up tables over appropriate subspaces of the state-space, and are derived by applying PI to the corresponding subsystem.

PI starts with a randomly initialized policy and iterates through the evaluation and improvement phases to converge to the optimal policy. In the evaluation phase an estimate of the value function for the current policy is derived by applying the policy at every state in the grid, estimating the value function at the subsequent states through interpolation, and updating the value function at grid points using the Bellman equation

$$V^{(\text{iter}+1)}(\boldsymbol{x}) = c(\boldsymbol{x}, \pi_{\boldsymbol{u}}(\boldsymbol{x})) + V^{\text{iter}}\left(\boldsymbol{x} + dt\, \boldsymbol{f}(\boldsymbol{x}, \pi_{\boldsymbol{u}}(\boldsymbol{x}))\right)$$

$$\#\text{operations}_{\text{evaluate}} = \prod_{i=1}^{n} \text{NS}_i \Big(\underbrace{2^n \text{ME}}_{\text{interpolate}} + \text{FLOP}_{\text{simulate}} \Big)$$

where n is the number of state variables, NS_i is the size of the look-up table along state dimension i, ME is the maximum number of evaluation iterations, and $\text{FLOP}_{\text{simulate}}$ is the number of floating point operations required to simulate one time-step of the system. The

improvement phase involves sampling NA_j candidate values for the j^{th} control input and greedily updating the policy based on the current value function estimates

$$\pi_{\boldsymbol{u}}^{(\text{iter}+1)}(\boldsymbol{x}) = \underset{\boldsymbol{u}\in[\boldsymbol{u}_{\min},\boldsymbol{u}_{\max}]}{\operatorname{argmin}} \left(c(\boldsymbol{x},\boldsymbol{u}) + V(\boldsymbol{x} + dt\,\boldsymbol{f}(\boldsymbol{x},\boldsymbol{u}))\right)$$

$$\#\text{operations}_{\text{update}} = \prod_{i=1}^{n} \text{NS}_i \prod_{j=1}^{m} \text{NA}_j \left(\underbrace{2^n}_{\text{interpolate}} + \text{FLOP}_{\text{simulate}} + \underbrace{3}_{\substack{\text{sample action+} \\ \text{update value}}} \right)$$

where $\boldsymbol{u}_{\min}$ and $\boldsymbol{u}_{\max}$ are control bounds and m is the number of control inputs. Estimate for total maximum operations is

$$\#\text{operations}_{\text{total}} = \text{M}\left(\#\text{operations}_{\text{evaluate}} + \#\text{operations}_{\text{update}}\right)$$

where M is the maximum iterations for PI. The estimated reduction in policy computation is the ratio of $\#\text{operations}_{\text{total}}$ for hierarchy δ, and for the optimal policy.

A.2.2 Proximal Policy Optimization

Here, we report our strategy to estimate the reduction in computation when deriving neural network policies using Policy Decomposition. We use a fully connected architecture for the policies and the value function, and compute them with the Proximal Policy Optimization (PPO) [80] algorithm. The architectures for the individual subpolicies in a hierarchy are derived by proportionally scaling down the number of hidden units in the optimal policy by dimensionality of the state and input subspaces for the corresponding subsystem. Therefore, the number of weights for the subpolicies are

$$|w_{\pi_{\boldsymbol{u}_i}}| = |w_{\pi^*}| \times \left(\frac{\dim(\boldsymbol{x}_i)}{\dim(\boldsymbol{x})}\right)^2 \times \left(\frac{\dim(\boldsymbol{u}_i)}{\dim(\boldsymbol{u})}\right)^2 \tag{A.6}$$

As described in algorithm 3, PPO iterates between applying the current policy to collect trajectories and updating the policy and value function through stochastic gradient descent (applying gradients for sampled batches of collected data and repeating it for some epochs E). The complexity of the forward pass and backward pass are both $\mathcal{O}(|w_{\pi_{\boldsymbol{u}_i}}| + |w_{V^{\pi_{\boldsymbol{u}_i}}}|)$. If

the algorithm is allowed T time-steps of system interactions, the estimate for the number of operations to optimize the policy for a subsystem turns out to be

$$\#\text{operations}_{\text{rollout}} = \text{T} \times (|w_{\pi_{\boldsymbol{u}_i}}| + |w_{V^{\pi_{\boldsymbol{u}_i}}}| + \text{FLOP}_{\text{simulate}})$$
$$\#\text{operations}_{\text{update}} = \text{T} \times \text{E} \times \left(|w_{\pi_{\boldsymbol{u}_i}}| + |w_{V^{\pi_{\boldsymbol{u}_i}}}|\right)$$

where $\text{FLOP}_{\text{simulate}}$ is the number of floating point operations required to simulate one time-step of the system. The estimated reduction in policy computation is the ratio of $\left(\#\text{operations}_{\text{rollout}} + \#\text{operations}_{\text{update}}\right)$ for hierarchy δ, and for the optimal policy.

A.3 Ablations for Genetic Algorithm based hierarchy search

Table A.1: Ablations for GA with the quadcopter (Figure 3.5(c))

	F_{δ} ($\times 10^{-19}$)	time (sec)	$\#\delta$ found
with hash-map	53.8 ± 12	254.3 ± 85.9	25599 ± 2469
without hash-map	17.8 ± 21	164.7 ± 62.2	89477 ± 2306
0% with crossover	8.95 ± 17.5	226.6 ± 156	90594 ± 2744
25% with crossover	8.95 ± 17.5	260.5 ± 124	80639 ± 3357
50% with crossover	0.18 ± 0	232 ± 188	70258 ± 2481

We evaluate the advantage of using a hash-map by running the GA based search with and without it for the quadcopter system for 600 seconds. Additionally, we apply crossover operator which swaps tree structure and state assignment between compatible input-trees to create a percentage of the next generation of population in every iteration of GA (Algorithm 1). In table A.1, we note the number of unique hierarchies discovered, the lowest fitness found and the time required to discover the hierarchy with that fitness. The search benefits with the use of hash-map as more hierarchies are evaluated and the time required to find the best is reduced. Moreover, the crossover operators do not provide a significant benefit but instead slow down the search (fewer hierarchies found) and so we do not use them.

Appendix B

System Representations

B.1 Singular Value Decomposition Regularization

A singular value decomposition of $\boldsymbol{K}^*$ is $\boldsymbol{U}_{\boldsymbol{K}^*}\boldsymbol{S}_{\boldsymbol{K}^*}\boldsymbol{V}_{\boldsymbol{K}^*}^T$ where $\boldsymbol{U}_{\boldsymbol{K}^*} \in \mathbb{R}^{m\times m}$ and $\boldsymbol{V}_{\boldsymbol{K}^*} \in \mathbb{R}^{n\times n}$ are orthogonal. Based on the singular values of $\boldsymbol{K}^*$ there are three possibilities

- ***Case I*** : If all singular values are unique and non-zero then $\boldsymbol{U}_{\boldsymbol{K}^*}$ and the first m columns of $\boldsymbol{V}_{\boldsymbol{K}^*}$ are uniquely defined. Last $(n-m)$ columns of $\boldsymbol{V}_{\boldsymbol{K}^*}$ span the null-space of $\boldsymbol{K}^*$ and can be arbitrarily chosen such that they are mutually orthogonal and orthogonal to the first m columns.
- ***Case II*** : Subsets of columns of $\boldsymbol{U}_{\boldsymbol{K}^*}$ corresponding to repeated and non-zero singular values span orthogonal subspaces (of dimension > 1) in the range of $\boldsymbol{K}^*$. Each of these subsets of columns can be chosen arbitrarily such that they are mutually orthogonal and orthogonal to the subspaces spanned by the remaining columns. Note that the orthogonality constraints between columns corresponding to different subsets are tied to the subspace they span rather than the choice of the columns (basis) themselves. Consequently, these subsets of columns of $\boldsymbol{U}_{\boldsymbol{K}^*}$ can be solved for in tandem. The first m columns of $\boldsymbol{V}_{\boldsymbol{K}^*}$ can be derived using $\boldsymbol{U}_{\boldsymbol{K}^*}$, $\boldsymbol{S}_{\boldsymbol{K}^*}$ and the first m columns of $\boldsymbol{K}^*$. The last $(n-m)$ columns of $\boldsymbol{V}_{\boldsymbol{K}^*}$ can then be obtained as in ***Case I***.
- ***Case III*** : If some of the singular values are zero the corresponding columns of $\boldsymbol{U}_{\boldsymbol{K}^*}$ can be derived as in ***Case II***. Furthermore, corresponding columns in $\boldsymbol{V}_{\boldsymbol{K}^*}$ belong

to the null-space of $\boldsymbol{K}^*$ and can be derived along with the last $(n-m)$ ones as in ***Case I***.

We solve the following optimization to determine the last $(n-m)$ columns of $\boldsymbol{V}_{\boldsymbol{K}^*}$ in ***Case I*** and ***Case II***

$$\begin{aligned} &\min \sum_{i=m+1}^{n} \|\boldsymbol{V}^i_{\boldsymbol{K}^*}\|_1 \\ \text{s.t. } &(\boldsymbol{V}^i_{\boldsymbol{K}^*})^T \boldsymbol{V}^j_{\boldsymbol{K}^*} = \mathbf{1}_{i==j} \; i,j \in \{m+1,\cdots,n\} \\ &(\boldsymbol{V}^i_{\boldsymbol{K}^*})^T \boldsymbol{V}^k_{\boldsymbol{K}^*} = 0 \; i \in \{m+1,\cdots,n\}, \; k \in \{1,\cdots,m\} \end{aligned}$$

For ***Case II***, a similar optimization is also posed over the subsets of columns of $\boldsymbol{U}_{\boldsymbol{K}^*}$. Intuitively, we find directions that closely align with the original state (or input) basis, $\boldsymbol{x}$ (or $\boldsymbol{u}$), and span an appropriate subspace either in the null-space (or range) of $\boldsymbol{K}^*$. We solve these optimization problems using the method described in [97]. A feasible initialization to the optimization is obtained using [28].

Appendix C

System Descriptions and Hyperparameters

C.1 System Descriptions

The dynamics for all the systems presented in this work can be found in standard dynamics textbooks such as [68], barring the biped shown in Fig. 2.5(c). For the biped the legs are massless and make contact with the ground at fixed locations d_f distance apart. A leg breaks contact if its length exceeds l_0. In contact, legs can exert forces ($F_{l/r}$) and hip torques ($\tau_{l/r}$).

$$
\begin{aligned}
m\ddot{x} &= F_r \cos\alpha_r + \frac{\tau_r}{l_r}\sin\alpha_r + F_l\cos\alpha_l + \frac{\tau_l}{l_l}\sin\alpha_l \\
m\ddot{z} &= F_r \sin\alpha_r - \frac{\tau_r}{l_r}\cos\alpha_r + F_l\sin\alpha_l - \frac{\tau_l}{l_l}\cos\alpha_l - mg \\
I\ddot{\theta} &= \tau_r(1+\frac{d}{l_r}\sin(\alpha_r-\theta)) + F_r d\cos(\alpha_r-\theta) + \tau_l(1+\frac{d}{l_l}\sin(\alpha_l-\theta)) + F_l d\cos(\alpha_l-\theta) \\
\text{where, } l_l &= \sqrt{l_r^2 + d_f^2 + 2l_r d_f\cos\alpha_r}, \text{ and } \alpha_l = \arcsin\frac{l_r\sin\alpha_r}{l_l}
\end{aligned}
\tag{C.1}
$$

C.2 Hyperparameters

All policies are represented as look-up tables over the state-space ($\mathcal{S}_{\text{full}}$) and we use Policy Iteration (PI) [10] to compute them, except for the optimal policies for the 4 degree of freedom manipulator and the quadcopter in chapters 3 and 4 which are approximated using Advantage Actor Critic [65] and Proximal Policy Optimization [80]. For hierarchical policies the individual subpolicies are look-up tables over the appropriate subspace of the state-space. Hyper-parameters for PI as described in appendix A.2.1 are also reported in table C.1. We compute err^{δ}, $\text{err}^{\delta}_{\text{lqr}}$ and $\text{err}^{\delta}_{\text{ddp}}$ over a smaller set $\mathcal{S} \subset \mathcal{S}_{\text{full}}$. The set $\mathcal{S}$ for different systems are reported in table C.2. For $\text{err}^{\delta}_{\text{ddp}}$ calculations, we compute trajectories starting from the corners of set $\mathcal{S}$, over a horizon of $T = 4\text{s}$ with $dt = 1\text{ms}$

In chapter 3, for the 4 link manipulator and the quadcopter, we use the Stable Baselines implementation [75] for Advantage Actor Critic (A2C) [65] and Proximal Policy Optimization (PPO) [80] to approximate the optimal policy using neural networks. We use the RMSProp optimizer with linearly decaying learning rate to train for 20 million steps. The decaying learning rate was crucial to stable policy optimization. We experimented with fully connected neural networks of different sizes, smaller ones with two hidden layers of dimensions $[256, 256]$ for both the systems, and larger ones with three hidden layers of dimensions $[1729, 1729, 1729]$ and $[2700, 2700, 2700]$ for the manipulator and quadcopter respectively. For both systems, the smaller ones led to better policies and we report results

Table C.1: Dynamics, cost function and hyper-parameters for policy computation of all systems presented in chapters 2 to 4

	type	parameters
Cart-pole	Dynamics	$[m_c, m_p] = [5, 1]$kg, $l_p = 0.9$m, $g = 9.81$m/s^2
	Bounds	$\lvert F\rvert \leq 9$N, $\lvert\tau\rvert \leq 9$Nm, $\mathcal{S}_{\text{full}} : x \in [-1.5, 1.5], \dot{x} \in [-3, 3], \theta \in [0, 2\pi], \dot{\theta} \in [-3, 3]$
	Cost	$\boldsymbol{Q} = \text{diag}([25, 0.02, 25, 0.02])$, $\boldsymbol{R} = 10^{-3}\text{diag}([1, 1])$, $\lambda = 3$
	Hyper-parameters	**NS** $= [31, 31, 31, 31]$, **NA** $= [12, 12]$, ME $= 500$, M $= 2000$
Biped	Dynamics	$m = 72$kg, $I = 3.6$kgm^2, $g = 9.81$m/s^2, $l_0 = 1.15$, $d = 0.2$m, $d_f = 0.5$m
	Bounds	$\lvert F_{r/l}\rvert \leq 3mg$ (N), $\lvert\tau_{r/l}\rvert \leq 0.25mgl_0$ (Nm) $\mathcal{S}_{\text{full}} : \begin{matrix} l_r \in [0.85, 1.25], \ (\alpha_r - \pi/2) \in [0, 0.6], \\ \dot{x} \in [-0.3, 0.5], \ \dot{z} \in [-0.5, 1], \ \theta \in [-\pi/8, \pi/8], \ \dot{\theta} \in [-2, 2] \end{matrix}$
	Cost	$\boldsymbol{Q} = \text{diag}([350, 700, 1.5, 1.5, 500, 5])$, $\boldsymbol{R} = 10^{-6}\text{diag}([1, 1, 10, 10])$, $\lambda = 1$
	Hyper-parameters	**NS** $= [13, 13, 14, 19, 14, 21]$, **NA** $= [5, 5, 2, 2]$, ME $= 100$, M $= 2000$
2 DoF Manipulator	Dynamics	$[m_1, m_2] = [1.25, 0.25]$kg, $g = 9.81$m/s^2, $[l_1, l_2] = [0.25, 0.125]$m
	Bounds	$\lvert\tau_1\rvert \leq 5$Nm, $\lvert\tau_2\rvert \leq 0.5$Nm, $\mathcal{S}_{\text{full}} : \theta_1 \in [0, 2\pi], \ \theta_2 \in [-\pi, \pi], \ \dot{\theta}_1, \dot{\theta}_2 \in [-3, 3]$
	Cost	$\boldsymbol{Q} = \text{diag}([1.6, 1.6, 0.12, 0.12])$, $\boldsymbol{R} = \text{diag}([0.003, 0.3])$, $\lambda = 3$
	Hyper-parameters	**NS** $= [31, 31, 31, 31]$, **NA** $= [15, 5]$, ME $= 300$, M $= 3000$
3 DoF Manipulator	Dynamics	$[m_1, m_2, m_3] = [2.75, 0.55, 0.11]$kg, $[l_1, l_2, l_3] = [0.5, 0.25, 0.125]$m, $g = 9.81$m/s^2
	Bounds	$\lvert\tau_1\rvert \leq 16$Nm, $\lvert\tau_2\rvert \leq 7.5$Nm, $\lvert\tau_3\rvert \leq 1$Nm $\mathcal{S}_{\text{full}} : \theta_1 \in [0, 2\pi], \theta_2, \theta_3 \in [-\pi, \pi], \dot{\theta}_1, \dot{\theta}_2, \dot{\theta}_3 \in [-3, 3]$
	Cost	$\boldsymbol{Q} = \text{diag}([1.6, 1.6, 1.6, 0.12, 0.12, 0.12])$, $\boldsymbol{R} = \text{diag}([0.004, 0.04, 0.4])$, $\lambda = 3$
	Hyper-parameters	**NS** $= [17, 17, 17, 13, 13, 13]$, **NA** $= [8, 3, 2]$, ME $= 300$, M $= 3000$
4 DoF Manipulator	Dynamics	$[m_1, m_2, m_3, m_4] = [5.4, 1.8, 0.6, 0.2]$kg, $[l_1, l_2, l_3, l_4] = [0.2, 0.5, 0.25, 0.125]$m, $g = 9.81$m/s^2
	Bounds	$\lvert\tau_1\rvert \leq 24$Nm, $\lvert\tau_2\rvert \leq 15$Nm, $\lvert\tau_3\rvert \leq 7.5$Nm, $\lvert\tau_4\rvert \leq 1$Nm $\mathcal{S}_{\text{full}} : \theta_1 \in [0, 2\pi], \ \theta_2, \theta_3, \theta_4 \in [-\pi, \pi], \ \dot{\theta}_1, \dot{\theta}_2, \dot{\theta}_3, \dot{\theta}_4 \in [-12, 12]$
	Cost	$\boldsymbol{Q} = \text{diag}([4, 4, 4, 4, 0.1, 0.1, 0.1, 0.1])$, $\boldsymbol{R} = \text{diag}([0.002, 0.004, 0.024, 0.1440])$, $\lambda = 3$
	Hyper-parameters	**PI** : **NS** $= [13, 13, 13, 13, 21, 21, 21, 21]$, **NA** $= [12, 8, 6, 6]$, ME $= 300$, M $= 3000$
		A2C : Layers $= [256, 256]$, Learning rate : $0.0015 \xrightarrow{20\text{M}} 0$ **PPO** : Layers $= [256, 256]$, Learning rate : $0.0005 \xrightarrow{20\text{M}} 0$
Quadcopter	Dynamics	$m = 0.5$kg, $I = 10^{-3}\text{diag}([4.86, 4.86, 8.8])$kgm^2, $l = 0.225$m, $k_M = 0.0383$m, $g = 9.81$m/s^2
	Bounds	$T \in [0, 2mg]$ (N), $\lvert F_{\text{roll/pitch}}\rvert \leq 0.25mg$ (N), $\lvert F_{\text{yaw}}\rvert \leq 0.125mg$ (N) $\mathcal{S}_{\text{full}} : \begin{matrix} z \in [0.5, 1.5], \ \phi, \theta \in [-0.7, 0.7], \ \psi \in [-\pi, \pi], \\ \dot{x}, \dot{y} \in [-2, 2], \ \dot{z} \in [-1.5, 1.5], \ \dot{\phi}, \dot{\theta} \in [-6, 6], \ \dot{\psi} \in [-2.5, 2.5] \end{matrix}$
	Cost	$\boldsymbol{Q} = \text{diag}([5, 1\text{e-}3, 1\text{e-}3, 5, 0.5, 0.5, 0.05, 0.075, 0.075, 0.05])$, $\boldsymbol{R} = \text{diag}([2\text{e-}3, 0.01, 0.01, 4\text{e-}3])$, $\lambda = 3$
	Hyper-parameters	**PI** : **NS** $= [7, 7, 7, 35, 7, 7, 7, 11, 11, 35]$, **NA** $= [12, 3, 3, 10]$, ME $= 250$, M $= 2500$
		A2C : Layers $= [256, 256]$, Learning rate : $0.0015 \xrightarrow{20\text{M}} 0$ **PPO** : Layers $= [256, 256]$, Learning rate : $0.0005 \xrightarrow{20\text{M}} 0$

Table C.2: Regions of state-space ($\mathcal{S} \subset \mathcal{S}_{\text{full}}$) over which value errors are computed for different systems (Chapter 2: Figure 2.6, Figure 2.7, Figure 2.8, Figure 2.9 and Chapter 3: Table 3.1)

	$\mathcal{S}$
Cart-pole	$x \in [-0.5, 0.5],\ \dot{x} \in [-1, 1],\ \theta \in [2\pi/3, 4\pi/3],\ \dot{\theta} \in [-1, 1]$
Biped	$l_r \in [0.92, 1],\ (\alpha_r - \pi/2) \in [0.2, 0.3],$ $\dot{x} \in [-0.1, 0.1],\ \dot{z} \in [-0.3, 0.3],\ \theta \in [-0.2, 0.2],\ \dot{\theta} \in [-0.2, 0.2]$
2 DoF Manipulator	$\theta_1 \in [2\pi/3, 4\pi/3],\ \theta_2 \in [-\pi/3, \pi/3],\ \dot{\theta}_1, \dot{\theta}_2 \in [-0.5, 0.5]$
3 DoF Manipulator	$\theta_1 \in [2\pi/3, 4\pi/3],\ \theta_2, \theta_3 \in [-\pi/3, \pi/3],\ \dot{\theta}_1, \dot{\theta}_2, \dot{\theta}_3 \in [-0.5, 0.5]$
4 DoF Manipulator	$\theta_1 \in [3\pi/5, 7\pi/5],\ \theta_2, \theta_3, \theta_4 \in [-2\pi/5, 2\pi/5],\ \dot{\theta}_1, \dot{\theta}_2, \dot{\theta}_3, \dot{\theta}_4 \in [-0.6, 0.6]$
Quadcopter	$z \in [0.7, 1.3],\ \phi, \theta \in [-0.7, 0.7],\ \psi \in [-\pi/4, \pi/4],$ $\dot{x}, \dot{y} \in [-1, 1],\ \dot{z} \in [-0.75, 0.75],\ \dot{\phi}, \dot{\theta} \in [-0.5, 0.5],\ \dot{\psi} \in [-0.25, 0.25]$

Bibliography

[1] Murad Abu-Khalaf and Frank L Lewis. Nearly optimal control laws for nonlinear systems with saturating actuators using a neural network hjb approach. *Automatica*, 41(5):779–791, 2005. 1

[2] Asma Al-Tamimi, Frank L Lewis, and Murad Abu-Khalaf. Discrete-time nonlinear hjb solution using approximate dynamic programming: Convergence proof. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 38(4):943–949, 2008. 1

[3] A. Alleyne, F. Allgöwer, A.D. Ames, et al. *Control for Societal-Scale Challenges: Roadmap 2030*. IEEE Control Systems Society Publication, May 2023. URL https://eprints.whiterose.ac.uk/197454/. © 2023 Institute of Electrical and Electronics Engineers (IEEE). 1

[4] Athanasios C Antoulas. *Approximation of large-scale dynamical systems*. SIAM, 2005. 1

[5] Miguel Castaño Arranz, Wolfgang Birk, and George Nikolakopoulos. A survey on control configuration selection and new challenges in relation to wireless sensor and actuator networks. *IFAC-PapersOnLine*, 50(1):8810–8825, 2017. 2.3, 4

[6] Christopher G Atkeson and Chenggang Liu. Trajectory-based dynamic programming. In *Modeling, simulation and optimization of bipedal walking*, pages 1–15. Springer, Berlin, Heidelberg, 2013. 1, 2.2.2

[7] Jie Bao, Kwong H Chan, Wen Z Zhang, and Peter L Lee. An experimental pairing method for multi-loop control based on passivity. *Journal of Process Control*, 17(10): 787–798, 2007. 1

[8] Richard Bellman. On the theory of dynamic programming. *Proceedings of the National Academy of Sciences*, 38(8):716–719, 1952. doi: 10.1073/pnas.38.8.716. URL https://www.pnas.org/doi/abs/10.1073/pnas.38.8.716. 1

[9] Richard Bellman. Dynamic programming. *Science*, 153(3731):34–37, 1966. 1

[10] Dimitri P Bertsekas. *Dynamic programming and optimal control*, volume 1. Athena Scientific, Belmont, MA, 1995. 1, 2.3, 3.1, 3.1.3.2, 3.3.2, 4.3, 5, A.2.1, C.2

[11] Dimitri P. Bertsekas. Approximate dynamic programming, 2008. 1

[12] Jalaj Bhandari and Daniel Russo. Global optimality guarantees for policy gradient methods. *arXiv preprint arXiv:1906.01786*, 2019. 1

[13] Leonora Bianchi, Marco Dorigo, Luca Maria Gambardella, and Walter J Gutjahr. A survey on metaheuristics for stochastic combinatorial optimization. *Natural Computing*, 8:239–287, 2009. 3

[14] Norman Biggs, E Keith Lloyd, and Robin J Wilson. *Graph Theory, 1736-1936*. Oxford University Press, 1986. 3.1.1.1, II

[15] Christian Blum and Andrea Roli. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Comput. Surv.*, 35(3), September 2003. ISSN 0360-0300. doi: 10.1145/937503.937505. 3

[16] Timothy J. Boerner, Stephen Deems, Thomas R. Furlani, Shelley L. Knuth, and John Towns. Access: Advancing innovation: Nsf's advanced cyberinfrastructure coordination ecosystem: Services & support. In *Practice and Experience in Advanced Research Computing*, PEARC '23, page 173–176, New York, NY, USA, 2023. Association for Computing Machinery. ISBN 9781450399852. doi: 10.1145/3569951.3597559. URL https://doi.org/10.1145/3569951.3597559. 6

[17] Samir Bouabdallah and Roland Siegwart. Full control of a quadrotor. In *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 153–158, 2007. doi: 10.1109/IROS.2007.4399042. (document), 3.1.3.5, 3.8

[18] Bruno Bouzy and Tristan Cazenave. Computer go: An ai oriented survey. *Artificial Intelligence*, 132(1), 2001. ISSN 0004-3702. doi: https://doi.org/10.1016/S0004-3702(01)00127-8. 3

[19] E. Bristol. On a new measure of interaction for multivariable process control. *IEEE Transactions on Automatic Control*, 11(1):133–134, 1966. doi: 10.1109/TAC.1966.1098266. 1, 4

[20] C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton. A survey of monte carlo tree search methods. *IEEE T-CIAIG*, 4(1), 2012. doi: 10.1109/TCIAIG.2012.2186810. 3

[21] C. D. Chapman, K. Saitou, and M. J. Jakiela. Genetic Algorithms as an Approach to Configuration and Topology Design. *J. Mech. Des*, 116(4), Dec. 1994. ISSN 1050-0472. doi: 10.1115/1.2919480. 3

[22] Yu-Ming Chen, Hien Bui, and Michael Posa. Reinforcement learning for reduced-order models of legged robots. In *Accepted to IEEE International Conference on Robotics and Automation (ICRA)*, May 2024. 1

[23] Mung Chiang, Steven H Low, A Robert Calderbank, and John C Doyle. Layering as optimization decomposition: A mathematical theory of network architectures.

Proceedings of the IEEE, 95(1):255–312, 2007. 1

[24] Arthur Conley and Mario E Salgado. Gramian based interaction measure. In *Proceedings of the 39th IEEE Conference on Decision and Control (Cat. No. 00CH37187)*, volume 5, pages 5020–5022. IEEE, 2000. 2.3, 4

[25] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, Third Edition*. The MIT Press, 3rd edition, 2009. ISBN 0262033844. 3.1.1.3

[26] Kalyanmoy Deb. Multi-objective optimisation using evolutionary algorithms: an introduction. In *Multi-objective evolutionary optimisation for product design and manufacturing*. Springer, 2011. 3.2

[27] Marc Peter Deisenroth, Gerhard Neumann, Jan Peters, et al. A survey on policy search for robotics. *Foundations and Trends® in Robotics*, 2(1–2):1–142, 2013. 1

[28] James Demmel and W. Kahan. Accurate singular values of bidiagonal matrices. *SIAM Journal on Scientific and Statistical Computing*, 11(5):873–912, 1990. doi: 10.1137/0911052. URL https://doi.org/10.1137/0911052. B.1

[29] Jared Di Carlo, Patrick M. Wensing, Benjamin Katz, Gerardo Bledt, and Sangbae Kim. Dynamic locomotion in the mit cheetah 3 through convex model-predictive control. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1–9, 2018. doi: 10.1109/IROS.2018.8594448. 1

[30] Randall T. Fawcett, Kereshmeh Afsari, Aaron D. Ames, and Kaveh Akbari Hamed. Toward a data-driven template model for quadrupedal locomotion. *IEEE Robotics and Automation Letters*, 7(3):7636–7643, 2022. doi: 10.1109/LRA.2022.3184007. 1

[31] Siyuan Feng, Eric Whitman, X Xinjilefu, and Christopher G Atkeson. Optimization-based full body control for the darpa robotics challenge. *Journal of field robotics*, 32 (2):293–312, 2015. 1

[32] Shuang Gao and Aditya Mahajan. Optimal control of network-coupled subsystems: Spectral decomposition and low-dimensional solutions. *IEEE Transactions on Control of Network Systems*, 2021. 1

[33] Hartmut Geyer and Uluc Saranli. *Gait Based on the Spring-Loaded Inverted Pendulum*, pages 923–947. Springer Netherlands, Dordrecht, 2019. ISBN 978-94-007-6046-2. doi: 10.1007/978-94-007-6046-2_43. URL https://doi.org/10.1007/978-94-007-6046-2_43. 1

[34] Fred Glover and Manuel Laguna. *Tabu Search*. Springer, Boston, MA, 1998. ISBN 978-1-4613-0303-9. doi: 10.1007/978-1-4613-0303-9_33. 3

[35] Alex Gorodetsky, Sertac Karaman, and Youssef Marzouk. High-dimensional stochastic optimal control using continuous tensor decompositions. *The International Journal of Robotics Research*, 37(2-3):340–377, 2018. 1

[36] Olivier Goury and Christian Duriez. Fast, generic, and reliable control and simulation of soft robots using model order reduction. *IEEE Transactions on Robotics*, 34 (6):1565–1576, 2018. doi: 10.1109/TRO.2018.2861900. 1

[37] Kevin Green, Yesh Godse, Jeremy Dao, Ross Hatton, Alan Fern, and Jonathan Hurst. Learning spring mass locomotion: Guiding policies with a reduced-order model. *IEEE Robotics and Automation Letters*, PP:1–1, 03 2021. doi: 10.1109/LRA.2021.3066833. (document), 1, 2.3, 3.1.3.3, 3.6

[38] Pierre Grosdidier and Manfred Morari. Interaction measures for systems under decentralized control. *Automatica*, 22(3):309–319, 1986. 1, 4

[39] Björn Halvarsson, Bengt Carlsson, and Torsten Wik. A new input/output pairing strategy based on linear quadratic gaussian control. In *IEEE International Conference on Control and Automation*, pages 978–982. IEEE, 2009. 2.3

[40] Morris H. Hansen and William N. Hurwitz. On the theory of sampling from finite populations. *The Annals of Mathematical Statistics*, 14(4):333–362, 1943. ISSN 00034851. URL http://www.jstor.org/stable/2235923. 3.1.1.1

[41] John H Holland. *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. MIT press, 1992. 3

[42] Bin Hu, Kaiqing Zhang, Na Li, Mehran Mesbahi, Maryam Fazel, and Tamer Başar. Toward a theoretical foundation of policy optimization for learning control policies. *Annual Review of Control, Robotics, and Autonomous Systems*, 6:123–158, 2023. 3.7

[43] Andrew Ilyas, Logan Engstrom, Shibani Santurkar, Dimitris Tsipras, Firdaus Janoos, Larry Rudolph, and Aleksander Madry. A closer look at deep policy gradients. In *International Conference on Learning Representations*, 2020. 1, 3.1.3.2, 4.3, 5.3

[44] Alberto Isidori. The zero dynamics of a nonlinear system: From the origin to the latest progresses of a long successful story. *European Journal of Control*, 19(5): 369–378, 2013. 4

[45] Shuuji Kajita, Fumio Kanehiro, Kenji Kaneko, Kiyoshi Fujiwara, Kensuke Harada, Kazuhito Yokoi, and Hirohisa Hirukawa. Biped walking pattern generation by using preview control of zero-moment point. In *2003 IEEE international conference on robotics and automation (Cat. No. 03CH37422)*, volume 2, pages 1620–1626. IEEE, 2003. 1

[46] Dmitry Kalashnikov, Alex Irpan, Peter Pastor, Julian Ibarz, Alexander Herzog, Eric Jang, Deirdre Quillen, Ethan Holly, Mrinal Kalakrishnan, Vincent Vanhoucke, et al. Scalable deep reinforcement learning for vision-based robotic manipulation. In *Conference on robot learning*, pages 651–673. PMLR, 2018. 1, 5.3

[47] Morton I Kamien and Nancy Lou Schwartz. *Dynamic optimization: the calculus of variations and optimal control in economics and management*. courier corporation,

2012. 1

[48] Lixia Kang, Wentao Tang, Yongzhong Liu, and Prodromos Daoutidis. Control configuration synthesis using agglomerative hierarchical clustering: A graph-theoretic approach. *Journal of Process Control*, 46:43–54, 2016. 1, 4

[49] Ashwin Khadke and Hartmut Geyer. Policy decomposition: Approximate optimal control with suboptimality estimates. In *IEEE-RAS Humanoids*, 2020. doi: 10.1109/HUMANOIDS47582.2021.9555796. 1.1

[50] Ashwin Khadke and Hartmut Geyer. Search methods for policy decompositions, 2022. URL https://arxiv.org/abs/2203.15200. 1.1, 5.3

[51] Ashwin Khadke and Hartmut Geyer. Sparsity inducing system representations for policy decompositions. In *CDC*, 2022 (under review). 1.1

[52] Ali Khaki-Sedigh and Bijan Moaveni. *Control configuration selection for multivariable plants*, volume 391. Springer, 2009. 1, 4

[53] Levente Kocsis and Csaba Szepesvári. Bandit based monte-carlo planning. In *ECML*, Berlin, Heidelberg, 2006. Springer. ISBN 978-3-540-46056-5. 3.1.2

[54] Vijay Konda and John Tsitsiklis. Actor-critic algorithms. *Advances in neural information processing systems*, 12, 1999. 5.3

[55] Sanjay Lall, Jerrold E. Marsden, and Sonja Glavaški. A subspace approach to balanced truncation for model reduction of nonlinear control systems. *International Journal of Robust and Nonlinear Control*, 12(6):519–535, 2002. doi: https://doi.org/10.1002/rnc.657. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/rnc.657. 1

[56] Sanjay Lall, Petr Krysl, and Jerrold E Marsden. Structure-preserving model reduction for mechanical systems. *Physica D: Nonlinear Phenomena*, 184(1-4):304–318, 2003. 1

[57] Toni Lassila, Andrea Manzoni, Alfio Quarteroni, and Gianluigi Rozza. Model order reduction in fluid dynamics: challenges and perspectives. *Reduced Order Methods for modeling and computational reduction*, pages 235–273, 2014. 1

[58] Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research*, 17(1):1334–1373, 2016. 1

[59] Yuanqi Mao, Michael Szmuk, and Behcet Acikmese. Successive convexification of non-convex optimal control problems and its convergence properties. In *2016 IEEE 55th Conference on Decision and Control (CDC)*. IEEE, dec 2016. doi: 10.1109/cdc.2016.7798816. 2.3

[60] William C. Martin, Albert Wu, and Hartmut Geyer. Experimental evaluation of deadbeat running on the atrias biped. *IEEE RAL*, 2(2), 2017. doi: 10.1109/LRA.2017.2658020. (document), 2.3, 3.1.3.3, 3.6

[61] Nikolai Matni, Aaron D. Ames, and John C. Doyle. Towards a theory of control architecture: A quantitative framework for layered multi-rate control, 2024. 1

[62] David Q Mayne. Differential dynamic programming–a unified approach to the optimization of dynamic systems. In *Control and Dynamic Systems*, volume 10, pages 179–254. Elsevier, 1973. 2.2.2

[63] Thomas Mc Avoy, Yaman Arkun, Rong Chen, Derek Robinson, and P David Schnelle. A new approach to defining a dynamic relative gain. *Control Engineering Practice*, 11(8):907–914, 2003. 1, 4

[64] Melanie Mitchell. *An introduction to genetic algorithms*. MIT press, 1998. 3

[65] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937. PMLR, 2016. 1, 3.1.3.2, 4.3, C.2, C.2

[66] Bruce Moore. Principal component analysis in linear systems: Controllability, observability, and model reduction. *IEEE transactions on automatic control*, 26(1): 17–32, 1981. (document), 4.2, 4.1

[67] Iulian Munteanu, Antoneta Iuliana Bratcu, Emil CeangĂ, and Nicolaos-Antonio Cutululis. *Optimal control of wind energy systems: towards a global approach*, volume 22. Springer, 2008. 1

[68] Richard M Murray, Zexiang Li, and S Shankar Sastry. *A mathematical introduction to robotic manipulation*. CRC press, 2017. C.1

[69] Umashankar Nagarajan and Katsu Yamane. Automatic task-specific model reduction for humanoid robots. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2578–2585. IEEE, 2013. 1

[70] Michael Neunert, Markus Stäuble, Markus Giftthaler, Carmine D Bellicoso, Jan Carius, Christian Gehring, Marco Hutter, and Jonas Buchli. Whole-body nonlinear model predictive control through contacts for quadrupeds. *IEEE Robotics and Automation Letters*, 3(3):1458–1465, 2018. 1

[71] Reza Olfati-Saber. *Nonlinear control of underactuated mechanical systems with application to robotics and aerospace vehicles*. PhD thesis, Massachusetts Institute of Technology, 2001. 4

[72] M. Palanisamy, H. Modares, F. L. Lewis, and M. Aurangzeb. Continuous-time q-learning for infinite-horizon discounted cost linear quadratic regulator problems. *IEEE Transactions on Cybernetics*, 45(2):165–176, 2015. 2.2, 2.2.1

[73] Jan Peters, Katharina Mulling, and Yasemin Altun. Relative entropy policy search. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 24, pages 1607–1612, 2010. 1

[74] Martin L. Puterman and Shelby L. Brumelle. On the convergence of policy iteration in stationary dynamic programming. *Mathematics of Operations Research*, 4(1):60–69, 1979. ISSN 0364765X, 15265471. URL http://www.jstor.org/stable/3689239. 1

[75] Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8, 2021. URL http://jmlr.org/papers/v22/20-1364.html. 4.3, C.2

[76] Sivaguru S Ravindran. A reduced-order approach for optimal control of fluids using proper orthogonal decomposition. *International journal for numerical methods in fluids*, 34(5):425–448, 2000. 1

[77] Mario E Salgado and Arthur Conley. Mimo interaction measure and controller structure selection. *International Journal of Control*, 77(4):367–383, 2004. 1, 4

[78] Manuel S Santos and John Rust. Convergence properties of policy iteration. *SIAM Journal on Control and Optimization*, 42(6):2094–2115, 2004. 4.3, 5.3

[79] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In Francis Bach and David Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 1889–1897, Lille, France, 07–09 Jul 2015. PMLR. URL https://proceedings.mlr.press/v37/schulman15.html. 1

[80] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017. (document), 1, 3.1.3.2, 4.3, 5, 5.3, 5.1, A.2.2, C.2, C.2

[81] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation, 2018. 4.3, 5, 5.3

[82] Jennie Si, Andrew G. Barto, Warren Buckler Powell, and Don Wunsch. *Handbook of Learning and Approximate Dynamic Programming (IEEE Press Series on Computational Intelligence)*. Wiley-IEEE Press, 2004. ISBN 047166054X. 1

[83] Dragoslav D Siljak. *Decentralized control of complex systems*. Courier Corporation, North Chelmsford, MA, 2011. 1, 2.3

[84] Adam Slowik and Halina Kwasnicka. Evolutionary algorithms and their applications to engineering problems. *Neural Computing and Applications*, 32:12363–12379, 2020. 3

[85] Kenneth Stanley and Risto Miikkulainen. Evolving neural networks through augmenting topologies. *Evol. Comput.*, 10(2), 2002. 3

[86] Richard P Stanley. What is enumerative combinatorics? In *Enumerative combinatorics*, pages 1–63. Springer, Boston, MA, 1986. I

[87] Mike Stilman, Christopher G Atkeson, James J Kuffner, and Garth Zeglin. Dynamic programming in reduced dimensional spaces: Dynamic planning for robust biped locomotion. In *International Conference on Robotics and Automation*, pages 2399–2404. IEEE, 2005. 1

[88] George W Swan et al. *Applications of optimal control theory in biomedicine*. M. Dekker New York, 1984. 1

[89] Yuval Tassa, Nicolas Mansard, and Emo Todorov. Control-limited differential dynamic programming. In *IEEE International Conference on Robotics and Automation*, pages 1168–1175. IEEE, 2014. 2.2, 2.2.2, 2.2.2

[90] Russ Tedrake, Ian R Manchester, Mark Tobenkin, and John W Roberts. Lqr-trees: Feedback motion planning via sums-of-squares verification. *The International Journal of Robotics Research*, 29(8):1038–1052, 2010. 1

[91] Emanuel Todorov and Weiwei Li. A generalized iterative lqg method for locally-optimal feedback control of constrained nonlinear stochastic systems. In *American Control Conference*, pages 300–306. IEEE, 2005. 2.2.2

[92] Emanuel Todorov and Yuval Tassa. Iterative local dynamic programming. In *IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning*, pages 90–95. IEEE, 2009. 2.2.2

[93] Emanuel Todorov, Weiwei Li, and Xiuchuan Pan. From task parameters to motor synergies: A hierarchical framework for approximately optimal control of redundant manipulators. *Journal of robotic systems*, 22(11):691–710, 2005. 1

[94] Marc Van De Wal and Bram De Jager. Survey a review of methods for input/output selection. *Automatica*, 37(4):487–510, apr 2001. ISSN 0005-1098. doi: 10.1016/S0005-1098(00)00181-3. URL https://doi.org/10.1016/S0005-1098(00)00181-3. 1

[95] Pradnya A Vikhar. Evolutionary algorithms: A critical review and its future prospects. In *2016 International conference on global trends in signal processing, information computing and communication (ICGTSPICC)*, pages 261–265. IEEE, 2016. 3

[96] Yuh-Shyang Wang, Nikolai Matni, and John C Doyle. Separable and localized system-level synthesis for large-scale systems. *IEEE Transactions on Automatic Control*, 63(12):4234–4249, 2018. 1

[97] Zaiwen Wen and Wotao Yin. A feasible method for optimization with orthogonality constraints. *Mathematical Programming*, 142(1):397–434, 2013. B.1

[98] Björn Wittenmark and Mario E Salgado. Hankel-norm based interaction measure

for input-output pairing. *IFAC Proceedings Volumes*, 35(1):429–434, 2002. 1, 4

[99] Zhaoming Xie, Xingye Da, Buck Babich, Animesh Garg, and Michiel van de Panne. Glide: Generalizable quadrupedal locomotion in diverse environments with a centroidal model. In *International Workshop on the Algorithmic Foundations of Robotics*, pages 523–539. Springer, 2022. 1

[100] Nan Xue and Aranya Chakrabortty. Optimal control of large-scale networks using clustering based projections. *arXiv:1609.05265*, 2016. 1

[101] Kemin Zhou and John Comstock Doyle. *Essentials of robust control*, volume 104. Prentice Hall, Upper Saddle River, NJ, 1998. 2.3