Effective Search for Control Hierarchies within the Policy Decomposition Framework

Ashwin Khadke and Hartmut Geyer

***Abstract*—Policy decomposition is a novel framework for approximating optimal control policies of complex dynamical systems. The framework belongs to the class of hierarchical control methods which compose a control policy from optimal control solutions of smaller but tractable sub-systems. What stands out about policy decomposition is that it can estimate *a priori* how well the closed-loop behavior of different control hierarchies will match the unknown optimal policy of the entire system. Policy decomposition can therefore discover tractable control policies that give up little in closed-loop performance. However, this advantage does not come for free, as even for systems with moderate numbers of states and inputs, their vast number of possible hierarchies makes it impossible to estimate the closed-loop performance for all hierarchies. Here we demonstrate that this combinatorial challenge does not thwart the utility of policy decomposition. We identify a tree-like representation of any control hierarchy that policy decomposition can produce and use it to adapt two combinatorial search methods, Genetic Algorithm and Monte Carlo Tree Search, to the search for promising control hierarchies. Applying the so adapted search methods to characteristic regulator problems in robotics, we find that the Genetic Algorithm in particular discovers promising hierarchies, which produce control policies that perform better than common heuristic policies and policies obtained with popular neural network learning strategies. These results suggest the utility of the policy decomposition framework, and we conclude with directions of future research that would broaden the type of problems to which policy decomposition can be applied.**

***Index Terms*—policy optimization, curse of dimensionality, hierarchical control**

I. Introduction

Synthesizing global policies for optimal control of complex dynamical systems, in general, requires approximation methods to be tractable [1]. These methods are broadly based on the ideas of novel policy representations [2], model order reduction [3], and hierarchical control [4], [5]. Novel policy representations include look-up tables [1] and their compact variants [6], locally optimized state and input trajectories [7]–[9], radial basis functions [10], or neural networks [11] to approximate the optimal control policy. Look-up tables and similar representations provide guaranteed convergence to the optimal solution but scale poorly in terms of the required memory and computation time to solve for the optimal policy. Neural networks, on the other hand, have shown tremendous success in representing control policies over high-dimensional state-spaces, but convergence guarantees exist in very restrictive settings [12] and in general the resulting policies are at best locally optimal.

Ashwin Khadke and Hartmut Geyer are with the Robotics Institute, Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, PA 15213, USA `{akhadke, hgeyer}@andrew.cmu.edu`

Complementary to novel policy representation methods, model reduction and hierarchical control methods explicitly reduce the complexity of the underlying dynamical system. Model reduction methods construct a lower-dimensional representation for the state and input spaces, solve for the optimal control policy of this reduced system, and then project the solution back onto the full system [3]. The model reductions are either hand designed [13], [14], or derived by minimizing a projection error [15] or the loss of controllability [16]. Hierarchical control methods, by contrast, impose a simplifying hierarchy on the control structure. In this hierarchy, control sub-policies are derived for components of the entire system and then assembled into a combined policy. The hierarchies are obtained by using interaction measures [17]–[19], passivity measures [20], controllability and observability measures [21], [22], and the notion of timescale separation in the open-loop dynamics [23]. However, none of these methods account for the objective of the original optimal control problem, and the resulting control policies can have poor closed-loop performance when controlling the full system. Very recently, an approach to model reduction that explicitly minimizes the loss in closed-loop performance was proposed [24] but a counterpart for hierarchical control is missing.

Policy Decomposition [25] is a framework that belongs to the class of hierarchical control methods but incorporates estimates of closed-loop performance in the process of identifying suitable control hierarchies. Similar to other hierarchical control methods, it reduces the optimal control problem for a complex dynamical system into a hierarchy of lower-dimensional subproblems that are much more tractable to solve (Fig. I). In contrast to the other methods, it estimates the suboptimality of different hierarchies *a priori* and uses this information to discover hierarchies that offer sharp reductions in computation time while sacrificing minimally on closed-loop performance.

While our initial work [25] introduced the theory of Policy Decomposition, it also pointed to a shortcoming of this framework. The number of hierarchies for which the suboptimality has to be estimated grows exponentially with system complexity. For example, a system with five degrees of freedom has about 10^{10} possible hierarchies if all degrees are actuated. Here we investigate how two widely applied search algorithms, Genetic Algorithm (GA) [26] and Monte Carlo Tree Search (MCTS) [27], can be adopted to effectively search for promising hierarchies in this large combinatorial space. This work extends the policy decomposition framework with

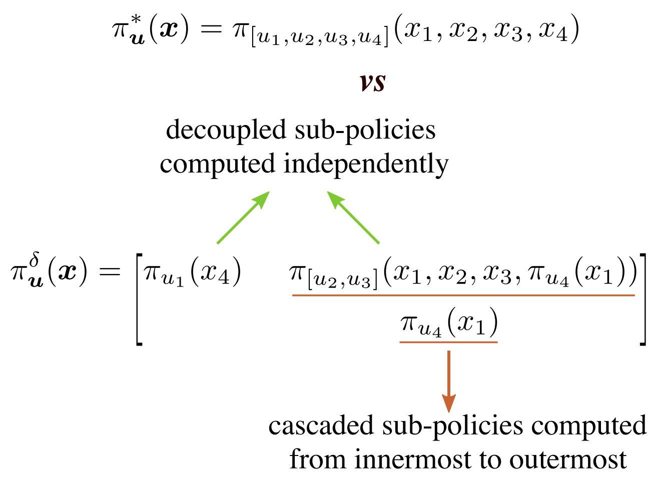

$\pi^*_{\boldsymbol{u}}(\boldsymbol{x}) = \pi_{[u_1,u_2,u_3,u_4]}(x_1,x_2,x_3,x_4)$
vs
decoupled sub-policies computed independently
$\pi^\delta_{\boldsymbol{u}}(\boldsymbol{x}) = \begin{bmatrix} \pi_{u_1}(x_4) & \dfrac{\pi_{[u_2,u_3]}(x_1,x_2,x_3,\pi_{u_4}(x_1))}{\pi_{u_4}(x_1)} \end{bmatrix}$
cascaded sub-policies computed from innermost to outermost

Fig. 1. Concept of policy decomposition shown for a fictive system with four states and four inputs. An example hierarchy constructed using the two fundamental strategies of cascading and decoupling is shown.

- a tree-like abstraction that encodes any control hierarchy constructed using the two fundamental strategies of decoupling and cascading, and facilitates deriving suboptimality estimates for such hierarchies;
- adaptations to GA and MCTS to search the space of such tree abstractions for promising hierarchies using
  - a fitness function that balances the suboptimality of the resulting policies and the computational reduction offered,
  - a procedure to uniformly sample hierarchy trees, and
  - operators for modifying (mutating) these trees with closure;
- experiments with optimal control problems from three domains including balancing a simplified biped, swinging up planar manipulators, and hovering a quadcopter, which demonstrate the efficacy of the search methods in finding promising hierarchies, and the advantage of the so discovered hierarchies over heuristic ones and ones identified with popular neural network learning strategies.

The remainder of this paper is organized as follows: We provide an overview of the main ideas behind Policy Decomposition with an example problem in section II and more formal development in section III. In section IV, we detail how GA and MCTS are adapted to search for hierarchies, discuss the efficacy of these methods, and compare the performance of the resulting policies with known control hierarchies for three example systems. Finally, we summarize the main findings and conclude with directions for future research in section V.

II. Overview

As an illustrative example, consider the design of a control policy for the swing up of a pole on a cart while moving the cart to a goal position (Fig. 2). The system can be completely described using the position and velocity of the cart $(x, \dot{x})$ and the pole $(\theta, \dot{\theta})$, and is controlled using a force F that can be applied to the cart and a torque τ applied at the joint between the cart and the pole. To simplify the joint optimization of policies for τ and F as functions of all four state variables, one could first optimize an inner control policy $\pi_\tau(\theta, \dot{\theta})$ for τ to swing up the pole assuming the cart is locked and then optimize an outer policy $\pi_F\left(x, \dot{x}, \theta, \dot{\theta}, \pi_\tau(\theta, \dot{\theta})\right)$ for F to move the cart with the torque control of the pole set to π_τ. An alternative to this cascade is to treat the cart and pole as decoupled subsystems and design independent control policies $\pi_\tau(x, \dot{x})$ and $\pi_F(\theta, \dot{\theta})$ (Fig. 2-b). Both possibilities reduce the dimensionality of the problem and make it computationally much more tractable, but the cascaded policy optimization suggested first (blue trace, Fig. 2-c) performs about as well as the true optimal control (red trace) whereas the decoupled one performs much worse (green trace). It is possible to intuit the cascaded strategy by observing that the pole dynamics are independent of the position and velocity of the cart. Therefore, a cascaded control optimization with an inner policy $\pi_\tau(\theta, \dot{\theta})$ disregarding the cart should perform well. For more complex systems, however, intuition quickly fades, and it gets difficult to predict the closed-loop performance 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, err^δ, defined as the average difference between the value functions V^δ and V^* of policies obtained with and without the hierarchy,

$$\mathrm{err}^\delta = \frac{\int_{\mathcal{S}} \left(V^\delta(\boldsymbol{x}) - V^*(\boldsymbol{x})\right) d\boldsymbol{x}}{\int_{\mathcal{S}} V^*(\boldsymbol{x})\, d\boldsymbol{x}} \tag{1}$$

where $\mathcal{S}$ is the state space and $\boldsymbol{x} \in \mathcal{S}$. 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 resulting control induced by a hierarchy. Since this measure cannot be evaluated without knowing the policies, we explore two methods for estimating the value error. The first method linearizes the system dynamics about a fixed point and approximates V^* and V_δ with LQR solutions [28], leading to the estimate $\mathrm{err}^\delta_{\mathrm{lqr}}$ derived in section III-A. The second method computes control-limited DDP solutions [29] for the original and hierarchical control system from a few initial states to estimate V^*, V_δ, and thence $\mathrm{err}^\delta_{\mathrm{ddp}}$ (Sec. III-B). The second method improves the accuracy of the error estimate but at great computational cost.

The actual and estimated value errors for all hierarchies of the cart-pole system are summarized in Fig. 3. We use look-up table based Policy Iteration [1] to compute the actual value functions and thereby the value errors (triangles). Both estimates rank the cascaded hierarchy described before (Fig. 2(b)) as the one with least value error, and it reduces policy computation times by a factor of 66. In summary, an algorithm using either estimate could have identified this hierarchy *a priori* and then derived the corresponding policy in minimal time without notably sacrificing closed-loop performance (blue vs. red trace, Fig. 2-c). While the cart-pole system has only 44 possible hierarchies, this number grows exponentially with the system complexity as shown in figure 4. A brute force evaluation of value error estimates for all possible hierarchies quickly becomes impractical. To tackle this combinatorial challenge, we adapt GA and MCTS to efficiently search for promising ones in section IV.

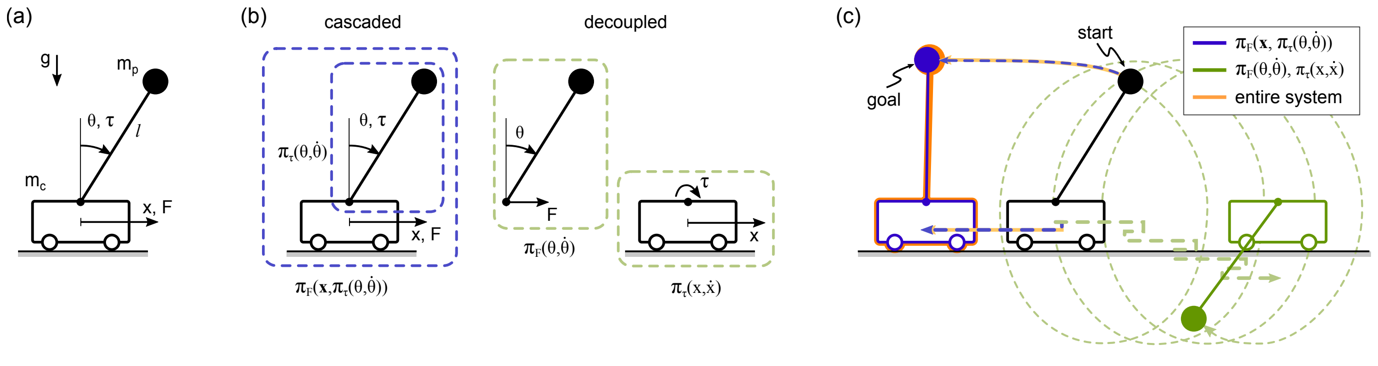

(a)
g
m_p
θ, τ
l
m_c
x, F
(b)
cascaded
$\pi_\tau(\theta,\dot\theta)$
θ, τ
x, F
$\pi_F(\mathbf{x},\pi_\tau(\theta,\dot\theta))$
decoupled
θ
F
$\pi_F(\theta,\dot\theta)$
τ
x
$\pi_\tau(x,\dot x)$
(c)
start
goal
$\pi_F(\mathbf{x}, \pi_\tau(\theta,\dot\theta))$
$\pi_F(\theta,\dot\theta),\ \pi_\tau(x,\dot x)$
entire system

Fig. 2. Control of the cart-pole system using policy decomposition. (**a**) The system is completely described using position and velocity of the cart $(x, \dot{x})$ and the pole $(\theta, \dot{\theta})$, and has two control inputs: force F and torque τ. (**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); parameters $m_c = 5$kg, $m_p = 1$kg, and $l = 0.9$m, and bounds on the inputs, $|F| \leq 9$N and $|\tau| \leq 9$Nm.

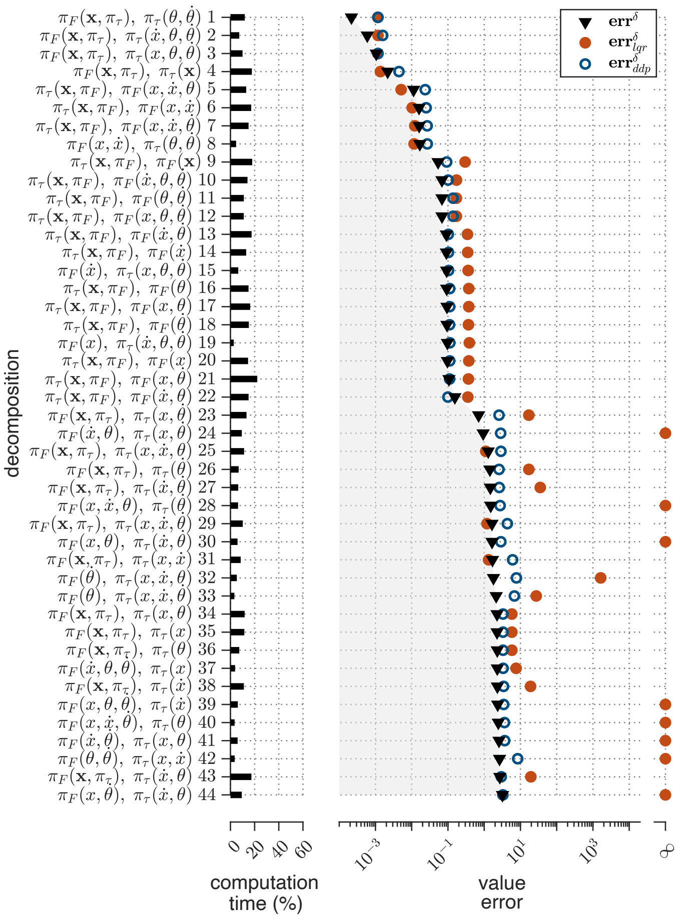


Fig. 3. Hierarchies for cart-pole listed in order of true value error. The computation times (relative to optimal control) and value errors of all hierarchies (triangles) are shown together with their LQR and DDP estimates (filled and open circles). LQR estimates are set to infinity for uncontrollable systems after linearization.

III. Policy Decomposition

Consider the general dynamical system

$$\dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u}) \tag{2}$$

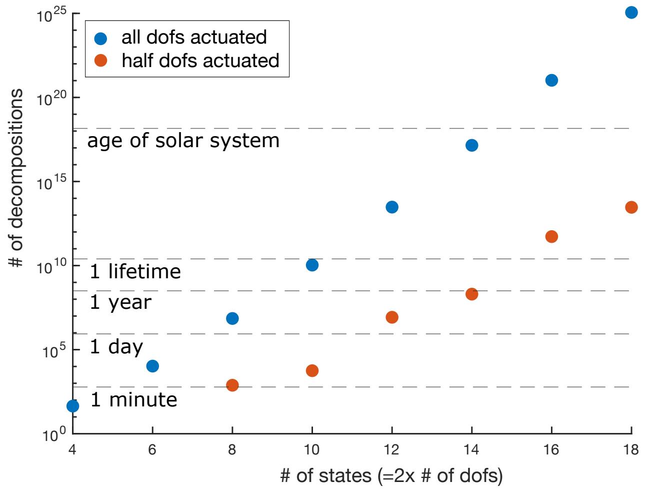


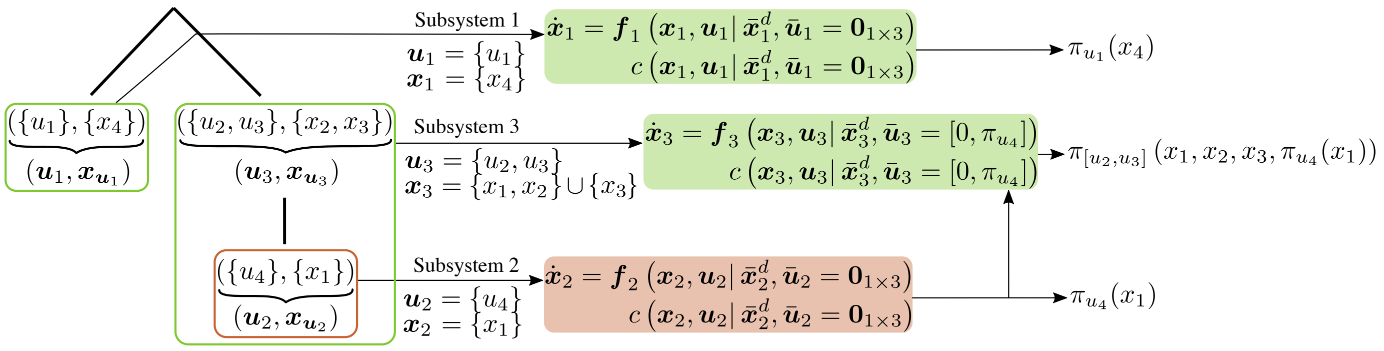
Fig. 4. Number of possible hierarchies as the number of state variables and control inputs increase. Assuming it takes **0.1 sec/value error estimate** computation, the number of hierarchies that can be evaluated in some fixed times are marked with dotted lines. Appendix A describes the enumeration.

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

$$J = \int_0^\infty e^{-\lambda t} c(\boldsymbol{x}(t), \boldsymbol{u}(t)) \, dt. \tag{3}$$

which describes the discounted sum of costs accrued over time. We consider optimal regulation control problems where the objective is to drive the system to a fixed desired state $\boldsymbol{x}^d$. Furthermore, we assume quadratic costs $c(\boldsymbol{x}, \boldsymbol{u}) = (\boldsymbol{x} - \boldsymbol{x}^d)^T \boldsymbol{Q} (\boldsymbol{x} - \boldsymbol{x}^d) + (\boldsymbol{u} - \boldsymbol{u}^d)^T \boldsymbol{R} (\boldsymbol{u} - \boldsymbol{u}^d)$ where $\boldsymbol{u}^d$ is the input that stabilizes the system at $\boldsymbol{x}^d$. The discount factor λ characterizes the trade-off between immediate and future costs.

Policy Decomposition approximates the search for one high-dimensional policy $\pi^*_{\boldsymbol{u}}(\boldsymbol{x})$ with a hierarchy of lower-dimensional sub-policies of smaller optimal control problems that are much faster to compute. These sub-policies are computed in either a cascaded or decoupled fashion and then reassembled into one hierarchical policy $\pi^\delta_{\boldsymbol{u}}(\boldsymbol{x})$ for the entire system (Fig. I). Sub-policy $\pi_{\boldsymbol{u}_i}(\boldsymbol{x}_i)$ is the solution to the



Subsystem 1
$\dot{\boldsymbol{x}}_1 = \boldsymbol{f}_1\left(\boldsymbol{x}_1, \boldsymbol{u}_1 \mid \bar{\boldsymbol{x}}_1^d, \bar{\boldsymbol{u}}_1 = \boldsymbol{0}_{1\times 3}\right)$
$c\left(\boldsymbol{x}_1, \boldsymbol{u}_1 \mid \bar{\boldsymbol{x}}_1^d, \bar{\boldsymbol{u}}_1 = \boldsymbol{0}_{1\times 3}\right)$
$\boldsymbol{u}_1 = \{u_1\}$
$\boldsymbol{x}_1 = \{x_4\}$
$\pi_{u_1}(x_4)$
$\underbrace{(\{u_1\}, \{x_4\})}_{(\boldsymbol{u}_1, \boldsymbol{x}_{\boldsymbol{u}_1})}$
$\underbrace{(\{u_2, u_3\}, \{x_2, x_3\})}_{(\boldsymbol{u}_3, \boldsymbol{x}_{\boldsymbol{u}_3})}$
Subsystem 3
$\dot{\boldsymbol{x}}_3 = \boldsymbol{f}_3\left(\boldsymbol{x}_3, \boldsymbol{u}_3 \mid \bar{\boldsymbol{x}}_3^d, \bar{\boldsymbol{u}}_3 = [0, \pi_{u_4}]\right)$
$c\left(\boldsymbol{x}_3, \boldsymbol{u}_3 \mid \bar{\boldsymbol{x}}_3^d, \bar{\boldsymbol{u}}_3 = [0, \pi_{u_4}]\right)$
$\boldsymbol{u}_3 = \{u_2, u_3\}$
$\boldsymbol{x}_3 = \{x_1, x_2\} \cup \{x_3\}$
$\pi_{[u_2,u_3]}(x_1, x_2, x_3, \pi_{u_4}(x_1))$
$\underbrace{(\{u_4\}, \{x_1\})}_{(\boldsymbol{u}_2, \boldsymbol{x}_{\boldsymbol{u}_2})}$
Subsystem 2
$\dot{\boldsymbol{x}}_2 = \boldsymbol{f}_2\left(\boldsymbol{x}_2, \boldsymbol{u}_2 \mid \bar{\boldsymbol{x}}_2^d, \bar{\boldsymbol{u}}_2 = \boldsymbol{0}_{1\times 3}\right)$
$c\left(\boldsymbol{x}_2, \boldsymbol{u}_2 \mid \bar{\boldsymbol{x}}_2^d, \bar{\boldsymbol{u}}_2 = \boldsymbol{0}_{1\times 3}\right)$
$\boldsymbol{u}_2 = \{u_4\}$
$\boldsymbol{x}_2 = \{x_1\}$
$\pi_{u_4}(x_1)$

Fig. 5. An input-tree T^δ that represents the policy decomposition depicted in figure I is shown (**left**). The three resulting subsystems are shown (**mid**). Policies for u_1 and u_4, i.e. the inputs at the leaf nodes, are obtained first by independently solving the optimal control problems for subsystems 1 and 2 respectively. Policies for inputs u_2 and u_3 are computed jointly by solving the optimal control problem for subsystem 3. The resulting policies for different inputs are a function of different state variables (**right**).

optimal control problem defined by dynamics and cost,

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

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 $\bar{\boldsymbol{u}}_i = \boldsymbol{u} \setminus \boldsymbol{u}_i$ comprises inputs that are decoupled from $\boldsymbol{u}_i$ and those 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 $\boldsymbol{u}_j \subseteq \bar{\boldsymbol{u}}_i$ are inputs that appear lower in the cascade to $\boldsymbol{u}_i$, and $\boldsymbol{x}_j \subseteq \boldsymbol{x}_i$. The inputs $\bar{\boldsymbol{u}}_i \setminus \boldsymbol{u}_j$ 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

$$T^\delta = (\mathcal{V}, \mathcal{E}) \tag{4}$$

where all nodes except the root node are tuples of disjoint subsets of inputs and state variables $\boldsymbol{v}_i = (\boldsymbol{u}_i, \boldsymbol{x}_{\boldsymbol{u}_i})\, \forall \boldsymbol{v}_i \in \mathcal{V} \setminus \{\boldsymbol{v}_{\text{root}}\}$. 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. Inputs that belong to different branches are decoupled for the sake of policy computation. A sub-tree rooted at node $\boldsymbol{v}_i$ characterizes a subsystem with control inputs $\boldsymbol{u}_i$ and state $\boldsymbol{x}_i = \cup_j \boldsymbol{x}_{\boldsymbol{u}_j}$, where $\boldsymbol{x}_{\boldsymbol{u}_j}$ are state variables belonging to nodes in the sub-tree. Note, $\boldsymbol{x}_{\boldsymbol{u}_i} \subseteq \boldsymbol{x}_i$ and $\boldsymbol{x}_{\boldsymbol{u}_i}$ may be empty if it does not belong to a leaf-node. Figure 5 depicts the input-tree for the hierarchy shown in figure I and describes the resulting subsystems. Policies are computed in a child-first order, starting from leaf nodes followed by their parents and so on.

The value error (err^δ) (Eq. 1) provides a suboptimality measure for hierarchy δ. However, exactly evaluating it requires knowing the optimal and the hierarchical policies. We thus explore two methods to estimate this error.

A. LQR Suboptimality Estimate

The first method relies on the system obtained by linearizing the dynamics (Eq. 2) about the goal state and input,

$$\dot{\boldsymbol{x}} = \boldsymbol{A}(\boldsymbol{x} - \boldsymbol{x}^d) + \boldsymbol{B}(\boldsymbol{u} - \boldsymbol{u}^d) \tag{5}$$
$$\boldsymbol{A} = \left.\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}}\right|_{(\boldsymbol{x}^d, \boldsymbol{u}^d)}, \quad \boldsymbol{B} = \left.\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{u}}\right|_{(\boldsymbol{x}^d, \boldsymbol{u}^d)}$$

As we assumed quadratic costs, the optimal control of this linear system is an LQR, whose value function $V^*_{\text{lqr}}(\boldsymbol{x})$ can be computed by solving the algebraic Riccati equation [30]. The value error estimate for decomposition δ then becomes

$$\text{err}^\delta_{\text{lqr}} = \frac{\int_{\mathcal{S}} \left(V^\delta_{\text{lqr}}(\boldsymbol{x}) - V^*_{\text{lqr}}(\boldsymbol{x})\right) d\boldsymbol{x}}{\int_{\mathcal{S}} V^*_{\text{lqr}}(\boldsymbol{x})\, d\boldsymbol{x}} \tag{6}$$

where $V^\delta_{\text{lqr}}(\boldsymbol{x})$ is the value function for the equivalent hierarchical policy 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, and then assembling them into a policy for the full linear system. The sub-policies $\pi_{\boldsymbol{u}_i}(\boldsymbol{x}_i)$ are LQR solutions to the corresponding linear subsystems,

$$\dot{\boldsymbol{x}}_i = \boldsymbol{A}_i(\boldsymbol{x}_i - \boldsymbol{x}_i^d) + \boldsymbol{B}_i(\boldsymbol{u}_i - \boldsymbol{u}_i^d) \tag{7}$$
$$\boldsymbol{A}_i = \left.\frac{\partial \boldsymbol{f}_i}{\partial \boldsymbol{x}_i}\right|_{(\boldsymbol{x}^d, \boldsymbol{u}^d)} + \left.\frac{\partial \boldsymbol{f}_i}{\partial \bar{\boldsymbol{u}}_i}\right|_{(\boldsymbol{x}^d, \boldsymbol{u}^d)} \left.\frac{\partial \pi_{\bar{\boldsymbol{u}}_i}}{\partial \boldsymbol{x}_i}\right|_{\boldsymbol{x}_i^d},$$
$$\boldsymbol{B}_i = \left.\frac{\partial \boldsymbol{f}_i}{\partial \boldsymbol{u}_i}\right|_{(\boldsymbol{x}^d, \boldsymbol{u}^d)}$$

and cost,

$$c_i(\boldsymbol{x}_i, \boldsymbol{u}_i) = (\boldsymbol{x}_i - \boldsymbol{x}_i^d)^T \boldsymbol{Q}_i (\boldsymbol{x}_i - \boldsymbol{x}_i^d) + \left(\begin{bmatrix}\boldsymbol{u}_i \\ \pi_{\bar{\boldsymbol{u}}_i}\end{bmatrix} - \boldsymbol{u}^d\right)^T \boldsymbol{R} \left(\begin{bmatrix}\boldsymbol{u}_i \\ \pi_{\bar{\boldsymbol{u}}_i}\end{bmatrix} - \boldsymbol{u}^d\right)$$

Note, $\pi_{\bar{\boldsymbol{u}}_i}(\boldsymbol{x}_i) = [0, \ldots, 0, \boldsymbol{u}_j^d - \boldsymbol{K}_j(\boldsymbol{x}_j - \boldsymbol{x}_j^d), 0, \ldots, 0]$ where $\boldsymbol{K}_j$ is the LQR solution for a subsystem characterized by $\boldsymbol{u}_j \subseteq \bar{\boldsymbol{u}}_i$ and $\boldsymbol{x}_j \subseteq \boldsymbol{x}_i$, which appears lower in cascade to the subsystem defined by $\boldsymbol{u}_i$ and $\boldsymbol{x}_i$. Inputs $\bar{\boldsymbol{u}}_i \setminus \boldsymbol{u}_j$ are decoupled from $\boldsymbol{u}_i$ in the hierarchy, and are set to 0. $\boldsymbol{Q}_i$ is the appropriate sub-matrix of the original cost function matrix $\boldsymbol{Q}$.

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

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

whose gain $\boldsymbol{K}^{\delta}$ is a block matrix composed of all the subsystem LQR gains $\boldsymbol{K}_i$. For the purely decoupled and cascaded hierarchies with r subsystems, the gain takes on the following forms

$$\boldsymbol{K}^{\delta_{\text{dec}}} = \begin{bmatrix} \overbrace{\boldsymbol{K}_1}^{\dim(\boldsymbol{x}_1)} & 0 & \cdots & 0 \\ 0 & \overbrace{\boldsymbol{K}_2}^{\dim(\boldsymbol{x}_2)} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & \overbrace{\boldsymbol{K}_r}^{\dim(\boldsymbol{x}_r)} \end{bmatrix}$$

and

$$\boldsymbol{K}^{\delta_{\text{cas}}} = \begin{bmatrix} \overbrace{\boldsymbol{K}_1}^{\dim(\boldsymbol{x}_1)} & 0 & \cdots & 0 \\ \multicolumn{2}{l}{\overbrace{\boldsymbol{K}_2 \qquad\qquad}^{\dim(\boldsymbol{x}_2)}} & \cdots & 0 \\ \vdots & & \ddots & \vdots \\ \multicolumn{4}{l}{\overbrace{\boldsymbol{K}_r \qquad\qquad\qquad\qquad\qquad\qquad}^{\dim(\boldsymbol{x})}} \end{bmatrix}$$

respectively. For the example hierarchy depicted in Fig. I the gain has the following form

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

With the gain $\boldsymbol{K}^{\delta}$ defined, the value function resolves to

$$V^{\delta}_{\text{lqr}}(\boldsymbol{x}) = (\boldsymbol{x} - \boldsymbol{x}^d)^T \boldsymbol{P}^{\delta}(\boldsymbol{x} - \boldsymbol{x}^d) \tag{8}$$

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

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

The LQR suboptimality estimate can be computed in minimal time, but it only accounts for linearized system dynamics at the goal state, is agnostic to bounds on the control inputs, and controllers obtained by decomposing the equivalent linear system may be closed-loop unstable ($\boldsymbol{P}^{\delta}$ in Eq. 8 has negative eigenvalues) resulting in $\text{err}^{\delta}_{\text{lqr}} = \infty$.

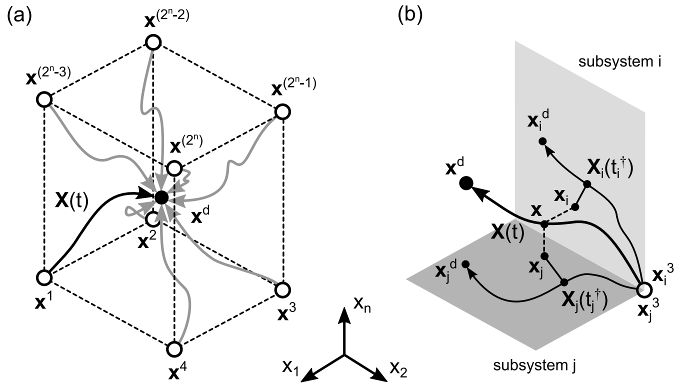


Fig. 6. DDP approximation of value function. (**a**) Local DDP solutions $\boldsymbol{X}(t)$ for $k = 2^n$ initial points $\boldsymbol{x}^s$ located at edges of hyper-cube that defines boundary of explored state space section. (**b**) Nearest neighbors $\boldsymbol{X}_i(t_i^{\dagger})$ on subsystem solutions $\boldsymbol{X}_i(t)$ for current state $\boldsymbol{x}$ along solution $\boldsymbol{X}(t)$.

B. DDP Suboptimality Estimate

We additionally explore the use of Control-Limited DDP [29] to estimate the value error defined in equation 1. (For compactness referred to as simply DDP hereafter.) In contrast to LQR, DDP can enforce input bounds and account for system dynamics away from the goal state. However, these benefits are bought with a costly increase in computation time. Although we use DDP in this work, a different trajectory optimization algorithm may just as easily be applied.

DDP optimizes the closed-loop performance of a system about an initial reference trajectory $\boldsymbol{X}^0(t)$ generated from an input guess $\boldsymbol{U}^0(t)$. It returns a locally optimal trajectory $\boldsymbol{X}(t)$, $\boldsymbol{U}(t)$ from which an estimate for $V^*(\boldsymbol{X}(0))$ is derived,

$$V^*_{\text{ddp}}(\boldsymbol{X}(0)) = \int_0^{t_{\max}} e^{-\lambda t} c\left(\boldsymbol{X}(t), \boldsymbol{U}(t)\right) dt$$

The suboptimality estimate,

$$\text{err}^{\delta}_{\text{ddp}} = \frac{\sum_{s=1}^{k}\left(V^{\delta}_{\text{ddp}}(\boldsymbol{x}^s) - V^*_{\text{ddp}}(\boldsymbol{x}^s)\right)}{\sum_{s=1}^{k} V^*_{\text{ddp}}(\boldsymbol{x}^s)} \tag{9}$$

averages the difference in value function estimates obtained from trajectories of the original and decomposed optimal control problems for k initial points centered on the goal state $\boldsymbol{x}^d$ (Fig. 6(a)). DDP uses quadratic approximations of the system dynamics to optimize the state and input trajectories, but to curb computational costs, we consider only linear approximations. While $V^*_{\text{ddp}}(\boldsymbol{x}^s)$ can be computed straightforwardly from DDP solutions for the full system, the process to derive $V^{\delta}_{\text{ddp}}(\boldsymbol{x}^s)$ requires further explanation.

A hierarchy δ with r subsystems creates r optimal control problems, whose individual DDP solutions need to be combined for computing the approximate value function $V^{\delta}_{\text{ddp}}(\boldsymbol{x}^s)$. We achieve this with the following procedure. First, starting from the initial sub-states $\{\boldsymbol{x}_i^s | s \in 1, \cdots, k\}$ we use DDP to find locally optimal state and input trajectories $\boldsymbol{X}_i^s(t)$ and $\boldsymbol{U}_i^s(t)$ respectively for each subsystem i. Additionally, DDP produces a linear feedback controller in the vicinity of the converged trajectory, characterized by local linear gains $\boldsymbol{K}_i^s(t)$ and reference trajectory $\tilde{\boldsymbol{X}}_i^s(t)$. The feedforward input

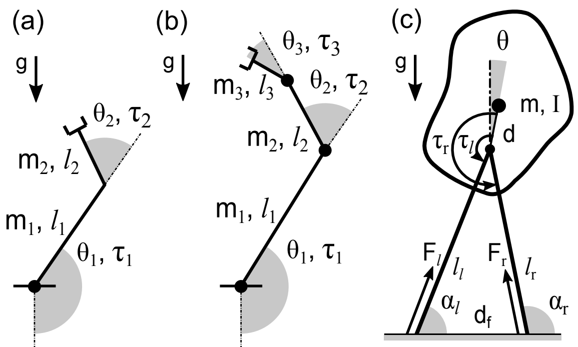


Fig. 7. 2 and 3 link planar manipulators are shown in (a) and (b) respectively, and a simplified model of a biped in stance is depicted in (c). Policies to swing-up the manipulators into an upright position, and to balance the biped midway between the two foot-holds are derived using several hierarchies. Appendix C describes the system dynamics and the cost functions used.

trajectories $\boldsymbol{U}_i^s(t)$ and the feedback controller ($\boldsymbol{K}_i^s(t)$, $\tilde{\boldsymbol{X}}_i^s(t)$) together result in the subsystem following $\boldsymbol{X}_i^s(t)$. Next, using all the k trajectories we define the subsystem control policy as the nearest neighbor policy [8],

$$\pi_{\boldsymbol{u}_i}(\boldsymbol{x}_i) = \boldsymbol{U}_i^{s^\dagger}(t^\dagger) - \boldsymbol{K}_i^{s^\dagger}(t^\dagger)\left(\boldsymbol{x}_i - \tilde{\boldsymbol{X}}_i^{s^\dagger}(t^\dagger)\right) \tag{10}$$

where $s^\dagger$ and $t^\dagger$ respectively mark the trajectory ID and time at which $\boldsymbol{X}_i^s(t)$ is closest to the subsystem state $\boldsymbol{x}_i$ (Fig. 6-b),

$$s^\dagger, t^\dagger = \arg\min_{s,t} \|\boldsymbol{X}_i^s(t) - \boldsymbol{x}_i\|_2 \,. \tag{11}$$

Lastly, we run the policy $\pi_{\boldsymbol{u}}^\delta(\boldsymbol{x}) = (\pi_{\boldsymbol{u}_1}(\boldsymbol{x}_1), \dots, \pi_{\boldsymbol{u}_r}(\boldsymbol{x}_r))$ on the complete system (Eq. 2) initialized at $\boldsymbol{x}^s$ and compute $V_{\text{ddp}}^\delta(\boldsymbol{x}^s)$ from the resulting trajectory $\boldsymbol{X}(t)$,

$$V_{\text{ddp}}^\delta(\boldsymbol{x}^s) = \int_0^{t_{\max}} e^{-\lambda t} c\left(\boldsymbol{X}(t), \pi_{\boldsymbol{u}}^\delta(\boldsymbol{X}(t))\right) dt. \tag{12}$$

Note that $\boldsymbol{X}(t)$ will differ from the collected trajectories of the individual DDP solutions, $(\boldsymbol{X}_1(t), \dots, \boldsymbol{X}_r(t))$, 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 III-A, 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 (Eq. 10). To generate the initial input sequence for a subsystem i, we use the LQR controller gain $\boldsymbol{K}_i$ (described in Sec. III-A), along with nearest neighbor policies (Eq. 10) for subsystems lower in the cascade, to roll-out trajectories and generate $\boldsymbol{U}_i^0(t)$.

The DDP suboptimality estimate generally improves the value error prediction in the cart-pole example (Fig. 3). But, computational costs for this improvement are high. For the cart-pole system, computing DDP estimates requires 60% of the time it takes to actually compute the decomposed policy.

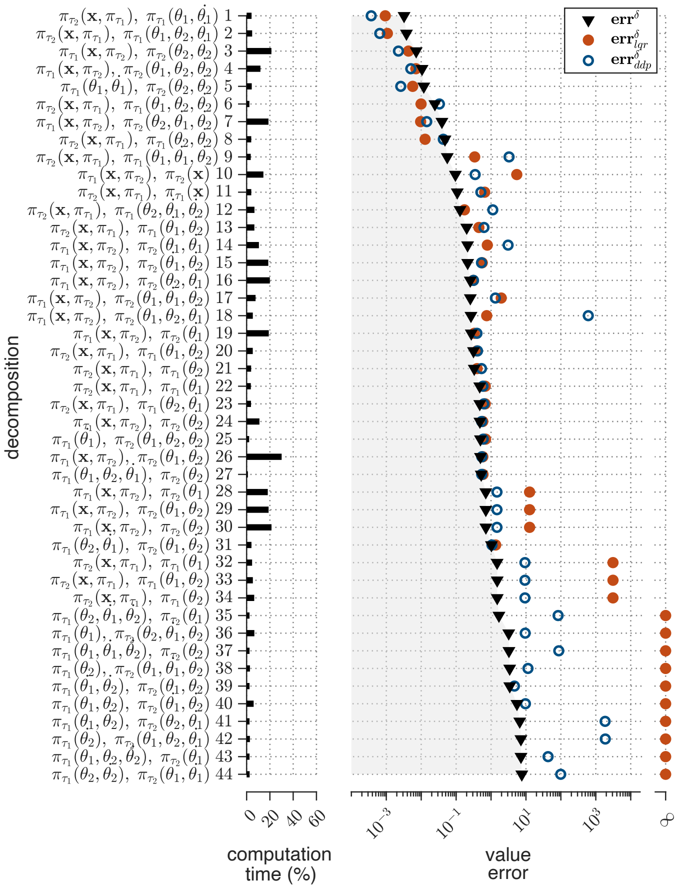


Fig. 8. Hierarchies for planar 2 link manipulator (analogous to Fig. 3).

C. Estimation Quality

We evaluate the quality of these estimates for three additional control problems: the swing-up of 2 and 3 link planar manipulators and the balancing of a simplified biped (Fig. 7). For the planar manipulators, the joint angles (θ_i) and velocities $(\dot{\theta}_i)$ form the state $\boldsymbol{x}$, and the control input $\boldsymbol{u}$ is composed of the joint torques (τ_i). For the biped model, the state variables comprise the center of mass position expressed using leg length (l_r) and leg angle (α_r), the center of mass velocities $(\dot{x}, \dot{z})$, and the torso orientation and angular velocity $(\theta, \dot{\theta})$, and the leg forces (F_r, F_l) and the hip torques (τ_r, τ_l) define the inputs. For all three systems, the dynamics and cost function encoding the desired behavior are summarized in appendix C. As in the cart-pole example, the 2 link manipulator has 44 possible hierarchies, and we compute the true value error and corresponding LQR and DDP estimates for all of them (Fig. 8). Contrarily, computing DDP estimates for every hierarchy of the 3 link manipulator and the biped is not feasible (10,512 and 396,716 possible hierarchies, respectively). We thus identify the Pareto optimal set of hierarchies (Figs. 9 and 10) based on the LQR estimates and the expected reduction in computation time for policy optimization (see Apx. B for details), and evaluate their true value errors and DDP estimates.

In general, the two estimates trend similarly and in line with the true value error. Importantly, as in the cart-pole example, the LQR and DDP estimates identify similar best policies for

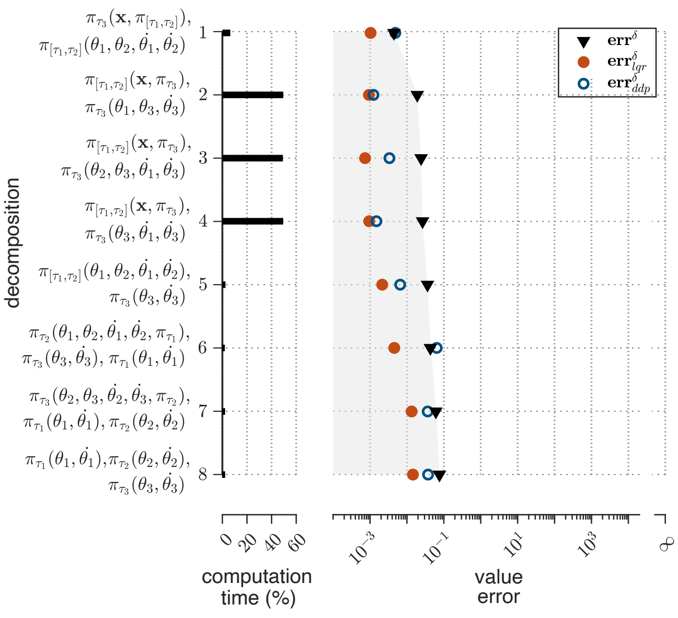


Fig. 9. Hierarchies for 3 link manipulator (Fig. 7(b)) (analogous to Fig. 3).

all three problems (top four hierarchies in Figs. 8-10), 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 [31], [32] (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 Fig. 10) 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 Fig. 9 and #5 in Fig. 10).

While the DDP estimate provides a closer estimate of the true value error in the manipulator problems, it performs significantly worse than the LQR estimate in the biped problem. The poor performance results from us ignoring the second order derivatives of the dynamics to speed up computation of the DDP estimate (compare Sec. III-B). More recent trajectory optimization methods with better convergence properties such as [33] could help to correct this issue. In any case, however, the results of the example problems do not indicate that the simpler LQR estimate performs worse when identifying promising hierarchies, and we will use it for its much lower computational costs when searching for policy decompositions in more complex systems.

IV. Search for Promising Policy Decompositions

The number of possible hierarchies grows substantially with the number of state variables and control inputs (see Apx. A for an explicit count of the number of possible hierarchies). Even evaluating the LQR suboptimality estimates becomes intractable for systems with more than 12 state variables (Fig. 4), and efficient search methods that can operate in this combinatorial space are needed to discover promising hierarchies. Several methods have been proposed to tackle

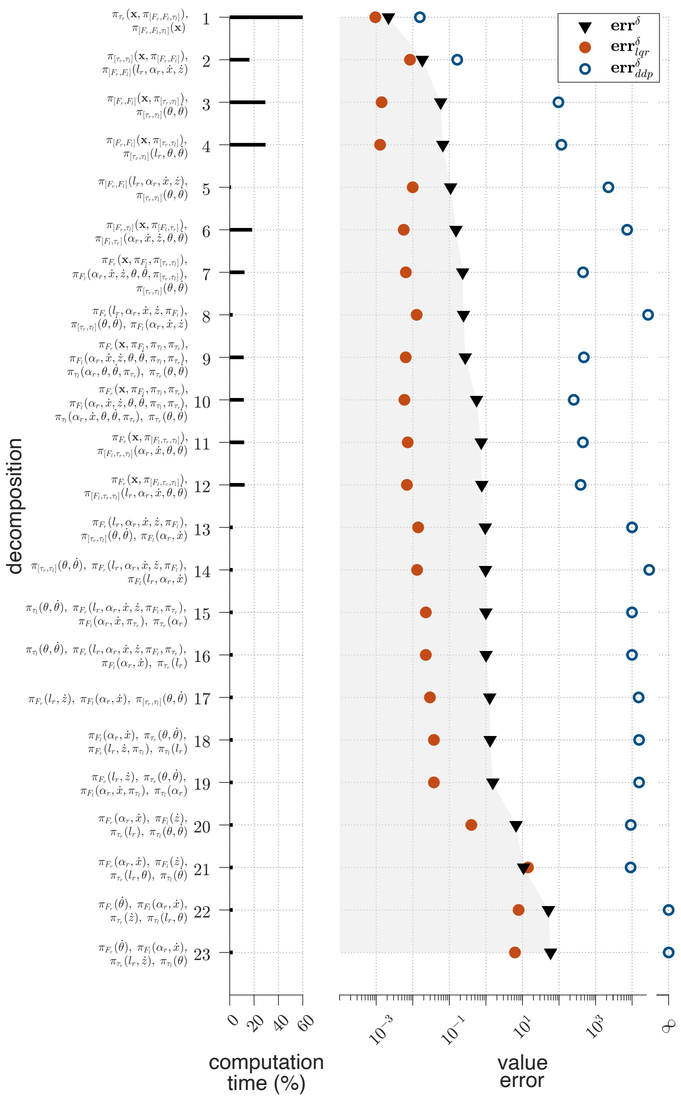


Fig. 10. Hierarchies for the biped (Fig. 7(c)) (analogous to Fig. 3).

combinatorial problems [34], [35]. Genetic Algorithm (GA) is one of them [36], [37]. GA starts off with a randomly generated set of candidate solutions and combines promising candidates to find better ones. A top-down alternative to GA is to start from the original system and decompose it step-by-step. The search for good 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 [38]. Among these, Monte-Carlo Tree Search (MCTS) [27] 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. We will first adapt GA and MCTS for searching over the space of input-trees, and then explore their performance for finding promising hierarchies in significantly more complex systems than investigated so far.

A. Genetic Algorithm

GA evolves a randomly generated initial population of candidates through mutation, selection and crossover to find promising candidates based on a fitness function. In our case, these candidates are input-trees. In section IV-A1, we provide a strategy to uniformly sample input-trees from the set of all possibilities. We then introduce a fitness function that balances the value error estimate with an estimate of the reduction in computation offered by a hierarchy (Sec. IV-A2), and operators for mutating the input-trees with closure (Sec. IV-A3). (Crossover operators did not markedly improve the search results, and we excluded them to minimize search time.) Finally, we add a hash table to the GA to avoid re-evaluating the fitness of previously seen candidates (Sec. IV-A4).

1) Uniform Sampling of Input-Trees: The sampling strategy is tightly linked to the process of constructing input-trees, 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 [39] in each step of the construction process to uniformly draw random input-trees.

For a system with m inputs and n state variables, we first sample the number of input groups in an input-tree, $r \in \{2, \cdots, m\}$, with probability proportional to the number of such input-trees (each entry in the outermost summation in Eq. 15). 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 probability proportional to the number of input-trees with r input groups and k leaf-nodes (each entry in the inner summation in Eq. 15). To sample the tree structure, we use Prufer codes [40]: A tree with N labelled nodes can be bijectively encoded with an $(N-2)$ character long sequence of its node labels called a Prufer code. Moreover, labels for the leaf-nodes are always absent from the encoding. We generate an encoding for an input-tree with k leaf-nodes by uniformly sampling a sequence of length $(r-1)$ with exactly $(r-k)$ distinct entries from $\{1, \cdots, r+1\}$. Finally, to assign state variables to different nodes, we uniformly sample a label for every variable from $\{1, \cdots, r\}$. Variables with label i are assigned to node i. We resample labels if an assignment is invalid i.e. no variables are assigned to leaf-nodes.

2) Fitness Function: Our fitness function for a hierarchy δ is the product of two components

$$F(\delta) = F_{\text{err}}(\delta) \times F_{\text{comp}}(\delta) \tag{13}$$

where $F_{\text{err}}(\delta)$ and $F_{\text{comp}}(\delta)$ quantify the suboptimality and the potential reduction in computation respectively. $F(\delta) \in [0, 1]$ with lower values indicating a more promising hierarchy. $F_{\text{err}}(\delta) = 1 - \exp(-\text{err}^{\delta}_{\text{lqr}})$ is the value error estimate (Sec. III-A, Eq. 6) scaled to the range $[0, 1]$. $F_{\text{comp}}(\delta)$ is the ratio of estimates of floating point operations required to compute the hierarchical and the optimal policies. We use look-up tables to represent policies and Policy Iteration [1] to compute them. Appendix B details how we evaluate $F_{\text{comp}}(\delta)$.

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

(i) Swap state variables between nodes,
(ii) Move one state variable from a node to another,
(iii) Move an entire sub-tree,
(iv) Couple two nodes into one,

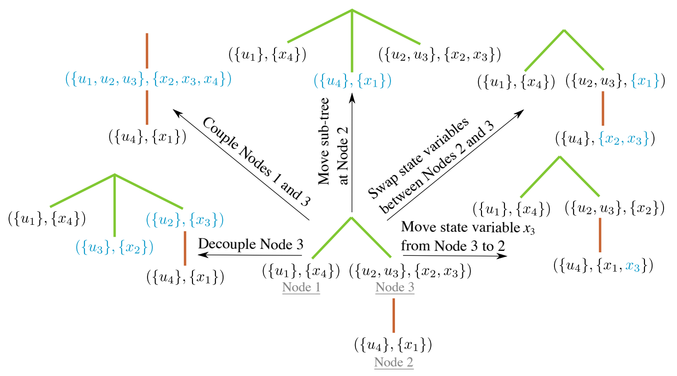


Fig. 11. Mutation operators described in section IV-A3 demonstrated on the input-tree T^{δ} depicted in figure 5.

(v) Decouple a node into two.

Figure 11 showcases examples of these operations and their outcome when applied to the input-tree depicted in figure 5. 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 none of them is applied, then two distinct inputs are randomly selected and, depending on whether they are decoupled or coupled, operators (iv) or (v) are applied, respectively.

4) Hashing Input-Trees: To avoid re-evaluating the fitness for previously seen candidates, we maintain an associative array [41]. This array maps a unique identifier or *key* for an input-tree to its computed fitness values. The key is composed of two binary matrices, $C^{m \times m}$ and $S^{m \times n}$, describing the input connectivity and state dependence of an input-tree. The j^{th} row in C has entries 1 for all inputs that either belong to the same node as u_j or its respective parent node. The j^{th} row in S corresponds to input u_j and has entries 1 for all state variables that belong to the same node as u_j. For example, for the input-tree in figure 5, these matrices are

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

B. Monte-Carlo Tree Search

MCTS takes a top-down approach, and creates a search tree of input-trees T^{δ} (Eq. 4) with the input-tree of the original, fully connected system at its root (Fig. 12). MCTS builds the search tree through continual rollouts, each representing a sequence of node expansions starting from the root and expanding until a terminal node is reached (Sec. IV-B1). After each rollout, a backup operation is performed, in which a value $Q(T^{\delta})$ for every node in the explored branch of the search tree is computed or updated (Sec. IV-B2). The value for an individual node is the best fitness (Eq. 13) found so far for the input trees T^{δ} in the MCTS sub-tree starting at that node. This value is used during subsequent rollouts to make an informed pick about further node expansions.

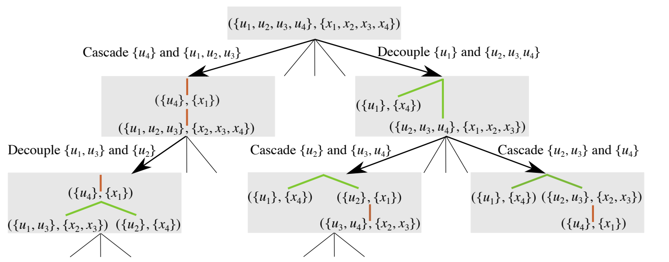

$(\{u_1,u_2,u_3,u_4\},\{x_1,x_2,x_3,x_4\})$
Cascade $\{u_4\}$ and $\{u_1,u_2,u_3\}$
Decouple $\{u_1\}$ and $\{u_2,u_3,u_4\}$
$(\{u_4\},\{x_1\})$
$(\{u_1,u_2,u_3\},\{x_2,x_3,x_4\})$
$(\{u_1\},\{x_4\})$
$(\{u_2,u_3,u_4\},\{x_1,x_2,x_3\})$
Decouple $\{u_1,u_3\}$ and $\{u_2\}$
Cascade $\{u_2\}$ and $\{u_3,u_4\}$
Cascade $\{u_2,u_3\}$ and $\{u_4\}$
$(\{u_4\},\{x_1\})$
$(\{u_1,u_3\},\{x_2,x_3\})$ $(\{u_2\},\{x_4\})$
$(\{u_1\},\{x_4\})$ $(\{u_2\},\{x_1\})$
$(\{u_3,u_4\},\{x_2,x_3\})$
$(\{u_1\},\{x_4\})$ $(\{u_2,u_3\},\{x_2,x_3\})$
$(\{u_4\},\{x_1\})$

Fig. 12. Example of a search tree generated from MCTS rollouts for a fictive system with four inputs and four state variables. Each node is an input-tree T^{δ} representing a hierarchy δ. Starting with the original system at the root node (top), subsequent layers represent more and more decomposed hierarchies (toward bottom) until no further decomposition is possible (leaf nodes).

1) Node Expansion: Expanding a tree node T^{δ} entails evaluating the fitness $F(\delta)$ (Eq. 13) of the corresponding hierarchy δ and enumerating possible children. These children are constructed by splitting the input and state variable sets into two for any leaf in T^{δ}, and arranging them in a cascaded or decoupled fashion (Fig. 12). To decide which children to pick for node expansion, we use the UCT strategy [42],

$$T^{\delta_{\text{expand}}} = \underset{T^{\delta'} \in \text{children}(T^{\delta})}{\text{argmin}} \; Q(T^{\delta'}) - \sqrt{\frac{2\ln(N_{T^{\delta}}+1)}{N_{T^{\delta'}}}} \tag{14}$$

where $Q(T^{\delta'})$ is initialized to $F(\delta')$, $N_{T^{\delta'}}$ denotes the number of times $T^{\delta'}$ was visited, and we break ties randomly. Additionally, to prevent redundant rollouts, we remove a node from consideration in the UCT strategy (Eq. 14) if all possible nodes reachable through it have been expanded.

2) Backup: At the end of each rollout, the values for the nodes visited during the rollout are updated as follows

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

starting from the terminal node and going towards the root of the search tree. The value of the root node is the fitness value of the best hierarchy identified in the search. Similar to GA, we use a hash table to avoid re-evaluating the fitness for previously evaluated input-trees.

C. Search Performance

We evaluate the search performance of GA and MCTS for three example problems common in robotics applications: balancing the simplified biped model introduced before (Fig. 7(c)), swinging up a 4 link planar manipulator (Fig. 13(a)), and hovering a quadcopter (Fig. 13(b)). Table V in appendix C details the dynamics and cost function parameters as well as the hyper-parameters for policy optimization that feature in the $F_{\text{comp}}(\delta)$ evaluation. Table IV reports the subsets of state-space over which the value errors are evaluated. As a baseline for search performance, we generate hierarchies through uniform random sampling (Sec. IV-A1). We run each search algorithm for a fixed time period (150, 600 and 1200 seconds for the biped, the manipulator, and the quadcopter, respectively) and note the hierarchy with the lowest fitness as well as the number of unique hierarchies discovered (Tab. I).

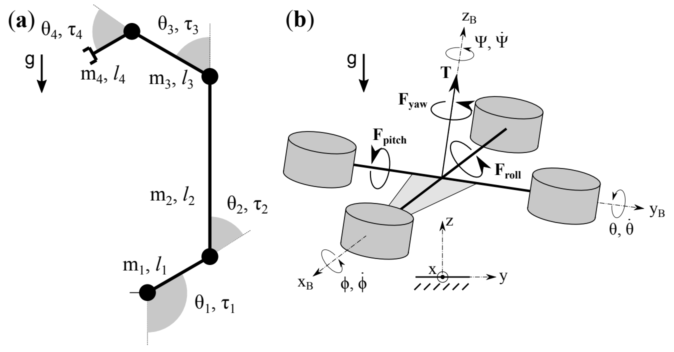


Fig. 13. Systems to test the efficacy of search methods in finding promising hierarchies: (a) 4 link planar manipulator and (b) quadcopter. Policies to swing-up the manipulator into an upright position, and to hover the quadcopter to rest at a fixed height are derived using the discovered hierarchies. Appendix C describes the system dynamics and the cost functions used.

Overall, GA performs consistently better than MCTS and random sampling. For all three example problems, GA discovers the largest number of unique control hierarchies and identifies hierarchies with the lowest fitness values. In the biped and manipulator problems, the suboptimality errors of these hierarchies are also significantly lower than those of the best hierarchies identified with MCTS or random sampling. In the quadcopter problem, all three search algorithms find hierarchies with negligible errors.

The GA search performed well not only in relation to MCTS and random sampling but also in absolute terms. For the biped and manipulator problems, we could evaluate the fitness for all possible hierarchies (396,716 and 7,147,628; respectively) by brute force enumeration. (With over 120 million possible hierarchies, this was not possible for the quadcopter problem.) The brute force evaluation took 2,433 seconds for the biped and 81,318 seconds for the manipulator problem. In comparison, GA evaluated only a fraction of the possible hierarchies (in a fraction of the time, compare Tab. I), but across all five runs identified the hierarchy with the *absolute* lowest fitness in either problem.

D. Evaluation of Identified Control Policies

To evaluate the identified control policies, we use two steps. First, we use Policy Iteration [1] with a look-up table representation to obtain the hierarchical controllers and evaluate their computation time. Second, to evaluate 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) [43] and Proximal Policy Optimization (PPO) [44], to obtain reference policies. In either case, we then simulate 100 closed-loop trajectories from the computed policies to evaluate the value error (Eq. 1) 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

TABLE I

SUMMARY OF SEARCH RESULTS. THE NUMBER OF UNIQUE HIERARCHIES DISCOVERED AS WELL AS 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 PROBLEMS, RESPECTIVELY.

	BIPED			MANIPULATOR			QUADCOPTER		
method	$\boldsymbol{F(\delta)}$ ($\times10^{-8}$)	$\textbf{err}^{\delta}_{\text{lqr}}$ ($\times10^{-2}$)	**$\#\delta$ found**	$\boldsymbol{F(\delta)}$ ($\times10^{-12}$)	$\textbf{err}^{\delta}_{\text{lqr}}$	**$\#\delta$ found**	$\boldsymbol{F(\delta)}$ ($\times10^{-20}$)	$\textbf{err}^{\delta}_{\text{lqr}}$ ($\times10^{-16}$)	**$\#\delta$ found**
GA	$\mathbf{4.89 \pm 0}$	$\mathbf{2.95 \pm 0}$	18604 ± 1037	$\mathbf{7.76 \pm 0}$	$\mathbf{0.0327 \pm 0}$	$\mathbf{58465 \pm 2549}$	$\mathbf{0.29 \pm 0.18}$	0.66 ± 0.38	$\mathbf{185797 \pm 5290}$
MCTS	6.36 ± 0.12	3.78 ± 0	11716 ± 171	1756 ± 334	4.17 ± 9.29	18962 ± 399	3.9 ± 4.7	2.33 ± 1.3	33882 ± 1404
RANDOM	13.8 ± 12.5	8.8 ± 8.4	$\mathbf{19684 \pm 447}$	2627 ± 791	4.61 ± 8.41	37499 ± 1341	833 ± 742	$\mathbf{0.31 \pm 0.22}$	70776 ± 1296

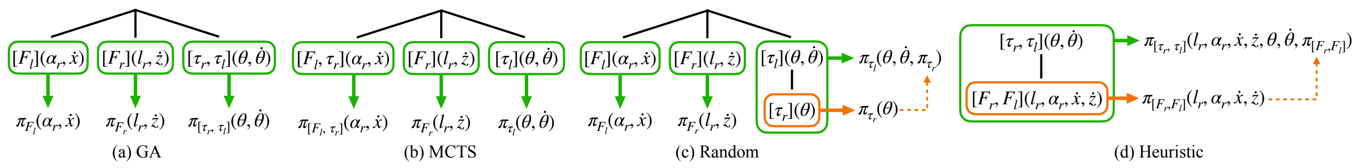

$[F_l](\alpha_r, \dot{x})$
$[F_r](l_r, \dot{z})$
$[\tau_r, \tau_l](\theta, \dot{\theta})$
$\pi_{F_l}(\alpha_r, \dot{x})$
$\pi_{F_r}(l_r, \dot{z})$
$\pi_{[\tau_r, \tau_l]}(\theta, \dot{\theta})$
(a) GA
$[F_l, \tau_r](\alpha_r, \dot{x})$
$[F_r](l_r, \dot{z})$
$[\tau_l](\theta, \dot{\theta})$
$\pi_{[F_l, \tau_r]}(\alpha_r, \dot{x})$
$\pi_{F_r}(l_r, \dot{z})$
$\pi_{\tau_l}(\theta, \dot{\theta})$
(b) MCTS
$[F_l](\alpha_r, \dot{x})$
$[F_r](l_r, \dot{z})$
$[\tau_l](\theta, \dot{\theta})$
$[\tau_r](\theta)$
$\pi_{\tau_l}(\theta, \dot{\theta}, \pi_{\tau_r})$
$\pi_{\tau_r}(\theta)$
$\pi_{F_l}(\alpha_r, \dot{x})$
$\pi_{F_r}(l_r, \dot{z})$
(c) Random
$[\tau_r, \tau_l](\theta, \dot{\theta})$
$[F_r, F_l](l_r, \alpha_r, \dot{x}, \dot{z})$
$\pi_{[\tau_r, \tau_l]}(l_r, \alpha_r, \dot{x}, \dot{z}, \theta, \dot{\theta}, \pi_{[F_r, F_l]})$
$\pi_{[F_r, F_l]}(l_r, \alpha_r, \dot{x}, \dot{z})$
(d) Heuristic

Fig. 14. Hierarchies for the simplified biped shown in figure 7(c). Hierarchies discovered by GA, MCTS and random sampling decouple the fore-aft control, the height regulation and the torso balance. The heuristic hierarchy is based on several reported works in the literature [31], [32].

closed-loop trajectories using the hierarchical, A2C and PPO policies, respectively (Tab. II).

1) Balance control of simplified biped: A common heuristic hierarchy for the balance control of a bipedal system [31], [32] is to regulate 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 to design a controller for the hip torques (τ_r and τ_l) in cascade to balance the torso (θ, $\dot{\theta}$). (Compare Sec. III-C and Fig. 7 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 (Fig. 14). This aggressive decoupling reduces the computation cost by an order of magnitude when compared to the heuristic hierarchy but it comes at the cost of clearly worse performance (biped columns, Tab. II).

2) Swing-up control for planar manipulator: The 4 link manipulator shown in figure 13(a) is similar in design to the 2 and 3 link ones presented earlier (Figs. 7(a) and (b)). (See Apx. 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 (Tab. II), a fully decoupled manipulator control (Fig. 15(a)). The best hierarchy that MCTS discovers is similar but retains a coupled control for the first two joints (Fig. 15(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 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 (Fig. 15(c)) and does not perform as well as the other two hierarchies.

3) Hover control of a quadcopter: With ten states and four inputs, the quadcopter shown in figure 13(b) is the most complex system studied in this paper. The quadcopter is described with its center of mass height (z) and velocity

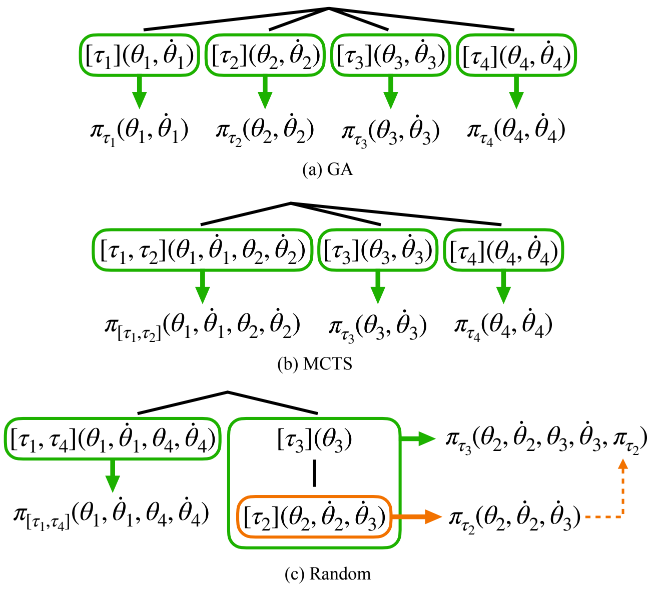

$[\tau_1](\theta_1, \dot{\theta}_1)$
$[\tau_2](\theta_2, \dot{\theta}_2)$
$[\tau_3](\theta_3, \dot{\theta}_3)$
$[\tau_4](\theta_4, \dot{\theta}_4)$
$\pi_{\tau_1}(\theta_1, \dot{\theta}_1)$
$\pi_{\tau_2}(\theta_2, \dot{\theta}_2)$
$\pi_{\tau_3}(\theta_3, \dot{\theta}_3)$
$\pi_{\tau_4}(\theta_4, \dot{\theta}_4)$
(a) GA
$[\tau_1, \tau_2](\theta_1, \dot{\theta}_1, \theta_2, \dot{\theta}_2)$
$[\tau_3](\theta_3, \dot{\theta}_3)$
$[\tau_4](\theta_4, \dot{\theta}_4)$
$\pi_{[\tau_1, \tau_2]}(\theta_1, \dot{\theta}_1, \theta_2, \dot{\theta}_2)$
$\pi_{\tau_3}(\theta_3, \dot{\theta}_3)$
$\pi_{\tau_4}(\theta_4, \dot{\theta}_4)$
(b) MCTS
$[\tau_1, \tau_4](\theta_1, \dot{\theta}_1, \theta_4, \dot{\theta}_4)$
$[\tau_3](\theta_3)$
$[\tau_2](\theta_2, \dot{\theta}_2, \dot{\theta}_3)$
$\pi_{\tau_3}(\theta_2, \dot{\theta}_2, \theta_3, \dot{\theta}_3, \pi_{\tau_2})$
$\pi_{[\tau_1, \tau_4]}(\theta_1, \dot{\theta}_1, \theta_4, \dot{\theta}_4)$
$\pi_{\tau_2}(\theta_2, \dot{\theta}_2, \dot{\theta}_3)$
(c) Random

Fig. 15. Hierarchies discovered by GA, MCTS and random sampling for the manipulator shown in figure 13(a).

($\dot{x}$, $\dot{y}$, $\dot{z}$), 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 Apx. 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 16. 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

TABLE II
COMPARISON OF POLICY COMPUTATION 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. APPENDIX C DETAILS THE CHOICE OF POLICY REPRESENTATION AND ALGORITHM FOR OPTIMIZATION.

	BIPED		MANIPULATOR			QUADCOPTER		
	time (sec)	**err**$^\delta$	**time** (sec)	**err**$^\delta_{\text{A2C}}$	**err**$^\delta_{\text{PPO}}$	**time** (sec)	**err**$^\delta_{\text{A2C}}$	**err**$^\delta_{\text{PPO}}$
π^*	12288	-	-	-	-	-	-	-
GA	**93.4**	0.28	**104.5**	$\mathbf{-0.025}$	$\mathbf{-0.098}$	30420	$\mathbf{-0.114}$	$\mathbf{-0.26}$
MCTS	103.2	0.33	465.9	-0.0235	-0.096	64061	0.22	0.02
RANDOM	109.1	0.34	465.2	-0.016	-0.09	-	-	-
HEURISTIC	1936.6	**0.01**	-	-	-	**6502**	-0.038	-0.20

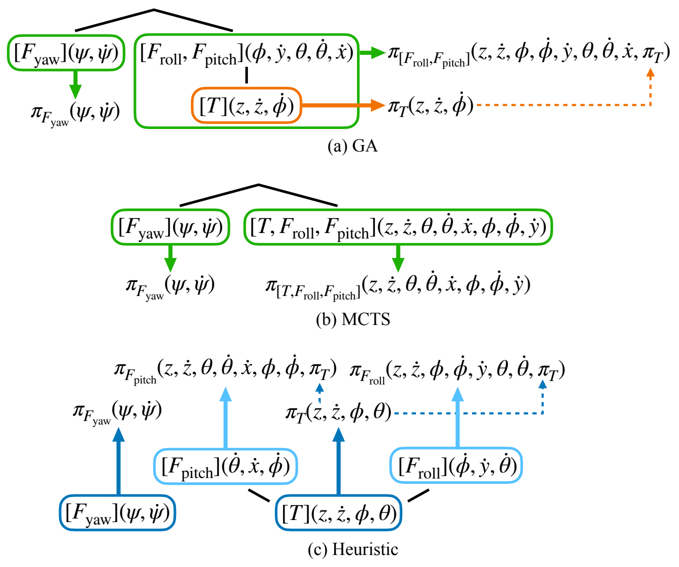

$[F_{\text{yaw}}](\psi, \dot\psi)$
$[F_{\text{roll}}, F_{\text{pitch}}](\phi, \dot y, \theta, \dot\theta, \dot x)$
$\pi_{[F_{\text{roll}},F_{\text{pitch}}]}(z, \dot z, \phi, \dot\phi, \dot y, \theta, \dot\theta, \dot x, \pi_T)$
$\pi_{F_{\text{yaw}}}(\psi, \dot\psi)$
$[T](z, \dot z, \dot\phi)$
$\pi_T(z, \dot z, \dot\phi)$
(a) GA
$[F_{\text{yaw}}](\psi, \dot\psi)$
$[T, F_{\text{roll}}, F_{\text{pitch}}](z, \dot z, \theta, \dot\theta, \dot x, \phi, \dot\phi, \dot y)$
$\pi_{F_{\text{yaw}}}(\psi, \dot\psi)$
$\pi_{[T,F_{\text{roll}},F_{\text{pitch}}]}(z, \dot z, \theta, \dot\theta, \dot x, \phi, \dot\phi, \dot y)$
(b) MCTS
$\pi_{F_{\text{pitch}}}(z, \dot z, \theta, \dot\theta, \dot x, \phi, \dot\phi, \pi_T)$
$\pi_{F_{\text{roll}}}(z, \dot z, \phi, \dot\phi, \dot y, \theta, \dot\theta, \pi_T)$
$\pi_{F_{\text{yaw}}}(\psi, \dot\psi)$
$\pi_T(z, \dot z, \phi, \theta)$
$[F_{\text{pitch}}](\dot\theta, \dot x, \dot\phi)$
$[F_{\text{roll}}](\dot\phi, \dot y, \dot\theta)$
$[F_{\text{yaw}}](\psi, \dot\psi)$
$[T](z, \dot z, \phi, \theta)$
(c) Heuristic

Fig. 16. Hierarchies discovered by GA and MCTS for the quadcopter shown in figure 13(b), as well as a heuristic hierarchy based on [45].

not report on it further. However, figure 16(c) depicts a common controller structure used to design Integral Backstepping Control for the quadcopter [45]. We designate this structure as a heuristic hierarchy[1] and compare its performance with the algorithmically identified ones.

As for the previous problems, the hierarchy discovered with GA offers the best closed-loop performance (Tab. II). 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). In contrast, the control policy derived from the GA hierarchy provides significant improvements in closed-loop performance over both the policies identified with A2C and PPO and the heuristic hierarchy policy, highlighting the utility of the policy decomposition search framework.

E. *Influence of System Representation on Search Performance*

System representation is important to constructing promising hierarchies with policy decomposition. The quadcopter problem highlights this point. We chose the commonly used inputs of net thrust and roll, pitch and yaw differential thrusts. However, these inputs are linear combinations of the actual quadcopter motor thrusts F_i: $F_{\text{roll}} = F_4 - F_2$, $F_{\text{pitch}} = F_3 - F_1$ and $F_{\text{yaw}} = F_2 + F_4 - F_1 - F_3$. Had we instead used these motor thrusts as the inputs, the hierarchies discovered through search would hardly provide any reduction in computation time, as the system cannot readily be decomposed. Identifying favorable input and state representations of a system is therefore critical to the performance of policy decomposition, and in [46] we present initial steps to do so in a principled way.

F. *Bi-objective Search*

The search for promising hierarchies is a bi-objective optimization with suboptimality and computation time as the two competing goals. The fitness function (Eq. 13) reduces this optimization to a single score and can thereby introduce bias toward one of the two goals. For example, in the biped problem, 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, the suboptimality criterion dominates the fitness. To avoid such bias, an alternative strategy is to find the Pareto optimal set of hierarchies, analyze their trade-off in computation time and suboptimality, and then choose the most suitable hierarchy based on design constraints on closed-loop control performance and available computational resources.

Figure 17 depicts the Pareto front of hierarchies for the biped and quadcopter. In both cases, we used the NSGA-II algorithm [47] with the adaptations specific to GA introduced in section IV-A to construct the front with the competing criteria defined as $F_{\text{err}}(\delta)$ and $F_{\text{comp}}(\delta)$ (Eq. 13). 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 17, 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

[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.

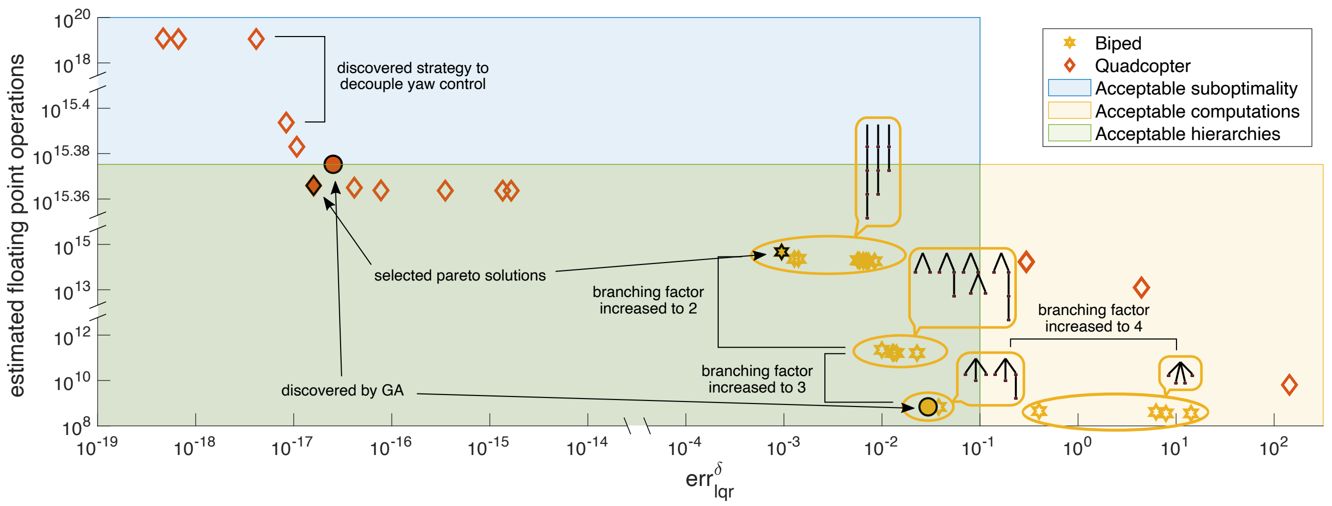


Fig. 17. Pareto fronts of hierarchies for the biped and the quadcopter. Hierarchies towards the top-left exhibit low suboptimality but require more floating point operations to compute control policies whereas those towards the bottom-right are highly suboptimal but offer dramatic reduction in computation. Discernible structural changes in the hierarchies for the biped and the quadcopter are highlighted.

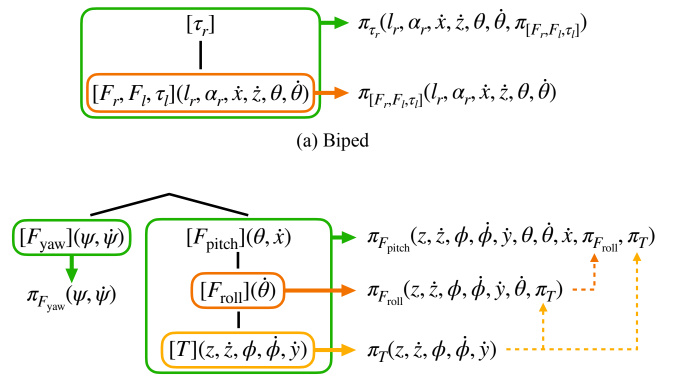

$[\tau_r]$
$[F_r, F_l, \tau_l](l_r, \alpha_r, \dot{x}, \dot{z}, \theta, \dot{\theta})$
$\pi_{\tau_r}(l_r, \alpha_r, \dot{x}, \dot{z}, \theta, \dot{\theta}, \pi_{[F_r,F_l,\tau_l]})$
$\pi_{[F_r,F_l,\tau_l]}(l_r, \alpha_r, \dot{x}, \dot{z}, \theta, \dot{\theta})$
(a) Biped
$[F_{\mathrm{yaw}}](\psi, \dot{\psi})$
$\pi_{F_{\mathrm{yaw}}}(\psi, \dot{\psi})$
$[F_{\mathrm{pitch}}](\theta, \dot{x})$
$[F_{\mathrm{roll}}](\dot{\theta})$
$[T](z, \dot{z}, \phi, \dot{\phi}, \dot{y})$
$\pi_{F_{\mathrm{pitch}}}(z, \dot{z}, \phi, \dot{\phi}, \dot{y}, \theta, \dot{\theta}, \dot{x}, \pi_{F_{\mathrm{roll}}}, \pi_T)$
$\pi_{F_{\mathrm{roll}}}(z, \dot{z}, \phi, \dot{\phi}, \dot{y}, \dot{\theta}, \pi_T)$
$\pi_T(z, \dot{z}, \phi, \dot{\phi}, \dot{y})$
(b) Quadcopter

Fig. 18. Hierarchies for the biped (Fig. 7(c)) and the quadcopter (Fig. 13(b)) selected from the Pareto fronts depicted in figure 17.

TABLE III
POLICY COMPUTATION TIMES AND VALUE ERROR ESTIMATES FOR THE PARETO OPTIMAL HIERARCHIES FOR THE BIPED AND QUADCOPTER DEPICTED IN FIGURE 18.

BIPED		QUADCOPTER		
time (sec)	$\mathbf{err}^{\delta}$	**time** (sec)	$\mathbf{err}^{\delta}_{\mathrm{A2C}}$	$\mathbf{err}^{\delta}_{\mathrm{PPO}}$
8220	0.0005	12725	-0.085	-0.236

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 17. The input-trees for these highlighted hierarchies are shown in figure 18, and the required time for policy optimization as well as the value error estimates for the resulting policies are reported in table III. For the quadcopter, the highlighted hierarchy reduces policy computation time by more than half in comparison to the hierarchy with the lowest fitness found previously by GA (compare quadcopter columns in Tabs. II and III) 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 Tabs. II and III).

In general, 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 seconds for the biped problem and 1200 seconds for the quadcopter problem).

V. CONCLUSION AND FUTURE WORK

Policy decomposition [25] is a novel framework for approximately solving intractable optimal control problems of complex dynamical systems. Similar to other hierarchical control methods, it achieves significant reduction in computational costs by composing a control policy for the entire system from optimal controllers of tractable sub-systems. What stands out about policy decomposition is that it incorporates *apriori* estimates of suboptimality, which enables this framework to discover hierarchies that can sharply reduce the computational cost without giving up much closed-loop performance. This advantage does not come for free, however, as the framework faces a combinatorial challenge. Even for a system with moderate numbers of states and inputs, the number of possible hierarchies is staggering, and computing the suboptimality estimates for all hierarchies itself becomes intractable.

Here, we investigated if two common methods for combinatorial search, Genetic Algorithm (GA) [26] and Monte Carlo Tree Search (MCTS) [27], can be adapted to discover promising hierarchies in the vast combinatorial space of all possible ones. To this end, we introduced input-trees as intuitive abstractions for hierarchies generated by policy decomposition (Sec. III), identified a way to enumerate all possible hierarchies based on these abstractions, and adapted GA and MCTS to manipulate them while maintaining closure (Sec. IV). Applying these adapted search methods to biped, manipulator, and quadcopter control problems, we found that GA in particular discovers promising hierarchies, which generate control policies that perform better than common heuristic policies and policies obtained with popular neural network learning strategies (A2C [43] and PPO [44]). Our results suggest that the combinatorial challenge which policy decomposition is facing does not thwart its utility.

Three future research directions would further broaden the utility of the policy decomposition framework. First, all the control problems addressed so far with this framework are regulator problems. Yet many practical applications require trajectory tracking; thus, extending policy decomposition to handle optimal trajectory tracking would largely increase its utility. Second, to incorporate the effects of sensing and actuation noise in the evaluation of the different control hierarchies. Finally, as discussed for the quadcopter example (Sec. IV-E), the quality of discovered hierarchies depends on state and input representations, and although we have made some progress [46], a more general approach to identifying representations that lead to better hierarchies could further improve the quality of the control policies obtained with the policy decomposition framework.

ACKNOWLEDGMENTS

We thank Christopher Atkeson for suggesting the use of DDP for estimating value errors, and Arpit Agarwal for his help with the experiments. This work was supported by NSF grant #1563807. We used the Bridges2 GPU nodes at the Pittsburgh Supercomputing Cluster through allocation CIS220132 from the Advanced Cyberinfrastructure Coordination Ecosystem: Services and Support (ACCESS) program, which is supported by NSF grants #2138259, #2138286, #2138307, #2137603, and #2138296.

APPENDIX A
COUNTING THE NUMBER OF POSSIBLE HIERARCHIES

Since any hierarchy can be represented using a unique input-tree, 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 can be constructed as follows

(I) Group the m inputs into r non-empty subsets.

(II) Arrange the r input subsets into input-trees such that k of these subsets are leaves[2].

(III) Distribute the n state variables into r groups such that the groups corresponding to the leafs 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

$$\sum_{r=2}^{m} \mathbf{I}(m,r) \sum_{k=1}^{r} \mathbf{II}(r,k)\mathbf{III}(n,r,k) \tag{15}$$

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

$$\mathbf{I}(m,r) = \frac{\Delta(m,r)}{r!}$$

ways [48], where $\Delta(m,r)$ is as follows

$$\begin{aligned}\Delta(m,r) =& r^m - \binom{r}{1}(r-1)^m + \binom{r}{2}(r-2)^m \\ &+ \cdots + (-1)^{r-1}\binom{r}{r-1}(1)^m\end{aligned} \tag{16}$$

(II) Any input-tree structure made up of r input subsets has $(r+1)$ nodes (including the root node), and can be bijectively encoded with a sequence of length $(r-1)$ consisting of labels for the $(r+1)$ nodes. Such an encoding is called a Prüfer code [40]. Moreover, the **only labels missing** from the encoding **are labels of the leaves**. If k of the input subsets are selected to be assigned to leaf-nodes ($\binom{r}{k}$ possible ways to make this selection), then depending on whether the root is a leaf

a) If root node **is** a leaf, the encoding consists of only the remaining $(r-k)$ labels, each of which must appear at least once: $\Delta(r-1, r-k)$ possibilities (Eq. (16)).

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

Thus the number of possible input-trees made up of r input subsets, k of which belong to leaf-nodes, are[3]

$$\begin{aligned}\mathbf{II}(r,k) = \binom{r}{k}\Big(&\Delta\big(r-1, r-k\big) \\ &+ \Delta\big(r-1, r-k+1\big)\Big)\end{aligned}$$

(III) Valid state assignment for an input-tree consisting of r input subsets, k of which belong to leaf-nodes, is obtained by assigning $i \in \{0, \cdots, n-k\}$ of the n variables to the $(r-k)$ non-leaf-nodes, and distributing the remaining $(n-i)$ into k non-empty groups corresponding to the k leaf-nodes. Number of possibilities

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

APPENDIX B
ESTIMATES FOR REDUCTION IN POLICY COMPUTATION

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) [1]. A look-up table based policy stores control values corresponding to a uniform grid over the state-space. For

[2]Leaf-nodes have a degree of 1. If the root node has a single child then the root node is also a leaf.

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

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

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

$$\#\text{operations}_{\text{evaluate}} = \prod_{i=1}^{n} \text{NS}_i \Big(\underbrace{2^n\text{ME}}_{\text{interpolate}} + \underbrace{4n}_{\text{step}} \Big)$$

where n is the number of state variables, NS_i is the size of the look-up table along state dimension i, and ME is the maximum number of evaluation iterations. 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

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

$$\#\text{operations}_{\text{update}} = \prod_{i=1}^{n} \text{NS}_i \prod_{j=1}^{m} \text{NA}_j \Big(\underbrace{2^n}_{\text{interpolate}} + \underbrace{4n}_{\text{step}} + \underbrace{3}_{\substack{\text{sample action+}\\ \text{update value}}} \Big)$$

where $\boldsymbol{u}_{\text{min}}$ and $\boldsymbol{u}_{\text{max}}$ 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.

Appendix C
System Descriptions

The dynamics for all the systems presented in this work can be found in standard dynamics textbooks such as [49], barring the biped shown in figure 7(c) which we describe here: The legs are massless and contact 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 ($\tau_{l/r}$). $m\ddot{x} = F_r\cos\alpha_r + \frac{\tau_r}{l_r}\sin\alpha_r + F_l\cos\alpha_l + \frac{\tau_l}{l_l}\sin\alpha_l$, $m\ddot{z} = F_r\sin\alpha_r - \frac{\tau_r}{l_r}\cos\alpha_r + F_l\sin\alpha_l - \frac{\tau_l}{l_l}\cos\alpha_l - mg$, and $I\ddot{\theta} = \tau_r(1+\frac{d}{l_r}\sin(\alpha_r-\theta)) + F_r d\cos(\alpha_r-\theta) + \tau_l(1+\frac{d}{l_l}\sin(\alpha_l-\theta)) + F_l d\cos(\alpha_l-\theta)$, where $l_l = \sqrt{l_r^2 + d_f^2 + 2l_r d_f\cos\alpha_r}$ and $\alpha_l = \arcsin\frac{l_r\sin\alpha_r}{l_l}$. The dynamics and cost function parameters for the optimal control problems showcased in this work are reported in table V. All policies are represented as look-up tables over the state-space ($\mathcal{S}_{\text{full}}$) and we use Policy Iteration (PI) [1] to compute them. 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 B are also reported in table V. 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 IV. For $\text{err}^{\delta}_{\text{ddp}}$ calculations, we compute trajectories starting from the corners of set $\mathcal{S}$, over a horizon of $T = 4$s with $dt = 1$ms

TABLE IV
Regions of state-space ($\mathcal{S} \subset \mathcal{S}_{\text{full}}$) over which value errors are computed for different systems presented in this work

	$\mathcal{S}$
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	$\theta_1 \in [2\pi/3, 4\pi/3],\ \theta_2 \in [-\pi/3, \pi/3],$ $\dot{\theta}_1, \dot{\theta}_2 \in [-0.5, 0.5]$
3 DOF	$\theta_1 \in [2\pi/3, 4\pi/3], \theta_2, \theta_3 \in [-\pi/3, \pi/3],$ $\dot{\theta}_1, \dot{\theta}_2, \dot{\theta}_3 \in [-0.5, 0.5]$
4 DOF	$\theta_1 \in [3\pi/5, 7\pi/5],\ \theta_2, \theta_3, \theta_4 \in [-2\pi/5, 2\pi/5],$ $\dot{\theta}_1, \dot{\theta}_2, \dot{\theta}_3, \dot{\theta}_4 \in [-0.6, 0.6]$
Quadcopter	$z \in [0.7, 1.3],\ \phi, \theta \in [-0.7, 0.7],\ \psi \in [-\pi/4, \pi/4],$ $\dot{x}, \dot{y} \in [-1, 1],\ \dot{z} \in [-0.75, 0.75],$ $\dot{\phi}, \dot{\theta} \in [-0.5, 0.5],\ \dot{\psi} \in [-0.25, 0.25]$

For the 4 link manipulator and the quadcopter, we use the Stable Baselines implementation [50] for Advantage Actor Critic (A2C) [43] and Proximal Policy Optimization (PPO) [44] 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

References

[1] D. P. Bertsekas, *Dynamic programming and optimal control.* Belmont, MA: Athena Scientific, 1995, vol. 1, no. 2.

[2] J. Si, A. G. Barto, W. B. Powell, and D. Wunsch, *Handbook of Learning and Approximate Dynamic Programming (IEEE Press Series on Computational Intelligence).* Wiley-IEEE Press, 2004.

[3] A. C. Antoulas, *Approximation of large-scale dynamical systems.* SIAM, 2005.

[4] R. R. Burridge, A. A. Rizzi, and D. E. Koditschek, "Sequential composition of dynamically dexterous robot behaviors," *The International Journal of Robotics Research*, vol. 18, no. 6, pp. 534–555, 1999. [Online]. Available: https://doi.org/10.1177/02783649922066385

[5] D. Precup, *Temporal abstraction in reinforcement learning.* University of Massachusetts Amherst, 2000.

[6] A. A. Gorodetsky, S. Karaman, and Y. M. Marzouk, "Efficient high-dimensional stochastic optimal motion control using tensor-train decomposition." in *Robotics: Science and Systems.* Citeseer, 2015.

[7] R. Tedrake, I. R. Manchester, M. Tobenkin, and J. W. Roberts, "Lqr-trees: Feedback motion planning via sums-of-squares verification," *The International Journal of Robotics Research*, vol. 29, no. 8, pp. 1038–1052, 2010.

[8] C. G. Atkeson and C. Liu, "Trajectory-based dynamic programming," in *Modeling, simulation and optimization of bipedal walking.* Berlin, Heidelberg: Springer, 2013, pp. 1–15.

[9] M. Zhong, M. Johnson, Y. Tassa, T. Erez, and E. Todorov, "Value function approximation and model predictive control," in *IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning*. IEEE, April 2013, pp. 100–107.
[10] J. Murray, C. Cox, G. Lendaris, and R. Saeks, "Adaptive dynamic programming," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 32, no. 2, pp. 140–153, 2002.
[11] D. Bertsekas and J. N. Tsitsiklis, *Neuro-dynamic programming*. Athena Scientific, 1996.
[12] A. Al-Tamimi, F. L. Lewis, and M. Abu-Khalaf, "Discrete-time nonlinear hjb solution using approximate dynamic programming: Convergence proof," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 38, no. 4, pp. 943–949, 2008.
[13] M. Stilman, C. G. Atkeson, J. J. Kuffner, and G. Zeglin, "Dynamic programming in reduced dimensional spaces: Dynamic planning for robust biped locomotion," in *International Conference on Robotics and Automation*. IEEE, 2005, pp. 2399–2404.
[14] E. C. Whitman and C. G. Atkeson, "Control of instantaneously coupled systems applied to humanoid walking," in *2010 10th IEEE-RAS International Conference on Humanoid Robots*. IEEE, 2010, pp. 210–217.
[15] O. Goury and C. Duriez, "Fast, generic, and reliable control and simulation of soft robots using model order reduction," *IEEE Transactions on Robotics*, vol. 34, no. 6, pp. 1565–1576, 2018.
[16] S. Lall, J. E. Marsden, and S. Glavaški, "A subspace approach to balanced truncation for model reduction of nonlinear control systems," *International Journal of Robust and Nonlinear Control*, vol. 12, no. 6, pp. 519–535, 2002. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/rnc.657
[17] E. Bristol, "On a new measure of interaction for multivariable process control," *IEEE Transactions on Automatic Control*, vol. 11, no. 1, pp. 133–134, 1966.
[18] P. Grosdidier and M. Morari, "Interaction measures for systems under decentralized control," *Automatica*, vol. 22, no. 3, pp. 309–319, 1986.
[19] T. Mc Avoy, Y. Arkun, R. Chen, D. Robinson, and P. D. Schnelle, "A new approach to defining a dynamic relative gain," *Control Engineering Practice*, vol. 11, no. 8, pp. 907–914, 2003.
[20] J. Bao, K. H. Chan, W. Z. Zhang, and P. L. Lee, "An experimental pairing method for multi-loop control based on passivity," *Journal of Process Control*, vol. 17, no. 10, pp. 787–798, 2007.
[21] B. Wittenmark and M. E. Salgado, "Hankel-norm based interaction measure for input-output pairing," *IFAC Proceedings Volumes*, vol. 35, no. 1, pp. 429–434, 2002.
[22] M. E. Salgado and A. Conley, "Mimo interaction measure and controller structure selection," *International Journal of Control*, vol. 77, no. 4, pp. 367–383, 2004.
[23] V. R. Saksena, J. O'reilly, and P. V. Kokotovic, "Singular perturbations and time-scale methods in control theory: survey 1976–1983," *Automatica*, vol. 20, no. 3, pp. 273–293, 1984.
[24] Y.-M. Chen and M. Posa, "Optimal reduced-order modeling of bipedal locomotion," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2020, pp. 8753–8760.
[25] A. Khadke and H. Geyer, "Policy decomposition: Approximate optimal control with suboptimality estimates," in *IEEE-RAS Humanoids*, 2020.
[26] M. Mitchell, *An introduction to genetic algorithms*. MIT press, 1998.
[27] 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*, vol. 4, no. 1, 2012.
[28] R. E. Kalman *et al.*, "Contributions to the theory of optimal control," *Bol. soc. mat. mexicana*, vol. 5, no. 2, pp. 102–119, 1960.
[29] Y. Tassa, N. Mansard, and E. Todorov, "Control-limited differential dynamic programming," in *IEEE International Conference on Robotics and Automation*. IEEE, 2014, pp. 1168–1175.
[30] 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*, vol. 45, no. 2, pp. 165–176, 2015.
[31] W. C. Martin, A. Wu, and H. Geyer, "Experimental evaluation of deadbeat running on the atrias biped," *IEEE Robotics and Automation Letters*, vol. 2, no. 2, pp. 1085–1092, 2017.
[32] K. Green, Y. Godse, J. Dao, R. Hatton, A. Fern, and J. Hurst, "Learning spring mass locomotion: Guiding policies with a reduced-order model," *IEEE Robotics and Automation Letters*, vol. PP, pp. 1–1, 03 2021.
[33] Y. Mao, M. Szmuk, and B. 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. [Online]. Available: https://doi.org/10.1109%2Fcdc.2016.7798816
[34] F. Glover and M. Laguna, *Tabu Search*. Boston, MA: Springer, 1998.
[35] C. Blum and A. Roli, "Metaheuristics in combinatorial optimization: Overview and conceptual comparison," *ACM Comput. Surv.*, vol. 35, no. 3, Sep. 2003.
[36] C. D. Chapman, K. Saitou, and M. J. Jakiela, "Genetic Algorithms as an Approach to Configuration and Topology Design," *J. Mech. Des*, vol. 116, no. 4, Dec. 1994.
[37] K. Stanley and R. Miikkulainen, "Evolving neural networks through augmenting topologies," *Evol. Comput.*, vol. 10, no. 2, 2002.
[38] B. Bouzy and T. Cazenave, "Computer go: An ai oriented survey," *Artificial Intelligence*, vol. 132, no. 1, 2001.
[39] M. H. Hansen and W. N. Hurwitz, "On the theory of sampling from finite populations," *The Annals of Mathematical Statistics*, vol. 14, no. 4, pp. 333–362, 1943. [Online]. Available: http://www.jstor.org/stable/2235923
[40] N. Biggs, E. K. Lloyd, and R. J. Wilson, *Graph Theory, 1736-1936*. Oxford University Press, 1986.
[41] M. T. Goodrich, R. Tamassia, and M. H. Goldwasser, *Data Structures and Algorithms in Java*, 6th ed. Wiley Publishing, 2014.
[42] L. Kocsis and C. Szepesvári, "Bandit based monte-carlo planning," in *ECML*. Berlin, Heidelberg: Springer, 2006.
[43] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *International conference on machine learning*. PMLR, 2016, pp. 1928–1937.
[44] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.
[45] S. Bouabdallah and R. Siegwart, "Full control of a quadrotor," in *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2007, pp. 153–158.
[46] A. Khadke and H. Geyer, "Sparsity inducing system representations for policy decompositions," in *2022 IEEE 61st Conference on Decision and Control (CDC)*, 2022, pp. 6824–6829.
[47] K. Deb, "Multi-objective optimisation using evolutionary algorithms: an introduction," in *Multi-objective evolutionary optimisation for product design and manufacturing*. Springer, 2011.
[48] R. P. Stanley, "What is enumerative combinatorics?" in *Enumerative combinatorics*. Boston, MA: Springer, 1986.
[49] R. M. Murray, Z. Li, and S. S. Sastry, *A mathematical introduction to robotic manipulation*. CRC press, 2017.
[50] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann, "Stable-baselines3: Reliable reinforcement learning implementations," *Journal of Machine Learning Research*, vol. 22, no. 268, pp. 1–8, 2021. [Online]. Available: http://jmlr.org/papers/v22/20-1364.html

Ashwin Khadke received a B.Tech. in mechanical engineering from the Indian Institute of Technology Bombay, Mumbai, India in 2016. He received an MS in robotics in 2018 and is currently a Ph.D. candidate at the Robotics Institute at Carnegie Mellon University, Pittsburgh, PA, USA.

His research interests include optimal control, legged locomotion and rigid body simulations.

Hartmut Geyer (M'13) received the Dipl. degree in physics and the Ph.D. degree in biomechanics from the Friedrich-Schiller-University of Jena, Jena, Germany, in 2001 and 2005, respectively. He is currently an Associate Professor with the Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, USA. His research interests include the principles of legged dynamics and control, their relation to human motor control, and resulting applications in humanoid and rehabilitation robotics.

TABLE V
DYNAMICS, COST FUNCTION AND HYPER-PARAMETERS FOR POLICY OPTIMIZATION OF ALL SYSTEMS PRESENTED IN THIS WORK

	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) $\mathcal{S}_{\text{full}}$: $l_r \in [0.85, 1.25]$, $(\alpha_r - \pi/2) \in [0, 0.6]$, $\dot{x} \in [-0.3, 0.5]$, $\dot{z} \in [-0.5, 1]$, $\theta \in [-\pi/8, \pi/8]$, $\dot{\theta} \in [-2, 2]$
	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 Link 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 Link Manipulator	Dynamics	$[m_1, m_2, m_3] = [2.75, 0.55, 0.11]$kg, $g = 9.81$m/s^2, $[l_1, l_2, l_3] = [0.5, 0.25, 0.125]$m
	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 Link Manipulator	Dynamics	$[m_1, m_2, m_3, m_4] = [5.4, 1.8, 0.6, 0.2]$kg, $g = 9.81$m/s^2, $[l_1, l_2, l_3, l_4] = [0.2, 0.5, 0.25, 0.125]$m
	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 : $0.0015 \xrightarrow{\text{20M}} 0$ **PPO** : Layers $= [256, 256]$, Learning rate : $0.0005 \xrightarrow{\text{20M}} 0$
Quadcopter	Dynamics	$m = 0.5$kg, $I = 10^{-3}\text{diag}([4.86, 4.86, 8.8])$kgm^2, $g = 9.81$m/s^2, $l = 0.225$m, $k_M = 0.0383$m
	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) $\mathcal{S}_{\text{full}}$: $z \in [0.5, 1.5]$, $\phi, \theta \in [-0.7, 0.7]$, $\psi \in [-\pi, \pi]$, $\dot{x}, \dot{y} \in [-2, 2]$, $\dot{z} \in [-1.5, 1.5]$, $\dot{\phi}, \dot{\theta} \in [-6, 6]$, $\dot{\psi} \in [-2.5, 2.5]$
	Cost	$\boldsymbol{Q} = \text{diag}([5, \text{1e-3}, \text{1e-3}, 5, 0.5, 0.5, 0.05, 0.075, 0.075, 0.05])$, $\boldsymbol{R} = \text{diag}([\text{2e-3}, 0.01, 0.01, \text{4e-3}])$, $\lambda = 3$
	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 : $0.0015 \xrightarrow{\text{20M}} 0$ **PPO** : Layers $= [256, 256]$, Learning rate : $0.0005 \xrightarrow{\text{20M}} 0$