Learning Basketball Dribbling Skills Using Trajectory Optimization and Deep Reinforcement Learning

LIBIN LIU, DeepMotion Inc., USA
JESSICA HODGINS, Carnegie Mellon University, USA

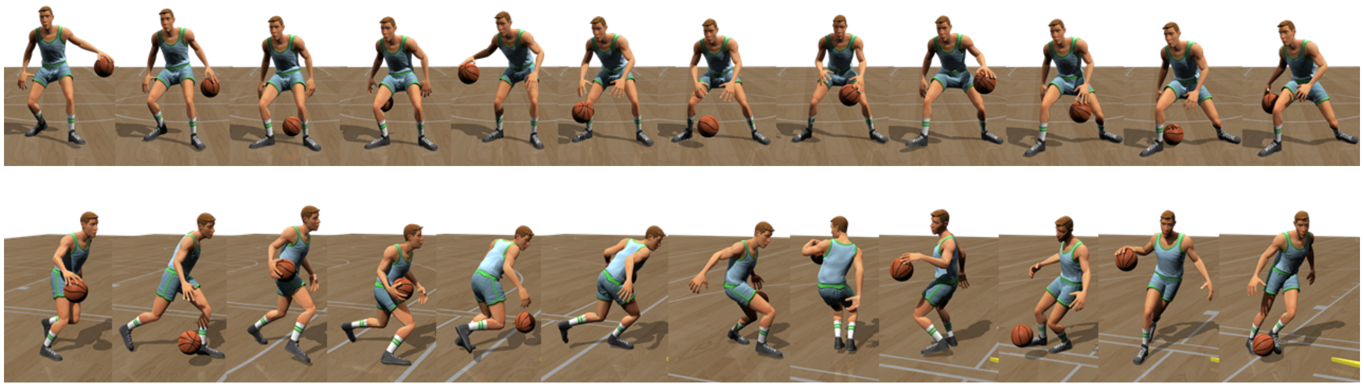


Fig. 1. Real-time simulation of the learned basketball skills. Top: A player dribbles a ball behind the back and between the legs. Bottom: A player performs crossover moves.

Basketball is one of the world's most popular sports because of the agility and speed demonstrated by the players. This agility and speed makes designing controllers to realize robust control of basketball skills a challenge for physics-based character animation. The highly dynamic behaviors and precise manipulation of the ball that occur in the game are difficult to reproduce for simulated players. In this paper, we present an approach for learning robust basketball dribbling controllers from motion capture data. Our system decouples a basketball controller into locomotion control and arm control components and learns each component separately. To achieve robust control of the ball, we develop an efficient pipeline based on trajectory optimization and deep reinforcement learning and learn non-linear arm control policies. We also present a technique for learning skills and the transition between skills simultaneously. Our system is capable of learning robust controllers for various basketball dribbling skills, such as dribbling between the legs and crossover moves. The resulting control graphs enable a simulated player to perform transitions between these skills and respond to user interaction.

CCS Concepts: • **Computing methodologies → Physical simulation**; *Neural networks*; *Motion capture*;

Authors' addresses: Libin Liu, DeepMotion Inc. 3 Twin Dolphin Dr. Suite 295, Redwood City, CA, 94065, USA, libin@deepmotion.com; Jessica Hodgins, Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, PA, 15213, USA, jkh@cs.cmu.edu.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
© 2018 Copyright held by the owner/author(s). Publication rights licensed to the Association for Computing Machinery.
0730-0301/2018/8-ART142 $15.00
https://doi.org/10.1145/3197517.3201315

Additional Key Words and Phrases: physics-based characters, human simulation, motion control, basketball, deep deterministic policy gradient

ACM Reference Format:
Libin Liu and Jessica Hodgins. 2018. Learning Basketball Dribbling Skills Using Trajectory Optimization and Deep Reinforcement Learning. *ACM Trans. Graph.* 37, 4, Article 142 (August 2018), 14 pages. https://doi.org/10.1145/3197517.3201315

1 INTRODUCTION

Basketball is one of the world's most popular sports. Animating basketball sports with motion capture data is challenging because the precise coupling between the movement of the ball and the motion of the player can be easily damaged during kinematic operations such as blending and deformation. Although the state-of-the-art basketball video games, such as the NBA 2K series and the NBA Live series, have demonstrated very natural motion with motion capture data and have provided a lot of enjoyment to fans, they still contain artifacts. For example, the ball may move in a physically implausible trajectory or appear to be stuck to the player's hand. Designing physics-based controllers for basketball skills has the potential to create high-quality, physically realistic basketball animation. However, designing such controllers is challenging because of the details of the physical interactions with the ball and the ground, the agility and grace required to perform the required tasks, and the coupling between the locomotion control and the manipulation of the fast-moving basketball.

In this paper, we describe a method for learning robust basketball dribbling controllers for simulated players from motion capture data. Our method treats a basketball controller as a combination of locomotion control and arm control components. The system learns the locomotion control first then trains the arm control to achieve robust control of the ball in a variety of dribbling tasks. We develop an efficient learning pipeline based on trajectory optimization and deep reinforcement learning and learn non-linear arm control policies. The control of agile locomotion in these basketball skills is learned using the sampling-based method proposed by Liu and his colleagues [2016]. To enable the player to transition between skills, we develop a learning procedure that allows a basketball skill and the transitions between the skill and other skills to be trained simultaneously. Our system is capable of learning a variety of dribbling skills, such as dribbling between the legs and crossover moves, as well as the transitions between those behaviors, while responding to user interaction.

Our work makes two principal contributions: (1) We present the first realtime, physics-based control of complex basketball skills. Our clips-to-controllers pipeline does not require that the input motion capture data contain the ball movement. Instead, it automatically recovers this information using trajectory optimization and learns robust basketball controllers that produce physically plausible ball movement and coordinated arm motion. (2) We demonstrate a successful adaptation of deep reinforcement learning to allow efficient learning of non-linear arm control policies that enable robust control of the ball during highly dynamic basketball skills. Our learning pipeline supports both the learning of individual skills and the learning of transitions between these skills. We find that initializing the learning using linear policies is key for efficient learning. We believe our design of the network structure and learning process will provide useful insights into the application of deep reinforcement learning to other tasks with dynamically manipulated objects.

2 RELATED WORK

Locomotion has been a central topic of physics-based character animation for several decades. Numerous successful control systems have been developed for various walking styles [Coros et al. 2010; Lee et al. 2010a; Yin et al. 2007], running [Ding et al. 2015; Mordatch et al. 2010], balancing [Macchietto et al. 2009], gymnastics [Hodgins et al. 1995; Kwon and Hodgins 2017], and acrobatic stunts [Al Borno et al. 2013; Ha et al. 2012; Zordan et al. 2014].

Recently, Liu and his colleagues [2016] proposed a sampling-based method for learning motion controllers from motion capture data. This method had been demonstrated to be applicable to a variety of locomotion skills, but we find that it cannot be used to create successful control of basketball skills. To successfully dribble a ball, basketball players must accurately control the ball before it leaves their hands and anticipate when and where the ball will next be contacted. A purely sampling-based method cannot easily accomplish such a task without an efficient lookahead policy.

Compared to the volume of research on locomotion control systems, research on controlling simulated humanoid characters that actively interact with moving objects is relatively sparse. Jain and Liu [2009] developed a physics-based system that animates ball movement corresponding to kinematic motion sequences. Tan et al. [2014] optimized neural networks to achieve robust control of bicycle stunts. Mordatch et al. [2012b] solved for complex interactions between characters and objects using contact-invariant optimization. Peng et al. [2017] demonstrated a soccer dribbling controller learned with the deep reinforcement learning. Liu and Hodgins [2017] demonstrated that by scheduling tracking control at runtime, a simulated character can perform various skateboarding skills, balancing on a bongo board, and even walking on a ball. In this paper, we realize robust control of agile basketball skills, where the fast movement of the ball, the small amount of time for controlling the ball, and the coupling between the ball's movement and the character's locomotion make it a non-trivial control problem.

Synthesizing grasping and manipulation has a long history in computer animation, robotics, and biomechanics [Wheatland et al. 2015]. Physics-based methods are often employed to generate detailed, physically realistic hand motion in these tasks. For example, realistic grasping can be achieved by tracking a few example poses using PD servos [Pollard and Zordan 2005], the interaction between fingers and objects can be transferred to other tasks by capturing contact forces [Kry and Pai 2006], and detailed hand manipulations can be optimized by carefully modeling contact constraints [Andrews and Kry 2013; Liu 2009; Mordatch et al. 2012a; Ye and Liu 2012; Zhao et al. 2013]. Dribbling or juggling a ball using a robot arm is also studied in robotics [Bätz et al. 2009, 2010; Haddadin et al. 2011; Reist and D'Andrea 2012], where the control policy is designed by carefully analyzing and modeling the ball's motion in every phase. Despite their success, these research works usually focus on controlling fingers, hands, and optionally arms, while the other parts of the character's body are neglected or controlled using baked animation sequences [Bai et al. 2012]. In comparison with these existing works, we focus on controlling fast moving objects like a basketball by coordinating a simulated character's arms, hands, and fingers, where the motions of the character's arms and the other part of its body are fully coupled.

Trajectory optimization is often employed in physics-based character animation to generate motions for both human [Al Borno et al. 2013; Mordatch et al. 2012b] and imaginary creatures [Wampler and Popović 2009] with or without a reference motion. It can compute a physically valid motion trajectory and the corresponding control signal trajectory. Feedback policies can be learned to realize robust tracking control systems based on these open-loop trajectories [Liu et al. 2016; Muico et al. 2009]. Some research also reveals that the solutions obtained by trajectory optimization can dramatically facilitate the learning of control policies in complex, high-dimensional tasks [Levine and Koltun 2014; Mordatch et al. 2015]. Inspired by these works, our control system utilizes trajectory optimization to learn a linear control policy for each basketball task. This control policy is then used to initialize the learning of a more complicated non-linear policy.

Reinforcement learning offers a convenient framework for learning good control strategies. In the field of character animation, a number of previous works have demonstrated successful adaptations of reinforcement learning approaches in both interactive motion synthesis [Lee et al. 2010b; McCann and Pollard 2007; Treuille et al. 2007] and physics-based control [Coros et al. 2009; Liu and Hodgins 2017; Peng et al. 2017]. The reinforcement learning problem for controlling a physically simulated character has high-dimensional, continuous state space and action space, which makes it hard to tackle [van Hasselt 2012]. A possible solution is to discretize the

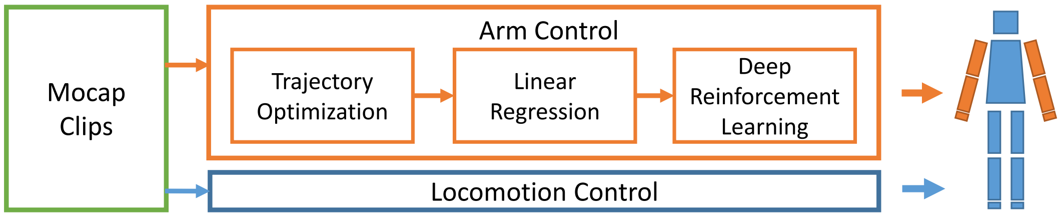


Fig. 2. System Overview

action space by learning a number of motion primitives [Coros et al. 2009; Liu and Hodgins 2017] so that value-based methods such as deep Q-learning [Mnih et al. 2015b] can be applied. In recent years, significant advances have been made by directly solving such high-dimensional continuous control problems, using methods such as guided policy search [Levine and Koltun 2013; Mordatch et al. 2015] and actor-critic approaches [Lillicrap et al. 2015; Peng et al. 2017; Schulman et al. 2015a,b]. Our work adapts the deep deterministic policy gradient (DDPG) method [Lillicrap et al. 2015] to learn arm control in basketball skills. In addition to training control policies for individual skills like many of the existing works, we developed a learning pipeline that trains the transitions between the skills as well, enabling the character to perform multiple skills and respond to user interaction.

3 SYSTEM OVERVIEW

Our system takes motion capture clips as input and creates robust controllers that allow a simulated basketball player to perform various basketball skills. As sketched in Figure 2, a controller in our system consists of two coupled components, each controlling a disjoint set of joints: the arm control component coordinates the player's arms, hands, and fingers for stable manipulation of the ball; while the locomotion control component handles the rest of the joints, keeping the player moving and maintaining balance.

We learn the locomotion control component and the arm control component separately. The locomotion control component is learned first using the method developed by [Liu et al. 2016]. In this stage, the movement of the ball is not taken into account, and the simulated player tries to mimic the arm motion in the reference motion capture data. This simplification is reasonable because a basketball is light compared to a human player and the adjustments to the arm for variations in the ball trajectory are small. In practice, the learned locomotion control component is robust enough to deal with the unmodeled perturbations caused by the ball and the change in player's arm motion.

The learning of the arm control component begins with the reconstruction of the ball's movement. Our system does not require that the motion of the ball be captured in the input motion clips. Instead, it solves an optimization problem to compute open-loop arm control for each control fragment that produces a simulated ball trajectory which best matches the motion clip.

Using the optimized open-loop arm control, we can learn a stepwise linear feedback policy for each skill using the regression method demonstrated in [Liu et al. 2016]. In practice, this initial arm control

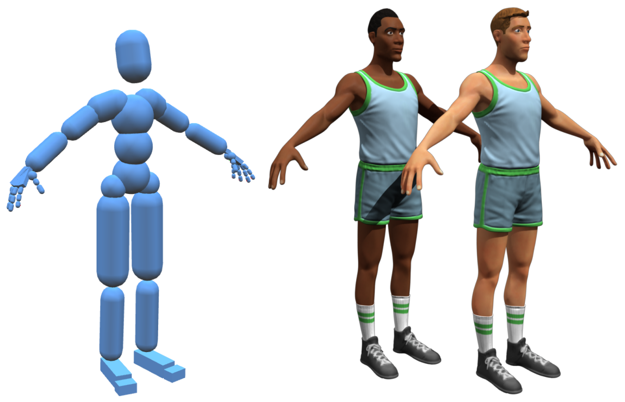


Fig. 3. Character model. Left: skeleton and collision geometries. Right: rigged characters.

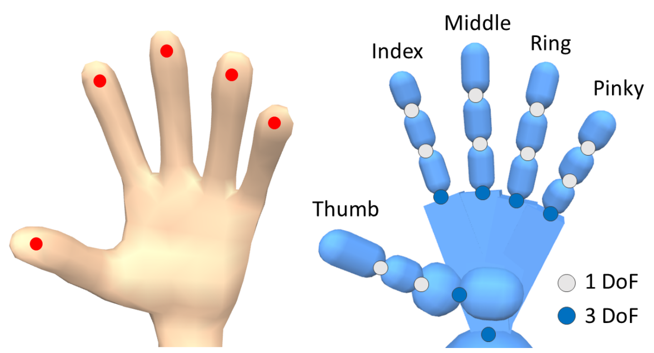
policy enables successful control of simple skills such as carrying a ball with both hands, but is not capable of controlling more complex skills such as dribbling. Because the player cannot apply control to the ball when it is bouncing or in flight, a small error in the ball's state amplifies quickly and drives the simulation out of the basin of attraction of a linear policy.

Our system thus learns a non-linear arm control policy modeled with artificial neural networks for each basketball skill. We use the Deep Deterministic Policy Gradient (DDPG) method [Lillicrap et al. 2015] to learn this control policy. In the learning process, the arm control policy is repeatedly applied and updated according to the simulation results until a robust policy is reached. This process is initialized with the examples generated from the linear feedback policies. This initialization is crucial for efficient learning.

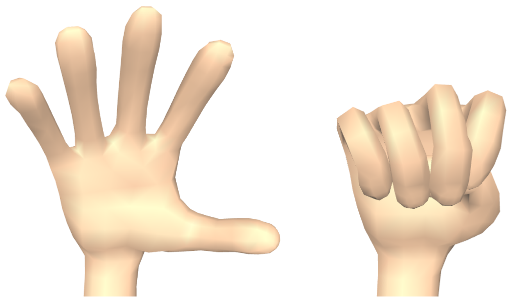
In addition to learning individual basketball skills, our system follows the same learning process to learn control graphs that consist of multiple basketball skills and transitions between the skills. To ensure successful transitions, a control graph is learned in an incremental manner, where a new skill is added to the graph by training a combined motion that consists of the new skill, the learned skills that the new skill connects to, and the transitions between the learned and new skills. In this process, the learned control policies are reused in this learning process to facilitate the training.

3.1 Simulation Framework

We model our basketball player as an articulated skeletal system (Figure 3), which consists of a floating root joint and internal joints



(a) Hand model used in this paper. We use 1-DoF pin joint for proximal interphalangeal joints and distal interphalangeal joints, and 3-DoF ball and socket joint for metacarpophalangeal joints. The red dots on the fingertips are proxies used to calculate the distance between the ball and the hand.



(b) The flat hand pose (left) and the fist pose (right) used for computing the target pose for hands.

Fig. 4. Hand model

actuated with PD-servos. At every time step, the locomotion and arm control components compute a target pose containing target angles for each internal joint. The joint torques are then computed by tracking this target pose using the PD-servos.

The player's hands are modeled with fingers, each having five degrees of freedom as shown in Figure 4a, and each being controlled individually. Finger motions are known to contain redundant degrees of freedom [Wheatland et al. 2015]. To make the control problem simpler, our system computes the target poses for each hand as the interpolation between a flat hand pose and a fist pose as shown in Figure 4b, i.e.

$$\hat{\theta}_i = (1-\lambda_i)\theta_i^{\text{flat}} + \lambda_i\theta_i^{\text{fist}} \tag{1}$$

where $i \in \{$ *Thumb*, *Index*, *Middle*, *Ring*, *Pinky* $\}$ is the index of a finger, $\hat{\theta}_i$, θ_i^{flat}, θ_i^{fist} represent the pose of finger i in the target pose, the flat hand pose, and the fist pose, respectively. λ_i is the interpolation factor for finger i. To achieve coordinated finger motions, our system controls each hand using three control signals $\boldsymbol{\alpha} = \{\lambda_{\text{Thumb}}, \lambda_{\text{Index}}, \lambda_{\text{Pinky}}\}$. The interpolation factors for the other

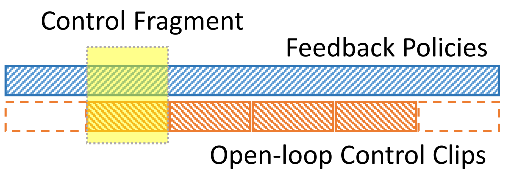


Fig. 5. Structure of Controllers

two fingers are computed as:

$$\lambda_{\text{Middle}} = \frac{2}{3}\lambda_{\text{Index}} + \frac{1}{3}\lambda_{\text{Pinky}} \tag{2}$$

$$\lambda_{\text{Ring}} = \frac{1}{3}\lambda_{\text{Index}} + \frac{2}{3}\lambda_{\text{Pinky}} \tag{3}$$

3.2 Control Fragments

Both the locomotion control component and the arm control component of our basketball controllers share the structure shown in Figure 5. Specifically, each control component consists of a feedback policy and an open-loop control trajectory that has been segmented into short clips. A *control fragment*, represented with a tuple $C_k = (\boldsymbol{m}_k, \boldsymbol{\pi})$, is then defined as the combination of an open-loop control clip, $\boldsymbol{m}_k$, and an associated control policy, $\boldsymbol{\pi}$. A controller is thus a series of control fragments, $\{C_k\}$.

A control policy $\boldsymbol{\pi} : X \longrightarrow A$ is a mapping between a state $\boldsymbol{x} \in X$ and an action $\boldsymbol{a} \in A$, where X and A represent the state space and the action space respectively. In our system, the state vector $\boldsymbol{x} = (\boldsymbol{s}, k)$ consists of the current state of the simulation, $\boldsymbol{s}$, and the index of the control fragment, k. The action $\boldsymbol{a}$ is a corrective offset that will be added to the open-loop control clip $\boldsymbol{m}_k$ to compensate for unexpected perturbations in the simulation.

At runtime, our system evaluates the feedback policies at the beginning of each control fragment and computes compensative offsets for both the locomotion and arm control according to the state of the simulation. By tracking the modified control clip, the simulation can be stabilized within the vicinity of the reference motion.

3.3 Review of Sampling-based Motion Control Method

We adapt the sampling-based motion control method proposed by Liu et al. [2016] for learning the locomotion control components of our basketball controllers. In this section, we briefly review a few concepts of this method that relate to our work. We refer interested readers to the original paper for more details.

When given a reference motion as input, the method of [Liu et al. 2016] first segments the input into short control fragments and learns a stepwise linear control policy for these fragments. This feedback policy is defined as

$$\boldsymbol{a} = \boldsymbol{\pi}(\boldsymbol{x}) := \boldsymbol{\pi}(\boldsymbol{s}, k) = M_k\boldsymbol{s} + \hat{a}_k \tag{4}$$

where M_k and $\hat{a}_k$ are the gain matrix and the affine term of the policy associated with the control fragment C_k respectively.

To learn M_k and $\hat{a}_k$, Liu et al. [2016] construct a long open-loop control trajectory which can be tracked using PD-servos to reproduce the target motion a number of times. In this process, every fragment of the target motion has been performed multiple times, thus a number of example state-action pairs can be extracted from the construction for each control fragment. Then the policy parameters $(M_k, \hat{a}_k)$ are learned with linear regression using the example state-action pairs corresponding to the control fragment C_k.

Liu et al. [2016] developed a sampling-based method, called Guided SAMCON, to effectively construct the long open-loop control trajectory. Guided SAMCON can be seen as a special Sequential Monte-Carlo method. It constructs open-loop control for every control fragment in sequence by generating sample actions, running simulation while applying the actions, and recording the samples which produce the motions that are the closest to the reference. These samples are generated from the current feedback policy combined with a Gaussian exploration noise. Once the long open-loop control trajectory is constructed, the feedback policy is updated using the recorded samples. This process continues until a robust policy is found.

4 LEARNING OF LOCOMOTION CONTROL

Our system follows the same learning process as detailed in [Liu et al. 2016] to learn the locomotion control component for a basketball skill. Because basketball players can only control the ball during the short time the ball is in contact with their hands, we use control fragments of 0.05 s in length in our control system, instead of the 0.1 s fragments suggested by [Liu et al. 2016]. By shortening the length of the control fragments, we allow tighter control of the ball's motion as a new control fragment is selected more frequently.

As discussed above, the ball's motion is not included in the learning of the locomotion component. This simplification allows us to treat each hand as a single rigid body during the learning of the locomotion control component, which reduces the total number of degrees of freedom (DoF) involved in the simulation and significantly speeds up the learning process.

Once learned, the locomotion control component is fixed during the learning of the arm control component. For all of the skills learned in this paper, the linear policy is robust enough to allow the locomotion controller to deal with the disturbance caused by the movement of the ball and the player's arms.

5 LEARNING OF ARM CONTROL

The method developed by [Liu et al. 2016] relies on the Guided SAMCON algorithm to generate example open-loop control for training the linear policies. Although this method is applicable to a variety of locomotion skills, we find that it cannot generate open-loop arm control that allows a player to successfully dribble a ball. Dribbling tasks require that the basketball players should accurately control the ball before it leaves their hands and anticipate when and where the ball will next be contacted. A purely sampling-based method cannot accomplish such tasks without an efficient lookahead policy. To solve this problem, our system uses trajectory optimization to compute successful open-loop arm control for dribbling tasks.

When our system constructs the open-loop locomotion control clip for each control fragment C_k, the open-loop control targets for shoulders, elbows, and wrists are also computed from the input motion clip. The open-loop arm control clip $\boldsymbol{m}_k$ is thus initialized with these control targets. The fingers are ignored during the learning of the locomotion control components, so the target angles for the finger joints are initialized to zero in $\boldsymbol{m}_k$. Note that these open-loop control clips, $\{\boldsymbol{m}_k\}$, are not accurate enough to produce successful dribbling because the ball's movement is not taken into account. The learning of the arm control component thus starts by improving the physical accuracy of these open-loop arm control clips while reconstructing the ball's movement.

5.1 Trajectory Optimization

We use trajectory optimization to find a set of corrective offsets to apply to the initial open-loop arm control clips so that the player can successfully dribble the ball. Specifically, given a reference motion clip as input, we first specify frames in which the ball should be touching the player's hands. This information will be used as soft constraints in the optimization and does not need to be precise. We represent this contact information as $\mathcal{H} = \{\mathrm{H}_t\}$, where $\mathrm{H}_t \in \{\emptyset, \{\mathrm{L}\}, \{\mathrm{R}\}, \{\mathrm{L}, \mathrm{R}\}\}$ is the set of hands that should control the ball in frame t. The objective of the optimization is to adjust the arm control to minimize the distance between the ball and the player's hands in these frames. In addition, a few frames are designated as *checkpoints* for the skill and will be treated as hard constraints where the contact between the ball and the player's fingers will be enforced by the optimizer. We choose the frames in which the ball reaches the highest point in a dribbling cycle as the checkpoints. Additional checkpoints can be included for more complicated skills. For example, for a spin crossover move, we set two checkpoints at the beginning and the end of the spin so that the system ensures the completion of the move during the optimization.

The optimization variables are the set of corrective offsets $\{\chi_k^{\mathrm{L}}, \chi_k^{\mathrm{H}}\}$ which will be used to correct the open-loop arm control clips $\{\boldsymbol{m}_k\}$. Here χ_k^{L} and χ_k^{H} represent the corrections being applied to the left arm and the right arm in control clip $\boldsymbol{m}_k$ respectively. The value of χ is a 10-DoF vector

$$\chi = (\boldsymbol{q}_{\mathrm{shoulder}}, q_{\mathrm{elbow}}, \boldsymbol{q}_{\mathrm{wrist}}, \boldsymbol{\alpha}_{\mathrm{fingers}}) \tag{5}$$

where $\boldsymbol{q}_{\mathrm{shoulder}}$ and $\boldsymbol{q}_{\mathrm{wrist}}$ are both three-dimensional rotation vectors representing the offset rotations for the shoulder joint and the wrist joint respectively, q_{elbow} is the offset angle for the elbow joint, and the 3-DoF vector $\boldsymbol{\alpha}_{\mathrm{fingers}}$ contains the finger control parameters defined in Section 3.1.

A basketball player often controls the ball using one hand at a time, which allows our system to only optimize the corrective offsets for a single arm. We represent the set of such corrective offsets with $\mathcal{X} = \{\chi_k^{\mathrm{h}}\}$, in which each χ_k^{h} indicates that the hand $\mathrm{h} \in \{\mathrm{L}, \mathrm{R}\}$ is actively controlling the ball in control fragment C_k. $\mathcal{X}$ can be derived from the contact information $\mathcal{H}$. For example, when the player dribbles the ball using one hand, $\mathcal{X}$ contains only the corrective offsets for the corresponding arm; when the player controls the ball with both hands alternately, $\mathcal{X}$ contains the corrective offsets for the first arm before the ball contacts with the ground, and the corrective offsets for the second hand thereafter. When a hand is inactive in a control fragment, a default corrective offset $\tilde{\chi} = \{\mathbf{0}, 0, \mathbf{0}, \tilde{\boldsymbol{\alpha}}\}$ is used to keep it in the pose of the original motion capture clip, where $\tilde{\boldsymbol{\alpha}}$ is empirically set to $(0.2, 0.2, 0.2)$.

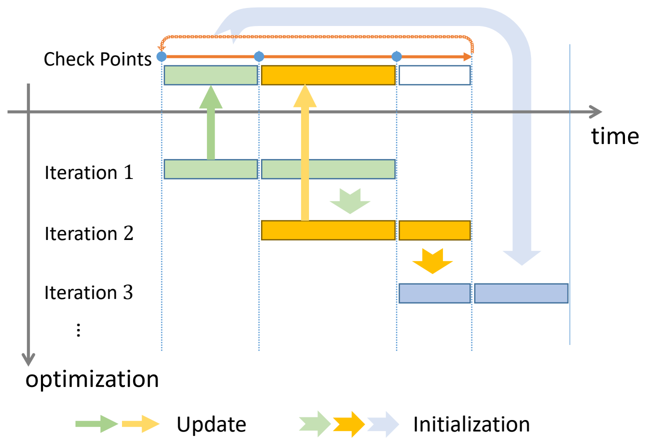


Fig. 6. Optimization schedule for a cyclic basketball skill consists of three optimization windows defined by three checkpoints. In Iteration 1, the first and second windows are initialized to zero and optimized together. The optimized corrective offsets for the first window are recorded by the system, while those for the second windows are kept as the initial solution for the next iteration. In Iteration 2, the second and third windows are optimized together, with the third window initialized to zero. In Iteration 3, the first window is optimized again. Our system initializes it according to the recorded corrective offsets.

For a long, complex basketball skill consisting of many control fragments, the optimization variable $\mathcal{X}$ can contain a large number of parameters, making the optimization prohibitively expensive and prone to falling into local minima. To mitigate this problem, our system uses a sliding window scheme like many previous works [Al Borno et al. 2013; Liu et al. 2006]. Specifically, our system divides a basketball skill into optimization windows at the checkpoints and then optimizes two consecutive windows each time. Figure 6 illustrates this process: The first two windows are optimized together with all the corrective offsets initialized to zero. When the optimization finishes, the first window is removed from the optimization problem, and the second and third windows are optimized together. The optimized corrective offsets for the second window are kept as the initial solution for this new optimization problem, while the corrective offsets for the third window are initialized to zero. This process is repeated until the whole motion is optimized.

The objective of the optimization problem is formulated as:

$$\min_{\mathcal{X}} \quad \frac{1}{\sum |\mathrm{H}_t|} \sum_{\mathrm{H}_t \in \mathcal{H}} \sum_{\mathrm{h} \in \mathrm{H}_t} D^{\mathrm{h}}_t + \frac{w_\chi}{|\mathcal{X}|} \sum_{\chi \in \mathcal{X}} ||\chi|| + w_{\tilde{E}} \tilde{E} \tag{6}$$

where the first term computes the average distance between the player's fingers and the ball in the frames specified in $\mathcal{H}$, the second term penalizes excessive corrections with the weight $w_\chi = 0.1$, and the last term is a helper cost that guides the optimization towards the feasible region.

Our system employs five proxy points placed on the fingertips of each hand to measure the distance between the ball and the player's hands. Figure 4a shows the position of these proxy points with red dots. The distance between a ball and a hand is then computed as:

$$D^{\mathrm{h}} = \frac{1}{|\mathcal{P}^{\mathrm{h}}|} \sum_{p \in \mathcal{P}^{\mathrm{h}}} ||\boldsymbol{d}_p|| - r_{\mathrm{ball}} \tag{7}$$

where $\mathcal{P}^{\mathrm{h}}$ represents the set of proxy points on hand $h \in \{\mathrm{L}, \mathrm{R}\}$, $\boldsymbol{d}_p = \boldsymbol{p}_p - \boldsymbol{p}_{\mathrm{ball}}$ is the distance vector between proxy point $p \in \mathcal{P}$ and the center of the ball, and r_{ball} is the radius of the ball.

Even with the sliding window scheme, the optimization problem is still difficult to solve because the optimization process can easily stall at a poor local minimum where the player cannot catch the ball again after dribbling it. The helper cost in Equation 6 is crucial to the success of the optimization. It serves as a large penalty when the simulated ball motion does not satisfy some requirements and becomes zero when the optimization converges. We formulate this helper cost as:

$$\tilde{E} = 2N_{\mathrm{check}} + \max(h_{\mathrm{ref}} - h_{\mathrm{ball}}, 0) + \frac{T_0 - T_{\mathrm{cont}}}{T_0} \tag{8}$$

where the first term $N_{\mathrm{check}} \in \{0, 1, 2\}$ counts the number of failed checkpoints at which the distance between the ball and the corresponding hands (Equation 7) is greater than 5 cm. The second term of Equation 8. applies a penalty if the ball fails to reach a reference height h_{ref} at the last checkpoint, where h_{ref} is extracted from the reference motion as the height of the corresponding hand at the checkpoint, less the radius of the ball along the normal direction of the palm. The last term of Equation 8 prevents unwanted contacts between the ball and the player's body, where T_0 is the total length of the two windows being optimized and T_{cont} is either the time when the first bad contact occurs or T_0 if no such contact occurs. We use $w_{\tilde{E}} = 10$ as the weight of this helper cost term in Equation 6.

Our system employs the Covariance Matrix Adaption Evolution Strategy (CMA-ES) [Hansen 2006] to solve the optimization problem. CMA-ES is a sampling-based method that iteratively evaluates a number of random samples and updates the sample distribution according to their costs. For each CMA-ES sample $\mathcal{X}$, our system applies the corrective offsets to the corresponding control fragments, simulates the two optimization windows from a starting state, and evaluates the objective function of Equation 6. For efficiency, the simulation stops early if the distance between the ball and the corresponding hand is greater than 5 cm at a checkpoint. The starting state is initially obtained from the input motion clip, where the position of the ball is manually set to make the palm and all the fingers touch it. When the optimization finishes, this starting state is updated to the end state of the first optimization window.

5.2 Learning of the Linear Control Policy

Similar to the locomotion control policy, the arm control policy, $\boldsymbol{\pi}$, takes a state vector $\boldsymbol{x}$ as input and computes an action vector $\boldsymbol{a} = (\chi^{\mathrm{L}}, \chi^{\mathrm{R}})$ that contains the correction offsets for both of the character's arms. The state vector $\boldsymbol{x} = (\boldsymbol{s}, k)$ consists of the current simulation state, $\boldsymbol{s}$, and the index of the control fragment, k, where $\boldsymbol{s}$ captures the states of the player and the ball as an 165-DoF vector:

$$\boldsymbol{s} = \{\boldsymbol{p}_{\mathrm{ball}}, \dot{\boldsymbol{p}}_{\mathrm{ball}}, \boldsymbol{p}_j, \dot{\boldsymbol{p}}_j, \boldsymbol{d}_i, \boldsymbol{c}, \dot{\boldsymbol{c}}, \boldsymbol{L}\} \tag{9}$$

which consists of the positions and velocities of the basketball $(\boldsymbol{p}_{\mathrm{ball}}, \dot{\boldsymbol{p}}_{\mathrm{ball}})$ and every body part of the player except the fingers

$(\boldsymbol{p}_j, \dot{\boldsymbol{p}}_j)$, the distances between the fingertips and the center of the ball ($\boldsymbol{d}_i$), the position and velocity of the player's center of mass ($\boldsymbol{c}, \dot{\boldsymbol{c}}$), and the angular momentum of the player ($\boldsymbol{L}$). All these quantities are computed in a body local reference coordinate frame that moves horizontally with the player's root and with one axis pointing in the facing direction of the player.

Inspired by [Liu et al. 2016], we use linear regression to learn a stepwise linear control policy, $\boldsymbol{\pi}(\boldsymbol{s}, k) = M_k \boldsymbol{s} + \hat{\boldsymbol{a}}_k$, as the initial arm control policy. The learned linear policy enables robust control of simple skills such as carrying the ball while swinging the arms, but cannot control more complicated dribbling skills. We will use this linear policy to initialize the deep reinforcement learning process in the next section.

The input to the regression is an optimized open-loop arm control trajectory that produces a long motion sequence in which the target skill is repeated N_{rep} times. To facilitate the optimization of this long control trajectory, our system maintains a moving average $\tilde{\boldsymbol{a}}_k$ of the optimized actions $\{\boldsymbol{a}_{k,i}\}$ for each control fragment C_k and initializes new optimizations according to it. As illustrated in Figure 6, when a control fragment C_k has been optimized i times and is being removed from the optimization, our system updates the corresponding $\tilde{\boldsymbol{a}}_k$ using

$$\tilde{\boldsymbol{a}}_k \leftarrow (1-\beta)\tilde{\boldsymbol{a}}_k + \beta \boldsymbol{a}_{k,i} \tag{10}$$

where the decay factor $\beta = 1/i$ when $i <= 10$ and is fixed to 0.1 when $i > 10$. The next time this control fragment is optimized, the corresponding actions will be initialized with $0.9\tilde{\boldsymbol{a}}_k$, where the shrinkage factor 0.9 is employed to actively encourage the optimization process to use small corrective offsets.

In addition to learning the linear control policies, our system also computes the average ball positions for every control fragment from the optimized motion sequence and updates the contact information set $\mathcal{H}$ to include only the frames in which the average distance between the player's fingers and the ball (Equation 7) is not greater than 5 cm. This information will be referenced by the deep reinforcement learning algorithm described in the next section.

ALGORITHM 1: Learn Arm Control Policy Using DDPG

```
Input: control fragments $\{C_k\}$, $k = 1, \dots, K$ and
       associated linear control polices $\{(M_k, \hat{\boldsymbol{a}}_k)\}$
Input: starting states $\{s_{k_i}\}$, $k_i \in \{1, \dots, K\}$
Result: arm control policy $\pi$
1  initialize $\mathcal{D} \leftarrow \emptyset$
2  initialize critic network parameters $\theta_Q$ and actor network parameters
   $\theta_\pi$
3  initialize target function $\theta'_Q \leftarrow \theta_Q$, $\theta'_\pi \leftarrow \theta_\pi$
4  initialize simulation with random starting state $s_{k_i}$, set $\boldsymbol{x} \leftarrow (s_{k_i}, k_i)$
5  for $t \leftarrow 1, 2, \dots$ do
6      $k \leftarrow k(\boldsymbol{x})$ ;                    // get the '$k$' component of $\boldsymbol{x}$
7      if $t \le N_{init}$ then
8          $\boldsymbol{s} \leftarrow \boldsymbol{s}(\boldsymbol{x})$ ;                // get the '$s$' component of $\boldsymbol{x}$
9          compute action $\boldsymbol{a} = M_k \boldsymbol{s} + \hat{\boldsymbol{a}}_k + \varepsilon_t$ and $r = r(\boldsymbol{x}, \boldsymbol{a})$
10     else
11         compute action $\boldsymbol{a} = \pi(\boldsymbol{x}; \theta_\pi) + \varepsilon_t$ and $r = r(\boldsymbol{x}, \boldsymbol{a})$
12     end
13     execute control fragment $C_k$ and observe $\boldsymbol{x}' \leftarrow (\boldsymbol{s}', \text{next}(k))$
14     store transition $\boldsymbol{\tau} = (\boldsymbol{x}, \boldsymbol{a}, r, \boldsymbol{x}')$ in $\mathcal{D}$
15     if $t > N_{init}$ then
16         sample a minibatch of $N_{\text{batch}}$ transtions $\{\boldsymbol{\tau}_i\} \subset \mathcal{D}$
17         compute target values $y_i = y(\boldsymbol{\tau}_i; \theta'_Q, \theta'_\pi)$ using Equation 13
18         update $\theta_Q$ by minimizing the loss function of Equation 12
19         update $\theta_\pi$ using the policy gradient of Equation 16
20         $\theta'_Q \leftarrow (1-\eta)\theta'_Q + \eta\theta_Q$
21         $\theta'_\pi \leftarrow (1-\eta)\theta'_\pi + \eta\theta_\pi$
22     end
23     if $\boldsymbol{x}' \in X_{term}$ then
24         reset simulation to a random stating state $s_{k_i}$
25         $\boldsymbol{x} \leftarrow (s, k_i)$
26     else
27         $\boldsymbol{x} \leftarrow \boldsymbol{x}'$
28     end
29 end
```

5.3 Deep Reinforcement Learning

We model our control problem as a Markov Decision Process (MDP) and train a non-linear arm control policy using deep reinforcement learning to achieve robust control of basketball skills. Specifically, our control system receives an observation $\boldsymbol{x}_t \in X$ of the simulation at the beginning of a control fragment C_{k_t}, takes an action $\boldsymbol{a}_t \in A$, and receives a scalar reward $r_t = r(\boldsymbol{x}_t, \boldsymbol{a}_t)$. The simulation then transitions to a new state $\boldsymbol{x}_{t+1}$ when the control fragment finishes, resulting in a transition tuple $\boldsymbol{\tau}_t = (\boldsymbol{x}_t, \boldsymbol{a}_t, r_t, \boldsymbol{x}_{t+1})$. Repeatedly applying the policy π produces a sequence $(\boldsymbol{x}_0, \boldsymbol{a}_0, r_0, \boldsymbol{x}_1, \boldsymbol{a}_1, r_1, \dots)$. The return from a state $\boldsymbol{x}_t$ is then defined as the sum of the rewards $R_t = \sum_{k=t}^{\infty} \gamma^{(k-t)} r_k$ along this sequence with a discounting factor $\gamma = 0.99$. The goal of the reinforcement learning is to find the optimal policy that maximizes the expected returns from any starting state. This goal can be denoted with the objective $J(\pi)$.

The deep deterministic policy gradient (DDPG) algorithm [Lillicrap et al. 2015] is an actor-critic method based on the deterministic policy gradient (DPG) [Silver et al. 2014]. This method trains a policy $\pi(\boldsymbol{x}; \theta_\pi)$, also known as the *actor*, and an action-value function $Q(\boldsymbol{x}, \boldsymbol{a}; \theta_Q)$, or rather, the *critic*, both parameterized as artificial neural networks with the parameters θ_π and θ_Q respectively. These networks are updated iteratively based on past simulation results.

The action-value function $Q(\boldsymbol{x}, \boldsymbol{a})$ computes the expected return of taking an action $\boldsymbol{a}$ at a state $\boldsymbol{x}$. With the deterministic policy π, this function can be written as the recursive Bellman equation:

$$Q(\boldsymbol{x}_t, \boldsymbol{a}_t) = r_t + \gamma\, \mathbb{E}_{\boldsymbol{x}_{t+1}|\boldsymbol{x}_t, \boldsymbol{a}_t} \left[Q\left(\boldsymbol{x}_{t+1}, \pi(\boldsymbol{x}_{t+1})\right)\right] \tag{11}$$

The DDPG algorithm learns this critic function using Q-learning. It optimizes the network parameters θ_Q by minimizing the loss

$$L(\theta_Q) = \mathbb{E}\left[\|y(\boldsymbol{\tau}; \theta'_Q, \theta'_\pi) - Q(\boldsymbol{x}_t, \boldsymbol{a}_t; \theta_Q)\|^2 \right] + w_{\theta_Q} \|\theta_Q\|^2 \tag{12}$$

against a target function

$$y(\boldsymbol{\tau}; \theta'_Q, \theta'_\pi) = \begin{cases} r_t + \gamma Q\left(\boldsymbol{x}_{t+1}, \pi(\boldsymbol{x}_{t+1}; \theta'_\pi); \theta'_Q\right) & \boldsymbol{x}_{t+1} \notin X_{\text{T}} \\ r_t & \boldsymbol{x}_{t+1} \in X_{\text{T}} \end{cases} \tag{13}$$

where $X_{\text{T}} \subset X$ is the set of terminal states in which the player fails to control the ball. The regularization weight is $w_{\theta_Q} = 0.001$ in all our tests. The parameters θ_Q are then updated using the batch stochastic gradient descent method:

$$\theta_Q \leftarrow \theta_Q - \alpha_Q \nabla_{\theta_Q} L(\theta_Q) \tag{14}$$

where $\nabla_{\theta_Q} L(\theta_Q)$ is the derivative of the loss function with respect to the parameters θ_Q, and α_Q is the learning rate for the critic.

The actor is updated similarly using the *policy gradient* $\nabla_{\theta_\pi} J(\theta_\pi)$:

$$\theta_\pi \leftarrow \theta_\pi - \alpha_\pi \nabla_{\theta_\pi} J(\pi) \tag{15}$$

where α_π is the learning rate for the actor. Silver and colleagues [2014] proved that $\nabla_{\theta_\pi} J(\theta_\pi)$ can be computed using the chain rule:

$$\begin{aligned}\nabla_{\theta_\pi} J(\theta_\pi) &= \mathbb{E}\left[\nabla_{\theta_\pi} Q\left(\boldsymbol{x}, \pi(\boldsymbol{x};\theta_\pi)\right)\right] \\ &= \mathbb{E}\left[\nabla_{\theta_\pi} \pi(\boldsymbol{x};\theta_\pi) \cdot \nabla_{\boldsymbol{a}} Q(\boldsymbol{x}, \boldsymbol{a})|_{\boldsymbol{a}=\pi(\boldsymbol{x};\theta_\pi)}\right]\end{aligned} \tag{16}$$

In the learning process, the expectations in Equation 12 and Equation 16 are computed over a minibatch that consists of $N_{\text{batch}} = 32$ samples randomly selected from a replay buffer $\mathcal{D} = \{\boldsymbol{\tau}_i\}$, which stores up to $N_{\mathcal{D}} = 10^6$ most recent transitions tuples. In addition, instead of directly applying the update rule of Equation 14 and Equation 15, our system employs the Adam algorithm [Kingma and Ba 2014] to achieve stabler and more efficient learning.

Directly setting the parameters θ'_Q, θ'_π of the target function of Equation 13 to the updated critic parameters θ_Q and actor parameters θ_π at every learning step can make the learning process prone to divergence in many cases [Mnih et al. 2015a]. Following the suggestion of Lillicrap et al. [2015], our system fixes the problem of divergence by making the target function slowly track the update of θ_Q and θ_π by maintaining a moving average $\theta'_* \leftarrow (1-\eta)\theta'_* + \eta\theta_*$, where θ_* represents either θ_Q or θ_π and the decay factor $\eta = 0.001$.

Algorithm 1 outlines the major steps of the learning process. Starting from randomly initialized actor and critic networks, our system repeatedly executes the current arm control policy and simulates the control fragments in sequence. At the end of every control fragment, the transition is evaluated by the reward function and stored in the replay buffer. The critic θ_Q and the actor θ_π are then updated using $\nabla_{\theta_Q} L(\theta_Q)$ and $\nabla_{\theta_\pi} J(\theta_\pi)$ respectively. The replay buffer $\mathcal{D}$ is initialized with $N_{\text{init}} = 2 \times 10^5$ samples generated with the stepwise linear arm control policy learned above. In the learning process, random exploration noise $\varepsilon_t \sim \mathcal{N}(0, \sigma_\varepsilon^2)$ is added to each action, where $\sigma_\varepsilon = 0.05$ is the standard deviation of this noise.

Our system uses the reward function of Equation 17 to evaluate the transitions:

$$\begin{aligned}r(\boldsymbol{x}, \boldsymbol{a}) &\equiv r\left((\boldsymbol{s}, k), \boldsymbol{a}\right) \\ &= R_0 - w_{\text{a}}\|\boldsymbol{a}\| - w_{\text{D}} D(\boldsymbol{s}) \\ &\quad - w_{\text{ball}} \max\left(0, \|\boldsymbol{p}_{\text{ball}} - \tilde{\boldsymbol{p}}_{\text{ball},k}\| - r_{\text{ball}}\right)\end{aligned} \tag{17}$$

where $R_0 = 1$ is a constant default reward; D is the average distance between fingertips and the ball as defined in Equation 7, which is non-zero only in the frames specified in $\mathcal{H}$; $\tilde{\boldsymbol{p}}_{\text{ball},k}$ is the reference ball position corresponding to control fragment C_k, which was obtained during the trajectory optimization. All the weights of Equation 17 are empirically set to 1, except $w_D = 10$.

Our system utilizes two medium-sized neural networks to approximate the actor and critic functions. As depicted in Figure 7, the critic network has two fully connected hidden layers containing 400 and 600 leaky rectified linear units (ReLU) respectively. A leaky ReLU is defined as

$$f(z) = \begin{cases} z & z \geq 0 \\ 0.001z & z < 0 \end{cases} \tag{18}$$

where the scalar z represents the input signal to the unit. The action component $\boldsymbol{a}$ of the critic function $Q(\boldsymbol{x}, \boldsymbol{a})$ is not included into the network until the second hidden layer.

The actor network shown in Figure 8 has one fully connected hidden layer containing 500 leaky ReLUs. In practice, this network can produce excessive actions in the first few thousands of learning steps, which can lead to unstable learning because the gradient information corresponding to these actions is not reliable. The same issue has been observed in several recent works [Hausknecht and Stone 2015; Peng and van de Panne 2017]. They suggested enforcing reasonable action bounds to avoid this problem. Inspired by these works, we include a leaky bounded output layer in the actor network,

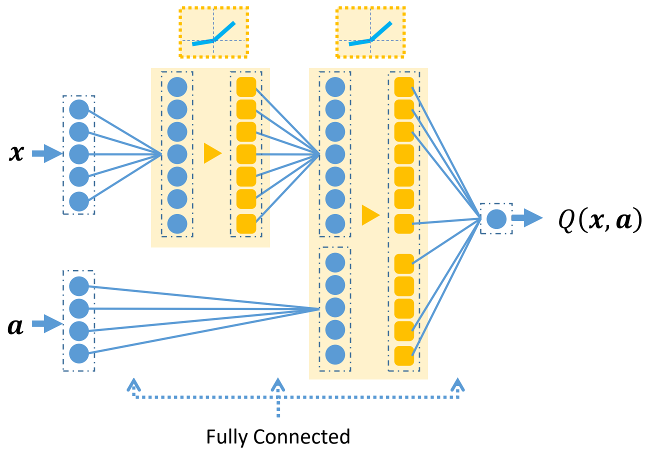


Fig. 7. Critic Network: The input layer consists of a state vector and an action vector, the output is a scalar representing the critic. The hidden layers consist of leaky rectified linear units (Equation 18). The action vector is not connected to the network until the second hidden layer. All the layers are fully connected.

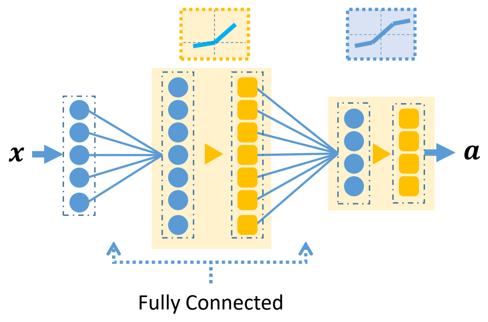


Fig. 8. Actor Network: The input layer is the state vector, the output layer is the action computed by the policy. The hidden layer consists of leaky ReLUs (Equation 18). The output layer is bounded by Equation 19. All the layers are fully connected.

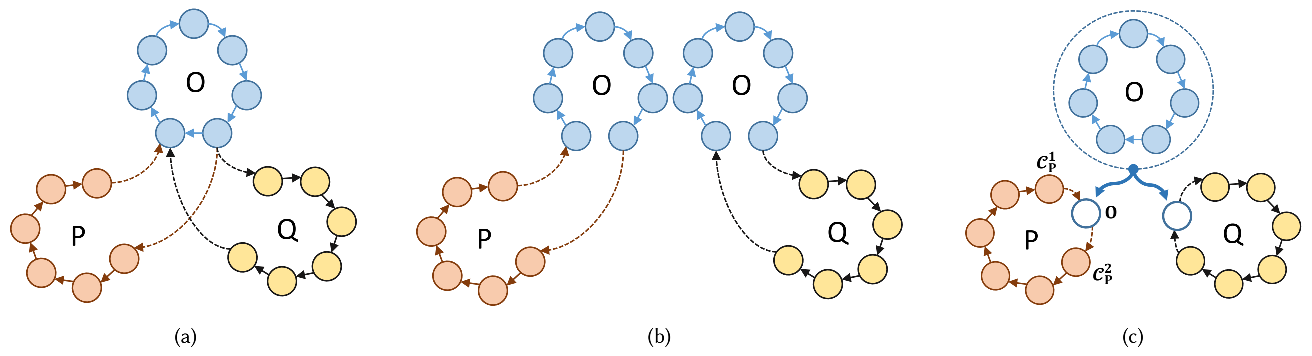


Fig. 9. Control graphs used in the learning process. (a) An example control graph consist of three basketball skills. (b) A control graph derived from (a) for the optimization of the open-loop arm control. (c) A control graph derived from (a) for the deep reinforcement learning of the arm control policy. The symbols C_P^1 and C_P^1 each represents a control fragment. See the text for more details.

which is defined as

$$f(z) = \begin{cases} 0.001(z+b) - b & z < -b \\ z & |z| \le b \\ 0.001(z-b) + b & z > b \end{cases} \qquad (19)$$

where $b = 1$ is an empirical bound on each DoF of the action. The policy gradient of Equation 16 is also rectified in the first 50k learning steps using

$$\nabla_{\theta_\pi} Q\left(\boldsymbol{x}, \pi(\boldsymbol{x};\theta_\pi);\theta_Q\right) = \nabla_{\theta_\pi} \pi(\boldsymbol{x};\theta_\pi) \cdot \nabla \boldsymbol{a} \qquad (20)$$

where each component of $\nabla \boldsymbol{a}$ is determined by

$$\nabla \boldsymbol{a}^i = \left.\begin{cases} b - \boldsymbol{a}^i, & \boldsymbol{a}^i > b \text{ and } \nabla_{\boldsymbol{a}} Q(\boldsymbol{x},\boldsymbol{a})^i > 0 \\ -b - \boldsymbol{a}^i, & \boldsymbol{a}^i < -b \text{ and } \nabla_{\boldsymbol{a}} Q(\boldsymbol{x},\boldsymbol{a})^i < 0 \\ \nabla_{\boldsymbol{a}} Q(\boldsymbol{x},\boldsymbol{a})^i, & \text{otherwise} \end{cases}\right|_{\boldsymbol{a}=\pi(\boldsymbol{x};\theta_\pi)}$$

In our early experiments, we find that the output of the critic network, $Q(\boldsymbol{x}, \boldsymbol{a})$, is prone to increasing quickly in the first thousands of learning steps, which causes the learning process to diverge. To solve this problem, we let the learning rate of the critic network decrease when the critic value increases. Specifically, the learning rate is updated to $\alpha_Q = 10^{-4} \times 10^{\alpha \bar{Q}}$ every one hundred learning steps, where $\bar{Q}$ is the average critic value in the past one hundred learning steps and α is a negative constant. We choose $\alpha = -0.04$ empirically, which decreases the learning rate to one percent of its original value when the average critic value is 50. This method is inspired by the commonly used approach where the learning rate decreases slowly at a fixed rate in the learning, but by letting the learning rate change according to the average Q-value instead of the fixed rate, we make the method independent from the convergence speed and thus generalize well to skills of different difficulty. For the actor network which is already bounded, we use a constant learning rate $\alpha_\pi = 10^{-4}$.

6 LEARNING OF CONTROL GRAPHS

The previous sections have introduced our learning pipeline for individual basketball skills. In this section, we will describe how we extend this method to learn control graphs that allow a simulated player to switch among a set of basketball skills.

A *control graph* is a graph data structure whose nodes are control fragments. Figure 9a shows an example control graph consisting of three basketball skills marked with (O), (P), and (Q), each consisting of a number of control fragments drawn in the same color. In this example graph, (O) is a cyclic skill like dribbling, (P) and (Q) are both noncyclic skills such as a crossover move. The player can perform each skill by executing the control fragments in sequence and can transition to another skill at designated transition points.

The structure of a control graph is created from the input motions by specifying the transitions between the skills. Transitions only happen at the frames where the states of the player are similar. A few frames near a transition point are blended to make the transition smooth. As with the learning of individual skills, we use the method of [Liu et al. 2016] to learn the locomotion control for the entire control graph.

Our system learns the arm control for the control graph in an incremental manner. Take the control graph in Figure 9a as an example. In both of the trajectory optimization stage and the deep reinforcement learning stage, we train the cyclic skill (O) first and then the noncyclic skills (P) and (Q), where the learned control policy of (O) is reused in the latter trainings.

Specifically, in the trajectory optimization stage, our system first optimizes the open-loop control clips for the cyclic skill (O) using the method discussed above and learns a stepwise linear policy $\boldsymbol{\pi}_O$. When training a noncyclic skill, e.g. (P), our system combines (O) and (P) into a cyclic motion (O+P) and optimizes open-loop control clips for it. Figure 9b shows this combined skill. This configuration ensures that both the skill (P) and the transitions between (O) and (P) are optimized simultaneously, so that the corresponding control policies are compatible with each other.

Because a linear policy usually cannot provide robust control of a basketball skill, our system still optimizes skill (O) when training the combined skill (O+P), but the corresponding learned linear feedback policy $\boldsymbol{\pi}_O$ will not be updated. To facilitate the optimization, $\boldsymbol{\pi}_O$ is used as a guiding policy for skill (O) in the CMA-ES algorithm, where a sample action is computed as

$$\begin{aligned} \boldsymbol{a}_k^{(j)} &= \boldsymbol{\pi}_O(\boldsymbol{s}, k) + \boldsymbol{a}_{\text{cma-es}}^{(j)} \\ &= M_{O,k}\boldsymbol{s} + \hat{a}_{O,k} + \boldsymbol{a}_{\text{cma-es}}^{(j)} \end{aligned} \tag{21}$$

Here $\{\boldsymbol{a}_{\text{cma-es}}^{(j)}\}$ are the actual samples generated by the CMA-ES algorithm. With the guiding policy $\boldsymbol{\pi}_O$, the optimization process can quickly converge to successful open-loop control for (O). Therefore the total computational cost for the combined skill is reduced.

In the deep reinforcement learning stage, our system first trains the non-linear control policy for skill (O) using the DDPG algorithm, then uses the combined skill (O+P) to train the noncyclic skill (P). During the latter training, the control policy of (O) is fixed, and the simulation of (O) is not used by the training algorithm. More specifically, as shown in Figure 9c, when the transition sequence

$$\begin{tikzcd} C_{\text{P}}^1 \arrow[r] & (\text{O}) \arrow[r] \arrow[loop above] & C_{\text{P}}^2 \end{tikzcd}$$

appears in the learning process, where (O) may be performed multiple times, the entire simulation will be recorded as a single transition tuple $(\boldsymbol{x}_{\text{P}}^1, \boldsymbol{a}_{\text{P}}^1, r, \boldsymbol{x}_{\text{P}}^2)$. If the player fails to perform (O), the corresponding state will be recorded in $\boldsymbol{x}_{\text{P}}^2$ and marked as a terminal state. In this way, our system learns a robust control policy that provides successful control of both the skill (P) and the transitions between (P) and (O).

7 RESULTS

We tested our method by letting a simulated player learn a set of basketball skills. The player model is 1.8 m tall, weighs 76 kg, and contains 101 DoFs in total. A full-sized basketball is used in these tests, which is 11.93 cm in radius and weighs 623.7 g. Our system is implemented in C++. We augment the Open Dynamics Engine (ODE) with the stable-PD control scheme as suggested in [Liu et al. 2013; Tan et al. 2011] to enable a large simulation time step so that the system can run faster than real-time. The built-in contact model of ODE is used in the simulation. This model treats each contact as a set of unilateral constraints and ensures that the ball's vertical velocity after a collision with the ground is proportional to that before the collision. We use a coefficient of restitution of 0.8 in our simulation. The learned arm control policies are executed by a neural network implementation based on the Eigen library [Guennebaud et al. 2010]. With the simulation time step of 0.01 s, our unoptimized single-threaded simulation pipeline runs at 3x faster than realtime on a desktop with a Intel Core i5-6500 @ 3.2 GHz CPU. The DDPG algorithm is implemented in Python 2 based on the Theano library [Theano Development Team 2016]. We run the learning algorithm on a separate workstation with dual 3.06 GHz 6-core, 12-thread Intel Xeon CPUs.

7.1 Cyclic Skills

We have tested a set of cyclic basketball skills to evaluate the capability of our method, including (A) carrying the ball while swinging arms, (B) dribbling in-place with the right hand, (C) rotating the ball around the waist, (D) dribbling in-place while switching hands, and (E) dribbling while running. The input motion clips for these skills were captured from human subjects of different heights and weights.

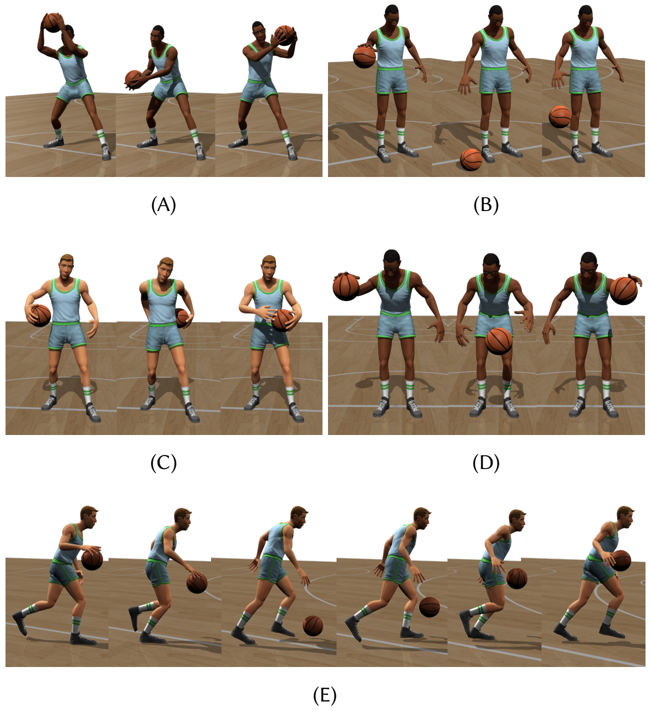


Fig. 10. Real-time simulation of the learned skills: (A) carrying a ball while swinging arms, (B) dribbling in-place with the right hand, (C) rotating a ball around the waist, (D) dribbling in-place while switching hands, (E) dribbling while running.

Table 1. Performance statistics for cyclic skills.

skill	T_{cycle} (second)	T_{opt} (hour)	N_{linear}	# of DDPG steps ($\times 10^6$)	T_{ddpg} (hour)
(A)	1.5	1.1	571	0.08	0.8
(B)	0.75	4.0	99	2.34	19.9
(C)	0.9	3.3	4	0.28	2.1
(D)	1.7	8.7	6	2.67	19.4
(E)	0.6	7.3	6	0.81	6.3

The skill: (A) carrying the ball while swinging arms, (B) dribbling in-place with the right hand, (C) rotating the ball around the waist, (D) dribbling in-place while switching hands, and (E) dribbling while running. T_{cycle} represent the period of the skill. T_{opt} is the computation time for the trajectory optimization processes. N_{linear} is the maximum number of cycles that the simulated player can perform a skill using a linear arm control policy. T_{ddpg} is the computation time for the deep reinforcement learning processes.

We retarget them onto our player model by copying the rotations of corresponding joints. A few frames at the beginning and the end of each clip are kinematically blended to make the reference motion cyclic. Although this kinematic process can cause artifacts such as foot skating and inaccurate arm motions, the learning pipeline automatically corrects these artifacts during the learning process.

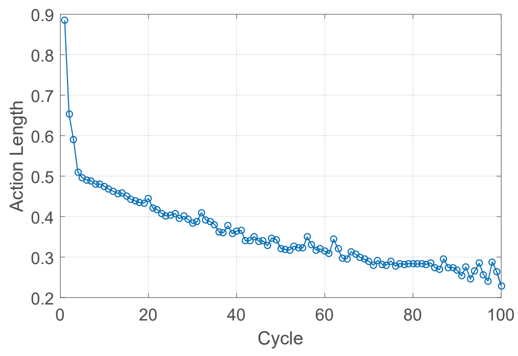


Fig. 11. Average length of the optimized corrective offsets in every trajectory optimization iteration of the dribbling skill (B).

The animation sequences shown in Figure 10 demonstrate the executions of these learned skills. We encourage readers to watch the supplemental video to better evaluate the motion quality.

Trajectory optimization. For each target skill, we optimize a motion sequence in which the player repeatedly performs the skill $N_{\text{rep}} = 100$ cycles. The sample distribution of the CMA-ES algorithm is initialized as a normal distribution $\mathcal{N}(0, \sigma_0^2)$, where the initial standard deviation σ_0 is set to 0.03 in the first cycle and is reduced to 0.01 in the following cycles as a good initial solution becomes available. The sampled corrective offsets for wrists are doubled to make the hands more flexible. The trajectory optimization process stops when either (a) the number of CMA-ES iterations exceeds 1000, (b) the optimization process stalls for 200 iterations, or (c) the average distance between the ball and the player's hands, i.e. the first term of Equation 6, is shorter than 1 cm for skills (A–D), and 2 cm for the dynamic skill (E).

The column labeled T_{opt} in Table 1 lists the optimization time for these skills. CMA-ES samples are independent, which allows us to parallelize the optimization process by evaluating each sample in a different working thread. The values listed under T_{opt} are measured with five working threads on a desktop with a 4-core, 8-thread CPU.

Figure 11 depicts the average length of the optimized corrective offsets during the trajectory optimization of the dribbling skill (B). We find that the length of the corrective offset can be large in the first few optimization iterations, which often causes abrupt movements in the resulting motion. The shrinkage factor effectively forces the optimization process to use smaller corrective offsets in the following iterations, thus producing smoother motion and more stable dribbling cycles.

Linear control policy. Our system then learns stepwise linear arm control policies from these optimized motion sequences using linear regression. The first 20 cycles of each motion sequence are discarded as they may contain unnecessarily large actions. The column labeled with N_{linear} in Table 1 shows the performance of these linear arm control policies, measured as the maximum number of cycles that the player can successfully perform a skill. This value is obtained based on one hundred experiments, each starting from a state randomly chosen from the optimized motion sequence.

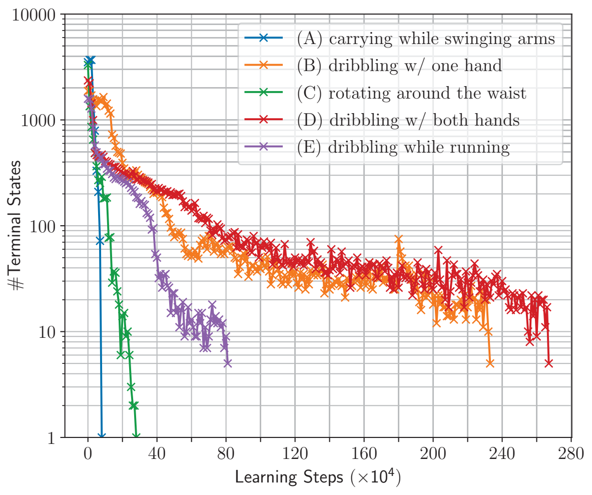


Fig. 12. Learning curves of cyclic skills. Each data point represents the number of terminal states in the past 10k transition tuples.

The learned linear control policy enables the player to repeatedly swing the arms hundreds of times while executing the carrying-a-ball skill (A). This skill is relatively easy to control because the player carries the ball with both hands and thus can apply control to the ball constantly. We can further improve the robustness of the linear policy by removing unnecessary information from the simulation state vector $\boldsymbol{s}$. Specifically, using a 18-DoF state vector that contains only the distance vectors between the ball and the thumb, index, and pinky fingers of each hand, the learned linear control policy allows the player to robustly perform this skill.

Our system cannot achieve robust control of the more complex skills (B–E) with linear feedback policies. The linear policy works the best for the dribbling in-place skill (B), where the ball is dribbled with one hand and has limited horizontal movement. The player can dribble the ball up to 99 times in our experiments. For the other skills, the player only finishes one cycle of the skill in most experiments. Despite significant experimentation, we did not find a representation of simulation state that can significantly improve the robustness of those linear policies.

Deep reinforcement learning. Our system then learns non-linear arm control policies for the skills using the DDPG algorithm sketched in Algorithm 1. During the learning process, our system monitors the number of terminal states in the last 10k transition tuples, which provides a measurement of the robustness of the control policy. Figure 12 depicts how the robustness of each control policy improves during the learning process. When no more than five terminal states occur in a 10k-transition interval, we check the robustness of the learned control policy and stop the learning process when the player can finish one thousand cycles of the target skill with the learned control policy.

The learned non-linear arm control policies enable robust control of all of these cyclic skills. The ball movement can change dramatically when the player performs a dynamic skill such as dribbling while running (E), where the highest position of the ball can change

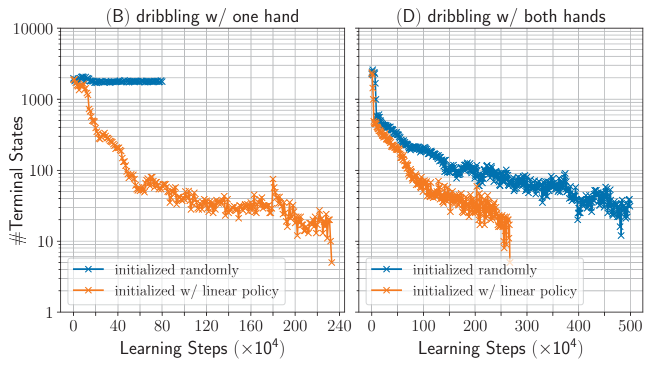


Fig. 13. Learning curves of skills (B) and (D) with different initialization strategies. Each data point represents the number of terminal states in the past 10k transition tuples. The learning process of skill (B) with random initialization (blue curve) stops early because of divergence.

horizontally up to 20 cm across dribbling cycles, but the learned control policy effectively handles the variation and produces robust dribbles. Although the dribbling control policies are learned on the ball with the coefficient of restitution of 0.8, the simulated player can successfully dribble a different ball whose coefficient of restitution is in the range of [0.78, 0.82]. The simulated players can also withstand some external pushes. For example, a player can keep running and dribbling the ball after being pushed horizontally by an up to 200Nx0.2s impulse. Larger pushes can cause dramatic change in player's state, and the player may fail to contact the ball again.

Initializing the replay buffers using the learned linear policies is important to the learning of the non-linear arm control policies. Figure 13 shows the learning processes of two in-place dribbling skills (B) and (D) with different initialization strategies, where the orange curves represent the learning processes that are initialized with linear policies, and the blue curves represent the learning processes that are initialized with the actions randomly sampled from the exploration strategy. The learning algorithm fails to find a successful control policy for skill (B) when it is initialized randomly, and the learning process diverges in the first million steps. For skill (D), the learning algorithm can eventually learn a robust policy from random initialization, but the learning process is much longer than the one initialized with the linear policy.

7.2 Control Graphs

We construct two control graphs, each containing one cyclic skill and two noncyclic skills, that allow the player to perform two integrated sets of basketball skills. The in-place skill graph consists of in-place dribbling in front of the body, between the legs, and behind the back. The running skill graph includes dribbling while running, front crossover, and spin crossover. The first skill of each graph is cyclic. All these skills and their mirrored counterparts are included in the graphs. The same control policy is used for both a skill and its reflection, where the states and actions are mirrored from left to right.

The noncyclic skills in these control graphs are more complicated than the cyclic skills and are difficult even for a skilled human player to perform without failing. To ensure that a successful control policy can be found, we increase the standard deviation of the exploration noise to $\sigma_\varepsilon = 0.1$ for these skills. We further update the open-loop arm control clips of the crossover moves by averaging the results of the trajectory optimization, which improves the physical accuracy of the control clips. In practice, the stopping criterion used in the deep reinforcement learning experiments on the cyclic skills is hard to satisfy on these noncyclic skills. Instead, our system performs five million DDPG learning steps and chooses the control policy corresponding to the learned step where the fewest terminal states occur in the past 10k transitions.

Table 2. Robustness of noncyclic skills in the learned control graphs.

control graph	noncyclic skill	success rate
in-place skills	dribbling between the legs	99.4%
	dribbling behind the back	99.6%
running skills	front crossover	98.1%
	spin crossover	95.7%

To measure the robustness of these learned control graphs, we let the player repeatedly perform a cyclic skill and a noncyclic skill from a random starting state recorded during the trajectory optimization, where the transition probability from the cyclic skill to the noncyclic skill is 0.5. Table 2 lists the success rate of those noncyclic skills in 1000 transitions. In practice, the player never fails at the cyclic skill and the transition, but may fail at the critical points of the noncyclic skill. For example, the player may miss the ball after the spin when performing the spin crossover skill.

We let the simulated basketball player perform the two learned control graphs. For the in-place skill graph, the player randomly chooses one of the skills to perform after finishing the current one. For the running skill graph, a greedy planner is implemented to enable interactive control of the direction of motion. The player dribbles the basketball while running and randomly performs one of the crossover skills when the difference between the current direction and the target direction is larger than 30 degrees. Figure 1 includes several keyframes generated during the simulation. We refer readers to the supplementary video for the animated performance.

8 DISCUSSION

Creating physics-based controllers for basketball skills is a challenging task. In this paper, we have proposed a framework for efficient learning of these skills from motion capture data. To the best of our knowledge, this is the first time that a variety of dribbling skills has been synthesized in realtime using a physics-based method. The learned basketball controllers can produce physically accurate ball movement and coordinated arm motion. Some motion details, such as that the ball may keep spinning for a while when it lightly contacts with the player's hands, can be difficult to create using kinematics approaches. Although our framework is designed for basketball skills, we believe that it can be extended to other motions, such as juggling, where the interaction between a simulated character and the manipulated object does not significantly affect the balance of the character. In future work, we are also interested in investigating other sports like soccer where the balance control is tightly coupled to the sports maneuvers.

Learning an arm control policy to realize robust control of the ball in a dynamic basketball skill is difficult, because the control task requires accurate control of the state of the ball but the ball is only in contact with the hand for a short period of time. We find that deep reinforcement learning is an effective way to achieve this goal, while trajectory optimization and learning of the linear control policies are both key components to the success and efficiency of our framework.

The trajectory optimization component of our framework can be used as an independent tool to construct physically plausible ball movement for a given locomotion sequence. Including fewer frames in the contact information set $\mathcal{H}$ and loosening the requirements on the distance between the ball and the player's hands can significantly reduce the total optimization time, but the resulting motions can be less natural because the ball may not move with the player's hands. We choose CMA-ES as our trajectory optimization method because it is derivative free, easy to implement, and independent from the choice of physics engine. When information such as the Jacobians and Hessians of the system can be easily obtained, our method can be extended to use more efficient trajectory optimization approaches such as differential dynamic programming (DDP) and iterative linear-quadratic regulator (iLQR).

Deep reinforcement learning is known to be a computationally costly process. It often takes millions of update steps to find a good control policy. Our medium-sized network structure works well with the full state vector, ensuring that the learning process can finish in a reasonable time. We note that the update of the neural network parameters is a bottleneck of the learning process with our CPU-based implementation. A speedup can be achieved via code optimization and a high performance GPU-based implementation of the learning algorithm.

Supervised learning can be an alternative to our method for learning control policies. Combining trajectory optimization and neural network policy regression has been shown to be an effective method for learning robust control of a diverse range of motions [Mordatch et al. 2015]. However, when learning a complicated non-linear control policy, a large amount of data are often necessary for supervised learning to prevent over-fitting. Obtaining such training data will need multiple runs of trajectory optimization, which can be a time-consuming process.

Directly optimizing the control policy is another alternative to reinforcement learning. A number of previous works learned successful control of a wide range of motions [Ding et al. 2015; Tan et al. 2014; Wang et al. 2010]. However, when learning a complex control policy with a large number of parameters, the optimization problem can be hard to solve and prone to poor local minima. Recently, Salimans et al. [2017] reported a method for directly optimizing complicated neural network policies using an enhanced evolution strategy and a distributive setting, which provides a possible way to solve this problem.

Similar to the sampling based method of [Liu et al. 2016], our framework learns static control graphs from input motion capture data. The use of reference motions greatly facilitates the learning and ensures the naturalness of the results, and potentially enable a user to alter a learned skill by providing a new reference motion or by modifying the current one, which is easier and more intuitive than tuning the hyperparameters of learning algorithms. However, a potential limitation of this approach is that the controllers can track the reference motions too rigidly. For example, a learned control graph in our system only allows a player to perform the skills via predefined transitions, while the locomotion component of a learned basketball controller does not allow the length of the steps to be changed at runtime. This limitation makes our framework not capable of learning controllers for the skills that require accurate control of steps, such as layups and slam dunks. Integrating more motions into control graphs can mitigate this problem by allowing the player to transition to other skills when necessary. In addition, combining the learning of arm control policies with other types of locomotion controllers and parameterizing the learned control graphs is a very interesting topic for future investigation.

REFERENCES

Mazen Al Borno, Martin de Lasa, and Aaron Hertzmann. 2013. Trajectory Optimization for Full-Body Movements with Complex Contacts. *IEEE Trans. Visual. Comput. Graph.* 19, 8 (Aug 2013), 1405–1414.

Sheldon Andrews and Paul G. Kry. 2013. Goal directed multi-finger manipulation: Control policies and analysis. *Comput. Graph.* 37, 7 (2013), 830 – 839.

Yunfei Bai, Kristin Siu, and C. Karen Liu. 2012. Synthesis of Concurrent Object Manipulation Tasks. *ACM Trans. Graph.* 31, 6, Article 156 (Nov. 2012), 9 pages.

Georg Bätz, Kwang-Kyu Lee, Dirk Wollherr, and Martin Buss. 2009. Robot basketball: A comparison of ball dribbling with visual and force/torque feedback. In *2009 IEEE International Conference on Robotics and Automation.* 514–519.

Georg Bätz, Uwe Mettin, Alexander Schmidts, Michael Scheint, Dirk Wollherr, and Anton S. Shiriaev. 2010. Ball dribbling with an underactuated continuous-time control phase: Theory & experiments. In *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems.* 2890–2895.

Stelian Coros, Philippe Beaudoin, and Michiel van de Panne. 2009. Robust Task-based Control Policies for Physics-based Characters. *ACM Trans. Graph.* 28, 5, Article 170 (Dec. 2009), 9 pages.

Stelian Coros, Philippe Beaudoin, and Michiel van de Panne. 2010. Generalized Biped Walking Control. *ACM Trans. Graph.* 29, 4, Article 130 (July 2010), 9 pages.

Kai Ding, Libin Liu, Michiel van de Panne, and KangKang Yin. 2015. Learning Reduced-order Feedback Policies for Motion Skills. In *Proceedings of the 14th ACM SIGGRAPH / Eurographics Symposium on Computer Animation (SCA '15).* ACM, 83–92.

Gaël Guennebaud, Benoît Jacob, and others. 2010. Eigen v3. http://eigen.tuxfamily.org. (2010).

Sehoon Ha, Yuting Ye, and C. Karen Liu. 2012. Falling and Landing Motion Control for Character Animation. *ACM Trans. Graph.* 31, 6, Article 155 (Nov. 2012), 9 pages.

Sami Haddadin, Kai Krieger, and Alin Albu-Schäffer. 2011. Exploiting elastic energy storage for cyclic manipulation: Modeling, stability, and observations for dribbling. In *2011 50th IEEE Conference on Decision and Control and European Control Conference.* 690–697.

Nikolaus Hansen. 2006. The CMA Evolution Strategy: A Comparing Review. In *Towards a New Evolutionary Computation.* Studies in Fuzziness and Soft Computing, Vol. 192. Springer Berlin Heidelberg, 75–102.

Matthew J. Hausknecht and Peter Stone. 2015. Deep Reinforcement Learning in Parameterized Action Space. *CoRR* abs/1511.04143 (2015). http://arxiv.org/abs/1511.04143

Jessica K. Hodgins, Wayne L. Wooten, David C. Brogan, and James F. O'Brien. 1995. Animating Human Athletics. In *Proceedings of SIGGRAPH.* 71–78.

Sumit Jain and C. Karen Liu. 2009. Interactive Synthesis of Human-object Interaction. In *Proceedings of the 2009 ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA '09).* 47–53.

Diederik P. Kingma and Jimmy Ba. 2014. Adam: A Method for Stochastic Optimization. *CoRR* abs/1412.6980 (2014). http://arxiv.org/abs/1412.6980

Paul G. Kry and Dinesh K. Pai. 2006. Interaction Capture and Synthesis. *ACM Trans. Graph.* 25, 3 (July 2006), 872–880.

Taesoo Kwon and Jessica K. Hodgins. 2017. Momentum-Mapped Inverted Pendulum Models for Controlling Dynamic Human Motions. *ACM Trans. Graph.* 36, 1, Article 10 (Jan. 2017), 14 pages.

Yoonsang Lee, Sungeun Kim, and Jehee Lee. 2010a. Data-driven Biped Control. *ACM Trans. Graph.* 29, 4, Article 129 (July 2010), 8 pages.

Yongjoon Lee, Kevin Wampler, Gilbert Bernstein, Jovan Popović, and Zoran Popović. 2010b. Motion Fields for Interactive Character Locomotion. *ACM Trans. Graph.* 29, 6, Article 138 (Dec. 2010), 8 pages.

Sergey Levine and Vladlen Koltun. 2013. Guided Policy Search. In *Proceedings of the 30th International Conference on Machine Learning*, Vol. 28(3). 1–9.

Sergey Levine and Vladlen Koltun. 2014. Learning Complex Neural Network Policies with Trajectory Optimization. In *Proceedings of the 31st International Conference on Machine Learning*, Vol. 32(2). 829–837.

Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. 2015. Continuous control with deep reinforcement learning. *CoRR* abs/1509.02971 (2015). http://arxiv.org/abs/1509.02971

C. Karen Liu. 2009. Dextrous Manipulation from a Grasping Pose. *ACM Trans. Graph.* 28, 3, Article 59 (July 2009), 6 pages.

C. Karen Liu, Aaron Hertzmann, and Zoran Popović. 2006. Composition of Complex Optimal Multi-character Motions. In *Proceedings of the 2006 ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA '06)*. 215–222.

Libin Liu and Jessica Hodgins. 2017. Learning to Schedule Control Fragments for Physics-Based Characters Using Deep Q-Learning. *ACM Trans. Graph.* 36, 3, Article 29 (June 2017), 14 pages.

Libin Liu, Michiel van de Panne, and KangKang Yin. 2016. Guided Learning of Control Graphs for Physics-Based Characters. *ACM Trans. Graph.* 35, 3, Article 29 (May 2016), 14 pages.

Libin Liu, KangKang Yin, Bin Wang, and Baining Guo. 2013. Simulation and Control of Skeleton-driven Soft Body Characters. *ACM Trans. Graph.* 32, 6 (2013), Article 215.

Adriano Macchietto, Victor Zordan, and Christian R. Shelton. 2009. Momentum control for balance. *ACM Trans. Graph.* 28, 3 (2009).

James McCann and Nancy Pollard. 2007. Responsive Characters from Motion Fragments. *ACM Trans. Graph.* 26, 3, Article 6 (July 2007).

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. 2015b. Human-level control through deep reinforcement learning. *Nature* 518, 7540 (26 Feb 2015), 529–533. Letter.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, and et al. 2015a. Human-level control through deep reinforcement learning. *Nature* 518, 7540 (26 Feb 2015), 529–533. Letter.

Igor Mordatch, Martin de Lasa, and Aaron Hertzmann. 2010. Robust Physics-based Locomotion Using Low-dimensional Planning. *ACM Trans. Graph.* 29, 4, Article 71 (July 2010), 8 pages.

Igor Mordatch, Kendall Lowrey, Galen Andrew, Zoran Popovic, and Emanuel V. Todorov. 2015. Interactive Control of Diverse Complex Characters with Neural Networks. In *Advances in Neural Information Processing Systems 28*. 3114–3122.

Igor Mordatch, Zoran Popović, and Emanuel Todorov. 2012a. Contact-invariant Optimization for Hand Manipulation. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA '12)*. 137–144.

Igor Mordatch, Emanuel Todorov, and Zoran Popović. 2012b. Discovery of Complex Behaviors Through Contact-invariant Optimization. *ACM Trans. Graph.* 31, 4, Article 43 (July 2012), 8 pages.

Uldarico Muico, Yongjoon Lee, Jovan Popović, and Zoran Popović. 2009. Contact-aware nonlinear control of dynamic characters. *ACM Trans. Graph.* 28, 3 (2009).

Xue Bin Peng, Glen Berseth, Kangkang Yin, and Michiel Van De Panne. 2017. DeepLoco: Dynamic Locomotion Skills Using Hierarchical Deep Reinforcement Learning. *ACM Trans. Graph.* 36, 4, Article 41 (July 2017), 13 pages.

Xue Bin Peng and Michiel van de Panne. 2017. Learning Locomotion Skills Using DeepRL: Does the Choice of Action Space Matter?. In *Proceedings of the ACM SIGGRAPH / Eurographics Symposium on Computer Animation (SCA '17)*. Article 12, 13 pages.

Nancy S. Pollard and Victor Brian Zordan. 2005. Physically Based Grasping Control from Example. In *Proceedings of the 2005 ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA '05)*. 311–318.

Philipp Reist and Raffaello D'Andrea. 2012. Design and Analysis of a Blind Juggling Robot. *IEEE Trans. Robot.* 28, 6 (Dec 2012), 1228–1243.

Tim Salimans, Jonathan Ho, Xi Chen, Szymon Sidor, and Ilya Sutskever. 2017. Evolution Strategies as a Scalable Alternative to Reinforcement Learning. *ArXiv e-prints* (March 2017). arXiv:stat.ML/1703.03864

John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. 2015a. Trust Region Policy Optimization. In *The 32nd International Conference on Machine Learning*. 1889–1897.

John Schulman, Philipp Moritz, Sergey Levine, Michael I. Jordan, and Pieter Abbeel. 2015b. High-Dimensional Continuous Control Using Generalized Advantage Estimation. *CoRR* abs/1506.02438 (2015). http://arxiv.org/abs/1506.02438

David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. 2014. Deterministic Policy Gradient Algorithms. In *The 31st International Conference on Machine Learning*. 387–395.

Jie Tan, Yuting Gu, C. Karen Liu, and Greg Turk. 2014. Learning Bicycle Stunts. *ACM Trans. Graph.* 33, 4, Article 50 (July 2014), 12 pages.

Jie Tan, Yuting Gu, Greg Turk, and C. Karen Liu. 2011. Articulated Swimming Creatures. *ACM Trans. Graph.* 30, 4, Article 58 (July 2011), 12 pages.

Theano Development Team. 2016. Theano: A Python framework for fast computation of mathematical expressions. *arXiv e-prints* abs/1605.02688 (May 2016). http://arxiv.org/abs/1605.02688

Adrien Treuille, Yongjoon Lee, and Zoran Popović. 2007. Near-optimal Character Animation with Continuous Control. *ACM Trans. Graph.* 26, 3, Article 7 (July 2007).

Hado van Hasselt. 2012. *Reinforcement Learning in Continuous State and Action Spaces.* Springer, Berlin, Heidelberg, 207–251.

Kevin Wampler and Zoran Popović. 2009. Optimal gait and form for animal locomotion. *ACM Trans. Graph.* 28, 3 (2009), Article 60.

Jack M. Wang, David J. Fleet, and Aaron Hertzmann. 2010. Optimizing Walking Controllers for Uncertain Inputs and Environments. *ACM Trans. Graph.* 29, 4, Article 73 (July 2010), 8 pages.

Nkenge Wheatland, Yingying Wang, Huaguang Song, Michael Neff, Victor Zordan, and Sophie Jörg. 2015. State of the Art in Hand and Finger Modeling and Animation. *Comput. Graph. Forum* 34, 2 (May 2015), 735–760.

Yuting Ye and C. Karen Liu. 2012. Synthesis of detailed hand manipulations using contact sampling. *ACM Trans. Graph.* 31, 4 (2012), Article 41.

KangKang Yin, Kevin Loken, and Michiel van de Panne. 2007. SIMBICON: Simple Biped Locomotion Control. *ACM Trans. Graph.* 26, 3 (2007), Article 105.

Wenping Zhao, Jianjie Zhang, Jianyuan Min, and Jinxiang Chai. 2013. Robust Realtime Physics-based Motion Control for Human Grasping. *ACM Trans. Graph.* 32, 6, Article 207 (Nov. 2013), 12 pages.

Victor Zordan, David Brown, Adriano Macchietto, and KangKang Yin. 2014. Control of Rotational Dynamics for Ground and Aerial Behavior. *IEEE Trans. Visual. Comput. Graph.* 20, 10 (Oct 2014), 1356–1366.