Motor Control and Strategy Discovery for Physically Simulated Characters

by

Zhiqi Yin

B.Sc., Simon Fraser University, 2020

Thesis Submitted in Partial Fulfillment of the
Requirements for the Degree of
Master of Science

in the
Department of Computing Science
Faculty of Applied Science

SIMON FRASER UNIVERSITY
Summer 2021

Declaration of Committee

Name:	**Zhiqi Yin**
Degree:	**Master of Science**
Thesis title:	**Motor Control and Strategy Discovery for Physically Simulated Characters**
Committee:	**Chair:** Yasutaka Furukawa Associate Professor, Computing Science
	KangKang Yin Supervisor Associate Professor, Computing Science
	Hang Ma Committee Member Assistant Professor, Computing Science
	Mo Chen Examiner Assistant Professor, Computing Science

Abstract

In physics-based character animation, motions are realized through control of simulated characters along with their interactions with the virtual environment. In this thesis, we study the problem of character control on two levels: joint-level motor control which transforms control signals to joint torques, and high-level motion control which outputs joint-level control signals given the current state of the character and the environment and the task objective. We propose a Modified Articulated-Body Algorithm (MABA) which achieves stable proportional-derivative (PD) low-level motor control with superior theoretical time complexity, practical efficiency and stability than prior implementations. We further propose a high-level motion control framework based on deep reinforcement learning (DRL) which enables the discovery of appropriate motion strategies without human demonstrations to complete a task objective. To facilitate the learning of realistic human motions, we propose a Pose Variational Autoencoder (P-VAE) to constrain the DRL actions to a subspace of natural poses. Our learning framework can be further combined with a sample-efficient Bayesian Diversity Search (BDS) algorithm and novel policy seeking to discover diverse strategies for tasks with multiple modes, such as various athletic jumping tasks.

Keywords: physics-based character animation; motor control; deep reinforcement learning; Bayesian optimization; variational autoencoder; motion strategy; proportional derivative control

Acknowledgements

I would like to express my sincere gratitude to my supervisor, Dr. KangKang Yin, who gave me the opportunity to study and do research in Simon Fraser University. She provided me with guidance and assisted me a lot throughout this research. Also, I would like to express my deep gratefulness to my parents, who supported my pursuit of graduate studies.

Table of Contents

List of Tables

List of Figures

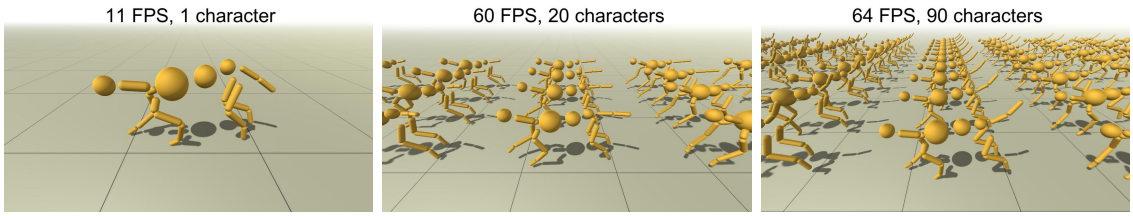
Chapter 1

Introduction

Physics-based character animation has made significant progresses in recent years. High quality controllers and skills can now be learned in real-time to generate motions that are indistinguishable from motion capture data [130, 88, 8, 87, 122]. Comparing with kinematic motion synthesis, physics-based methods achieve desired motions through control of simulated characters in virtual environments. Physically plausible responses to environmental perturbations can therefore be automatically generated without the need to capture and process additional motion data. However, the design of controllers for under-actuated characters given certain objectives remains a key challenge in physics-based methods.

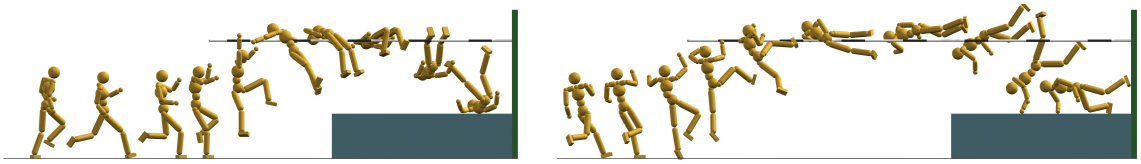
The challenge of controller design for physics-based characters is twofold. First, the simulation time for physics-based characters still remains one of the bottlenecks for efficient controller learning. State-of-the-art Deep Reinforcement Learning (DRL) based algorithms still require hours to days to learn motor skills successfully. For a fixed-duration motion sequence, wall clock simulation time depends highly on the choice of joint-level controllers. Vanilla Proportional-Derivative (PD) joint servos are widely adopted for conventional handcrafted or sampling-based full-body motion controllers like SIMBICON [127] and SAMCON [73]. PD servos are simple to implement and induce negligible computational cost during simulation. However, extremely small simulation time steps are usually required for controlling complex character models due to numerical instabilities of simple PD servos. Directly controlling joint torques is another straightforward option but has been shown to degrade learning performance and increase learning time for modern DRL-based motion controllers [93]. Stable PD (SPD) controllers proposed in [115] has become the default choice for modern physics-based character animation systems for its superior numerical stability that supports significantly larger simulation time steps [88, 90, 89, 104, 87, 61, 10]. However, the stability of existing SPD implementations comes with a cubic $O(n^3)$ computational cost in each step, where n is the number of degrees of freedom of the character model. As a result, SPD computation becomes the simulation bottleneck quickly as model complexity increases.

To mitigate this problem, we propose a fast and practical SPD computation algorithm for articulations parameterized in generalized coordinates. In particular, we derive a Modified



(a) PD (b) Cubic-time SPD (c) Our linear-time SPD

Figure 1.1: Performance comparison using a dog model tracking a canter motion with different motor controllers.



(a) Fosbury Flop – max height=200cm (b) Western Roll – max height=195cm

Figure 1.2: Two of the eight high jump strategies discovered by our strategy discovery framework, as achieved by physics-based control policies.

Articulated-Body Algorithm (MABA) based on Featherstone's Articulated-Body Algorithm (ABA) for forward dynamics [28]. The proposed algorithm computes SPD controls in worst case $O(n)$ time. As shown in Figure 1.1, our linear-time algorithm enables simulation and control of many more characters than PD and cubic-time SPD. We detail the derivation of MABA in Chapter 2 and demonstrate its performance advantage over conventional SPD implementations. We further show that our algorithm provides superior stability for controlling complex character models at large time steps, and improves the training speed and quality of a modern DRL-based motor skill learning system.

Given a simulated character in a virtual environment and a task objective, another long-standing challenge in physics-based character animation is to learn motion controllers to complete the task with natural-looking movements. Most existing methods learn character motion controllers by leveraging motion capture examples, where a mocap imitation objective is usually required during controller optimization to ensure high quality results. For example, DeepMimic [88] learns physics-based skills in a DRL framework, where an imitation reward and a task reward are combined together to learn skills. The learned controllers can thus complete the task while being similar to specified motion capture clips. Even though such imitation-based methods have demonstrated their effectiveness on achieving task-related goals, the imitation objective inherently restricts them from generalizing to novel motion strategies fundamentally different from the reference.

In this thesis, we propose a framework for discovering motion strategies without motion examples of the specific tasks. Our framework is based on DRL and takes the task goal as its main objective. To facilitate the learning of realistic human motions, we propose a

Pose Variational Autoencoder (P-VAE) to constrain the actions to a subspace of natural poses. We validate our learning framework on a set of athletic jumping tasks, including Olympics high jumps, obstacle jumps, and long jumps. Through exploring a low-dimensional feature space of take-off states, a diverse set of novel strategies can be discovered within a proposed Bayesian Diversity Search (BDS) procedure. A second stage of optimization that encourages novel policies can be further applied to enrich the unique strategies discovered. Our framework discovers diverse set of athletic jump skills, such as the well-known Fosbury Flop and Western Roll as shown in Figure 1.2. Comparing with previous works, our method allows the discovery of diverse and novel strategies for athletic jumping tasks with no motion examples and less reward engineering. We detail the implementation of our framework in Chapter 3.

In summary, the contributions of this thesis include:

- A novel algorithm for computing stable PD motor control with superior efficiency and stability, enabling efficient learning of high-level motion controllers.
- A system based on DRL and P-VAE for discovering novel and natural motion strategies without references, as achieved by physics-based high-level motion controllers.
- The use of Bayesian diversity search and novel policy seeking for discovering diverse strategies for the studied athletic jumping tasks.

The work in Chapter 2 is published at Computer Graphics Forum as [129]. The work in Chapter 3 will appear at ACM Transactions on Graphics as [128].

Chapter 2

Linear Time Stable PD Motor Controllers

2.1 Overview

In physics-based character animation, the choice of low-level motor controllers crucially affects the simulation speed and the success of learning high-level character motion controllers. Among all available options, Proportional-Derivative (PD) controllers are commonly used for joint motor actuation in physics-based animation systems, especially for tracking-based methods where kinematic reference motions are available [137, 127]. PD controllers are simple to implement, and have been shown beneficial for improving quality and efficiency of recent DRL-based motor skill learning systems [93]. However, numerical instability is one of the major disadvantages of vanilla PD controllers. Extremely small simulation time steps are required for accurate simulation of characters driven by PD servos resulting in significant drop of simulation speed.

Stable Proportional-Derivative (SPD) controllers proposed by Tan *et al.* greatly improve the numerical stability of traditional PD controllers by employing the idea of implicit integration [115]. Instead of computing control forces based on the current state, SPD formulates PD controls using the state at the next simulation time step. In practice, SPD formulation allows for fairly high gains at large time steps. SPD controllers are usually implemented for articulated rigid body systems in generalized coordinates [61, 10]. However, these implementations compute SPD control by solving an $n \times n$ linear system in $O(n^3)$ time based on dense matrix factorization, where n is the total number of Degrees of Freedom (DoFs) in the articulation. We show in our experiments that cubic time SPD computations significantly slow down the simulation when controlling complex articulated characters with large DoFs.

In this chapter, we derive and validate Modified Articulated-Body Algorithm (MABA) to compute SPD control for articulations parameterized in generalized coordinates. Our algorithm is based on Featherstone's Articulated-Body Algorithm (ABA) for forward dy-

namics [28]. The proposed algorithm computes SPD controls in worst case $O(n)$ time and is practically fast. In our experiments, we demonstrate the performance advantage of MABA over the conventional dense matrix factorization based SPD implementation, as well as an alternative method based on sparse matrix factorization. We report the simulation FPS (frames per second) of an entire motion tracking system, and then the extra time required for SPD computation. We show that our algorithm provides superior stability for controlling complex character models at large time steps. We further demonstrate that MABA improves the training speed and quality of a DRL system for learning physics-based skills.

2.2 Background and Related Work

2.2.1 SPD Formulation for a Single DoF

Standard PD controllers calculate forces based on position and velocity errors at the current time step. At time step n, we denote the position variable as q^n, the target position as $\bar{q}^n$, the velocity as $\dot{q}^n$, and the target velocity as $\bar{\dot{q}}^n$. Then the PD control force τ^n can be calculated as:

$$\tau^n = -k_p(q^n - \bar{q}^n) - k_d(\dot{q}^n - \bar{\dot{q}}^n) \tag{2.1}$$

where k_p is the stiffness parameter and k_d is the damping parameter. If no velocity tracking is required, Equation 2.1 can be simplified to:

$$\tau^n = -k_p(q^n - \bar{q}^n) - k_d\dot{q}^n \tag{2.2}$$

For notation simplicity, hereafter we will formulate different variations of SPD controllers based on the PD control in Equation 2.2.

SPD computes control forces using state at the next time step instead of the current state [115]:

$$\tau^n = -k_p(q^{n+1} - \bar{q}^{n+1}) - k_d\dot{q}^{n+1} \tag{2.3}$$

Since future position q^{n+1} and velocity $\dot{q}^{n+1}$ at the next time step are unknown, they are approximated by the first order Taylor expansion:

$$q^{n+1} = q^n + \Delta t\dot{q}^n$$

$$\dot{q}^{n+1} = \dot{q}^n + \Delta t\ddot{q}^n$$

Equation 2.3 can then be reformulated as:

$$\tau^n = -k_p(q^n + \Delta t\dot{q}^n - \bar{q}^{n+1}) - k_d(\dot{q}^n + \Delta t\ddot{q}^n) \tag{2.4}$$

Equation 2.4 is the most popular SPD formulation. Other SPD formulations do exist. For example, the PD formulation in [74, 71, 69] uses positions at the current time step but

velocities at the next time step. Such explicit-proportional implicit-derivative formulation still achieves better stability than the conventional PD. In this paper, we focus on designing and analyzing practical algorithms for SPD computation formulated as Equation 2.4.

2.2.2 SPD for Articulations

For an articulated rigid body system with n DoFs parameterized in generalized coordinates, we use n dimensional vectors $\boldsymbol{\tau}$, $\boldsymbol{q}$, $\dot{\boldsymbol{q}}$, $\ddot{\boldsymbol{q}}$ to denote the generalized force, position, velocity, and acceleration of the system. The equation of motion for the articulation can then be expressed as:

$$\boldsymbol{M}\ddot{\boldsymbol{q}} = \boldsymbol{\tau} - \boldsymbol{C} \tag{2.5}$$

where $\boldsymbol{M}$ is the generalized inertia matrix, and $\boldsymbol{C}$ is the bias force term including centrifugal, Coriolis, and external forces.

The SPD formula for an articulation parameterized in generalized coordinates is a linear equation relating the acceleration $\ddot{\boldsymbol{q}}$ to the force $\boldsymbol{\tau}$, similar to Equation 2.4:

$$\boldsymbol{\tau} = -\boldsymbol{K}_p(\boldsymbol{q} + \Delta t\dot{\boldsymbol{q}} - \bar{\boldsymbol{q}}) - \boldsymbol{K}_d(\dot{\boldsymbol{q}} + \Delta t\ddot{\boldsymbol{q}}) \tag{2.6}$$

Here $\boldsymbol{K}_p$ and $\boldsymbol{K}_d$ are the diagonal stiffness and damping matrices. The position $\boldsymbol{q}$ and velocity $\dot{\boldsymbol{q}}$ can be obtained from the simulation state. The acceleration $\ddot{\boldsymbol{q}}$ is unknown and needs to be solved for by a forward dynamics algorithm that obeys Equation 2.5. Therefore, we substitute $\boldsymbol{\tau}$ in Equation 2.6 to Equation 2.5 to solve for the acceleration $\ddot{\boldsymbol{q}}$ first:

$$(\boldsymbol{M} + \boldsymbol{K}_d\Delta t)\ddot{\boldsymbol{q}} = -\boldsymbol{C} - \boldsymbol{K}_p(\boldsymbol{q} + \Delta t\dot{\boldsymbol{q}} - \bar{\boldsymbol{q}}) - \boldsymbol{K}_d\dot{\boldsymbol{q}} \tag{2.7}$$

Then $\boldsymbol{\tau}$ can be calculated by substituting $\ddot{\boldsymbol{q}}$ into either Equation 2.5 or 2.6.

2.2.3 Solving SPD by Matrix Factorization

The key step in solving SPD is to solve the linear system in Equation 2.7. Conventionally, it is solved directly by numerical methods which result in $O(n^3)$ running time in general. We briefly review these implementations in this section.

The most straight forward SPD implementation uses direct matrix inversion to solve the system [61, 104, 87]. This approach takes $O(n^3)$ time and may encounter numerical stability issues. A better approach is to apply Cholesky factorization LDL^T or LL^T on $\boldsymbol{M} + \boldsymbol{K}_d\Delta t$. Such factorization is applicable because the inertia matrix $\boldsymbol{M}$ is symmetric positive definite, and $\boldsymbol{K}_d\Delta t$ is a diagonal matrix with non-negative elements. This approach still takes $O(n^3)$ time to compute because it treats $\boldsymbol{M} + \boldsymbol{K}_d\Delta t$ as a dense matrix. We shall refer to this approach as **dense factorization** (DF) hereafter. Due to its simplicity, the DF method is widely adopted by both the research community and the industry [10, 88, 90, 89, 131].

To reduce the time complexity of the DF method, our first thought is to examine the inertia matrix $\boldsymbol{M}$. Featherstone's dynamics formulation of articulated rigid body systems results in branch-induced sparsity of $\boldsymbol{M}$ [28]. More specifically, for tree-like articulations that contain no loops, $\boldsymbol{M}_{ij}$ is non zero if and only if node i is an ancestor or a decedent of node j. We thus can employ a topology dependent sparse LDL^T or LL^T factorization algorithm [28] for $\boldsymbol{M} + \boldsymbol{K}_d\Delta t$, because again the additional diagonal matrix $\boldsymbol{K}_d\Delta t$ does not change the sparsity of the inertia matrix $\boldsymbol{M}$. We shall refer to this approach as **sparse factorization** (SF) hereafter. The time complexity of the SF method is $O(nd^2)$, where d is the maximum number of DoFs among all branches of the articulation tree. In the worst case where the tree becomes a chain, SF degenerates into DF and requires $O(n^3)$ time as well, as the inertia matrix is not sparse anymore. To the best of our knowledge, we are the first to implement the sparse factorization method for SPD computation.

2.2.4 Articulated-Body Forward Dynamics Algorithm

Forward dynamics is the problem of solving accelerations from the equation of motion indicated by Equation 2.5. From Equations 2.5 and 2.7 we can see that both forward dynamics and SPD control involve solving for the accelerations from a linear system. The cubic time SPD computation can significantly slow down the performance for systems with large DoFs, if linear time forward dynamics algorithms are used for simulation [28, 7]. One well-known linear time forward dynamics algorithm from the robotics literature is Featherstone's Articulated-Body Algorithm (ABA) [28]. This algorithm is sometimes abbreviated as ABM (Articulated Body Method) [56], but we follow Featherstone's original acronym.

ABA solves for the accelerations recursively in linear time without explicitly solving the $n \times n$ closed-form linear system. Its efficiency comes from the use of recurrence relations. A similar and more well known case is the Newton-Euler inverse dynamics algorithm: the recursive Newton-Euler is much faster than its non-recursive predecessor. ABA calculates the forward dynamics of a kinematic tree by three passes over the tree: an outward pass (root to leaves) to calculate velocity and bias terms; an inward pass to calculate articulated-body inertias and bias forces; and a second outward pass to calculate the accelerations.

ABA does not directly handle constraints such as collisions and joint limits. Additional mechanisms, either ABA-based or independent, are required to impose penalty, impulse, or constraint forces to enforce such constraints. For example, a modified version of ABA is used to compute acceleration constraint matrices in [56]. Another ABA-based procedure is used for computing and propagating impulse responses through articulated bodies [79]. In our work, we solely modify ABA for the purpose of SPD computation, which is complementary to other works that integrate other types of constraints into ABA-based algorithms. We will discuss more ABA details in Section 2.3.1, and derive our linear time SPD implementation, which is based on ABA, in Section 2.3.2 .

2.3 Modified Articulated-Body Algorithm

Since SPD computation has a similar structure as forward dynamics, we propose to solve SPD in linear time by adapting the Articulated-Body Algorithm. The fundamental intuition is that solving SPD accelerations is simply computing forward dynamics while enforcing the SPD constraints on the control forces indicated by Equation 2.6. We name such a linear time SPD algorithm as Modified Articulated-Body Algorithm (MABA). We will derive MABA by enforcing the SPD control force constraints where necessary in the ABA derivation. MABA shares the same algorithm structure as ABA, and therefore can be computed with the same set of tree traversals as in ABA.

2.3.1 ABA Preliminaries

In this section, we review a few key concepts and the three tree traversal passes of ABA, starting with necessary symbol definitions. We refer readers to [28] for more detailed explanations. However, ABA was developed by roboticists and its original derivation as presented in [28] is hard to understand for a typical graphics audience [7, 79]. We encourage readers who do not have any knowledge on spatial notations to first follow these excellent tutorials [26, 27]. Interested readers are further referred to an ABA derivation from basic principles of rigid body dynamics[79], which should be easier to understand.

Let T be a kinematic tree with m links L_1 to L_m, where L_1 is the root. The subtree rooted at L_i is denoted as T_i. The set of link indices of L_i's children is denoted as $\mu(i)$. The index of L_i's parent link is denoted as $\lambda(i)$. The inbound joint of L_i is denoted as J_i.

For each link L_i, we denote its 6D spatial motion vector as $\boldsymbol{v}_i$, which includes both the linear and angular motion of the rigid body. Similarly, we denote its 6D spatial force vector as $\boldsymbol{f}_i$, which includes both the linear force and the angular torque. Spatial acceleration of L_i is denoted as $\boldsymbol{a}_i$. Generalized joint variables for joint J_i are denoted as $\boldsymbol{q}_i$, $\dot{\boldsymbol{q}}_i$, $\ddot{\boldsymbol{q}}_i$ and $\boldsymbol{\tau}_i$. The motion subspace matrix $\boldsymbol{S}_i$ relates the generalized coordinates to spatial quantities as follows:

$$\boldsymbol{v}_{J_i} = \boldsymbol{S}_i \dot{\boldsymbol{q}}_i \tag{2.8}$$

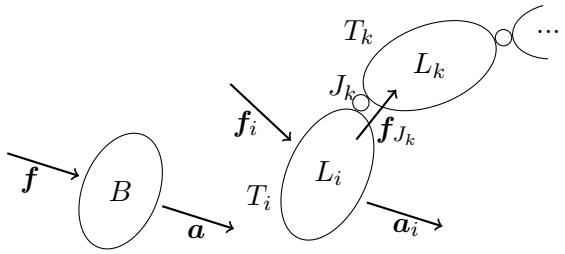
$$\boldsymbol{\tau}_i = \boldsymbol{S}_i^T \boldsymbol{f}_{J_i} \tag{2.9}$$

where $\boldsymbol{v}_{J_i}$ and $\boldsymbol{f}_{J_i}$ are the spatial velocity and spatial force of joint J_i.

Figure 2.1 illustrates the concept of a single body system and an articulated-body system. When applying a spatial force $\boldsymbol{f}$ on a single rigid body B as shown in Figure 2.1a, the acceleration $\boldsymbol{a}$ of body B is determined by the Newton-Euler equation:

$$\boldsymbol{f} = \boldsymbol{I}\boldsymbol{a} + \boldsymbol{p} \tag{2.10}$$

where $\boldsymbol{I}$ is a 6×6 single-body spatial inertia matrix including both the mass and moment of inertia, and $\boldsymbol{p}$ is a bias force term including the centrifugal and Coriolis forces.



(a) a single body (b) an articulated body

Figure 2.1: Illustration of a single body and an articulated body.

An articulated body is a system of rigid bodies articulated by joints as shown in Figure 2.1b. When spatial force $\boldsymbol{f}_i$ is applied on link L_i, we cannot compute the acceleration $\boldsymbol{a}_i$ from Equation 2.10 anymore because of the unknown joint force $\boldsymbol{f}_{J_k}$ from the subtree T_k. However, $\boldsymbol{f}_i$ and $\boldsymbol{a}_i$ still satisfy a linear equation

$$\boldsymbol{f}_i = \boldsymbol{I}_i^A \boldsymbol{a}_i + \boldsymbol{p}_i^A \tag{2.11}$$

where unknowns $\boldsymbol{I}_i^A$ and $\boldsymbol{p}_i^A$ depend on the whole structure of the articulated body. $\boldsymbol{I}_i^A$ and $\boldsymbol{p}_i^A$ are termed the articulated-body inertia and bias force of T_i. L_i is called the *handle* of the articulated-body system, as Equation 2.11 describes the acceleration response of body L_i jointed with all bodies in T_i. $\boldsymbol{I}_i^A$ and $\boldsymbol{p}_i^A$ have the following two properties that enable the ABA implementation by three tree traversals:

1. $\boldsymbol{I}_i^A$ and $\boldsymbol{p}_i^A$ can be computed recursively from the leaves to the root. Specifically, $\boldsymbol{I}_i^A$ and $\boldsymbol{p}_i^A$ can be computed from $\boldsymbol{I}_j^A$ and $\boldsymbol{p}_j^A$ where $j \in \mu(i)$.

2. Once we know $\boldsymbol{I}_i^A$ and $\boldsymbol{p}_i^A$ for each i, all joint and link accelerations can be computed recursively from the root to the leaves.

ABA requires three passes of tree traversal of the articulation tree. These tree traversals solve for the auxiliary variables and final accelerations following the topological order of the tree:

- **Pass 1** (top down): Compute auxiliaries including joint and link velocities, and single-body centrifugal, Coriolis, and external forces, from the root to the leaves.

- **Pass 2** (bottom up): Compute articulated-body inertias and bias forces based on auxiliaries computed in Pass 1, from the leaves to the root.

- **Pass 3** (top down): Compute joint and link accelerations based on the articulated-body inertias and bias forces computed in Pass 2, from the root to the leaves.

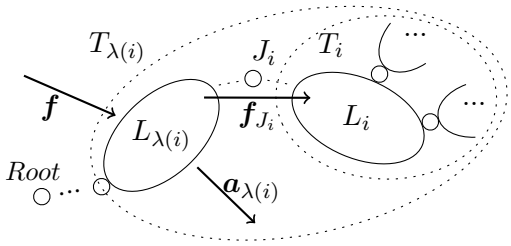


Figure 2.2: Articulated-body $T_{\lambda(i)}$ with handle $L_{\lambda(i)}$.

2.3.2 MABA Derivation

Our MABA derivation is similar to the standard ABA derivation, which makes three traversal passes of the kinematic tree. In particular, MABA shares the exact same Pass 1 as ABA, for which we omit the details in this paper and refer the interested readers to [28]. Hereafter we assume quantities computed by Pass 1 are known, including joint and link velocities, and single-body bias forces. We will first derive Pass 3 then Pass 2, since derivation for Pass 2 requires a relationship between joint accelerations and link accelerations derived in Pass 3. For notation simplicity and ease of comprehension, we omit necessary coordinate transformations in this section. For ease of reimplementation though, we list the complete set of equations with proper coordinate transformations in Appendix A.

MABA Pass 3

Pass 3 of MABA does a similar job as Pass 3 of ABA, with additional SPD constraints taken into account when accumulating accelerations from the root to the leaves. The root link acceleration serves as the base case of the recursion. For fixed-base articulations, the root acceleration is set to zero. For floating-base articulations, we treat the root link as connected to a fixed base by a virtual 6-DoF joint. Now we derive recurrent formulas to solve

- (Objective 1): $\boldsymbol{a}_i$ based on $\boldsymbol{a}_{\lambda(i)}$ and $\ddot{\boldsymbol{q}}_i$, and
- (Objective 2): $\ddot{\boldsymbol{q}}_i$ based on $\boldsymbol{a}_{\lambda(i)}$.

For Objective 1, we consider the articulated body $T_{\lambda(i)}$ shown in Figure 2.2. The joint velocity $\boldsymbol{v}_{J_i}$ is the relative spatial velocity of L_i with respect to $L_{\lambda(i)}$. From Equation 2.8 we have

$$\boldsymbol{v}_{J_i} = \boldsymbol{v}_i - \boldsymbol{v}_{\lambda(i)} = \boldsymbol{S}_i \dot{\boldsymbol{q}}_i. \tag{2.12}$$

By differentiating the above equation, we get

$$\boldsymbol{a}_i - \boldsymbol{a}_{\lambda(i)} = \boldsymbol{S}_i \ddot{\boldsymbol{q}}_i + \dot{\boldsymbol{S}}_i \dot{\boldsymbol{q}}_i. \tag{2.13}$$

So the link acceleration $\boldsymbol{a}_i$ can be computed from the parent link acceleration $\boldsymbol{a}_{\lambda(i)}$ and the inbound joint acceleration $\ddot{\boldsymbol{q}}_i$, achieving Objective 1 of Pass 3.

For Objective 2, we take the SPD constraints into consideration. After Pass 2, the articulated-body inertia $\boldsymbol{I}_i^A$ and the bias force $\boldsymbol{p}_i^A$ for T_i are knowns that satisfy

$$\boldsymbol{f}_i = \boldsymbol{I}_i^A \boldsymbol{a}_i + \boldsymbol{p}_i^A \tag{2.14}$$

where $\boldsymbol{f}_i$ is the net external force acting on articulated body T_i through handle L_i. Such force comes only from joint J_i, therefore,

$$\boldsymbol{f}_i = \boldsymbol{f}_{J_i} \tag{2.15}$$

Now we expand Equation 2.9 by Equation 2.15, 2.14 and 2.13:

$$\begin{aligned} \boldsymbol{\tau}_i &= \boldsymbol{S}_i^T \boldsymbol{f}_{J_i} \\ &= \boldsymbol{S}_i^T \left(\boldsymbol{I}_i^A \boldsymbol{a}_i + \boldsymbol{p}_i^A \right) \\ &= \boldsymbol{S}_i^T \left(\boldsymbol{I}_i^A \left(\boldsymbol{a}_{\lambda(i)} + \boldsymbol{S}_i \ddot{\boldsymbol{q}}_i + \dot{\boldsymbol{S}}_i \dot{\boldsymbol{q}}_i \right) + \boldsymbol{p}_i^A \right) \end{aligned} \tag{2.16}$$

Equation 2.16 is a linear equation relating the generalized joint force $\boldsymbol{\tau}_i$ and the generalized joint acceleration $\ddot{\boldsymbol{q}}_i$. Here we must enforce that $\boldsymbol{\tau}_i$ equals the SPD control force. Following Equation 2.4, the SPD constraint for a single joint J_i can be written as

$$\boldsymbol{\tau}_i = -\boldsymbol{K}_{pi}(\boldsymbol{q}_i + \Delta t \dot{\boldsymbol{q}}_i - \bar{\boldsymbol{q}}_i) - \boldsymbol{K}_{di}(\dot{\boldsymbol{q}}_i + \Delta t \ddot{\boldsymbol{q}}_i) \tag{2.17}$$

Combining Equation 2.16 and 2.17, we get

$$\ddot{\boldsymbol{q}}_i = (\boldsymbol{S}_i^T \boldsymbol{I}_i^A \boldsymbol{S}_i + \boldsymbol{K}_i)^{-1} \left(\boldsymbol{Q}_i - \boldsymbol{S}_i^T \left(\boldsymbol{I}_i^A (\boldsymbol{a}_{\lambda(i)} + \dot{\boldsymbol{S}}_i \dot{\boldsymbol{q}}_i) + \boldsymbol{p}_i^A \right) \right) \tag{2.18}$$

where we define

$$\boldsymbol{K}_i = \boldsymbol{K}_{di} \Delta t, \tag{2.19}$$

$$\boldsymbol{Q}_i = -\boldsymbol{K}_{pi}(\boldsymbol{q}_i + \Delta t \dot{\boldsymbol{q}}_i - \bar{\boldsymbol{q}}_i) - \boldsymbol{K}_{di} \dot{\boldsymbol{q}}_i \tag{2.20}$$

Equation 2.18 achieves Objective 2, as joint acceleration $\ddot{\boldsymbol{q}}_i$ can be computed from the parent link acceleration $\boldsymbol{a}_{\lambda(i)}$. Since we have enforced the SPD constraint in the derivation, accelerations computed by this algorithm strictly follow the SPD control. Now we have completed the derivation of MABA Pass 3.

MABA Pass 2

Pass 2 of MABA recursively accumulates articulated-body inertias and bias forces from the leaves to the root. All leaf links of the kinematic tree form the recursion base cases.

As leaf links have no children, their articulated-body inertia and bias force equal to their single-body counterparts. Next we need to derive the recurrent formulas to solve for a link's articulated-body inertia and bias force from those of its children. We first consider the case shown in Figure 2.2 where $L_{\lambda(i)}$ has only one child L_i. Later we will generalize the computation for links with multiple children.

To compute $\boldsymbol{I}^A_{\lambda(i)}$ and $\boldsymbol{p}^A_{\lambda(i)}$ from $\boldsymbol{I}^A_i$ and $\boldsymbol{p}^A_i$, the strategy is to derive an equation of the form

$$\boldsymbol{f} = \boldsymbol{A}\boldsymbol{a}_{\lambda(i)} + \boldsymbol{b} \tag{2.21}$$

where $\boldsymbol{f}$ is the net spatial force acting on $L_{\lambda(i)}$ external to the articulated body $T_{\lambda(i)}$. Then the coefficients $\boldsymbol{A}$ and $\boldsymbol{b}$ will correspond to the desired quantities $\boldsymbol{I}^A_{\lambda(i)}$ and $\boldsymbol{p}^A_{\lambda(i)}$.

We first consider the spatial forces acting on the single body $L_{\lambda(i)}$. Apart from $\boldsymbol{f}$, there is also a joint reaction force $-\boldsymbol{f}_{J_i}$ acting on $L_{\lambda(i)}$. Then from the Newton-Euler equation similar to Equation 2.10:

$$\boldsymbol{f} - \boldsymbol{f}_{J_i} = \boldsymbol{I}_{\lambda(i)}\boldsymbol{a}_{\lambda(i)} + \boldsymbol{p}_{\lambda(i)} \tag{2.22}$$

where $\boldsymbol{I}_{\lambda(i)}$ and $\boldsymbol{p}_{\lambda(i)}$ are the single-body spatial inertia and bias force terms. Expanding Equation 2.22 with Equations 2.15, 2.14, 2.13 and 2.18, we get:

$$\begin{aligned}
\boldsymbol{f} &= \boldsymbol{I}_{\lambda(i)}\boldsymbol{a}_{\lambda(i)} + \boldsymbol{p}_{\lambda(i)} + \boldsymbol{f}_{J_i} \\
&= \boldsymbol{I}_{\lambda(i)}\boldsymbol{a}_{\lambda(i)} + \boldsymbol{p}_{\lambda(i)} + \boldsymbol{I}^A_i\boldsymbol{a}_i + \boldsymbol{p}^A_i \\
&= \boldsymbol{I}_{\lambda(i)}\boldsymbol{a}_{\lambda(i)} + \boldsymbol{p}_{\lambda(i)} + \boldsymbol{I}^A_i(\boldsymbol{a}_{\lambda(i)} + \boldsymbol{S}_i\ddot{\boldsymbol{q}}_i + \dot{\boldsymbol{S}}_i\dot{\boldsymbol{q}}_i) + \boldsymbol{p}^A_i \\
&= \boldsymbol{I}_{\lambda(i)}\boldsymbol{a}_{\lambda(i)} + \boldsymbol{p}_{\lambda(i)} + \boldsymbol{I}^A_i(\boldsymbol{a}_{\lambda(i)} + \boldsymbol{S}_i(\boldsymbol{S}_i^T\boldsymbol{I}^A_i\boldsymbol{S}_i + \boldsymbol{K}_i)^{-1} \\
&\quad \left(\boldsymbol{Q}_i - \boldsymbol{S}_i^T\left(\boldsymbol{I}^A_i(\boldsymbol{a}_{\lambda(i)} + \dot{\boldsymbol{S}}_i\dot{\boldsymbol{q}}_i) + \boldsymbol{p}^A_i\right)\right) + \dot{\boldsymbol{S}}_i\dot{\boldsymbol{q}}_i) + \boldsymbol{p}^A_i
\end{aligned} \tag{2.23}$$

By rearranging terms in the above equation, we can achieve the desired form in Equation 2.21. If we define

$$\boldsymbol{I}^a_i = \boldsymbol{I}^A_i - \boldsymbol{I}^A_i\boldsymbol{S}_i(\boldsymbol{S}_i^T\boldsymbol{I}^A_i\boldsymbol{S}_i + \boldsymbol{K}_i)^{-1}\boldsymbol{S}_i^T\boldsymbol{I}^A_i, \tag{2.24}$$

$$\boldsymbol{p}^a_i = \boldsymbol{p}^A_i + \boldsymbol{I}^a_i\dot{\boldsymbol{S}}_i\dot{\boldsymbol{q}}_i + \boldsymbol{I}^A_i\boldsymbol{S}_i(\boldsymbol{S}_i^T\boldsymbol{I}^A_i\boldsymbol{S}_i + \boldsymbol{K}_i)^{-1}(\boldsymbol{Q}_i - \boldsymbol{S}_i^T\boldsymbol{p}^A_i) \tag{2.25}$$

then we can get the formulas for $\boldsymbol{I}^A_{\lambda(i)}$ and $\boldsymbol{p}^A_{\lambda(i)}$ as follows:

$$\boldsymbol{I}^A_{\lambda(i)} = \boldsymbol{I}_{\lambda(i)} + \boldsymbol{I}^a_i, \tag{2.26}$$

$$\boldsymbol{p}^A_{\lambda(i)} = \boldsymbol{p}_{\lambda(i)} + \boldsymbol{p}^a_i \tag{2.27}$$

Similarly, Equations 2.26 and 2.27 can be generalized for links with multiple children. For an arbitrary link L_i:

$$\boldsymbol{I}^A_i = \boldsymbol{I}_i + \sum_{j\in\mu(i)} \boldsymbol{I}^a_j, \tag{2.28}$$

$$\boldsymbol{p}_i^A = \boldsymbol{p}_i + \sum_{j \in \mu(i)} \boldsymbol{p}_j^a. \tag{2.29}$$

We therefore use Equations 2.28 and 2.29 to compute the articulated-body inertias and bias forces from the leaves to the root in MABA Pass 2. Again, a complete set of equations for the whole algorithm is given in Appendix A.

2.3.3 Algorithm Complexity

MABA and ABA share most of their essential computations. The key difference is that MABA requires the extra computation of $\boldsymbol{K}_i$ and $\boldsymbol{Q}_i$ by Equation 2.19 and 2.20, which takes at most $O(n)$ time. Since ABA runs in worst case $O(n)$ time, we conclude that MABA runs in worst case $O(n)$ time as well.

2.3.4 Practical Implementation

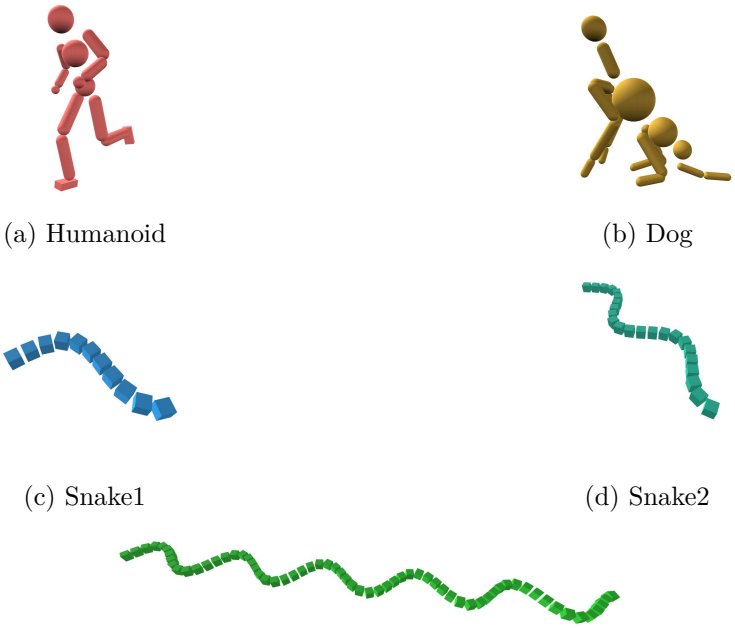
MABA can be directly implemented by modifying essentially just two lines of the original ABA equations, for which more details are given in Appendix A. This makes it easy to embed SPD control directly into simulation systems that already use ABA for forward dynamics, such as PhysX [94], Bullet [10] and DART [61]. In such an embedded implementation, forward dynamics accelerations are computed directly under constraints imposed by SPD, without the actual control forces explicitly calculated. Such implementation is simple to code on top of ABA, and incurs negligible cost as we will show in our experiments.

As MABA is simply ABA with SPD constraints satisfied, our embedded implementation of MABA naturally works with additional constraint solvers, either ABA-based or independent. Simulation engines with independent constraint solvers can call MABA instead of ABA to solve forward dynamics with SPD controls [94]. ABA-based solvers just need to incorporate our minor modifications into their algorithm to use SPD controllers [56]. For our experiments done on PhysX, we do not need to do anything other than replacing the original ABA code with our MABA code to incorporate contact and ground reaction forces into the simulation, as PhysX handles these constraints independently of ABA.

2.4 Experiments

In this section we validate the accuracy, stability, and performance of MABA in motion tracking tasks and Deep Reinforcement Learning (DRL) tasks. We also compare MABA with the dense factorization (DF) method and the sparse factorization (SF) method. The DF method is implemented using the dense LL^T factorization provided by the Eigen library [33]. The SF implementation strictly follows the pseudo-code for sparse LL^T factorization presented in [28].

Our experiments are performed on a Dell Precision 7920 Tower workstation with an Intel Xeon Gold 6128 CPU (3.4 GHz, 12 threads) and a GeForce GTX 1080 Ti GPU.



(a) Humanoid (b) Dog

(c) Snake1 (d) Snake2

(e) Snake3

Figure 2.3: Character models used in our experiments.

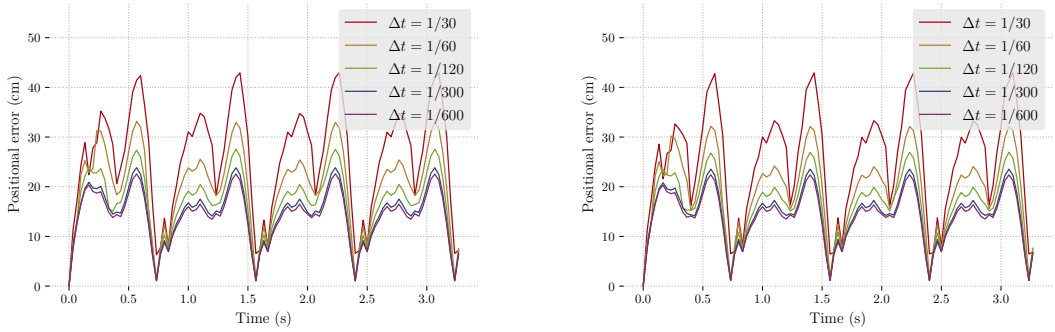
Model	Humanoid	Dog	Snake1	Snake2	Snake3
n	34	72	36	72	195
d	13	24	36	72	195

Table 2.1: Model parameters: n is the degrees of freedom; and d is the maximum number of DoFs from the root to the leaves.

NVIDIA PhysX (version 2019.8) is used as our physics simulation engine. Five simulated virtual character models of different complexity are studied in our experiments, including a humanoid, a dog, and three snakes of different length. We visualize these models in Figure 2.3 and list their important model parameters in Table 2.1. The humanoid model and motions are obtained from Peng *et al.* [88]. The dog model and motions are obtained from Zhang *et al.* [132].

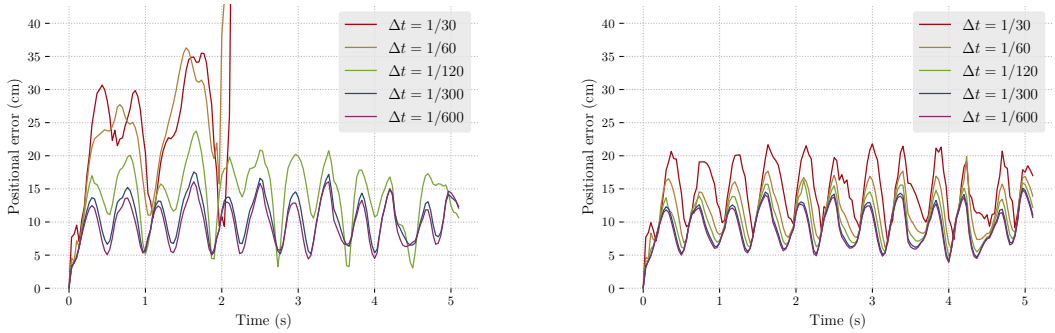
2.4.1 Accuracy and Stability

We investigate the accuracy and stability of MABA through motion tracking tasks on the humanoid and dog models. More specifically, we use SPD controllers to track both the root and internal joints in predefined motion trajectories. As the root is controlled by external "hand-of-God" forces, such tasks are quasi-static in nature and used purely for accessing the tracking performance. Tracking accuracy is measured by end-effector to root errors between the reference and simulated vectors. Here we use a fixed set of SPD parameters



(a) Dense factorization method (b) MABA

Figure 2.4: SPD tracking errors measured at the right ankle of the humanoid model in a running motion.



(a) Dense factorization method (b) MABA

Figure 2.5: SPD tracking errors measured at the right front toe of the dog model in a cantering motion.

in our experiment: $k_p = 20000, k_d = 2000$ for the root, and $k_p = 75000, k_d = 4000$ for all the internal joints. We show the tracking errors using different simulation time steps in Figure 2.4 and 2.5. Figure 2.4 corresponds to the tracking errors measured by the right ankle to root vector of the humanoid model during a running motion. Figure 2.5 corresponds to the tracking errors measured by the right front toe to root vector of the dog model during a cantering motion. We only compare the DF method and MABA here, as SF only differs from DF in terms of efficiency, but not accuracy nor stability.

Figure 2.4 shows that MABA and the dense factorization method produce near identical accuracy curves on the humanoid model. Both controllers are stable for time steps up to $\Delta t = 1/30\ s$. Figure 2.5 demonstrates that MABA achieves significantly better accuracy and stability than DF on the dog model. MABA is stable for all tested time steps, while DF diverges for $\Delta t \geq 1/60\ s$. For $\Delta t = 1/120\ s$, although DF does not fail, it produces visibly larger tracking errors than MABA. For small time steps $\Delta t \leq 1/300\ s$, the two implemen-

Model	Motion	DF	PCG	SF	MABA
Humanoid	walk	13,752	14,108	17,083	20,224
	run	14,109	14,409	17,048	20,523
	cartwheel	13,729	14,436	16,962	20,213
	backflip	13,827	14,537	16,943	19,387
Dog	pace	6,572	4,838	8,764	11,804
	trot	6,551	4,682	8,969	11,540
	canter	6,498	4,553	8,738	11,700
Snake1	slither	11,897	11,935	12,259	16,444
Snake2	slither	5,099	5,069	3,979	8,334
Snake3	slither	1,036	944	417	3,036

Table 2.2: Simulation FPS (frames per second) of different SPD implementations: DF (dense factorization), PCG (preconditioned conjugate gradient with Jacobi preconditioning), SF (sparse factorization), and MABA.

Model	DF(%)	SF(%)	MABA(%)
Humanoid	31.1	18.2	2.1
Dog	44.3	25.7	3.6
Snake1	28.0	28.3	1.7
Snake2	39.9	53.9	2.5
Snake3	64.4	86.3	3.7

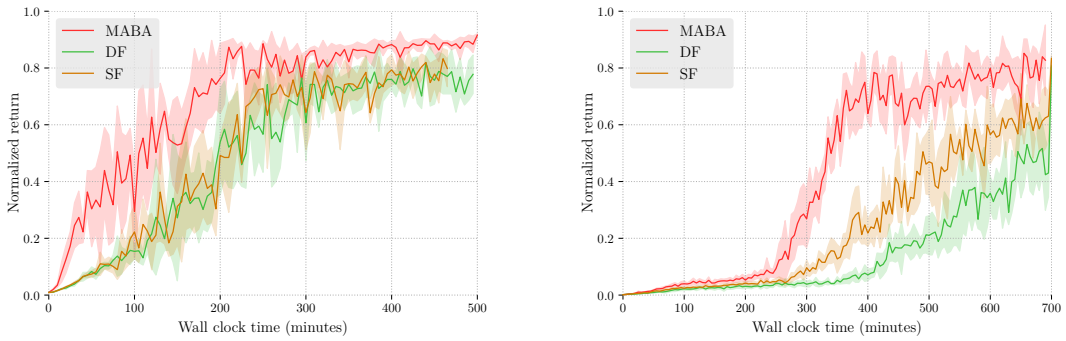
Table 2.3: Percentage of extra time required by SPD computation over total simulation time.

tations become comparable. Based on these experiments, we conclude that MABA delivers better stability for complex models at large time steps. Even though different implementations solve the same SPD formulation, MABA generally produces less numerical errors than matrix factorization due to its computational simplicity. Therefore MABA degrades more gracefully than matrix factorization based methods for large integration time steps and complex models, which tend to amplify the accumulated numerical errors.

2.4.2 Simulation Performance

We compare the runtime performance of different SPD implementations on four character models of different complexity using the same motion tracking tasks described in Section 2.4.1. Performance is measured by the number of simulated Frames Per Second (FPS) on a single CPU thread. We set $\Delta t = 1/30\ s$ for the humanoid and snake models. We use $\Delta t = 1/240\ s$ for the dog model, as factorization-based methods cannot achieve stable simulation when using large time steps as shown in Figure 2.5.

We record the simulation FPS for different combinations of models, motions and SPD implementations (DF, SF, and MABA). Table 2.2 shows that MABA is significantly more efficient than DF, which is currently the only option in both research and industry to the



(a) Humanoid run (b) Dog canter

Figure 2.6: DRL learning curves measured in wall-clock time using different SPD implementations.

best of our knowledge. MABA is also always faster than SF. SF achieves better performance than DF when controlling models with low DoFs, or models with shallow tree structures. However, SF is less efficient than DF for models with high DoFs or deep tree structures. Our snake models correspond to the worst case scenario for SF, as their kinematic trees degenerate to chains so the time complexity becomes $O(n^3)$ just as in DF. In fact, SF runs slower than DF due to necessary overhead incurred for sparse factorization.

Iterative methods, such as Preconditioned Conjugate Gradient (PCG), can also be used to solve this problem. However, as our matrix is neither very large nor sparse, PCG does not provide any performance gain over Cholesky factorization. As shown in Table 2.2, PCG results in comparable speed to DF. There is a better mechanism proposed in [119] that uses PCG to achieve $O(n^2)$ performance. Our proposed MABA is much simpler and faster, however, so we will not include PCG-based methods in further comparative studies.

We also report the percentages of extra time required for SPD computation over total simulation time for different methods. Table 2.3 shows that matrix factorization-based methods can dominate the simulation process for complex models while MABA solves SPD with consistently negligible cost. Therefore, we recommend MABA for SPD computation wherever possible. We note that for Table 2.3, reported performances are averaged values across all available motions for the humanoid and dog models.

2.4.3 DRL Training Performance

The motion tracking task that we have used for testing and comparison so far is only a quasi physics-based method, as it tracks the root joint with a hand-of-God type control. In order to see the effect of different solvers in true physics-based learning and control systems, where only internal joint actuation and ground reaction forces are used for control of the full body, we employ these SPD solvers within a state-of-the-art Deep Reinforcement Learning (DRL) framework named DeepMimic [88]. DeepMimic employs both an imitation reward and a

task reward to encourage physics-based characters to learn high-quality motor skills. We re-implement DeepMimic on NVIDIA PhysX and train several skills on our machine with 12 CPU threads in parallel and one GPU. We use exactly the same network architecture and training parameters as those given in the original implementation [88]. Figure 2.6 reports the learning curves in wall-clock time. As DeepMimic is non-deterministic, the curves we show are the average of five training runs. The shaded region indicates the standard deviation. Our results show that for both the humanoid and the dog model, MABA not only improves the learning speed, but also helps DeepMimic converge to better solutions. This is because first MABA saves training time by directly reducing the SPD computational cost; and second DeepMimic requires less training samples when using MABA as its greater stability helps reduce the learning difficulty.

2.5 Summary and Discussions

In this chapter, we have presented the Modified Articulated-Body Algorithm for SPD computation of articulated rigid body systems parameterized in generalized coordinates. We show that MABA runs in linear time, which is the theoretical minimum under the presented SPD formulation. We demonstrate the performance and stability advantages of MABA for physics-based character animation. Since SPD controllers are fundamental components in many time-critical or time-consuming systems, such as computer games and DRL-based algorithms, our proposed algorithm could potentially benefit a wide range of applications and research.

Our current MABA implementation is embedded into an ABA forward dynamics solver by directly modifying the ABA code in the PhysX simulation engine. A standalone implementation is necessary when the SPD control forces need to be computed explicitly. For example, when the SPD control forces need to be monitored and tailored before being sent to the forward dynamics simulation. The standalone implementation is roughly equivalent to running a linear time forward dynamics solver twice at each simulation time step. The theoretical time complexity is still $O(n)$ but the speed and accuracy of the standalone MABA implementation will be inferior to the embedded MABA, due to the two passes of the ABA-type solver and error accumulations during this process. In our experiments, we found that standalone MABA can be slower than dense factorization for small models such as the humanoid, but still shows its scalability on complex models such as the snakes.

In the future, we plan to investigate linear time SPD controllers for articulations parameterized in full coordinates. Such an algorithm should be developed on top of a linear forward dynamics algorithm in full coordinates, such as [7], to maximize the potential performance gain. It is also interesting to explore parallel algorithms for solving SPD, so that GPU acceleration can be utilized.

Chapter 3

Discover Diverse Athletic Jumping Strategies

3.1 Overview

Given a simulated character in a virtual environment, we propose a framework for the discovery of appropriate motion strategies for certain tasks. Given a task objective and an initial character configuration, the combination of physics simulation and deep reinforcement learning (DRL) provides a suitable starting point for automatic control policy training. To facilitate the learning of realistic human motions, we propose a Pose Variational Autoencoder (P-VAE) to constrain the actions to a subspace of natural poses. Comparing with prior work, our method requires no motion examples and less reward engineering.

We apply our learning framework to train take-off controllers for a set of athletic jumping tasks especially for Olympics high jumps, since high jumps are highly technically complex and strategically nuanced involving multiple strategies discovered throughout human history. In our work, we aim to discover as many of these strategies as possible without motion examples. We first identify that the initial character state before the take-off, or the take-off state for short, is a strong determinant of the learned strategy. We then apply a Bayesian Diversity Search (BDS) algorithm to explore a low-dimensional feature space of the take-off state, and maximize the diversity of the corresponding strategies. Given a desired take-off state, we first train a run-up controller that imitates a single generic run-up motion capture clip while also targeting the desired take-off state. The subsequent jump control policy is trained with the help of P-VAE and a curriculum, but without any recourse to motion capture data. Given a specific take-off state, we further enrich unique strategy variations by a second optimization stage which reuses the take-off state and encourages novel control policies.

In Section 3.4, we explain the implementation of our DRL framework and P-VAE for learning natural strategies. In Section 3.5, we describe BDS and novel policy seeking tech-

niques for learning diverse strategies. The conceptual formulation and the actual implementation of BDS are provided by Zeshi Yang.

3.2 Related Work

We build on prior work from several areas, including character animation, diversity optimization, human pose modeling, and high-jump analysis from biomechanics and kinesiology.

3.2.1 Character Animation

Synthesizing natural human motion is a long-standing challenge in computer animation. We first briefly review kinematic methods, and then provide a more detailed review of physics-based methods. To the best of our knowledge, there are no previous attempts to synthesize athletic high jumps or obstacle jumps using either kinematic or physics-based approaches. Both tasks require precise coordination and exhibit multiple strategies.

Kinematic Methods Data-driven kinematic methods have demonstrated their effectiveness for synthesizing high-quality human motions based on captured examples. Such kinematic models have evolved from graph structures [58, 99], to Gaussian Processes [66, 126], and recently deep neural networks [44, 132, 112, 113, 62, 67]. Non-parametric models that store all example frames have limited capability of generalizing to new motions due to their inherent nature of data interpolation [14]. Compact parametric models learn an underlying low-dimensional motion manifold. Therefore they tend to generalize better as new motions not in the training dataset can be synthesized by sampling in the learned latent space [45]. Completely novel motions and strategies, however, are still beyond their reach. Most fundamentally, kinematic models do not take into account physical realism, which is important for athletic motions. We thus cannot directly apply kinematic methods to our problem of discovering unseen strategies for highly dynamic motions. However, we do adopt a variational autoencoder (VAE) similar to the one in [67] as a means to improve the naturalness of our learned motion strategies.

Physics-based Methods Physics-based control and simulation methods generate motions with physical realism and environmental interactions. The key challenge is the design or learning of robust controllers. Conventional manually designed controllers have achieved significant success for locomotion, e.g., [127, 120, 17, 121, 31, 64, 29, 48]. The seminal work from Hodgins *et al.* demonstrated impressive controllers for athletic skills such as a handspring vault, a standing broad jump, a vertical leap, somersaults to different directions, and platform dives [43, 123]. Such handcrafted controllers are mostly designed with finite state machines (FSM) and heuristic feedback rules, which require deep human insight and domain knowledge, and tedious manual trial and error. Zhao and van de Panne [135] thus

proposed an interface to ease such a design process, and demonstrated controllers for diving, skiing and snowboarding. Controls can also be designed using objectives and constraints adapted to each motion phase, e.g., [48, 23], or developed using a methodology that mimics human coaching [34]. In general, manually designed controllers remain hard to generalize to different strategies or tasks.

With the wide availability of motion capture data, many research endeavors have been focused on tracking-based controllers, which are capable of reproducing high-quality motions by imitating motion examples. Controllers for a wide range of skills have been demonstrated through trajectory optimization [108, 21, 83, 63, 125, 64], sampling-based algorithms [73, 72, 71], and deep reinforcement learning [90, 88, 92, 76, 70, 104]. Tracking controllers have also been combined with kinematic motion generators to support interactive control of simulated characters [8, 87, 122]. Even though tracking-based methods have demonstrated their effectiveness on achieving task-related goals [88], the imitation objective inherently restricts them from generalizing to novel motion strategies fundamentally different from the reference. Most recently, style exploration has also been demonstrated within a physics-based DRL framework using spacetime bounds [76]. However, these remain style variations rather than strategy variations. Moreover, high jumping motion capture examples are difficult to find. We obtained captures of three high jump strategies, which we use to compare our synthetic results to.

Our goal is to discover as many strategies as possible, so example-free methods are most suitable in our case. Various tracking-free methods have been proposed via trajectory optimization or deep reinforcement learning. Heess *et al.* [39] demonstrate a rich set of locomotion behaviors emerging from just complex environment interactions. However, the resulting motions show limited realism in the absence of effective motion quality regularization. Better motion quality is achievable with sophisticated reward functions and domain knowledge, such as sagittal symmetry, which do not directly generalize beyond locomotion [130, 18, 124, 82, 80]. Synthesizing diverse physics-based skills without example motions generally requires optimization with detailed cost functions that are engineered specifically for each skill [4], and often only works for simplified physical models [81].

3.2.2 Diversity Optimization

Diversity Optimization is a problem of great interest in artificial intelligence [38, 117, 111, 15, 95, 65]. It is formulated as searching for a set of configurations such that the corresponding outcomes have a large diversity while satisfying a given objective. Diversity optimization has also been utilized in computer graphics applications [78, 3]. For example, a variety of 2D and simple 3D skills have been achieved through jointly optimizing task objectives and a diversity metric within a trajectory optimization framework [3]. Such methods are computationally prohibitive for our case as learning the athletic tasks involve expensive DRL training through non-differentiable simulations, e.g., a single strategy takes six hours

to learn even on a high-end desktop. We propose a diversity optimization algorithm based on the successful Bayesian Optimization (BO) philosophy for sample efficient black-box function optimization.

In Bayesian Optimization, objective functions are optimized purely through function evaluations as no derivative information is available. A Bayesian statistical *surrogate model*, usually a Gaussian Process (GP) [98], is maintained to estimate the value of the objective function along with the uncertainty of the estimation. An *acquisition function* is then repeatedly maximized for fast decisions on where to sample next for the actual expensive function evaluation. The next sample needs to be promising in terms of maximizing the objective function predicted by the surrogate model, and also informative in terms of reducing the uncertainty in less explored regions of the surrogate model [50, 30, 110]. BO has been widely adopted in machine learning for parameter and hyperparameter optimizations [107, 55, 52, 53, 57, 106]. Recently BO has also seen applications in computer graphics [60, 59], such as parameter tuning for fluid animation systems [9].

We propose a novel acquisition function to encourage discovery of diverse motion strategies. We also decouple the exploration from the maximization for more robust and efficient strategy discovery. We name this algorithm Bayesian Diversity Search (BDS). The BDS algorithm searches for diverse strategies by exploring a low-dimensional initial state space defined at the take-off moment. Initial states exploration has been applied to find appropriate initial conditions for desired landing controllers [35]. In the context of DRL learning, initial states are usually treated as hyperparameters rather than being explored.

Recently a variety of DRL-based learning methods have been proposed to discover diverse control policies in machine learning, e.g., [25, 133, 114, 1, 105, 36, 16, 46, 41, 101]. These methods mainly encourage exploration of unseen states or actions by jointly optimizing the task and novelty objectives [133], or optimizing intrinsic rewards such as heuristically defined curiosity terms [25, 105]. We adopt a similar idea for novelty seeking in Stage 2 of our framework after BDS, but with a novelty metric and reward structure more suitable for our goal. Coupled with the Stage 1 BDS, we are able to learn a rich set of strategies for challenging tasks such as athletic high jumping.

3.2.3 Natural Pose Space

In biomechanics and neuroscience, it is well known that muscle synergies, or muscle co-activations, serve as motor primitives for the central nervous system to simplify movement control of the underlying complex neuromusculoskeletal systems [86, 134]. In character animation, human-like character models are much simplified, but are still parameterized by 30+ DoFs. Yet the natural human pose manifold learned from motion capture databases is of much lower dimension [45]. The movement of joints are highly correlated as typically they are strategically coordinated and co-activated. Such correlations have been modelled

through traditional dimensionality reduction techniques such as PCA [11], or more recently, Variational AutoEncoders (VAE) [37, 67].

We rely on a VAE learned from mocap databases to produce natural target poses for our DRL-based policy network. Searching behaviors in low dimensional spaces has been employed in physics-based character animation to both accelerate the nonlinear optimization and improve the motion quality [100]. Throwing motions based on muscle synergies extracted from human experiments have been synthesized on a musculoskeletal model [20]. Recent DRL methods either directly imitate mocap examples [88, 122], which makes strategy discovery hard if possible; or adopt a *de novo* approach with no example at all [40], which often results in extremely unnatural motions for human like characters. Close in spirit to our work is [97], where a low-dimensional PCA space learned from a single mocap trajectory is used as the action space of DeepMimic for tracking-based control. We aim to discover new strategies without tracking, and we use a large set of generic motions to deduce a task-and-strategy-independent natural pose space. We also add action offsets to the P-VAE output poses so that large joint activation can be achieved for powerful take-offs.

Reduced or latent parameter spaces based on statistical analysis of poses have been used for grasping control [13, 5, 85]. A Trajectory Generator (TG) can provide a compact parameterization that can enable learning of reactive policies for complex behaviors [47]. Motion primitives can also be learned from mocap and then be composed to learn new behaviors [91].

3.2.4 History and Science of High Jump

The high jump is one of the most technically complex, strategically nuanced, and physiologically demanding sports among all track and field events [24]. Over the past 100 years, high jump has evolved dramatically in the Olympics. Here we summarize the well-known variations [51, 24], and we refer readers to our supplemental video for more visual illustrations.

- The Hurdle: the jumper runs straight-on to the bar, raises one leg up to the bar, and quickly raises the other one over the bar once the first has cleared. The body clears the bar upright.
- Scissor Kick: the jumper approaches the bar diagonally, throws first the inside leg and then the other over the bar in a scissoring motion. The body clears the bar upright.
- Eastern Cutoff: the jumper takes off like the scissor kick, but extends his back and flattens out over the bar.
- Western Roll:the jumper also approaches the bar diagonally, but the inner leg is used for the take-off, while the outer leg is thrust up to lead the body sideways over the bar.

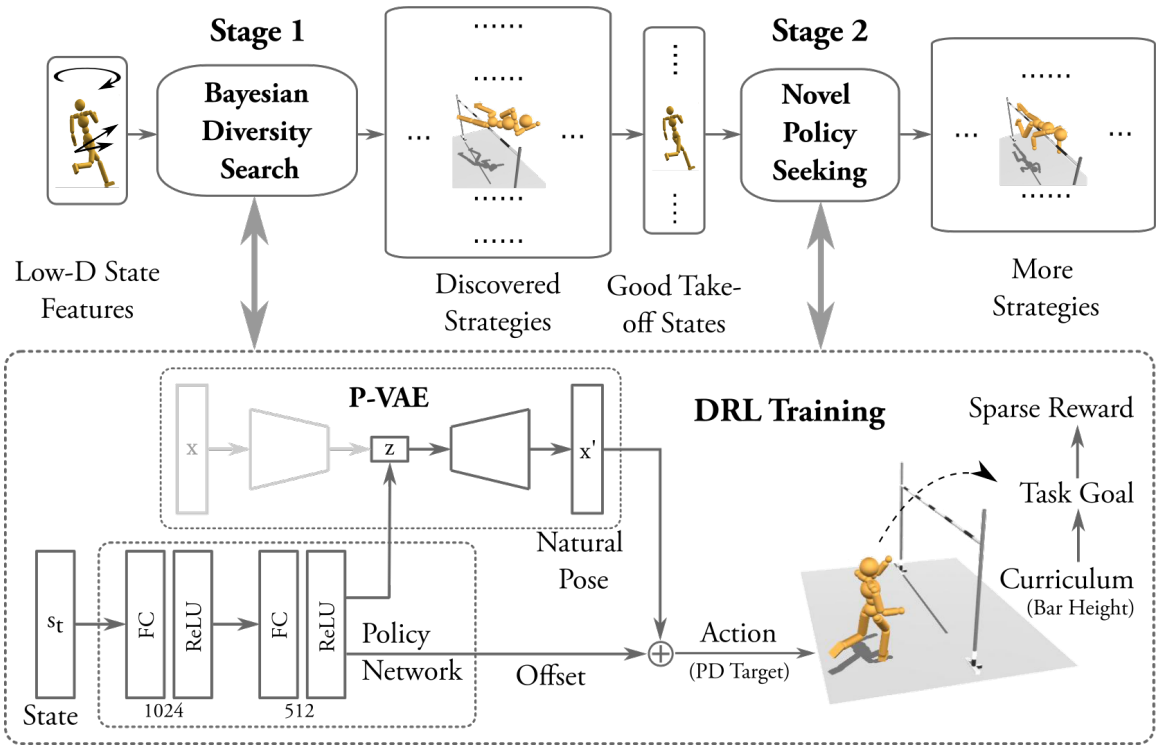


Figure 3.1: Overview of our strategy discovery framework.

- The Straddle: similar to Western Roll, but the jumper clears the bar face-down.
- Fosbury Flop: The jumper approaches the bar on a curved path and leans away from the bar at the take-off point to convert horizontal velocity into vertical velocity and angular momentum. In flight, the jumper progressively arches their shoulders, back, and legs in a rolling motion, and lands on their neck and back. The jumper's Center of Mass (CoM) can pass under the bar while the body arches and slide above the bar. It has been the favored high jump technique in Olympic competitions since used by Dick Fosbury in the 1968 Summer Olympics. It was concurrently developed by Debbie Brill.

In biomechanics, kinesiology, and physical education, high jumps have been analyzed to a limited extent. We adopt the force limits reported in [84] in our simulations. Dapena simulated a higher jump by making small changes to a recorded jump [22]. Mathematical models of the Center of Mass (CoM) movement have been developed to offer recommendations to increase the effectiveness of high jumps [2].

3.3 System Description

We now give a high-level description of our learning framework as illustrated in Figure 3.1. Our framework splits athletic jumps into two phases: a run-up phase and a jump phase. The

take-off state marks the transition between these two phases, and consists of a time instant midway through the last support phase before becoming airborne. The take-off state is key to our exploration strategy, as it is a strong determinant of the resulting jump strategy. We characterize the take-off state by a feature vector that captures key aspects of the state, such as the net angular velocity and body orientation. This defines a low-dimensional take-off feature space that we can sample in order to explore and evaluate a variety of motion strategies. While random sampling of take-off state features is straightforward, it is computationally impractical as evaluating one sample involves an expensive DRL learning process that takes hours even on modern machines. Therefore, we introduce a sample-efficient Bayesian Diversity Search (BDS) algorithm as a key part of our Stage 1 optimization process.

Given a specific sampled take-off state, we then need to produce an optimized run-up controller and a jump controller that result in the best possible corresponding jumps. This process has several steps. We first train a run-up controller, using deep reinforcement learning, that imitates a single generic run-up motion capture clip while also targeting the desired take-off state. For simplicity, the run-up controller and its training are not shown in Figure 3.1. These are discussed in Section 3.6.1. The main challenge lies with the synthesis of the actual jump controller which governs the remainder of the motion, and for which we wish to discover strategies without any recourse to known solutions.

The jump controller begins from the take-off state and needs to control the body during take-off, over the bar, and to prepare for landing. This poses a challenging learning problem because of the demanding nature of the task, the sparse fail/success rewards, and the difficulty of also achieving natural human-like movement. We apply two key insights to make this task learnable using deep reinforcement learning. First, we employ an action space defined by a subspace of natural human poses as modeled with a Pose Variational Autoencoder (P-VAE). Given an action parameterized as a target body pose, individual joint torques are then realized using PD-controllers. We additionally allow for regularized *offset* PD-targets that are added to the P-VAE targets to enable strong takeoff forces. Second, we employ a curriculum that progressively increases the task difficulty, i.e., the height of the bar, based on current performance.

A diverse set of strategies can already emerge after the Stage 1 BDS optimization. To achieve further strategy variations, we reuse the take-off states of the existing discovered strategies for another stage of optimization. The diversity is explicitly incentivized during this Stage 2 optimization via a novelty reward, which is focused specifically on features of the body pose at the peak height of the jump. As shown in Figure 3.1, Stage 2 makes use of the same overall DRL learning procedure as in Stage 1, albeit with a slightly different reward structure.

3.4 Learning Natural Strategies

Given a character model, an environment, and a task objective, we aim to learn feasible natural-looking motion strategies using deep reinforcement learning. We first describe our DRL formulation in Section 3.4.1. To improve the learned motion quality, we propose a Pose Variational Autoencoder (P-VAE) to constrain the policy actions in Section 3.4.2.

3.4.1 DRL Formulation

Our strategy learning task is formulated as a standard reinforcement learning problem, where the character interacts with the environment to learn a control policy which maximizes a long-term reward. The control policy $\pi_\theta(a|s)$ parameterized by θ models the conditional distribution over action $a \in \mathcal{A}$ given the character state $s \in \mathcal{S}$. At each time step t, the character interacts with the environment with action a_t sampled from $\pi(a|s)$ based on the current state s_t. The environment then responds with a new state s_{t+1} according to the transition dynamics $p(s_{t+1}|s_t, a_t)$, along with a reward signal r_t. The goal of reinforcement learning is to learn the optimal policy parameters θ^* which maximizes the expected return defined as

$$J(\theta) = \mathbb{E}_{\tau \sim p_\theta(\tau)} \left[\sum_{t=0}^{T} \gamma^t r_t \right], \tag{3.1}$$

where T is the episode length, $\gamma \leq 1$ is a discount factor, and $p_\theta(\tau)$ is the probability of observing trajectory $\tau = \{s_0, a_0, s_1, ..., a_{T-1}, s_T\}$ given the current policy $\pi_\theta(a|s)$.

States The state s describes the character configuration. We use a similar set of pose and velocity features as those proposed in DeepMimic [88], including relative positions of each link with respect to the root, their rotations parameterized in quaternions, along with their linear and angular velocities. Different from DeepMimic, our features are computed directly in the global frame without direction-invariant transformations for the studied jump tasks. The justification is that input features should distinguish states with different relative transformations between the character and the environment obstacle such as the crossbar. In principle, we could also use direction-invariant features as in DeepMimic, and include the relative transformation to the obstacle into the feature set. However, as proved in [75], there are no direction-invariant features that are always singularity free. Direction-invariant features change wildly whenever the character's facing direction approaches the chosen motion direction, which is usually the global up-direction or the Y-axis. For high jump techniques such as the Fosbury flop, singularities are frequently encountered as the athlete clears the bar facing upward. Therefore, we opt to use global features for simplicity and robustness. Another difference from DeepMimic is that time-dependent phase variables are not included in our feature set. Actions are chosen purely based on the dynamic state of the character.

Initial States The initial state s_0 is the state in which an agent begins each episode in DRL training. We explore a chosen low-dimensional feature space ($3 \sim 4D$) of the take-off states for learning diverse jumping strategies. As shown by previous work [76], the take-off moment is a critical point of jumping motions, where the volume of the feasible region of the dynamic skill is the smallest. In another word, bad initial states will fail fast, which in a way help our exploration framework to find good ones quicker. Alternatively, we could place the agent in a fixed initial pose to start with, such as a static pose before the run-up. This is problematic for several reasons. First, different jumping strategies need different length for the run-up. The planar position and facing direction of the root is still a three dimensional space to be explored. Second, the run-up strategies and the jumping strategies do not correlate in a one-to-one fashion. Visually, the run-up strategies do not look as diverse as the jumping strategies. Lastly, starting the jumps from a static pose lengthens the learning horizon, and makes our learning framework based on DRL training even more costly. Therefore we choose to focus on just the jumping part of the jumps in this work, and leave the run-up control learning to DeepMimic, which is one of the state-of-the-art imitation-based DRL learning methods. More details are given in Section 3.6.1.

Actions The action a is a target pose described by internal joint rotations. We parameterize 1D revolute joint rotations by scalar angles, and 3D spherical joint rotations by exponential maps [32]. Given a target pose and the current character state, joint torques are computed through the Stable Proportional Derivative (SPD) controllers [115]. Our control frequency f_{control} ranges from 10 Hz to 30 Hz depending on both the task and the curriculum. For challenging tasks like high jumps, it helps to quickly improve initial policies through stochastic evaluations at early training stages. A low-frequency policy enables faster learning by reducing the needed control steps, or in another word, the dimensionality and complexity of the actions $(a_0, ..., a_T)$. This is in spirit similar to the 10 Hz control fragments used in SAMCON-type controllers [71]. Successful low-frequency policies can then be gradually transferred to high-frequency ones according to a curriculum to achieve finer controls and thus smoother motions. We discuss the choice of control frequency in more detail in Section 3.6.1.

Reward We use a reward function consisting of the product of two terms for all our strategy discovery tasks as follows:

$$r = r_{\text{task}} \cdot r_{\text{naturalness}} \tag{3.2}$$

where r_{task} is the task objective and $r_{\text{naturalness}}$ is a naturalness reward term computed from the P-VAE to be described in Section 3.4.2. For diverse strategy discovery, a simple r_{task} which precisely captures the task objective is preferred. For example in high jumping, the agent receives a sparse reward signal at the end of the jump after it successfully clears

the bar. In principle, we could transform the sparse reward into a dense reward to reduce the learning difficulty, such as to reward CoM positions higher than a parabolic trajectory estimated from the bar height. However in practice, such dense guidance reward can mislead the training to a bad local optimum, where the character learns to jump high in place rather than clears the bar in a coordinated fashion. Moreover, the CoM height and the bar height may not correlate in a simple way. For example, the CoM passes underneath the crossbar in Fosbury flops. As a result, a shaped dense reward function could harm the diversity of the learned strategies. We will discuss reward function settings for each task in more details in Section 3.6.1.

Policy Representation We use a fully-connected neural network parameterized by θ to represent the control policy $\pi_\theta(a|s)$. Similar to the settings in [88], the network has two hidden layers with 1024 and 512 units respectively. ReLU activations are applied for all hidden units. Our policy maps a given state s to a Gaussian distribution over actions $a = \mathcal{N}(\mu(s), \Sigma)$. The mean $\mu(s)$ is determined by the network output. The covariance matrix $\Sigma = \sigma I$ is diagonal, where I is the identity matrix and σ is a scalar variable measuring the action noise. We apply an annealing strategy to linearly decrease σ from 0.5 to 0.1 in the first 1.0×10^7 simulation steps, to encourage more exploration in early training and more exploitation in late training.

Training We train our policies with the Proximal Policy Optimization (PPO) method [103]. PPO involves training both a policy network and a value function network. The value network architecture is similar to the policy network, except that there is only one single linear unit in the output layer. We train the value network with TD(λ) multi-step returns. We estimate the advantage of the PPO policy gradient by the Generalized Advantage Estimator GAE(λ) [102].

3.4.2 Pose Variational Autoencoder

The dimension of natural human poses is usually much lower than the true degrees of freedom of the character model. We propose a generative model to produce natural PD target poses at each control step. More specifically, we train a Pose Variational Autoencoder (P-VAE) from captured natural human poses, and then sample its latent space to produce desired PD target poses for control. Here a pose only encodes internal joint rotations without the global root transformations. We use publicly available human motion capture databases to train our P-VAE. Note that none of these databases consist of high jumps or obstacle jumps specifically, but they already provide enough poses for us to learn the natural human pose manifold successfully.

P-VAE Architecture and Training Our P-VAE adopts the standard Beta Variational Autoencoder (β-VAE) architecture [42]. The encoder maps an input feature x to a low-dimensional latent space, parameterized by a Gaussian distribution with a mean μ_x and a diagonal covariance Σ_x. The decoder maps a latent vector sampled from the Gaussian distribution back to the original feature space as x'. The training objective is to minimize the following loss function:

$$\mathcal{L} = \mathcal{L}_{\text{MSE}}(x, x') + \beta \cdot \text{KL}(\mathcal{N}(\mu_x, \Sigma_x), \mathcal{N}(0, I)), \tag{3.3}$$

where the first term is the MSE (Mean Squared Error) reconstruction loss, and the second term shapes the latent variable distribution to a standard Gaussian by measuring their Kulback-Leibler divergence. We set $\beta = 1.0 \times 10^{-5}$ in our experiments, so that the two terms in the loss function are within the same order of magnitude numerically.

We train the P-VAE on a dataset consisting of roughly $20,000$ poses obtained from the CMU and SFU motion capture databases. We include a large variety of motion skills, including walking, running, jumping, breakdancing, cartwheels, flips, kicks, martial arts, etc. The input features consist of all link and joint positions relative to the root in the local root frames, and all joint rotations with respect to their parents. We parameterize joint rotations by a 6D representation for better continuity, as described in [136, 67].

We model both the encoder and the decoder as fully connected neural networks with two hidden layers, each having 256 units with *tanh* activation. We perform PCA (Principal Component Analysis) on the training data and choose $d_{\text{latent}} = 13$ to cover 85% of the training data variance, where d_{latent} is the dimension of the latent variable. We use the Adam optimizer to update network weights [54], with the learning rate set to 1.0×10^{-4}. Using a mini-batch size of 128, the training takes 80 epochs within 2 minutes on an NVIDIA GeForce GTX 1080 GPU and an Intel i7-8700k CPU. We use this single pre-trained P-VAE for all our strategy discovery tasks to be described.

Composite PD Targets PD controllers provide actuation based on positional errors. So in order to reach the desired pose, the actual target pose needs to be offset by a certain amount. Such offsets are usually small to just counter-act the gravity for free limbs. However, for joints that interact with the environment, such as the lower body joints for weight support and ground takeoff, large offsets are needed to generate powerful ground reaction forces to propel the body forward or into the air. Such complementary offsets combined with the default P-VAE targets help realize natural poses during physics-based simulations. Our action space $\mathcal{A}$ is therefore $d_{\text{latent}} + d_{\text{offset}}$ dimensional, where d_{latent} is the dimension of the P-VAE latent space, and d_{offset} is the dimension of the DoFs that we wish to apply offsets for. We simply apply offsets to all internal joints in this work. Given $a = (a_{\text{latent}}, a_{\text{offset}}) \in \mathcal{A}$ sampled from the policy $\pi_\theta(a|s)$, where a_{latent} and a_{offset} correspond to the latent and offset

part of a respectively, the final PD target is computed by $D_{\text{pose}}(a_{\text{latent}})+a_{\text{offset}}$. Here $D_{\text{pose}}(\cdot)$ is a function that decodes the latent vector a_{latent} to full-body joint rotations. We minimize the usage of rotation offsets by a penalty term as follows:

$$r_{\text{naturalness}} = 1 - \text{Clip}\left(\left(\frac{||a_{\text{offset}}||_1}{c_{\text{offset}}}\right)^2, 0, 1\right), \tag{3.4}$$

where c_{offset} is the maximum offset allowed. For tasks with only a sparse reward signal at the end, $||a_{\text{offset}}||_1$ in Equation 3.4 is replaced by the average offset norm $\frac{1}{T}\sum_{t=0}^{T}||a_{\text{offset}}^{(t)}||_1$ across the entire episode. We use $L1$-norm rather than the commonly adopted $L2$-norm to encourage sparse solutions with fewer non-zero components [116, 12], as our goal is to only apply offsets to essential joints to complete the task while staying close to the natural pose manifold prescribed by the P-VAE.

3.5 Learning Diverse Strategies

Given a virtual environment and a task objective, we would like to discover as many strategies as possible to complete the task at hand. Without human insights and demonstrations, this is a challenging task. To this end, we propose a two-stage framework to enable stochastic DRL to discover solution modes such as the Fosbury flop.

The first stage focuses on strategy discovery by exploring the space of initial states. For example in high jump, the Fosbury flop technique and the straddle technique require completely different initial states at take-off, in terms of the approaching angle with respect to the bar, the take-off velocities, and the choice of inner or outer leg as the take-off leg. A fixed initial state may lead to success of one particular strategy, but can miss other drastically different ones. We systematically explore the initial state space through a novel sample-efficient Bayesian Diversity Search (BDS) algorithm to be described in Section 3.5.1.

The output of Stage 1 is a set of diverse motion strategies and their corresponding initial states. Taken such a successful initial state as input, we then apply another pass of DRL learning to further explore more motion variations permitted by the same initial state. The intuition is to explore different local optima while maximizing the novelty of the current policy, compared to previously found ones. We describe our detailed settings for the Stage 2 novel policy seeking algorithm in Section 3.5.2.

3.5.1 Stage 1: Initial States Exploration with Bayesian Diversity Search

In Stage 1, we perform diverse strategy discovery by exploring initial state variations, such as pose and velocity variations, at the take-off moment. We first extract a feature vector f from a motion trajectory to characterize and differentiate between different strategies. A straightforward way is to compute the Euclidean distance between time-aligned motion trajectories, but we hand pick a low-dimensional visually-salient feature set as detailed in

Section 3.6.1. We also define a low-dimensional exploration space $\mathcal{X}$ for initial states, as exploring the full state space is computationally prohibitive. Our goal is to search for a set of representatives $X_n = \{x_1, x_2, ..., x_n | x_i \in \mathcal{X}\}$, such that the corresponding feature set $F_n = \{f_1, f_2, ..., f_n | f_i \in \mathcal{F}\}$ has a large diversity. Note that as DRL training and physics-based simulation are involved in producing the motion trajectories from an initial state, the computation of $f_i = g(x_i)$ is a stochastic and expensive black-box function. We therefore design a sample-efficient Bayesian Optimization (BO) algorithm to optimize for motion diversity in a guided fashion.

Our BDS (Bayesian Diversity Search) algorithm iteratively selects the next sample to evaluate from $\mathcal{X}$, given the current set of observations $X_t = \{x_1, x_2, ..., x_t\}$ and $F_t = \{f_1, f_2, ..., f_t\}$. More specifically, the next point x_{t+1} is selected based on an acquisition function $a(x_{t+1})$ to maximize the diversity in $F_{t+1} = F_t \cup \{f_{t+1}\}$. We choose to maximize the minimum distance between f_{t+1} and all $f_i \in F_t$:

$$a(x_{t+1}) = \min_{f_i \in F_t} ||f_{t+1} - f_i||. \tag{3.5}$$

Since evaluating f_{t+1} through $g(\cdot)$ is expensive, we employ a surrogate model to quickly estimate f_{t+1}, so that the most promising sample to evaluate next can be efficiently found through Equation 3.5.

We maintain the surrogate statistical model of $g(\cdot)$ using a Gaussian Process (GP) [98], similar to standard BO methods. A GP contains a prior mean $m(x)$ encoding the prior belief of the function value, and a kernel function $k(x, x')$ measuring the correlation between $g(x)$ and $g(x')$. More details of our specific $m(x)$ and $k(x, x')$ are given in Section 3.6.1. Hereafter we assume a one-dimensional feature space $\mathcal{F}$. Generalization to a multi-dimensional feature space is straightforward as multi-output Gaussian Process implementations are readily available, such as [118]. Given $m(\cdot)$, $k(\cdot, \cdot)$, and current observations $\{X_t, F_t\}$, posterior estimation of $g(x)$ for an arbitrary x is given by a Gaussian distribution with mean μ_t and variance σ_t^2 computed in closed forms:

$$\begin{aligned} \mu_t(x) &= k(X_t, x)^T (K + \eta^2 I)^{-1} (F_t - m(x)) + m(x), \\ \sigma_t^2(x) &= k(x, x) + \eta^2 - k(X_t, x)^T (K + \eta^2 I)^{-1} k(X_t, x), \end{aligned} \tag{3.6}$$

where I is the identity matrix, η is the standard deviation of the observation noise, $X_t \in \mathbb{R}^{t \times \dim(\mathcal{X})}$, $F_t \in \mathbb{R}^t$, $K \in \mathbb{R}^{t \times t}$, $K_{i,j} = k(x_i, x_j)$, and $k(X_t, x) = [k(x, x_1), k(x, x_2), ...k(x, x_t)]^T$. Equation 3.5 can then be approximated by

$$\hat{a}(x_{t+1}) = \mathbb{E}_{\hat{f}_{t+1} \sim \mathcal{N}(\mu_t(x_{t+1}), \sigma_t^2(x_{t+1}))} \left[\min_{f_i \in F_t} ||\hat{f}_{t+1} - f_i|| \right]. \tag{3.7}$$

Equation 3.7 can be computed analytically for one-dimensional features, but gets more and more complicated to compute analytically as the feature dimension grows, or when

the feature space is non-Euclidean as in our case with rotational features. Therefore, we compute Equation 3.7 numerically with Monte-Carlo integration for simplicity.

The surrogate model is just an approximation to the true function, and has large uncertainty where observations are lacking. Rather than only maximizing the function value when picking the next sample, BO methods usually also take into consideration the estimated uncertainty to avoid being overly greedy. For example, GP-UCB (Gaussian Process Upper Confidence Bound), one of the most popular BO algorithms, adds a variance term into its acquisition function. Similarly, we could adopt a composite acquisition function as follows:

$$a'(x_{t+1}) = \hat{a}(x_{t+1}) + \beta\sigma_t(x_{t+1}), \tag{3.8}$$

where $\sigma_t(x_{t+1})$ is the heuristic term favoring candidates with large uncertainty, and β is a hyperparameter trading off exploration and exploitation (diversity optimization in our case). Theoretically well justified choice of β exists for GP-UCB, which guarantees optimization convergence with high probability [110]. However in our context, no such guarantees hold as we are not optimizing f but rather the diversity of f, the tuning of the hyperparameter β is thus not trivial, especially when the strategy evaluation function $g(\cdot)$ is extremely costly. To mitigate this problem, we decouple the two terms and alternate between exploration and exploitation following a similar idea proposed in [109]. During exploration, our acquisition function becomes:

$$a_{\text{exp}}(x_{t+1}) = \sigma_t(x_{t+1}). \tag{3.9}$$

The sample with the largest posterior standard deviation is chosen as x_{t+1} to be evaluated next:

$$x_{t+1} = \arg\max_x \sigma_t(x). \tag{3.10}$$

Under the condition that $g(\cdot)$ is a sample from GP function distribution $\mathcal{GP}(m(\cdot), k(\cdot,\cdot))$, Equation 3.10 can be shown to maximize the Information Gain I on function $g(\cdot)$:

$$x_{t+1} = \arg\max_x I\left(X_t \cup \{x\}, F_t \cup \{g(x)\}; g\right), \tag{3.11}$$

where $I(A;B) = H(A) - H(A|B)$, and $H(\cdot) = \mathbb{E}\left[-\log p(\cdot)\right]$ is the Shannon entropy [19].

We summarize our BDS algorithm in Algorithm 1. The alternation of exploration and diversity optimization involves two extra hyperparameters N_{exp} and N_{opt}, corresponding to the number of samples allocated for exploration and diversity optimization in each round. Compared to β in Equation 3.8, N_{exp} and N_{opt} are much more intuitive to tune. We also found that empirically the algorithm performance is insensitive to the specific values of N_{exp} and N_{opt}. The exploitation stage directly maximizes the diversity of motion strategies. We optimize $\hat{a}(\cdot)$ with a sampling-based method DIRECT (Dividing Rectangle) [49], since derivative information may not be accurate in the presence of function noise due to the Monte-Carlo integration. This optimization does not have to be perfectly accurate, since

Algorithm 1: Bayesian Diversity Search

Input: Strategy evaluation function $g(\cdot)$, exploration count N_{exp} and diversity optimization count N_{opt}, total sample count n.

Output: Initial states $X_n = \{x_1, x_2, ..., x_n\}$ for diverse strategies.

$t = 0$; $X_0 \leftarrow \varnothing$; $F_0 \leftarrow \varnothing$;
Initialize GP surrogate model with random samples;
while $t < n$ **do**
  if $t\%(N_{exp} + N_{opt}) < N_{exp}$ **then**
    $x_{t+1} \leftarrow \arg\max a_{\text{exp}}(\cdot)$ by L-BFGS; `// Equation 3.9`
  else
    $x_{t+1} \leftarrow \arg\max \hat{a}(\cdot)$ by DIRECT; `// Equation 3.7`
  end
  $f_{t+1} \leftarrow g(x_{t+1})$;
  $X_{t+1} \leftarrow X_t \cup \{x_{t+1}\}$; $F_{t+1} \leftarrow F_t \cup \{f_{t+1}\}$;
  Update GP surrogate model with X_{t+1}, F_{t+1}; `// Equation 3.6`
  $t \leftarrow t + 1$;
end
return X_n

the surrogate model is an approximation in the first place. The exploration stage facilitates the discovery of diverse strategies by avoiding repeated queries on well-sampled regions. We optimize $a_{\text{exp}}(\cdot)$ using a simple gradient-based method L-BFGS [68].

3.5.2 Stage 2: Novel Policy Seeking

In Stage 2 of our diverse strategy discovery framework, we explore potential strategy variations given a fixed initial state discovered in Stage 1. Formally, given an initial state x and a set of discovered policies $\Pi = \{\pi_1, \pi_2, ..., \pi_n\}$, we aim to learn a new policy π_{n+1} which is different from all existing $\pi_i \in \Pi$. This can be achieved with an additional policy novelty reward to be jointly optimized with the task reward during DRL training. We measure the novelty of policy π_i with respect to π_j by their corresponding motion feature distance $||f_i - f_j||$. The novelty reward function is then given by

$$r_{\text{novelty}}(f) = \text{Clip}\left(\frac{\min_{\pi_i \in \Pi} ||f_i - f||}{d_{\text{threshold}}}, 0.01, 1\right), \tag{3.12}$$

which rewards simulation rollouts showing different strategies to the ones presented in the existing policy set. $d_{\text{threshold}}$ is a hyperparameter measuring the desired policy novelty to be learned next. Note that the feature representation f here in Stage 2 can be the same as or different from the one used in Stage 1 for initial states exploration.

Our novel policy search is in principle similar to the idea of [133, 114]. However, there are two key differences. First, in machine learning, policy novelty metrics have been designed and validated only on low-dimensional control tasks. For example in [133], the policy novelty

is measured by the reconstruction error between states from the current rollout and previous rollouts encapsulated as a deep autoencoder. In our case of high-dimensional 3D character control tasks, however, novel state sequences do not necessarily correspond to novel motion strategies. We therefore opt to design discriminative strategy features whose distances are incorporated into the DRL training reward.

Second, we multiply the novelty reward with the task reward as the training reward, and adopt a standard gradient-based method PPO to train the policy. Additional optimization techniques are not required for learning novel strategies, such as the Task-Novelty Bisector method proposed in [133] that modifies the policy gradients to encourage novelty learning. Our novel policy seeking procedure always discovers novel policies since the character is forced to perform a different strategy. However, the novel policies may exhibit unnatural and awkward movements, when the given initial state is not capable of multiple natural strategies.

3.6 Task Setup and Implementation

We demonstrate diverse strategy discovery for two challenging motor tasks: high jumping and obstacle jumping. We also tackle several variations of these tasks. We describe task specific settings in Section 3.6.1, and implementation details in Section 3.6.2.

3.6.1 Task Setup

The high jump task follows the Olympics rules, where the simulated athlete takes off with one leg, clears the crossbar, and lands on a crash mat. We model the landing area as a rigid block for simplicity. The crossbar is modeled as a rigid wall vertically extending from the ground to the target height to prevent the character from cheating during early training, i.e., passing through beneath the bar. A rollout is considered successful and terminated when the character lands on the rigid box with all body parts at least 20 cm away from the wall. A rollout is considered as a failure and terminated immediately, if any body part touches the wall, or any body part other than the take-off foot touches the ground, or if the jump does not succeed within two seconds after the take-off.

The obstacle jump shares most of the settings of the high jump. The character takes off with one leg, clears a box-shaped obstacle of 50 cm in height with variable widths, then lands on a crash mat. The character is required to complete the task within two seconds as well, and not allowed to touch the obstacle with any body part.

Run-up Learning

A full high jump or obstacle jump consists of three phases: run-up, take-off and landing. Our framework described so far can discover good initial states at take-off that lead to diverse jumping strategies. What is lacking is the matching run-up control policies that can

Task	$z_{\min}$(cm)	$z_{\max}$(cm)	Δz(cm)	R_T
High jump	50	200	1	30
Obstacle jump	5	250	5	50

Table 3.1: Curriculum parameters for learning jumping tasks.

prepare the character to reach these good take-off states at the end of the run. We train the run-up controllers with DeepMimic [88], where the DRL learning reward consists of a task reward and an imitation reward. The task reward encourages the end state of the run-up to match the desired take-off state of the jump. The imitation reward guides the simulation to match the style of the reference run. We use a curved sprint as the reference run-up for high jump, and a straight sprint for the obstacle jump run-up. For high jump, the explored initial state space is four-dimensional: the desired approach angle α, the X and Z components of the root angular velocity ω, and the magnitude of the Z component of the root linear velocity v_z in a facing-direction invariant frame. We fix the desired root Y angular velocity to 3rad/s, which is taken from the reference curved sprint. In summary, the task reward r_{G} for the run-up control of a high jump is defined as

$$r_{\mathrm{G}} = \exp\left(-\frac{1}{3} \cdot ||\omega - \bar{\omega}||_1 - 0.7 \cdot (v_z - \bar{v}_z)^2\right), \tag{3.13}$$

where $\bar{\omega}$ and $\bar{v}_z$ are the corresponding targets for ω and v_z. α does not appear in the reward function as we simply rotate the high jump suite in the environment to realize different approach angles. For the obstacle jump, we explore a three-dimensional take-off state space consisting of the root angular velocities along all axes. Therefore the run-up control task reward r_{G} is given by

$$r_{\mathrm{G}} = \exp(-\frac{1}{3} \cdot ||\omega - \bar{\omega}||_1). \tag{3.14}$$

Reward Function

We use the same reward function structure for both high jumps and obstacle jumps, where the character gets a sparse reward only when it successfully completes the task. The full reward function is defined as in Equation 3.2 for Stage 1. For Stage 2, the novelty bonus r_{novelty} as discussed in Section 3.5.2 is added:

$$r = r_{\mathrm{task}} \cdot r_{\mathrm{naturalness}} \cdot r_{\mathrm{novelty}}. \tag{3.15}$$

$r_{\mathrm{naturalness}}$ is the motion naturalness term discussed in Section 3.4.2. For both stages, the task reward consists of three terms:

$$r_{\mathrm{task}} = r_{\mathrm{complete}} \cdot r_{\omega} \cdot r_{\mathrm{safety}}. \tag{3.16}$$

Parameter	Simulated Athlete	Mocap Athlete
Weight (kg)	60	70
Height (cm)	170	191
hip height (cm)	95	107
knee height (cm)	46	54

Table 3.2: Model parameters of our virtual athlete and the mocap athlete.

r_{complete} is a binary reward precisely corresponding to task completion. $r_\omega = \exp(-0.02||\omega||)$ penalizes excessive root rotations where $||\omega||$ is the average magnitude of the root angular velocities across the episode. r_{safety} is a term to penalize unsafe head-first landings. We set it to 0.7 for unsafe landings and 1.0 otherwise. r_{safety} can also be further engineered to generate more landing styles, such as a landing on feet as shown in Figure 3.8.

Curriculum and Scheduling

The high jump is a challenging motor skill that requires years of training even for professional athletes. We therefore adopt curriculum-based learning to gradually increase the task difficulty z, defined as the crossbar height in high jumps or the obstacle width in obstacle jumps. Detailed curriculum settings are given in Table 3.1, where $z_{\min}$ and $z_{\max}$ specify the range of z, and Δz is the increment when moving to a higher difficulty level.

We adaptively schedule the curriculum to increase the task difficulty according to the DRL training performance. At each training iteration, the average sample reward is added to a reward accumulator. We increase z by Δz whenever the accumulated reward exceeds a threshold R_T, and then reset the reward accumulator. Detailed settings for Δz and R_T are listed in Table 3.1. The curriculum could also be scheduled following a simpler scheme adopted in [124], where task difficulty is increased when the average sample reward in each iteration exceeds a threshold. We found that for athletic motions, such average sample reward threshold is hard to define uniformly for different strategies in different training stages.

Throughout training, the control frequency f_{control} and the P-VAE offset penalty coefficient c_{offset} in Equation 3.4 are also scheduled according to the task difficulty, in order to encourage exploration and accelerate training in early stages. We set $f_{\text{control}} = 10 + 20 \cdot \text{Clip}(\rho, 0, 1)$ and $c_{\text{offset}} = 48 - 33 \cdot \text{Clip}(\rho, 0, 1)$, where $\rho = 2z - 1$ for high jumps and $\rho = z$ for obstacle jumps. We find that in practice the training performance does not depend sensitively on these hyperparameters.

Strategy Features

We choose low-dimensional and visually discriminate features f of learned strategies for effective diversity measurement of discovered strategies. In the sports literature, high jump techniques are usually characterized by the body orientation when the athlete clears the bar

at his peak position. The rest of the body limbs are then coordinated in the optimal way to clear the bar as high as possible. Therefore we use the root orientation when the character's CoM lies in the vertical crossbar plane as f. This three-dimensional root orientation serves well as a Stage 2 feature for high jumps. For Stage 1, this feature can be further reduced to one dimension, as we will show in Section 3.7.1. More specifically, we only measure the angle between the character's root direction and the global up vector, which corresponds to whether the character clears the bar facing upward or downward. Such a feature does not require a non-Euclidean GP output space that we need to handle in Stage 1. We use the same set of features for obstacle jumps, except that root orientations are measured when the character's CoM lies in the center vertical plane of the obstacle.

Note that it is not necessary to train to completion, i.e., the maximum task difficulty, to evaluate the feature diversity, since the overall jumping strategy usually remains unchanged after a given level of difficulty, which we denote by z_{freeze}. Based on empirical observations, we terminate the training after reaching $z_{\text{freeze}} = 100cm$ for both high jump and obstacle jump tasks for strategy discovery.

GP Priors and Kernels

We set GP prior $m(\cdot)$ and kernel $k(\cdot, \cdot)$ for BDS based on common practices in the Bayesian optimization literature. Without any knowledge on the strategy feature distribution, we set the prior mean $m(\cdot)$ to be the mean of the value range of a feature. Among the many common choices for kernel functions, we adopt the Matérn$^5/_2$ kernel [77, 55], defined as:

$$k_{5/2}(x, x') = \theta(1 + \sqrt{5}d_\lambda(x, x') + \frac{5}{3}d_\lambda^2(x, x'))e^{-\sqrt{5}d_\lambda(x,x')} \tag{3.17}$$

where θ and λ are learnable parameters of the GP. $d_\lambda(x, x') = (x - x')^T \operatorname{diag}(\lambda)(x - x')$ is the Mahalanobis distance.

3.6.2 Implementation

We implemented our system in PyTorch [96] and PyBullet [18]. The simulated athlete has 28 internal DoFs and 34 DoFs in total. We run the simulation at 600 Hz. Torque limits for the hips, knees and ankles are taken from Biomechanics estimations for a human athlete performing a Fosbury flop [84]. Torque limits for other joints are kept the same as [88]. Joint angle limits are implemented by penalty forces. We captured three standard high jumps from a university athlete, whose body measurements are given in Table 3.2. For comparison, we also list these measurements for our virtual athlete.

For DRL training, we set $\lambda = 0.95$ for both TD(λ) and GAE(λ). We set the discounter factor $\gamma = 1.0$ since our tasks have short horizon and sparse rewards. The PPO clip threshold is set to 0.02. The learning rate is 2.5×10^{-5} for the policy network and 1.0×10^{-2} for the value network. In each training iteration, we sample 4096 state-action tuples in parallel

and perform five policy updates with a mini-batch size of 256. For Stage 1 diverse strategy discovery, we implement BDS using GPFlow [118] with both N_{exp} and N_{opt} set to three. $d_{\text{threshold}}$ in Stage 2 novel policy seeking is set to $\pi/2$. Our experiments are performed on a Dell Precision 7920 Tower workstation, with dual Intel Xeon Gold 6248R CPUs (3.0 GHz, 48 cores) and an Nvidia Quadro RTX 6000 GPU. Simulations are run on the CPUs. One strategy evaluation for a single initial state, i.e. Line 9 in Algorithm 1, typically takes about six hours. Network updates are performed on the GPU.

3.7 Results

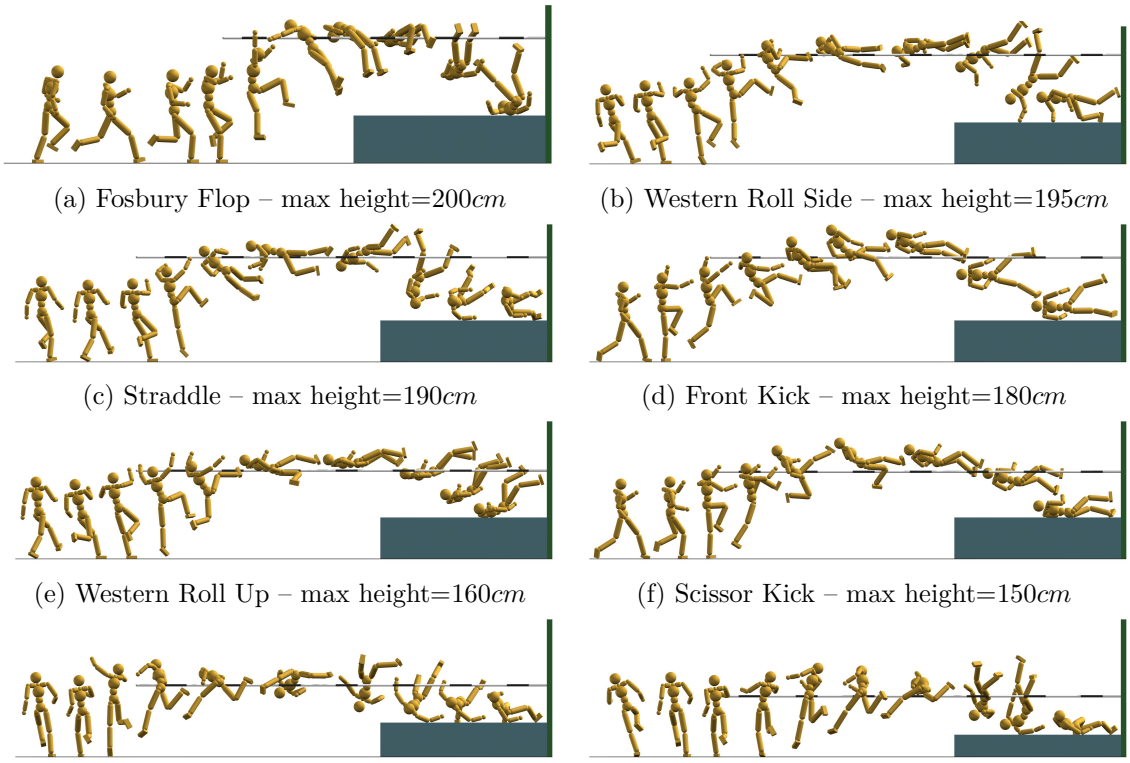
We demonstrate multiple strategies discovered through our framework for high jumping and obstacle jumping in Section 3.7.1. We validate the effectiveness of BDS and P-VAE in Section 3.7.2. Comparison with motion capture examples, and interesting variations of learned policies are given in Section 3.7.3. All results are best seen in the supplementary videos in order to judge the quality of the synthesized motions.

3.7.1 Diverse Strategies

High Jumps

In our experiments, six different high jump strategies are discovered during the Stage 1 initial state exploration within the first ten BDS samples: Fosbury Flop, Western Roll (facing up), Straddle, Front Kick, Side Jump, Side Dive. The first three are high jump techniques standard in the sports literature. The last three strategies are not commonly used in sporting events, but still physically valid so we name them according to their visual characteristics. The other four samples generated either repetitions or failures. Strategy repetitions are generally not avoidable due to model errors and large flat regions in the motion space. Since the evaluation of one BDS sample takes about six hours, the Stage 1 exploration takes about 60 hours in total. The discovered distinct strategies at $z_{\text{freeze}} = 100cm$ are further optimized separately to reach their maximum difficulty level, which takes another 20 hours. Representative take-off state feature values of the discovered strategies can be found in Table 3.3. Note that the approach angle α for high jumps is defined as the wall orientation in a facing-direction invariant frame. The orientation of the wall is given by the line $x\text{sin}\alpha - z\text{cos}\alpha = 0$.

In Stage 2, we perform novel policy search for five DRL iterations from each good initial state of Stage 1. Training is warm started with the associated Stage 1 policy for efficient learning. The total time required for Stage 2 is roughly 60 hours. More strategies are discovered in Stage 2, but most are repetitions and only two of them are novel strategies not discovered in Stage 1: Western Roll (facing sideways) and Scissor Kick. Western Roll (sideways) shares the same initial state with Western Roll (up). Scissor Kick shares the same initial state with Front Kick. The strategies discovered in each stage are summarized



(a) Fosbury Flop – max height=200cm

(b) Western Roll Side – max height=195cm

(c) Straddle – max height=190cm

(d) Front Kick – max height=180cm

(e) Western Roll Up – max height=160cm

(f) Scissor Kick – max height=150cm

(g) Side Dive – max height=130cm

(h) Side Jump – max height=110cm

Figure 3.2: Eight high jump strategies discovered by our learning framework, ordered by their maximum cleared height.

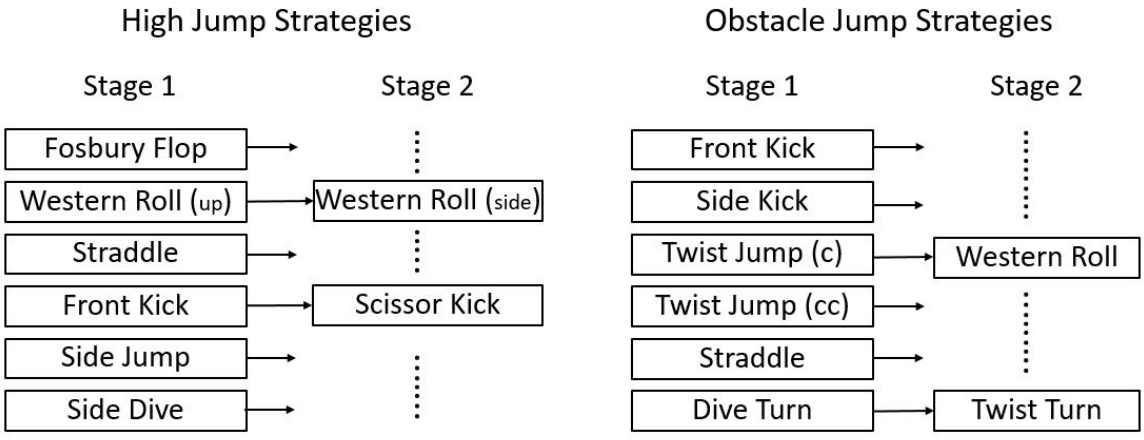
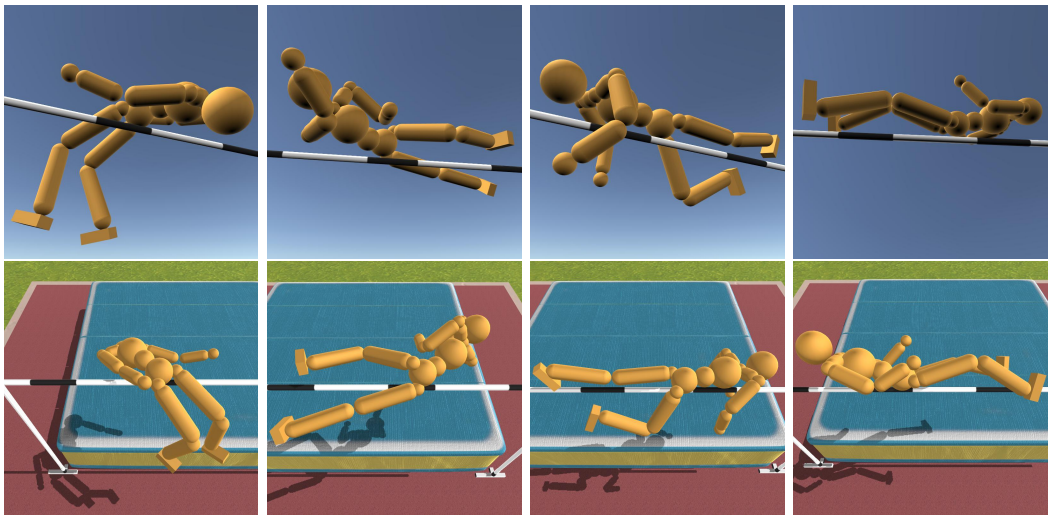
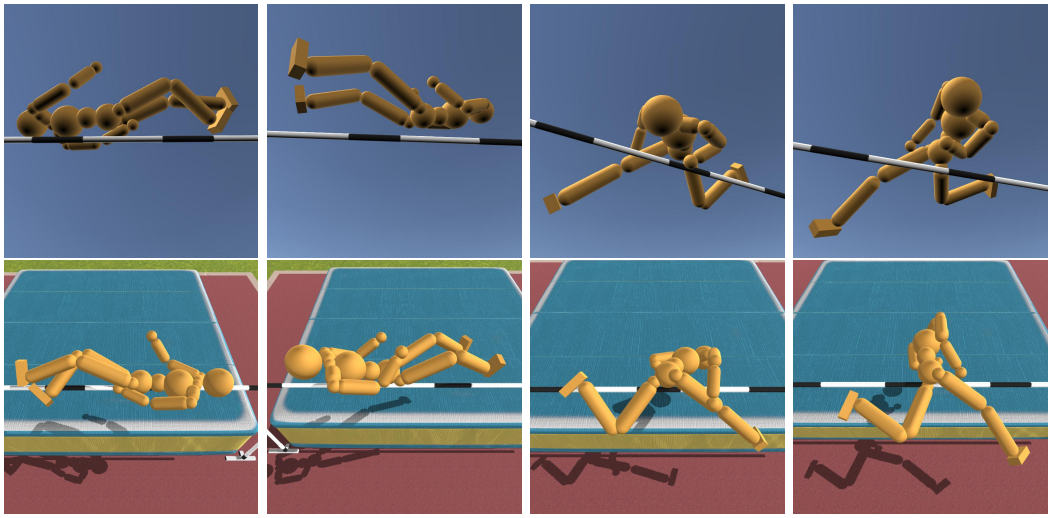


Figure 3.3: Diverse strategies discovered in each stage of our framework.



(a) Fosbury Flop (b) Western Roll (c) Straddle (d) Front Kick



(e) Western Roll* (f) Scissor (g) Side Dive (h) Side Jump

Figure 3.4: Peak poses of discovered high jump strategies including look-up views and look-down views, ordered by their maximum cleared height.

Strategy	v_z	ω_x	ω_z	α
Fosbury Flop	-2.40	-3.00	1.00	-0.05
Western Roll (up)	-0.50	1.00	-1.00	2.09
Straddle	-2.21	1.00	0.88	1.65
Front Kick	-0.52	1.00	-0.26	0.45
Side Dive	-1.83	-2.78	-0.32	1.18
Side Jump	-1.99	-1.44	0.44	0.70

Table 3.3: Representative take-off state features for discovered high jumps.

Strategy	ω_x	ω_y	ω_z
Front Kick	1.15	-1.11	3.89
Side Kick	3.00	3.00	-2.00
Twist Jump (c)	-1.50	1.50	-2.00
Straddle	0.00	0.00	1.00
Twist Jump (cc)	-2.67	0.00	-1.44
Dive Turn	-0.74	-2.15	-0.41

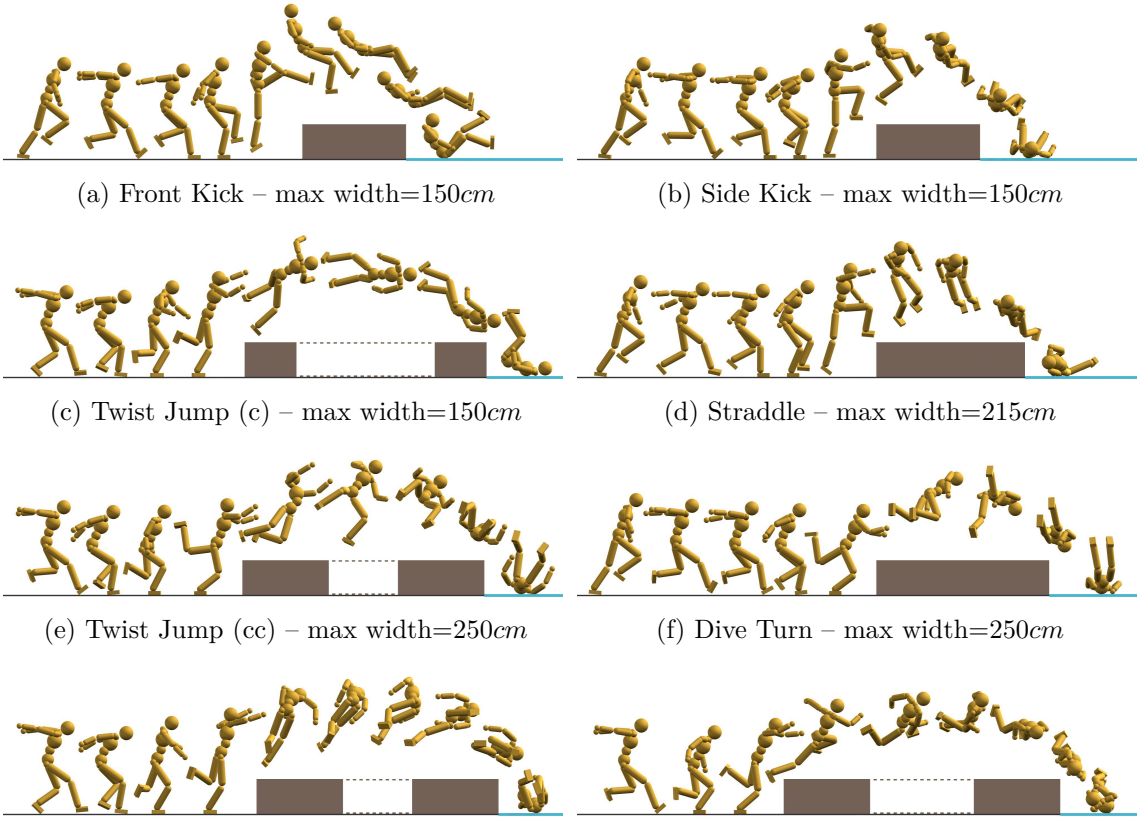
Table 3.4: Representative take-off state features for discovered obstacle jumps.

in Figure 3.3. We visualize all eight distinct strategies in Figure 3.2. We also visualize their peak poses in Figure 3.4. Note that Western Roll (facing up) and Scissor Kick differ in the choice of inner or outer leg as the take-off leg. The Western Roll (facing sideways) and the Scissor Kick are learned in Stage 2. All other strategies are discovered in Stage 1.

While the final learned control policies are stochastic in nature, the majority of the results shown in our supplementary video are the deterministic version of those policies, i.e., using the mean of the learned policy action distributions. In the video we further show multiple simulations from the final stochastic policies, to help give insight into the true final endpoint of the optimization. As one might expect for a difficult task such as a maximal-height high jump, these stochastic control policies will also fail for many of the runs, similar to a professional athlete.

Obstacle Jumps

Figure 3.5 shows the eight different obstacle jump strategies discovered by our learning framework. The first six strategies are discovered in Stage 1 within the first 17 BDS samples: Front Kick, Side Kick, Twist Jump (clockwise), Twist Jump (counterclockwise), Straddle and Dive Turn. The last two strategies are the novel ones discovered in Stage 2: Western Roll and Twist Turn. Western Roll shares the initial state with Twist Jump (clockwise). Twist Turn shares the initial state with Dive Turn. For some of the strategies, the obstacle is split into two parts connected with dashed lines to enable better visualization of the poses over the obstacle. The two stages together take about 180 hours. Representative take-off state



(a) Front Kick – max width=150cm

(b) Side Kick – max width=150cm

(c) Twist Jump (c) – max width=150cm

(d) Straddle – max width=215cm

(e) Twist Jump (cc) – max width=250cm

(f) Dive Turn – max width=250cm

(g) Twist Turn – max width=250cm

(h) Western Roll – max width=250cm

Figure 3.5: Eight obstacle jump strategies discovered by our learning framework.

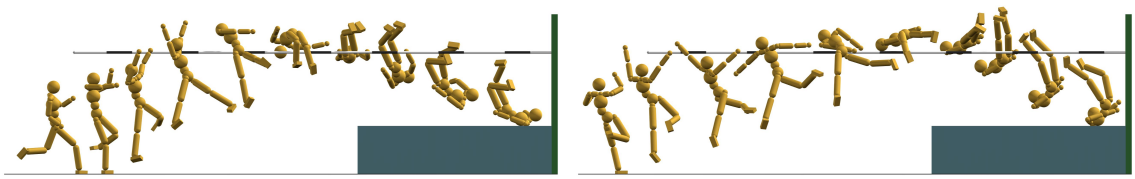
feature values of the discovered strategies can be found in Table 3.4. We encourage readers to watch the supplementary video for better visual perception of the learned strategies.

Although our obstacle jump task is not an Olympic event, it is analogous to a long jump in that it seeks to jump a maximum-length jumped. Setting the obstacle height to zero yields a standard long jump task. The framework discovers several strategies, including one similar to the standard long jump adopted in competitive events, with the strong caveat that the distance achieved is limited by the speed of the run up. Please refer to the supplementary video for the long jump results.

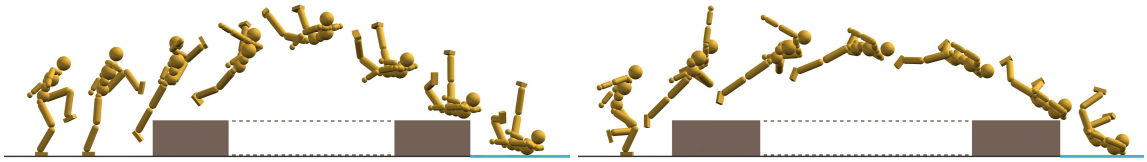
3.7.2 Validation and Ablation Study

BDS vs. Random Search

We validate the sample efficiency of BDS compared with a random search baseline. Within the first ten samples of initial states exploration in the high jump task, BDS discovered six distinct strategies as discussed in Section 3.7.1. Given the same computational budget, random search only discovered three distinct strategies: Straddle, Side Jump, and one strategy similar to Scissor Kick. Most samples result in repetitions of these three strategies, due to



(a) High jumps trained without P-VAE, given the initial state of Fosbury Flop and Straddle respectively. Please compare with Figure 3.2a and Figure 3.2c.



(b) Obstacle jumps trained without P-VAE, given the initial state of Straddle and Twist Jump (cc) respectively. Please compare with Figure 3.5d and Figure 3.5e.

Figure 3.6: Jumping strategies learned without P-VAE. Although the character can still complete the tasks, the poses are less natural.

the presence of large flat regions in the strategy space where different initial states lead to the same strategy. In contrast, BDS largely avoids sampling the flat regions thanks to the acquisition function for diversity optimization and guided exploration of the surrogate model.

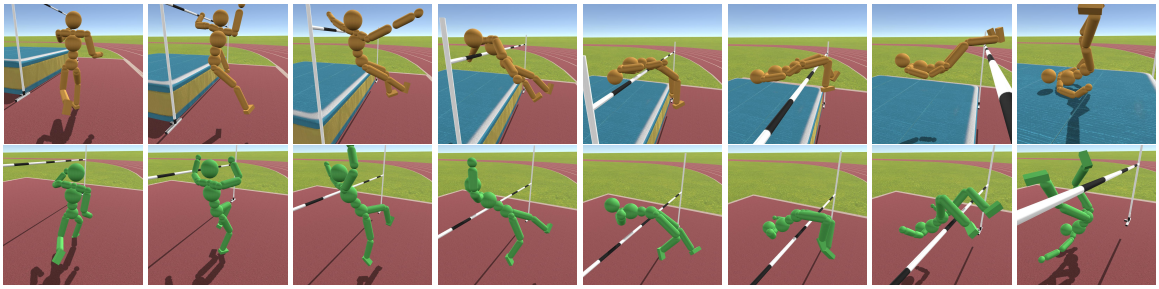
Motion Quality with/without P-VAE

We justify the usage of P-VAE for improving motion naturalness with results shown in Figure 3.6. Without P-VAE, the character can still learn physically valid skills to complete the tasks successfully, but the resulting motions usually exhibit unnatural behavior. In the absence of a natural action space constrained by the P-VAE, the character can freely explore any arbitrary pose during the course of the motion to complete the task, which is unlikely to be within the natural pose manifold all the time.

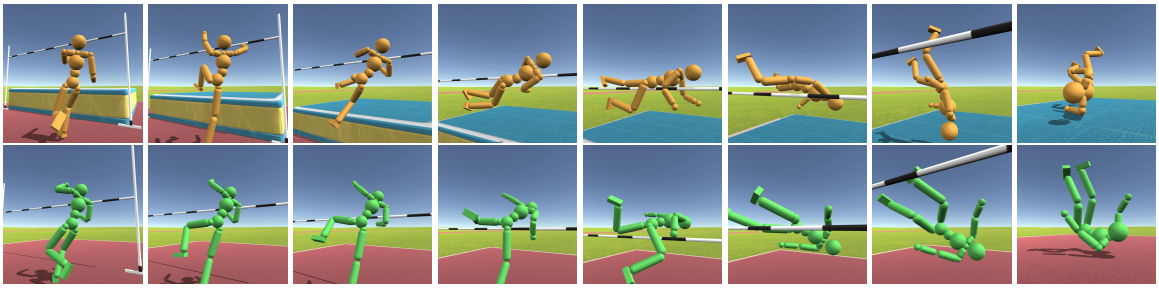
3.7.3 Comparison and Variations

Synthesized High Jumps vs. Motion Capture

We capture motion capture examples from a university athlete in a commercial motion capture studio for three well-known high jump strategies: Scissor Kick, Straddle, and Fosbury Flop. We retarget the mocap examples onto our virtual athlete, which is shorter than the real athlete as shown in Table 3.2. We visualize keyframes sampled from our simulated jumps and the retargeted mocap jumps in Figure 3.7. Note that the bar heights are set to the maximum heights achievable by our trained policies, while the bar heights for the mocap examples are just the bar heights used at the actual capture session. We did not set



(a) Fosbury Flop. First row: synthesized – max height=200cm; Second row: motion capture – capture height=130cm.



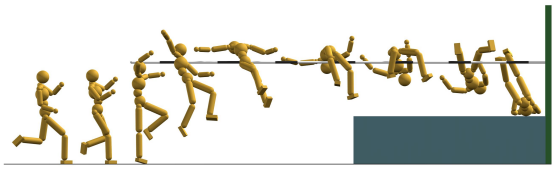
(b) Straddle. First row: synthesized – max height=195cm; Second row: motion capture – capture height=130cm.

Figure 3.7: Comparison of our synthesized high jumps with those captured from a human athlete.

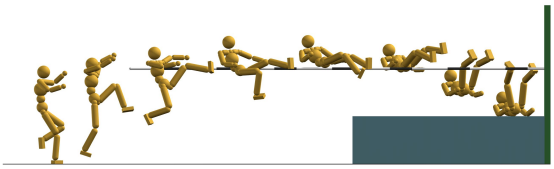
the mocap bar heights at the athlete's personal record height, as we wanted to ensure his safety and comfort while jumping in a tight mocap suit with a lot of markers on.

High Jump Variations

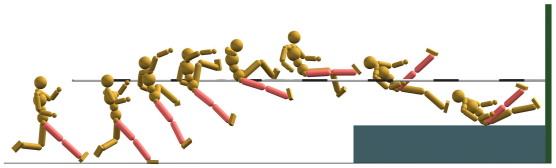
In addition to discovering multiple motion strategies, our framework can easily support physically valid motion variations. We show four high jump variations generated from our framework in Figure 3.8. We generate the first three variations by taking the initial state of the Fosbury Flop strategy discovered in Stage 1, and retrain the jumping policy with additional constraints starting from a random initial policy. Figure 3.8a shows a jump with a weaker take-off leg, where the torque limits are reduced to 60% of its original values. Figure 3.8b shows a character jumping with a spine that does not permit backward arching. Figure 3.8c shows a character jumping with a fixed knee joint. All these variations clear lower maximum heights, and are visually different from the original Fosbury Flop in Figure 3.2a. For the jump in Figure 3.8d, we take the initial state of the Front Kick, and train with an additional constraint that requires landing on feet. In Figure 3.9 we also show a high jump trained on Mars, where the gravity $g = 3.711 m/s^2$ is lower, from the initial state of the Fosbury flop discovered on Earth.



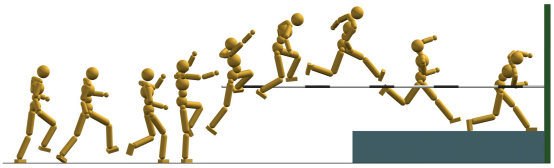
(a) Fosbury Flop performed by a character with a weaker take-off leg – max height=160cm.



(b) Fosbury Flop performed by a character with an inflexible spine – max height=150cm.



(c) Scissor Kick learned by a character with a cast on its take-off leg – max height=130cm.



(d) Front Kick with an additional constraint requiring landing on feet – max height=120cm.

Figure 3.8: High jump variations.

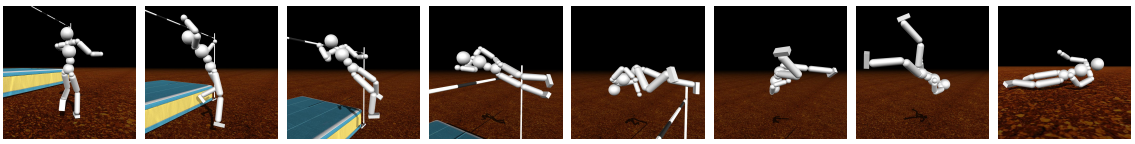


Figure 3.9: High jump policy trained on Mars with a lower gravity ($g = 3.711m/s^2$), given the initial state of the Fosbury Flop discovered on Earth.

3.7.4 Numerical Analysis

We plot Stage 1 DRL learning and curriculum scheduling curves for two high jump strategies in Figure 3.10. As DRL learning is stochastic, the curves shown are the average of five training runs. The shaded regions indicates the standard deviation. An initial solution for the starting bar height $0.5m$ can be learned relatively quickly. After a certain bar height has been reached (around $1.4m$), the return starts to drop because larger action offsets are needed to jump higher, which decreases the $r_{naturalness}$ in Equation 3.4 and therefore the overall return in Equation 3.2. Subjectively speaking, the learned motions remain as natural for high crossbars, as the lower return is due to the penalty on action offsets.

3.8 Summary and Discussions

In this chapter, we have presented a framework for discovering and learning multiple natural and distinct strategies for highly challenging athletic jumping motions. A key insight is to explore the take-off state, which is a strong determinant of the jump strategy that follows once airborne. In a second phase, we additionally use explicit rewards for novel motions. Another crucial aspect is to constrain the action space inside the natural human pose manifold. With the proposed two-stage framework and the pose variational autoencoder, natural and physically-nuanced jumping strategies emerge automatically without any reference to

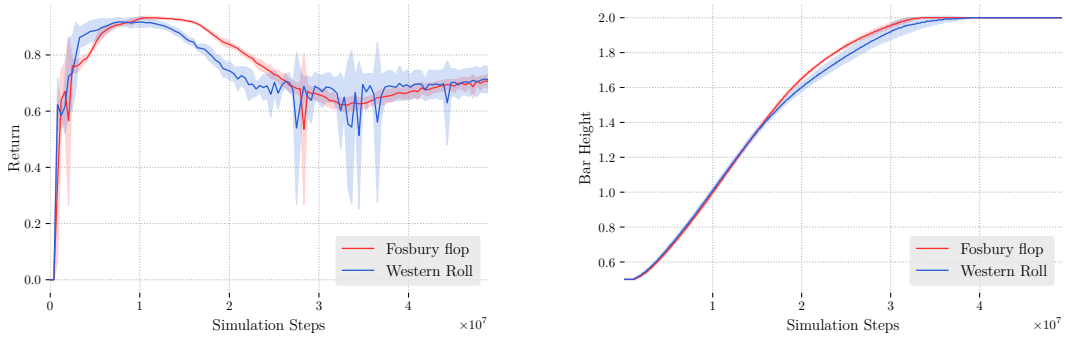


Figure 3.10: DRL learning and curriculum scheduling curves for two high jump strategies.

human demonstrations. Within the proposed framework, the take-off state exploration is specific to jumping tasks, while the diversity search algorithms in both stages and the P-VAE are task independent. We leave further adaptation of the proposed framework to additional motor skills as future work. We believe this work demonstrates a significant advance by being able to learn a highly-technical skill such as high-jumping.

The run-up phase of both jump tasks imitates reference motions, one single curved run for all the high jumps and one single straight run for all the obstacle jumps. The quality of the two reference runs affect the quality of not only the run-up controllers, but also the learned jump controllers. This is because the jump controllers couple with the run-up controllers through the take-off states, for which we only explore a low-dimensional feature space. The remaining dimensions of the take-off states stay the same as in the original reference run. As a result, the run-up controllers for our obstacle jumps remain in medium speed, and the swing leg has to kick backward sometimes in order for the body to dive forward. If we were to use a faster sprint with more forward leaning as the reference run, the discovered jumps could potentially be more natural and more capable to clear wider obstacles. Similarly, we did not find the Hurdle strategy for high jumping, likely due to the reference run being curved rather than straight. In both reference runs, there is a low in-place jump after the last running step. We found this jumping intention successfully embedded into the take-off states, which helped both jump controllers to jump up naturally. Using reference runs that anticipate the intended skills is definitely recommended, although retraining the run-up controller and the jump controller together in a post-processing stage may be helpful as well.

We were able to discover most well-known high-jump strategies, and some lesser-known variations. There remains a rich space of further parameters to consider for optimization, with our current choices being a good fit for our available computational budget. It would be exciting to discover a better strategy than the Fosbury flop, but a better strategy may not exist. We note that Stage 1 can discover most of the strategies shown in Figure 3.3. Stage 2 is only used to search for additional unique strategies and not to fine tune the strategies

already learned in Stage 1. We also experimented with simply running Stage 1 longer with three times more samples for the BDS. However, this could not discover any new strategies that can be discovered by Stage 2. In summary, Stage 2 is not absolutely necessary for our framework to work, but it complements Stage 1 in terms of discovering additional visually distinctive strategies. We also note that our Stage 2 search for novel policies is similar in spirit to the algorithm proposed in [133]. An advantage of our approach is its simplicity and the demonstration of its scalability to the discovery of visually distinct strategies for athletic skills.

There are many directions for further improving our system. First, we have only focused on strategy discovery for the take-off and airborne parts of jumping tasks. For landing, we only required not to land on head first. We did not model get-ups at all. How to seamlessly incorporate landing and get-ups into our framework is a worthy problem for future studies. Second, there is still room to further improve the quality of our synthesized motions. The P-VAE only constrains naturalness at a pose level, while ideally we need a mechanism to guarantee naturalness on a motion level. This is especially helpful for under-constrained motor tasks such as crawling, where feasible regions of the tasks are large and system dynamics cannot help prune a large portion of the state space as for the jumping tasks. Lastly, our strategy discovery is computationally expensive. We are only able to explore initial states in a four dimensional space, limited by our computational resources. If more dimensions could be explored, more strategies might be discovered. Parallel implementation is trivial for Stage 2 since searches for novel policies for different initial states are independent. For Stage 1, batched BDS where multiple points are queried together, similar to the idea of [6], may be worth exploring. The key challenge of such an approach is how to find a set of good candidates to query simultaneously.

Chapter 4

Conclusion and Future Work

In this thesis, we studied the problem of designing high-quality controllers for physically simulated characters. We studied the problem on two levels, joint-level motor controls and high-level character motion controls. We have presented a linear time Modified Articulated-Body Algorithm for stable PD motor control. MABA enables efficient learning of high-level motion controllers for its superior run-time complexity and stability. We further studied the problem of designing high-level motion controllers for the discovery of motion strategies in a DRL framework given a task objective. We have demonstrated a diverse set of natural-looking athletic jumping strategies discovered through our framework with the help of P-VAE, BDS and novel policy seeking.

There are two major avenues for future research endeavors. First, the proposed stable PD motor control algorithm and the strategy discovery system both assume that the characters are modeled with articulated rigid bodies. This simplification enables efficient model construction, simulation and control, but also bounds the performance and realism of the learned motions. For example, in our system for discovering diverse high jumps, we simplify the athlete's feet and specialized high jump shoes as rigid rectangular boxes, which reduces the maximum heights the virtual athlete can clear. We model the high jump crossbar as a wall at training time and as a rigid bar at run time, while real bars are made from more elastic materials such as fiberglass. We use a rigid box as the landing surface, while real-world landing cushions protect the athlete from breaking his neck and back, and also help him roll and get up in a fluid fashion. A more accurate modeling of the character and the environment could potentially lead to significant improvement to motion realism. Characters based on a muscle-skeletal-tendon system and soft-tissue simulation are worth exploring. However, the challenge is that muscle-skeletal-tendon systems and soft bodies cannot be efficiently simulated based on currently available algorithms and implementations. DRL-based motion learning frameworks are therefore impractical for such complicated systems due to the huge amount of simulation samples required for learning motion controllers successfully. Improving simulation efficiency of advanced character models and sample efficiency of DRL learning systems could be the key problem to be investigated in future research.

Synthesizing natural motion strategies without mocap demonstration is still hard to achieve for some motor tasks within our proposed learning system. Given a task objective, our learning system finds the best strategy to complete the task within the search region. The learned strategy is not guaranteed to match the expected ones from a human's perspective. For example, given an initial state for crawling and an objective to move forward with a desired velocity, the character usually learns a nice natural-looking rolling motion rather than crawling. The underlying reason is that rolling is easier than crawling given a simple reward function requiring it to maintain a certain speed. To synthesize a desired motion exactly, reward shaping and additional constraints are likely required and could be tedious for certain tasks. For example, to synthesize the desired crawling motion, we may need to add contact constraints and adjust the reward function to take body orientation into consideration. One potential approach to this problem worth exploring in future research is to decouple strategy discovery and full-body motion learning, where strategy discovery is performed first on a simplified character model, and the full-body motion is then learned with P-VAE to follow the discovered strategy on simplified models. Nonetheless, we believe that our work opens the door to efficiently synthesizing natural-looking human motions which can be hard or expensive to capture, and discovering novel strategies for certain tasks with less human insights.

Bibliography

[1] Joshua Achiam, Harrison Edwards, Dario Amodei, and Pieter Abbeel. Variational option discovery algorithms. *arXiv preprint arXiv:1807.10299*, 2018.

[2] V.M. Adashevskiy, S.S. Iermakov, and A.A. Marchenko. Biomechanics aspects of technique of high jump. *Physical Education of Students*, 17(2):11–17, 2013.

[3] Shailen Agrawal, Shuo Shen, and Michiel van de Panne. Diverse motion variations for physics-based character animation. In *Proceedings of the 12th ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 37–44, 2013.

[4] Mazen Al Borno, Martin de Lasa, and Aaron Hertzmann. Trajectory optimization for full-body movements with complex contacts. *TVCG*, 19(8):1405–1414, 2013.

[5] Sheldon Andrews and Paul G. Kry. Goal directed multi-finger manipulation: Control policies and analysis. *Computers & Graphics*, 37(7):830 – 839, 2013.

[6] Javad Azimi, Alan Fern, and Xiaoli Z Fern. Batch bayesian optimization via simulation matching. In *Advances in Neural Information Processing Systems*, pages 109–117. Citeseer, 2010.

[7] David Baraff. Linear-time dynamics using lagrange multipliers. In *SIGGRAPH*, pages 137–146, 1996.

[8] Kevin Bergamin, Simon Clavet, Daniel Holden, and James Forbes. DReCon: data-driven responsive control of physics-based characters. *ACM Transctions on Graphics*, 38(6), 2019.

[9] Eric Brochu, Abhijeet Ghosh, and Nando de Freitas. Preference galleries for material design. *SIGGRAPH Posters*, 105(10.1145):1280720–1280834, 2007.

[10] Bullet. Bullet physics library, 2015. http://bulletphysics.org.

[11] Jinxiang Chai and Jessica K. Hodgins. Performance animation from low-dimensional control signals. In *ACM SIGGRAPH 2005 Papers*. Association for Computing Machinery, New York, NY, USA, 2005.

[12] Scott Shaobing Chen, David L Donoho, and Michael A Saunders. Atomic decomposition by basis pursuit. *SIAM review*, 43(1):129–159, 2001.

[13] Matei Ciocarlie. *Low-Dimensional Robotic Grasping: Eigengrasp Subspaces and Optimized Underactuation*. PhD thesis, Columbia University, 2010.

[14] Simon Clavet. Motion matching and the road to next-gen animation. In *GCD*, 2016.

[15] Alexandra Coman and Hector Munoz-Avila. Generating diverse plans using quantitative and qualitative plan distance metrics. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 25, 2011.

[16] Edoardo Conti, Vashisht Madhavan, Felipe Petroski Such, Joel Lehman, Kenneth Stanley, and Jeff Clune. Improving exploration in evolution strategies for deep reinforcement learning via a population of novelty-seeking agents. In *Advances in neural information processing systems*, pages 5027–5038, 2018.

[17] Stelian Coros, Philippe Beaudoin, and Michiel van de Panne. Generalized biped walking control. *ACM Transctions on Graphics*, 29(4):Article 130, 2010.

[18] Erwin Coumans and Yunfei Bai. Pybullet, a python module for physics simulation for games, robotics and machine learning. `http://pybullet.org`, 2016–2019.

[19] Thomas M Cover. *Elements of information theory*. John Wiley & Sons, 1999.

[20] Ana Lucia Cruz Ruiz, Charles Pontonnier, Jonathan Levy, and Georges Dumont. A synergy-based control solution for overactuated characters: Application to throwing. *Computer Animation and Virtual Worlds*, 28(6):e1743, 2017.

[21] Marco da Silva, Yeuhi Abe, and Jovan Popović. Interactive simulation of stylized human locomotion. In *ACM SIGGRAPH 2008 papers*, pages 1–10. Association for Computing Machinery, 2008.

[22] Jesus Dapena. The evolution of high jumping technique: Biomechanical analysis. In *20th Internat. Symp. Biomech. Sports*, 2002.

[23] Martin de Lasa, Igor Mordatch, and Aaron Hertzmann. Feature-based locomotion controllers. In *ACM Transactions on Graphics (TOG)*, volume 29, page 131. ACM, 2010.

[24] Sean Donnelly. *An Introduction to the High Jump*. unknown, 2014.

[25] Benjamin Eysenbach, Abhishek Gupta, Julian Ibarz, and Sergey Levine. Diversity is all you need: Learning skills without a reward function. In *ICLR*, 2019.

[26] Roy Featherstone. A beginner's guide to 6-D vectors (part 1). *IEEE Robotics & Automation*, 17(3):83–94, 2010.

[27] Roy Featherstone. A beginner's guide to 6-D vectors (part 2). *IEEE Robotics & Automation*, 17(4):88–99, 2010.

[28] Roy Featherstone. *Rigid body dynamics algorithms*. Springer, 2014.

[29] Martin L Felis and Katja Mombaur. Synthesis of full-body 3d human gait using optimal control methods. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1560–1566. IEEE, 2016.

[30] Peter Frazier, Warren Powell, and Savas Dayanik. The knowledge-gradient policy for correlated normal beliefs. *INFORMS journal on Computing*, 21(4):599–613, 2009.

[31] Thomas Geijtenbeek, Michiel Van De Panne, and A Frank Van Der Stappen. Flexible muscle-based locomotion for bipedal creatures. *ACM Transactions on Graphics (TOG)*, 32(6):1–11, 2013.

[32] F Sebastian Grassia. Practical parameterization of rotations using the exponential map. *Journal of graphics tools*, 3(3):29–48, 1998.

[33] Gaël Guennebaud, Benoît Jacob, et al. Eigen v3, 2010.

[34] Sehoon Ha and C Karen Liu. Iterative training of dynamic skills inspired by human coaching techniques. *ACM Transactions on Graphics (TOG)*, 34(1):1–11, 2014.

[35] Sehoon Ha, Yuting Ye, and C Karen Liu. Falling and landing motion control for character animation. *ACM Transactions on Graphics (TOG)*, 31(6):1–9, 2012.

[36] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *arXiv preprint arXiv:1801.01290*, 2018.

[37] Ikhsanul Habibie, Daniel Holden, Jonathan Schwarz, Joe Yearsley, and Taku Komura. A recurrent variational autoencoder for human motion synthesis. In *28th British Machine Vision Conference*, 2017.

[38] Emmanuel Hebrard, Brahim Hnich, Barry O'Sullivan, and Toby Walsh. Finding diverse and similar solutions in constraint programming. In *AAAI*, volume 5, pages 372–377, 2005.

[39] Nicolas Heess, Dhruva TB, Srinivasan Sriram, Jay Lemmon, Josh Merel, Greg Wayne, Yuval Tassa, Tom Erez, Ziyu Wang, SM Eslami, et al. Emergence of locomotion behaviours in rich environments. *ArXiv*, abs/1707.02286, 2017.

[40] Nicolas Heess, Gregory Wayne, David Silver, Tim Lillicrap, Tom Erez, and Yuval Tassa. Learning continuous control policies by stochastic value gradients. In *Advances in Neural Information Processing Systems*, pages 2944–2952, 2015.

[41] Todd Hester and Peter Stone. Intrinsically motivated model learning for developing curious robots. *Artificial Intelligence*, 247:170–186, 2017.

[42] Irina Higgins, Loïc Matthey, Arka Pal, Christopher Burgess, Xavier Glorot, Matthew Botvinick, Shakir Mohamed, and Alexander Lerchner. beta-vae: Learning basic visual concepts with a constrained variational framework. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017.

[43] J. K. Hodgins, W. L. Wooten, D. C. Brogan, and J. F. O'Brien. Animating human athletics. In *Proceedings of SIGGRAPH 1995*, pages 71–78, 1995.

[44] Daniel Holden, Taku Komura, and Jun Saito. Phase-functioned neural networks for character control. *ACM Transctions on Graphics*, 36(4), 2017.

[45] Daniel Holden, Jun Saito, and Taku Komura. A deep learning framework for character motion synthesis and editing. *ACM Transctions on Graphics*, 35(4):Article 138, 2016.

[46] Rein Houthooft, Xi Chen, Yan Duan, John Schulman, Filip De Turck, and Pieter Abbeel. Vime: Variational information maximizing exploration. *Advances in neural information processing systems*, 29:1109–1117, 2016.

[47] Atil Iscen, Ken Caluwaerts, Jie Tan, Tingnan Zhang, Erwin Coumans, Vikas Sindhwani, and Vincent Vanhoucke. Policies modulating trajectory generators. In *Proceedings of The 2nd Conference on Robot Learning*, pages PMLR 87:916–926, 2018.

[48] Sumit Jain, Yuting Ye, and C Karen Liu. Optimization-based interactive motion synthesis. *ACM Transactions on Graphics (TOG)*, 28(1):1–12, 2009.

[49] Donald R. Jones. *Direct global optimization algorithmDirect Global Optimization Algorithm*, pages 431–440. Springer US, Boston, MA, 2001.

[50] Donald R Jones, Matthias Schonlau, and William J Welch. Efficient global optimization of expensive black-box functions. *Journal of Global optimization*, 13(4):455–492, 1998.

[51] Teen Jumper. 7 classic high jump styles, 2020.

[52] Kirthevasan Kandasamy, Willie Neiswanger, Jeff Schneider, Barnabas Poczos, and Eric P Xing. Neural architecture search with bayesian optimisation and optimal transport. In *Advances in neural information processing systems*, pages 2016–2025, 2018.

[53] Kirthevasan Kandasamy, Karun Raju Vysyaraju, Willie Neiswanger, Biswajit Paria, Christopher R Collins, Jeff Schneider, Barnabas Poczos, and Eric P Xing. Tuning hyperparameters without grad students: Scalable and robust bayesian optimisation with dragonfly. *Journal of Machine Learning Research*, 21(81):1–27, 2020.

[54] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[55] Aaron Klein, Stefan Falkner, Simon Bartels, Philipp Hennig, and Frank Hutter. Fast bayesian optimization of machine learning hyperparameters on large datasets. In *Artificial Intelligence and Statistics*, pages 528–536. PMLR, 2017.

[56] Evangelos Kokkevis. Practical physics for articulated characters. In *Game Developers Conference*, volume 2004, 2004.

[57] Ksenia Korovina, Sailun Xu, Kirthevasan Kandasamy, Willie Neiswanger, Barnabas Poczos, Jeff Schneider, and Eric Xing. Chembo: Bayesian optimization of small organic molecules with synthesizable recommendations. In *International Conference on Artificial Intelligence and Statistics*, pages 3393–3403. PMLR, 2020.

[58] Lucas Kovar, Michael Gleicher, and Frédéric Pighin. Motion graphs. *ACM Transctions on Graphics*, 21(3):473–482, 2002.

[59] Yuki Koyama, Issei Sato, and Masataka Goto. Sequential gallery for interactive visual design optimization. *ACM Transactions on Graphics*, 39(4), Jul 2020.

[60] Yuki Koyama, Issei Sato, Daisuke Sakamoto, and Takeo Igarashi. Sequential line search for efficient visual design optimization by crowds. *ACM Transactions on Graphics (TOG)*, 36(4):1–11, 2017.

[61] Jeongseok Lee, Michael Grey, Sehoon Ha, Tobias Kunz, Sumit Jain, Yuting Ye, Siddhartha Srinivasa, Mike Stilman, and Chuanjian Liu. Dart: Dynamic animation and robotics toolkit. *Journal of Open Source Software*, 3(22):500, 2018.

[62] Kyungho Lee, Seyoung Lee, and Jehee Lee. Interactive character animation by learning multi-objective control. *ACM Transactions on Graphics (TOG)*, 37(6):1–10, 2018.

[63] Yoonsang Lee, Sungeun Kim, and Jehee Lee. Data-driven biped control. In *ACM SIGGRAPH 2010 papers*, pages 1–8. Association for Computing Machinery, 2010.

[64] Yoonsang Lee, Moon Seok Park, Taesoo Kwon, and Jehee Lee. Locomotion control for many-muscle humanoids. *ACM Transactions on Graphics (TOG)*, 33(6):1–11, 2014.

[65] Joel Lehman and Kenneth O Stanley. Novelty search and the problem with objectives. In *Genetic programming theory and practice IX*, pages 37–56. Springer, 2011.

[66] Sergey Levine, Jack M Wang, Alexis Haraux, Zoran Popović, and Vladlen Koltun. Continuous character control with low-dimensional embeddings. *ACM Transactions on Graphics (TOG)*, 31(4):1–10, 2012.

[67] Hung Yu Ling, Fabio Zinno, George Cheng, and Michiel van de Panne. Character controllers using motion vaes. *ACM Trans. Graph.*, 39(4), 2020.

[68] Dong C Liu and Jorge Nocedal. On the limited memory bfgs method for large scale optimization. *Mathematical programming*, 45(1-3):503–528, 1989.

[69] Libin Liu and Jessica Hodgins. Learning to schedule control fragments for physics-based characters using deep q-learning. *ACM Transactions on Graphics (TOG)*, 36(3):1–14, 2017.

[70] Libin Liu and Jessica Hodgins. Learning basketball dribbling skills using trajectory optimization and deep reinforcement learning. *ACM Transactions on Graphics*, 37(4), August 2018.

[71] Libin Liu, Michiel van de Panne, and KangKang Yin. Guided learning of control graphs for physics-based characters. *ACM Transctions on Graphics*, 35(3):Article 29, 2016.

[72] Libin Liu, KangKang Yin, and Baining Guo. Improving sampling-based motion control. *Computer Graphics Forum*, 34(2):415–423, 2015.

[73] Libin Liu, KangKang Yin, Michiel van de Panne, Tianjia Shao, and Weiwei Xu. Sampling-based contact-rich motion control. *ACM Transctions on Graphics*, 29(4), 2010.

[74] Libin Liu, KangKang Yin, Bin Wang, and Baining Guo. Simulation and control of skeleton-driven soft body characters. *ACM Transactions on Graphics (TOG)*, 32(6):1–8, 2013.

[75] Li-Ke Ma, Zeshi Yang, Baining Guo, and KangKang Yin. Towards robust direction invariance in character animation. *Computer Graphics Forum*, 38(7):1–8, 2019.

[76] Li-ke Ma, Zeshi Yang, Xin Tong, Baining Guo, and Kangkang Yin. Learning and exploring motor skills with spacetime bounds. *Computer Graphics Forum*, 2021.

[77] B Matérn. Spatial variation: Meddelanden fran statens skogsforskningsinstitut. *Lecture Notes in Statistics*, 36:21, 1960.

[78] Paul Merrell, Eric Schkufza, Zeyang Li, Maneesh Agrawala, and Vladlen Koltun. Interactive furniture layout using interior design guidelines. *ACM transactions on graphics (TOG)*, 30(4):1–10, 2011.

[79] Brian Vincent Mirtich. *Impulse-based dynamic simulation of rigid body systems*. University of California, Berkeley, 1996.

[80] Igor Mordatch, Kendall Lowrey, Galen Andrew, Zoran Popovic, and Emanuel V Todorov. Interactive control of diverse complex characters with neural networks. *Advances in neural information processing systems*, 28:3132–3140, 2015.

[81] Igor Mordatch, Emanuel Todorov, and Zoran Popović. Discovery of complex behaviors through contact-invariant optimization. *ACM SIGGRAPH*, 31(4):Article 43, 2012.

[82] Igor Mordatch, Jack M Wang, Emanuel Todorov, and Vladlen Koltun. Animating human lower limbs using contact-invariant optimization. *ACM Transactions on Graphics (TOG)*, 32(6):1–8, 2013.

[83] Uldarico Muico, Yongjoon Lee, Jovan Popović, and Zoran Popović. Contact-aware nonlinear control of dynamic characters. In *ACM SIGGRAPH 2009 papers*, pages 1–9. Association for Computing Machinery, 2009.

[84] K. Okuyama, M. Ae, and T. Yokozawa. Three dimensional joint torque of the takeoff leg in the fosbury flop style. In *Proceedings of the XIXth Congress of the International Society of the Biomechanics (CD-ROM)*, 2003.

[85] T. Osa, J. Peters, and G. Neumann. Hierarchical reinforcement learning of multiple grasping strategies with human instructions. *Advanced Robotics*, 32:955–968, 2018.

[86] SA Overduin, A d'Avella, J. Roh, JM Carmena, and E. Bizzi. Representation of muscle synergies in the primate brain. *Journal of Neuroscience*, 37, 2015.

[87] Soohwan Park, Hoseok Ryu, Seyoung Lee, Sunmin Lee, and Jehee Lee. Learning predict-and-simulate policies from unorganized human motion data. *ACM Transctions on Graphics*, 38(6), 2019.

[88] Xue Bin Peng, Pieter Abbeel, Sergey Levine, and Michiel van de Panne. DeepMimic: Example-guided deep reinforcement learning of physics-based character skills. *ACM Transctions on Graphics*, 37(4), 2018.

[89] Xue Bin Peng, Glen Berseth, and Michiel Van de Panne. Terrain-adaptive locomotion skills using deep reinforcement learning. *ACM Transactions on Graphics (TOG)*, 35(4):1–12, 2016.

[90] Xue Bin Peng, Glen Berseth, KangKang Yin, and Michiel van de Panne. DeepLoco: Dynamic locomotion skills using hierarchical deep reinforcement learning. *ACM Transactions on Graphics (Proc. SIGGRAPH 2017)*, 36(4), 2017.

[91] Xue Bin Peng, Michael Chang, Grace Zhang, Pieter Abbeel, and Sergey Levine. Mcp: Learning composable hierarchical control with multiplicative compositional policies. In *Advances in Neural Information Processing Systems 32*, pages 3681–3692. Curran Associates, Inc., 2019.

[92] Xue Bin Peng, Angjoo Kanazawa, Jitendra Malik, Pieter Abbeel, and Sergey Levine. SFV: Reinforcement learning of physical skills from videos. *ACM Transctions on Graphics*, 37(6), 2018.

[93] Xue Bin Peng and Michiel van de Panne. Learning locomotion skills using DeepRL: Does the choice of action space matter? *CoRR*, abs/1611.01055, 2016.

[94] PhysX. NVIDIA PhysX SDK, 2019. https://github.com/NVIDIAGameWorks/PhysX.

[95] Justin K Pugh, Lisa B Soros, and Kenneth O Stanley. Quality diversity: A new frontier for evolutionary computation. *Frontiers in Robotics and AI*, 3:40, 2016.

[96] PyTorch. Pytorch, 2018. https://pytorch.org/.

[97] Avinash Ranganath, Pei Xu, Ioannis Karamouzas, and Victor Zordan. Low dimensional motor skill learning using coactivation. In *Motion, Interaction and Games*, pages 1–10. Association for Computing Machinery, 2019.

[98] Carl Edward Rasmussen. Gaussian processes in machine learning. In *Summer School on Machine Learning*, pages 63–71. Springer, 2003.

[99] Alla Safonova and Jessica K. Hodgins. Construction and optimal search of interpolated motion graphs. *ACM Transctions on Graphics*, 26(3):106–es, 2007.

[100] Alla Safonova, Jessica K Hodgins, and Nancy S Pollard. Synthesizing physically realistic human motion in low-dimensional, behavior-specific spaces. *ACM Transctions on Graphics*, 23(3):514–521, 2004.

[101] Jürgen Schmidhuber. Curious model-building control systems. In *Proc. international joint conference on neural networks*, pages 1458–1463, 1991.

[102] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2016.

[103] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017.

[104] Moonseok Park Seunghwan Lee, Kyoungmin Lee and Jehee Lee. Scalable muscle-actuated human simulation and control. *ACM Transactions on Graphics (Proc. SIGGRAPH 2019)*, 38(4), 2019.

[105] Archit Sharma, Shixiang Gu, Sergey Levine, Vikash Kumar, and Karol Hausman. Dynamics-aware unsupervised discovery of skills. *arXiv preprint arXiv:1907.01657*, 2019.

[106] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical bayesian optimization of machine learning algorithms. In *Advances in neural information processing systems*, pages 2951–2959, 2012.

[107] Jasper Snoek, Oren Rippel, Kevin Swersky, Ryan Kiros, Nadathur Satish, Narayanan Sundaram, Mostofa Patwary, Mr Prabhat, and Ryan Adams. Scalable bayesian optimization using deep neural networks. In *International conference on machine learning*, pages 2171–2180, 2015.

[108] Kwang Won Sok, Manmyung Kim, and Jehee Lee. Simulating biped behaviors from human motion data. In *ACM SIGGRAPH 2007 papers*, pages 107–es. Association for Computing Machinery, 2007.

[109] Jialin Song, Yuxin Chen, and Yisong Yue. A general framework for multi-fidelity bayesian optimization with gaussian processes. In *The 22nd International Conference on Artificial Intelligence and Statistics*, pages 3158–3167. PMLR, 2019.

[110] Niranjan Srinivas, Andreas Krause, Sham Kakade, and Matthias Seeger. Gaussian process optimization in the bandit setting: No regret and experimental design. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ICML'10, page 1015–1022, Madison, WI, USA, 2010. Omnipress.

[111] Biplav Srivastava, Tuan Anh Nguyen, Alfonso Gerevini, Subbarao Kambhampati, Minh Binh Do, and Ivan Serina. Domain independent approaches for finding diverse plans. In *IJCAI*, pages 2016–2022, 2007.

[112] Sebastian Starke, He Zhang, Taku Komura, and Jun Saito. Neural state machine for character-scene interactions. *ACM Trans. Graph.*, 38(6):209–1, 2019.

[113] Sebastian Starke, Yiwei Zhao, Taku Komura, and Kazi Zaman. Local motion phases for learning multi-contact character movements. *ACM Transactions on Graphics (TOG)*, 39(4):54–1, 2020.

[114] Hao Sun, Zhenghao Peng, Bo Dai, Jian Guo, Dahua Lin, and Bolei Zhou. Novel policy seeking with constrained optimization, 2020.

[115] Jie Tan, Karen Liu, and Greg Turk. Stable proportional-derivative controllers. *IEEE Computer Graphics and Applications*, 31(4):34–44, 2011.

[116] Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society: Series B (Methodological)*, 58(1):267–288, 1996.

[117] Rasmus K Ursem. Diversity-guided evolutionary algorithms. In *International Conference on Parallel Problem Solving from Nature*, pages 462–471. Springer, 2002.

[118] Mark van der Wilk, Vincent Dutordoir, ST John, Artem Artemev, Vincent Adam, and James Hensman. A framework for interdomain and multioutput Gaussian processes. *arXiv:2003.01115*, 2020.

[119] M. W. Walker and D. E. Orin. Efficient Dynamic Computer Simulation of Robotic Mechanisms. *Journal of Dynamic Systems, Measurement, and Control*, 104(3):205–211, 09 1982.

[120] Jack M Wang, David J Fleet, and Aaron Hertzmann. Optimizing walking controllers. In *ACM SIGGRAPH Asia 2009 papers*, pages 1–8. Association for Computing Machinery, 2009.

[121] Jack M Wang, Samuel R Hamner, Scott L Delp, and Vladlen Koltun. Optimizing locomotion controllers using biologically-based actuators and objectives. *ACM Transactions on Graphics (TOG)*, 31(4):1–11, 2012.

[122] Jungdam Won, Deepak Gopinath, and Jessica Hodgins. A scalable approach to control diverse behaviors for physically simulated characters. *ACM Transactions on Graphics (TOG)*, 39(4):Article 33, 2020.

[123] Wayne Lewis Wooten. *Simulation of Leaping, Tumbling, Landing, and Balancing Humans*. PhD thesis, Georgia Institute of Technology, USA, 1998. AAI9827367.

[124] Zhaoming Xie, Hung Yu Ling, Nam Hee Kim, and Michiel van de Panne. Allsteps: Curriculum-driven learning of stepping stone skills. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 2020.

[125] Yuting Ye and C Karen Liu. Optimal feedback control for character animation using an abstract model. In *ACM SIGGRAPH 2010 papers*, pages 1–9. Association for Computing Machinery, 2010.

[126] Yuting Ye and C Karen Liu. Synthesis of responsive motion using a dynamic model. In *Computer Graphics Forum*, volume 29, pages 555–562. Wiley Online Library, 2010.

[127] KangKang Yin, Kevin Loken, and Michiel van de Panne. SIMBICON: Simple biped locomotion control. *ACM Transctions on Graphics*, 26(3):Article 105, 2007.

[128] Zhiqi Yin, Zeshi Yang, Michiel van de Panne, and KangKang Yin. Discovering diverse athletic jumping strategies. *ACM Transactions on Graphics (Proc. SIGGRAPH 2021)*, 40(4), 2021.

[129] Zhiqi Yin and KangKang Yin. Linear time stable pd controllers for physics-based character animation. *Computer Graphics Forum*, 39(8):191–200, 2020.

[130] Wenhao Yu, Greg Turk, and C Karen Liu. Learning symmetric and low-energy locomotion. *ACM Transactions on Graphics*, 37(4), 2018.

[131] Ye Yuan and Kris Kitani. Ego-pose estimation and forecasting as real-time PD control. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 10082–10092, 2019.

[132] He Zhang, Sebastian Starke, Taku Komura, and Jun Saito. Mode-adaptive neural networks for quadruped motion control. *ACM Transctions on Graphics*, 37(4), 2018.

[133] Yunbo Zhang, Wenhao Yu, and Greg Turk. Learning novel policies for tasks. *arXiv preprint arXiv:1905.05252*, 2019.

[134] K. Zhao, Z. Zhang, H. Wen, Z. Wang, and J. Wu. Modular organization of muscle synergies to achieve movement behaviors. *Journal of Healthcare Engineering*, 2019.

[135] Peng Zhao and Michiel van de Panne. User interfaces for interactive control of physics-based 3d characters. In *I3D: ACM SIGGRAPH 2005 Symposium on Interactive 3D Graphics and Games*, 2005.

[136] Yi Zhou, Connelly Barnes, Jingwan Lu, Jimei Yang, and Hao Li. On the continuity of rotation representations in neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5745–5753, 2019.

[137] Victor B. Zordan and Jessica K. Hodgins. Motion capture driven simulations that hit and react. In *SCA*, pages 89–96, 2002.