Iterative Training Of Dynamic Skills Inspired By Human Coaching Techniques

Sehoon Ha and C. Karen Liu
Georgia Institute of Technology

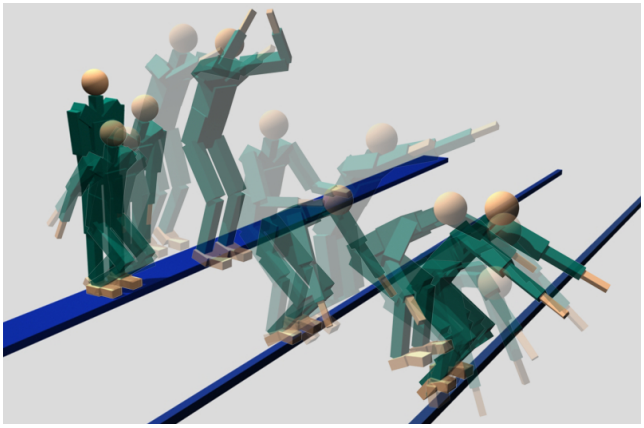

Fig. 1. Two precision jumps on narrow rails.

Inspired by how humans learn dynamic motor skills through progressive process of coaching and practices, we introduce an intuitive and interactive framework for developing dynamic controllers. The user only needs to provide a primitive initial controller and high-level, human-readable instructions as if s/he is coaching a human trainee, while the character has the ability to interpret the abstract instructions, accumulate the knowledge from the coach, and improve its skill iteratively. We introduce "control rigs" as an intermediate layer of control module to facilitate the mapping between high-level instructions and low-level control variables. Control rigs also utilize the human coach's knowledge to reduce the search space for control optimization. In addition, we develop a new sampling-based optimization method, Covariance Matrix Adaptation with Classification (CMA-C), to efficiently compute control rig parameters. Based on the observation of human ability to "learn from failure", CMA-C utilizes the failed simulation trials to approximate an infeasible region in the space of control rig parameters, resulting a faster convergence for the CMA optimization. We demonstrate the design process of complex dynamic controllers using our framework, including precision jumps, turnaround jumps, monkey vaults, drop-and-rolls, and wall-backflips.

Categories and Subject Descriptors: I.3.7 [**Computer Graphics**]: Three-Dimensional Graphics and Realism—*Animation*

General Terms: Character Animation

Additional Key Words and Phrases: Physics-based Animation, Optimal Control

1. INTRODUCTION

Mastering a dynamic motor skill, such as a handstand in gymnastics, or a precision jump in Parkour, usually requires an iterative process with interactive coaching and repetitive practices. Based on the current skill level of the trainee, the coach gives instructions that emphasize the key areas for improvement. The trainee then internalizes the new information and improves the skill through practices. The learning process alternates between coaching and practicing stages until the skill is acquired. In contrast, teaching a physically simulated character a new motor skill entirely depends on the effort of the controller developer, from the design of the control architecture to the tweaking of low-level control parameters. We hypothesize that the controller development can be greatly simplified by exploiting the same learning principles humans use to acquire new motor skills. In addition, if designing new controller can be done in the similar fashion as coaching a human trainee, the existing controllers can be easily adapted, extended, or concatenated for new situations.

This paper attempts to formalize the methodologies humans use to learn dynamic motor skills. We present an algorithmic framework to facilitate the iterative learning process of coaching and practicing. During the coaching stage, the user only needs to provide a primitive initial controller and high-level, human-readable instructions as if s/he is coaching a human trainee. For example, a human coach typically uses high-level instructions, such as "extend the legs" or "push the ground", rather than specific descriptions of joint angles. During the practicing stage, the character is capable of following the instructions and improving its motor skill effectively on its own. That is, the character has the autonomous ability to interpret the abstract instructions, accumulate the knowledge from the coach, and optimizes its motion based on the guidance.

The main challenge of this work is to formalize these elusive principles of human learning into mathematical models for controller design. Our underlying assumption is that any motor skill can be achieved by using simple proportion-derivative (PD) style control and Jacobian transpose control at every actuated joint and every body part, *if* the control parameters are properly determined. This formulation, however, introduces a prohibitively large space of control variables for the existing optimization methods. Our controller design framework solves this issue by utilizing human coaching knowledge to select an appropriate subset of control variables (coaching stage) and relying on a new sampling-based optimization method to determine the value of control variables (practicing stage).

Using high-level, human-readable instructions can potentially simplify the controller design, but directly mapping high-level instructions to low-level control variables is a challenging task. We introduce an intermediate layer of control module, called control rigs, to interpret the human instructions during the coaching stage. A control rig simultaneously controls a set of low-level control variables in a coordinated fashion. For example, a control rig that lexes legs uses a single parameter, the distance between the waist and the feet, to control the target joint angles of the PD controllers on the hips, knees, and ankles. With this intermediate layer, mapping human instructions to the low-level control variables can be done automatically by selecting appropriate control rigs. In addition, us-

ing control rigs instead of low-level variables reduces the search space for the optimization. We design a set of control rigs from frequently used instructions for Parkour training. These control rigs are general and can be reused for coaching different sports.

To determine the control variables efficiently during the practicing stage, we introduce the concept of "learning from failure" using a sampling-based optimization method. Our key insight is that the failed samples contain as much useful information as the successful ones. For example, falling on the ground or hitting obstacles are valuable experiences to learn vaulting. Instead of throwing away those failed simulation trials, our algorithm uses them to approximate the boundary of the feasible region in the control variable space. Having an approximated feasible region accelerates the optimizations by preventing the character to repeat failures committed before. Based on this idea, we build Support Vector Machines in concert with Covariance Matrix Adaptation (CMA), called Covariance Matrix Adaptation with Classification (CMA-C). The main advantage of CMA-C is that it exploits every simulated trial; the successful ones are used to contract the covariance matrix while the failed ones are used to refine the boundary of feasible region.

We demonstrate the design process of complex dynamic controllers using our framework, including precision jumps, turnaround jumps, monkey vaults, drop-and-rolls, and wall backflips. We show that the character started out with basic controllers; using PD control to track a few roughly specified poses; and were able to learn these complex dynamic skills within minutes with only a few high-level instructions from the user. Once a controller is developed, parameterizing it to a family of similar controllers for concatenation can be done without additional effort from the user.

2. RELATED WORK

Designing controllers for physically simulated biped characters is a challenging problem due to its nonlinear dynamics and under-actuated control. Earlier work demonstrated the potential of physics-based character animation by simulating a variety of human motor skills using manually constructed controllers [Hodgins et al. 1995; Wooten 1998; Faloutsos et al. 2001]. The process of design controllers required tedious parameter tuning, but the results were compelling and inspiring. Later, researchers simplified the design process by developing more general control principles resulting in much more robust locomotion controllers [Yin et al. 2007; Coros et al. 2010; Lee et al. 2010; Lasa et al. 2010]. The combination of PD servos and a balance control strategy, which regulates the center of mass and global orientation, has proven very effective in performing walking in various environments with disturbances. We use similar underlying controllers in this work: PD servos for controlling joint configurations and Jacobian transpose control [Sunada et al. 1994] for regulating the center of mass. However, the parameters for these low-level controllers, including the gains, target joint angles, and desired virtual forces, are all determined automatically without any user intervention.

Optimizing control parameters is a widely used approach in biomechanics and computer animation. Due to the large number of low-level control parameters, most previous work applied domain knowledge [Wang et al. 2009; 2010; Wang et al. 2012], used task goals [Yin et al. 2008; Wu and Popović 2010; Liu et al. 2012], or formulated smaller horizons [Sok et al. 2007] to reduce the problem domain. Similarly, our work reduces the domain by using prior knowledge to define a set of control rigs. These control rigs are intuitively associated with human-readable instructions and are general enough to be reused for new motor tasks. To solve the optimization problem, sampling-based methods are usually preferred over gradient-based methods due to the discontinuous nature of the problem. Among different sampling-based optimization methods, Covariance Matrix Adaptation Evolution Strategy [Hansen and Kern 2004] was frequently applied in computer animation in the recent years because it converges relatively fast for high-dimensional problems. In most cases, however, using CMA to optimize control parameters still requires hours or even days of computation. To provide a more interactive controller design framework, we introduce Covariance Matrix Adaptation with Classification, a new algorithm, which speeds up the standard CMA significantly.

Controller generalization is also a challenging and important research problem. Many researchers proposed various ways to combine or concatenate existing controllers for new tasks. The common difficulty of this problem is that the control parameters cannot be trivially interpolated or blended. da Silva *et al.* demonstrated that optimal controllers can be interpolated to create another optimal controller for a new task using linear Bellman combination [da Silva et al. 2009]. Similarly, Muico *et al.* proposed to track multiple trajectories in parallel and automatically reweight different actions to generate appropriate control force [Muico et al. 2011]. Besides linear Bellman combination, Faloutsos *et al.* [2001] developed a supervised learning method to predict whether a controller can successfully handle the transition between motor skills. Recently, Liu *et al.* used affined feedback laws to parameterize existing controllers [Liu et al. 2012]. They further showed that composition of these parameterized controllers can be done by developing special controllers for the transition motions.

We demonstrate our framework by training a virtual character to perform highly dynamic motor skills. Motions with relatively long airborne time can be physically simulated by designing state-machine-like controllers [Hodgins et al. 1995; Faloutsos et al. 2001], sampling around reference trajectories [Liu et al. 2010; Liu et al. 2012], or planning center of mass or momentum [Mordatch et al. 2010; Ha et al. 2012]. Alternatively, trajectory optimization under physics constraints is also a viable approach to produce highly dynamic motions [Popović and Witkin 1999; Liu and Popović 2002; Fang and Pollard 2003; Safonova et al. 2004; Coros et al. 2011; Borno et al. 2013] The optimization provides a mathematical way to formalize the energy expenditure or motion style as an objective function, but this approach is too costly for consecutive motion editing process. In contrast, our approach solves for a series of optimization problems in an efficient manner by exploiting the accumulated information from the previous problems.

We chose highly dynamic motions to showcase our work, but our controller design framework is generous to other types of motions as long as the appropriate control rigs are provided.

3. ITERATIVE CONTROLLER DESIGN

We introduce a general framework to design dynamic controllers using high-level, human-readable instructions (Figure 2). The iterative process begins with an input controller. We view the initial controller as a blackbox because our algorithm does not interact with its internal implementation. For all our experiments, we used a simple pose-tracking controller with 4 to 6 keyframes. During training, the initial controller is improved iteratively through alternating stages of *coaching* and *practicing*. The output is a new controller that meets the requirements of the user. Once a controller is developed, we can generalize it by parameterization or concatenation for new situations.

During each coaching stage, the user provides high-level instructions to correct undesired behaviors, change task objectives, or add different styles to the motion. According to the type of the instruc-

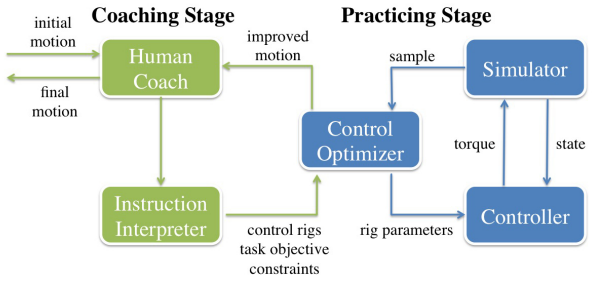

Fig. 2. Overview diagram.

tions, the *instruction interpreter* automatically selects the appropriate control rigs, and modifies constraints or the objective function for the optimization.

During each practicing stage, the *control optimizer* searches for control rig parameters using our new algorithm, Covariance Matrix Adaptation with Classification (CMA-C). The control optimizer uses the current estimate of rig parameters to generate motion samples. A controller can be represented as a function g, which takes the current state $\mathbf{q}_t$ as input and generates torque τ_t. In addition, we denote s as a function that simulates the motion from a given state under a given controller, and outputs the final state $\mathbf{q}_f$ of the simulation.

$$\mathbf{q}_f = s(\mathbf{q}_t, g) \tag{1}$$

4. COACHING STAGE

The coaching stage takes in high-level user instructions and interprets them by revising the task objective or adding constraints. We introduce "control rigs", as an intermediate layer between high-level instructions and low-level control variables. A control rig, predefined by our framework, is a function of a set of low-level control variables. It allows for more coordinated control of the low-level variables and provides more intuitive mapping to high-level instructions. The main advantage of using control rigs is that it reduces the optimization time significantly by suggesting the most effective directions to optimize. Although a control rig needs to be manually defined, it can be shared by different instructions or repurposed for new motor tasks.

4.1 Instructions

Although coaching strategies and styles vary widely, the basic instructions commonly used to break down a complex movement are surprisingly consistent. Most instructions provide a numerical or categorical correction to improve a particular part of the body. For example, "Lower the center of mass more" or "raise your arms to 45 degrees". From observing Parkour training sessions and tutorials, we define a set of coaching instructions in Table II.

4.2 Control Rigs

A control rig r is a pre-defined function of a set of low-level PD or Jacobian Transpose controllers. A PD controller tracks the target joint angle based on the feedback rule: $\tau = k_s(\hat{\theta} - \theta) - k_d(\dot{\theta})$, where k_s and k_d are the gain and the damping coefficient of the joint and $\hat{\theta}$ is the desired value for the joint angle. A Jacobian Transpose controller computes the required joint torques to produce the desired "virtual force", $\mathbf{f}_v$, at a point $\mathbf{x}$, using the Jacobian Transpose mapping: $\tau = \mathbf{J}^T(\mathbf{x})\mathbf{f}_v$, where $\mathbf{J}(\mathbf{x})$ is the Jacobian matrix at $\mathbf{x}$. These two types of low level controllers, when combined properly, can effectively control the pose and global motion of the character. A control rig, $\tau = r(\mathbf{q}_t, \mathbf{p})$, takes in the current state $\mathbf{q}_t$ and the rig parameters $\mathbf{p}$ to produce torques which are added to the total torques applied to the character.

We keep a set of active control rigs, $\mathcal{R} = \{r_1, ..., r_m\}$, during training. As the user provides more instructions, more control rigs are included to the active set. With optimized parameters $\mathcal{P} = \{\mathbf{p}_1, ..., \mathbf{p}_m\}$, we obtain an improved controller g_R.

$$g_{\mathcal{R}}(\mathbf{q}, \mathcal{P}) = g(\mathbf{q}) + \sum_{i=1}^{m} r_i(\mathbf{q}, \mathbf{p}_i) \tag{2}$$

In this paper, we designed five control rigs from common instructions used for learning gymnastics and Parkour. Each control rig, r_i, has a set of arguments determined by the instruction from which r_i is created, and a set of rig parameters which are optimized during the practicing stage (Table I).

(1) TargetJoints rig consists of the PD controllers on a set of joints specified by the instruction. The rig parameters are defined as the target joint angles of the PD controllers.

(2) IKPose rig solves for the joint angles to meet a desired relative position of two bodies specified by the instruction. The solved joint angles are then used as the target angles for a set of PD controllers.

(3) Stiffness rig adjusts the gains of the PD controllers on a set of joints specified by the instruction. The rig parameters are defined as the gains.

(4) VirtualForce rig consists of a Jacobian Transpose controller which computes required torques to produce a virtual force at the center of mass of a body link specified by the instruction. The rig parameter is defined as the desired virtual force

(5) FeedbackVirtualForce rig consists of a Jacobian Transpose controller on a body link specified by the instruction. Instead of treating the desired virtual force as a rig parameter, it is computed using the feedback rule: $f_v = k_s(\hat{\mathbf{C}} - \mathbf{C}) - k_d\dot{\mathbf{C}}$, where $\mathbf{C}$ and $\hat{\mathbf{C}}$ are the center of mass of the whole body and its desired position. Intuitively, this rig computes the torques that generate a virtual force to bring the center of mass closer to the desired position.

We show that these five control rigs are sufficient to generate the motor skills demonstrated in the result section.

4.3 Instruction Interpreter

The instruction interpreter translates a human-readable instruction into two possible actions: modifying the task objective or adding constraints for the optimization problem. In addition, if these actions involve new control rigs, the instruction interpreter will add them to the optimization domain. We define a set of simple rules that map the instructions to the control rigs. Please see details in Table II.

The task objective. The task objective is evaluated at the final state of the motion by an objective function .

$$f(\mathcal{P}) = \sum_{i=1}^{n} \|h_i(s(\mathbf{q}_0, g_{\mathcal{R}}) - \hat{h}_i\| \tag{3}$$

Table I. Control rigs.

Rig Type <Arguments>	Description	Rig Parameters
TargetJoints <joint a, $\cdots$ >	Command a set of joints to achieve desired angles simultaneously.	Desired joint angles
IKPose <body a, body b>	Compute a target pose to meet the desired relative position between body a and body b. Command joints to achieve the target pose.	Desired distance or desired angles
Stiffness < joint a, $\cdots$ >	Command a set of joints with desired stiffness.	Gains
VirtualForce <body a>	Apply torques which produce the desired virtual force at the center of mass of body a.	[1]Desired virtual force in end-effector frame
FeedbackVirtualForce <body a, $\hat{\mathbf{C}}$ >	[2]Apply torques which produce the virtual force at the cetner of mass of body a. The virtual force is computed by a feedback rule.	-

[1] f_u is in the direction of C - S. f_v is orthogonal to f_u and parallel to the contacting surface.
[2] We use the following feedback rule on the center of mass of the whole body: $f_v = k_s(\hat{\mathbf{C}} - \mathbf{C}) - k_d\dot{\mathbf{C}}$, where $k_s = 300$ and $k_d = 6$.

Table II. Interpretation of instructions. Each instruction is associated with a control rig, an objective term and/or a contraint.

Instruction	Introduced control rig	Introduced objective / constraint
FLEX|EXTEND joint BY θ	TargetJoints<joint>	$\|q_{joint}(\mathbf{q}_f) - \theta\|$
MOVE [1]direction BY δ	IKPose<[2]contact, root>	[3] $\|\mathbf{C}(\mathbf{q}_f)\cdot\mathbf{d} - (\mathbf{C}(\mathbf{q}_f^{prev})\cdot\mathbf{d} + \delta)\|$
TRANSLATE limb direction BY δ	IKPose<root, limb>	$\|\mathbf{pos}_{limb}(\mathbf{q}_f)\cdot\mathbf{d} - (\mathbf{pos}_{limb}(\mathbf{q}_f^{prev})\cdot\mathbf{d} + \delta)\|$
ROTATE limb direction BY δ	IKPose<root, limb>	$\|\mathbf{rot}_{limb}(\mathbf{q}_f)\cdot\mathbf{d} - (\mathbf{rot}_{limb}(\mathbf{q}_f^{prev})\cdot\mathbf{d} + \delta)\|$
SPEED NEAR $\hat{\mathbf{C}}$	VirtualForce<contact>	$\|\dot{\mathbf{C}}(\mathbf{q}_f) - \hat{\mathbf{C}}\|$
SPEEDUP/SLOWDOWN γ %	VirtualForce<contact>	$\|\mathbf{P}(\mathbf{q}_f) - (\mathbf{P}(\mathbf{q}_f^{prev}) * \gamma)\|$
TURNFASTER/TURNSLOWER γ %	VirtualForce<contact>	$\|\mathbf{L}(\mathbf{q}_f) - (\mathbf{L}(\mathbf{q}_f^{prev}) * \gamma)\|$
BALANCE	FeedbackVirtualForce<contact, $\hat{\mathbf{C}}_{bal}$ >	$\|\dot{\mathbf{C}}(\mathbf{q}_f)\|$
RELAX|STIFFEN joint BY γ %	Stiffness<joint>	$\|ks_{joint}(\mathbf{q}_f) - (ks_{joint}(\mathbf{q}_f^{prev})\cdot\gamma)\|$
PUSH/PULL limb AGAINST surface	VirtualForce<limb>	-
PLACE body|C|S NEAR body|C|S|constant	-	$\|\mathbf{pos}_{body\|C\|S}(\mathbf{q}_f) - \mathbf{pos}_{body\|C\|S\|constant}(\mathbf{q}_f)\|$
[4]body|C|S IN RELATION TO body|C|S|constant	-	relation(body|C|S, body|C|S|constant)

Notations. $\mathbf{q}_f$, $\mathbf{q}_f^{prev}$: the final state of the current motion and the previous motion. $\mathbf{C},\mathbf{S},\mathbf{P},\mathbf{L}$: the center of mass, center of pressure, linear momentum, and angular momentum. $\mathbf{pos}_{limb}$, $\mathbf{rot}_{limb}$: the limb position and orientation. q_{joint}, ks_{joint} : the joint angle and stiffness.
[1] The direction can be one of the predefined keywords, such as up, down, left, or right, or it can be an arbitrary 3×1 direction vector $\mathbf{d}$.
[2] "contact" indicates the body parts in contact
[3] We project $\mathbf{C}(\mathbf{q})$ in the desired direction, $\mathbf{d}$, specified in the instruction.
[4] The last instruction creates a constraint, instead of an objective function.

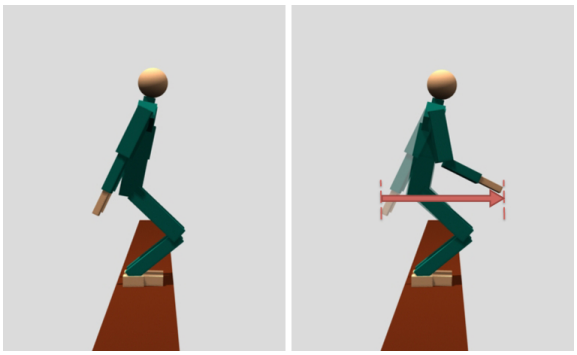

Fig. 3. The user can adjust the positions of hands by giving an instruction "TRANSLATE hands forward BY 0.5m". The instruction will add an IKPose rig for arms and modify the desired position of hands by 0.5m in the forward direction.

When the user specifies an instruction from Table II (except for the last instruction, which will be explained in the next paragraph), a new term, $\|h_i(\mathbf{q}_f) - \hat{h}_i\|$, will be added to the objective function. h_i is a function that evaluates a quantity derived from the final state. $\hat{h}_i$ is the desired value for that quantity. For example, to interpret the instruction "TRANSLATE hands forward BY 0.5m" (Figure 3), the keyword "forward" is first mapped to the predefined direction $\mathbf{d} = [1,0,0]^T$. Then $h_i(\mathbf{q}_f)$ is defined as the average position of the hands in the forward direction $(\mathbf{pos}_{hands}(\mathbf{q}_f)\cdot\mathbf{d})$. Finally, the desired value $\hat{h}_i$ is computed by adding 0.5m to the previous average position of the hands in the forward direction $(\mathbf{pos}_{hands}(\mathbf{q}_f^{prev})\cdot\mathbf{d} + 0.5)$. Although we use the final state in Equation 3, in our implementation the entire motion sequence is available for evaluation. Thus the cost function can depend on any state in the motion sequence. In addition, we can formulate a cost function that affects the timing of the motion by evaluating the elapsed time for each phase.

Constraints. Most dynamic motor skills are subject to constraints, such as maintaining balance or contact conditions. When the character's motion fails to meet any of the constraints, the simulation will terminate immediately. The failed motion adds a penalty term, $K(T_{max} - t)$, to the objective function (Equation 3) to penalize early failure. t indicates the time when failure occurs, and K and T_{max} are set to 1000 and 2 respectively. T_{max} can be adjusted based on the expected duration of the successful motion.

In our instruction set, the last instruction (Table II) introduces a constraint to enforce spatial relationship between two body parts. For example, "head IN FRONT OF root" instruction places a lower bound on the x position of head $(\mathbf{d}\cdot(pos_{head}(\mathbf{q}_f) - pos_{root}(\mathbf{q}_f)) > 0$ where $\mathbf{d} = (1,0,0)^T)$.

5. PRACTICING STAGE

The practicing stage optimizes the parameters of each control rig selected by the coaching stage. Although the search space is largely reduced by using control rigs rather than low-level control variables, we still need to solve a non-convex and discontinuous optimization problem. Much previous work in computer animation applied Covariance Matrix Adaptation Evolution Strategy (CMA-ES) to problem in this nature. The standard procedure at each iteration involves generating samples in the control variable space, using the samples to simulate motions, and evaluating each motion according to the cost function.

However, the standard CMA-ES does not exploit two distintive features of our problem. First, our problem has a clear definition of infeasible samples, such as the control parameters that result in an imbalanced motion. Second, because our framework is an iterative process, we have a series of optimization problems sharing very similar feasible regions. Without leveraging these features, the standard CMA-ES spends much computation time on repeatedly evaluating failed samples.

We designed a new algorithm, called CMA-C, based on the observation of human's ability to *learn from failure*. Because failure in the real world is usually associated with pain or injury, humans tend to be very effective in characterizing the cause of failure and trying to avoid the same mistakes in the future. To this end, CMA-C simultaneously builds a set of Support Vector Machines (SVMs) along with the evolution of CMA. Each SVM approximates the infeasible region of a particular type of failure. CMA-C accelerates the optimization by preventing redundant evaluations of failed samples. Moreover, the constructed SVMs can be reused by subsequent optimizations because they share similar feasible regions, further speeding up the computation significantly.

5.1 CMA-C

CMA-C is designed for solving a general optimization problem with multiple constraints.

$$\begin{aligned} \mathbf{x}^* &= \underset{\mathbf{x}}{\operatorname{argmin}}\, f(\mathbf{x}) \\ &\text{subject to } c_i(\mathbf{x}) = 0, \text{ where } i = 1 \cdots n \end{aligned} \tag{4}$$

In our formulation, the cost function $f(\mathbf{x})$ evaluates the performance of the simulated motion generated by a given set of control rig parameters (i.e. $\mathbf{x}$ refers to $\mathcal{P}$). Each constraint $c_i(\mathbf{x})$ represents a type of failure when the motion cannot satisfy it. $c_i(\mathbf{x})$ can be in the form of inequality constraint, but we only show equality constraints for clarity. CMA-C can be applied to any problem with the form of Equation (4), but it is particularly effective when evaluating $f(\mathbf{x})$ is costly or when the problem is highly constrained.

Our main idea is to construct classifiers using SVMs while running CMA. For each constraint, we build a SVM to represent its feasible region. Every time a sample is randomly drawn, we first use SVMs to predict whether this sample will satisfy all constraints. If so, we evaluate it using $f(\mathbf{x})$. Otherwise, the sample is discarded without evaluation. In the context of our problem, we use SVMs to predict whether a set of control variables yields successful motion before we simulate it. If a sample is predicted to satisfy all the constraints, we evaluate its cost and assign a label for each SVM: $+1$ if the sample indeed satisfies the corresponding constraint and -1 otherwise. At the end of each CMA iteration, we refine the boundary of each SVM using all the samples. Algorithm 1 and 2 outline the procedure of CMA-C.

Algorithm 1: CMA-C

```
Initialize m, C, σ;
Initialize SVM_{1..n};
while not terminate do
    for i = 1 → λ do
        x_i = selectSample(m, C, σ, SVM_{1..n});
        (f_i, e_{i,1}, ..., e_{i,n}) = fitness(x_i);
    sort(x_{1..λ});
    (m, C, σ) = updateCMA(x_{1..λ}, f_{1..λ}, m, C, σ);
    for i = 1 → n do
        if enough samples for SVM_i and γ_i is null then
            γ_i = 1/2(kσ)^2
        updateSVM(SVM_i, γ_i, x_{1..λ}, e_{1..λ,i});
```

Algorithm 2: selectSample()

```
Data: m, C, σ, SVM_{1..n}
Result: selected sample x
while not reach maximum trials do
    x = gaussianSelection(m, σ^2 C);
    reject = False;
    for i = 1 → n do
        if SVM_i.activated() and SVM_i.predict(x) < 0 then
            reject = True;
            break;
    if not reject then
        return x
return gaussianSelection(m, σ^2 C);
```

When the SVM makes a correct prediction, it speeds up the convergence of CMA. When the SVM makes an incorrect prediction (i.e. generates a sample with -1 label), our algorithm still utilizes the negative sample to improve the boundary of the feasible region. However, there are two issues with this algorithm when applied in practice. First, a SVM requires a sufficient number of positive and negative samples before it can be activated. This requirement imposes a long "warm-up" time if the initial CMA distribution has low likelihood to generate feasible samples. Second, although SVMs can use kernels to represent non-linear boundaries, tweaking kernel parameters in SVMs can greatly affect the results of classification.

The first issue is exasperated by problems with a large number of constraints and a relatively small feasible region, such as the problem of developing parkour controllers. To reduce the warm-up time, we represent each constraint with a SVM individually, instead of using one aggregate SVM to represent the intersection of all constraints. During the optimization, we superimpose all the activated SVMs and approximate the feasible region by taking the intersection of all positive regions. Building multiple SVMs significantly reduces the warm-up time because a positive sample for an aggregate SVM must satisfy all constraints, where as a positive sample for each separate SVM only needs to satisfy one constraint. Collecting enough positive samples to activate an aggregate SVM clearly takes much longer time than activating each individual SVM (Figure 4). In our toy problems, the first SVM is constructed after 17.2 evaluation on average, an aggregate SVM requires 101.7 evaluations.

Table III. CMA-C evaluation on five problems. CMA-C improves the computation by four to five times. μ, λ, and σ represents CMA parent size, population size, and step size. $\hat{\mathbf{C}}$, $\hat{\mathbf{P}}$, $\hat{\mathbf{L}}$, and $\hat{\mathbf{S}}$ indicate the desired COM, linear momentum, angular momentum, and the COP.

Problem	Objective function	Constraints ($c_1 \mid c_2 \mid$ ...)	μ	λ	σ	Domain	Feasible region	Noise	CMA-ES (evals/total time)	CMA-C (evals/total time)
Toy1	$f(x,y) = 0.1((x-3.9)^2+(y-3.9)^2)$	$(x>4$ *or* $y>4) \mid x+y<7.5$	4	8	5	$[-5,5]^2$	Convex	Asymmetric Random	663.2 / 8ms	144.8 / 70ms
Toy2	$f(x,y) = 0.1((x-3.9)^2+(y-3.9)^2)$	$(x>4$ *or* $y>4) \mid x+y<7.5$	4	8	5	$[-5,5]^2$	Convex	Symmetric Undulation	1026.4 / 9ms	229.6 / 116ms
Toy3	$f(x,y) = 0.1((x-3.1)^2+(y-3.6)^2)$	$(x>4$ *or* $y>4) \mid (x<3$ or $y<3$ or $(x<3.5$ and $y<3.5))$	4	8	5	$[-5,5]^2$	Non convex	Symmetric Undulation	1027.2 / 10ms	205.2 / 107ms
Leaning	$\|\mathbf{C}(\mathbf{q}_f)-\hat{\mathbf{C}}\|^2$	Fall forward | Fall backward | Timeout	5	10	1	$[-1,1]^4$	-	-	126.25 / 148s	23.75 / 32s
Thrusting	$\|\mathbf{P}(\mathbf{q}_f)-\hat{\mathbf{P}}\|^2 + \|\mathbf{L}(\mathbf{q}_f)-\hat{\mathbf{L}}\|^2$	Fall forward | Fall backward | Negative angular momentum	5	10	1	$[-1,1]^3$	-	-	101.25 / 15s	77.5 / 12s
Landing	$\|\mathbf{C}(\mathbf{q}_f)-\hat{\mathbf{C}}\|^2 + \|\mathbf{S}(\mathbf{q}_f)-\hat{\mathbf{S}}\|^2$	Fall forward | Fall backward | Invalid contact	5	10	1	$[-1,1]^4$	-	-	105.0 / 210s	22.5 / 44s

In addition, we propose a stochastic sampling scheme to address the general issue due to insufficient samples at the beginning of the optimization. This scheme accepts a sample with the probability of a sigmoid function $\frac{1}{1+e^{-\alpha x}}$, where x is the signed distance from the sample to the boundary. With a smaller α, this stochastic scheme is more conservative about accepting and rejecting samples. As more samples are available and the SVM boundary becomes more accurate, we can increase α to have a harder rule. In our experiments, we use $\alpha = 4$ throughout the entire optimization.

The second issue regards the tuning of the kernel parameters. We chose a frequently used kernel, Gaussian radial basis function (RBF), in our implementation.

$$k(\mathbf{x}_i, \mathbf{x}_j) = exp(-\gamma \|\mathbf{x}_i - \mathbf{x}_j\|^2) \tag{5}$$

Intuitively, a RBF adds a "bump" around each sample. The bump with the ideal size, determined by γ, should overlap with a few neighboring samples. If γ is too large, the trained SVMs will overfit the data. If it is too small, the classification will have very low accuracy. Consequently, tuning γ requires the knowledge of the current sample distribution.

Fortunately, we know the exact sample distribution because all the samples are drawn from the current estimate of Gaussian distribution, $\mathbf{x} \sim N(\mathbf{m}, \sigma^2\mathbf{C})$, provided by CMA. With this information, we set γ to cover only portion of the current CMA gaussian distribution, using a simple rule:

$$\gamma = 1/2(k\sigma)^2 \tag{6}$$

where k determines the proportion of the radial basis function with respect to the sample distribution. We use $k = 0.1$ for all the experiments.

5.2 Analysis on Toy Problems

In addition to the real problems on motion control, we designed three 2D problems to evaluate and visualize the results of CMA-C. We found that CMA-C outperforms CMA-ES significantly when the problem has a large portion of infeasible region or when the infeasible region is highly nonlinear or discontinuous. In other words, if the optimizer is more likely to be "trapped" in the infeasible region, CMA-C has a better chance to "escape" and converge at a feasible and optimal solution.

The first two toy problems have convex feasible regions, while the third one has a non-convex feasible region (Figure 5, Table III). To mimic the nonlinearity and discontinuity in the objective function of the real problems, we added random noise to the objective function of the toy problems. For the first toy problem, we add different levels of random noise in feasible and infeasible regions as follows:

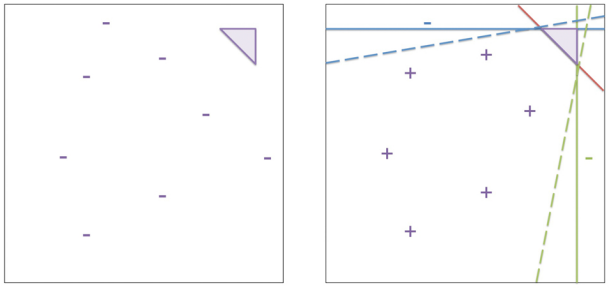

Fig. 4. A comparison of a single SVM and multiple SVMs. Left: Using a single SVM to represent the feasible region (purple triangle), the SVM cannot be activated after eight samples, due to the lack of positive samples. Right: If the feasible regions is represented by the intersection of three constraints, each of which is approximated by an individual SVM, eight samples are sufficient to active two SVMs (shown in green and blue). Dashed lines indicate the current SVM approximation of constraints.

$$f_{toy}(x,y) = \begin{cases} f(x,y) + k^f_{toy} \cdot (rand(0,1)) & \text{if } \forall i, c_i = 0 \\ f(x,y) + k^{inf}_{toy} \cdot (rand(0,1)) & \text{otherwise} \end{cases} \tag{7}$$

where $k^f_{toy} = 0.05$ and $k^{inf}_{toy} = 2.0$. In the second and third problems, we add same sinusoidal noise in both feasible and infeasible regions:

$$\begin{aligned} f_{toy}(x,y) = & f(x,y) \\ & + k_{toy} \cdot |sin(\omega_{toy} \cdot (x+y)) + sin(\omega_{toy} \cdot (x-y))| \end{aligned} \tag{8}$$

where the magnitude and frequency of the sinusoidal function $k_{toy} = 2.0$ and $\omega_{toy} = 10.0$.

Table IV. CMA-C on the toy problem I (Table III) with various ratios of the infeasible area to the feasible area. All other conditions are the same.

Area Ratio	CMA-ES (evals)	CMA-C (evals)	Performance gain
50	237.2	153.2	155%
200	663.2	144.8	458%
800	1343.6	200.0	672%

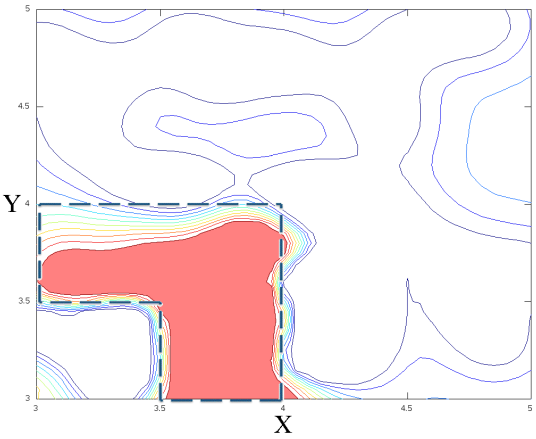

Fig. 5. Contour of the trained SVMs for the second toy problem (Table III). The feasible region classified by SVMs is filled with red. The ground truth feasible region is outlined by the dashed lines. For clarity, the figure is zoomed into the region from $[-5,5]^2$ to $[3,5]^2$.

Table V. CMA-C on the toy problem II (Table III) with various magnitude and frequency of noise. All other conditions are the same.

k_{toy}	ω_{toy}	CMA-ES (evals)	CMA-C (evals)	Performance gain
1.0	10.0	638.8	195.2	327%
2.0	10.0	1026.4	229.6	447%
4.0	10.0	1820.4	309.2	589%
1.0	20.0	735.2	220.0	334%
2.0	20.0	1079.6	272.0	397%
4.0	20.0	1574.8	348.0	452%

We compared the performance of standard CMA-ES and CMA-C on each problem by evaluating 20 times with different initial seeds. For a problem with costly sample evaluation routines, such as the controller design problem focused in this paper, CMA-C significantly reduces the total computation time. However, if the sample evaluation time is negligible, such as our toy examples, reducing the number of evaluations does not speed up the total computation time. In fact, CMA-C can be slower than CMA-ES due to the additional computation on constructing SVMs.

To further evaluate the performance of CMA-C, we conducted two sets of analyses by varying the ratio of the infeasible area to the feasible area (problem I) and the level of noise (problem II). Both analyses were designed to add difficulty in finding feasible solutions to the optimization problem. Our results, shown in Table IV and Table V, indicate that the performance of CMA-C increases when either the magnitude of the noise in the infeasible region or the infeasible area increases, while the frequency of noise does not have a significant correlation to the performance gain.

In addition, we validated the approximated feasible regions by visualizing the difference between constructed SVM and the ground truth (Figure 5). The results show that the non-convex SVM boundary in the second problems match the ground truth well. The only difference resides in the upper left corner of the L-shape feasible region in the second problem.

5.3 Analysis on Real Problems

We also evaluated the performance of CMA-C on the problems of controller design (Table III). On average, CMA-C performed five times faster than CMA-ES to achieve the same optimal value. In some cases, CMA-C reached desired objective value, while CMA-ES got stuck in the local minima. The advantage of CMA-C is even more prominent when solving a series of optimizations with gradually introduced constraints, because CMA-C can simply overlap the new SVM with previously constructed ones. However, when the infeasible region is relatively small, the difference in performance between CMA-C and CMA-ES is not obvious. For example, thrusting controller does not fully exploit the advantages of CMA-C due to its relatively small infeasible region.

6. PARAMETERIZATION AND CONCATENATION

Once a controller is developed, generalizing it to a family of similar controllers can be done easily by parameterizing the task objective function (Equation (3)):

$$f(\mathcal{P},\mathbf{u}) = \sum_{i=1}^{n} \|h_i(\mathbf{q}_f) - \hat{h}_i(\mathbf{q}_f,\mathbf{u})\| \tag{9}$$

where $\mathbf{u}$ is the varying task parameter among the parameterized controllers. For example, if $\mathbf{u}$ represents the forward leaning angle and h_i evaluates the center of mass position at the final state, we can produce a family of new targets for the center of mass by varying $\mathbf{u}$ in $\hat{h}_i(\mathbf{q}_f,\mathbf{u})$. Because all the controllers share the same constraints, the SVMs built for the original controller can be reused. As a result, optimizing a family of parameterized controllers can be done efficiently without additional user effort.

Our framework also supports concatenation of controllers by utilizing the parameterized controllers. Given two controller A and B, the naive way to optimize both controllers is putting them together in one big problem and optimizing control parameters $\mathcal{P}_A$ and $\mathcal{P}_B$ simultaneously. To resolve the issue of increasing dimensionality, we first parameterize A with the task parameter $\mathbf{u}$ to generate a family of controllers $\mathcal{A}$. When optimizing B, we include $\mathbf{u}$ as a free variable along with other rig parameters for B. The simulated motions depend on $\mathbf{u}$ because different $\mathbf{u}$ results in different initial states of the simulation. Once the optimizer obtains an optimal value $\mathbf{u}^*$, we choose one controller whose task parameter is the closest to $\mathbf{u}^*$ from $\mathcal{A}$. Finally we concatenate the chosen controller and B to generate a longer motion sequence.

7. RESULTS

We developed a few different Parkour movements to demonstrate the generality of our framework. All the results shown in the video were produced on a single core of 3.20GHz CPU. Our program runs at 1000 frames per second with the time step of 0.5 millisecond. We used RTQL8 [RTQL8 2012], an open-source simulator for multibody dynamics based on generalized coordinates. The character has 33 degrees of freedom. Except for the first six degrees of freedom that describe the global translation and orientation, all other degrees of freedom can be actuated.

Table VI. Instructions used to train a precision jump.

Phase	Instruction
Leaning	MOVE downward BY 0.1m
Leaning	MOVE downward BY 0.2m
Leaning	head IN FRONT OF root
Thrusting	SPEED near $[1.2, 2.4, 0.0]^T$
Thrusting	SLOWDOWN 20%
Thrusting	spin about z FORWARD
Airborne	PLACE feet NEAR $[0.8, 0.0, 0.0]^T$
Airborne	PLACE COM NEAR $[0.7, 0.4, 0.0]^T$
Airborne	MOVE upward BY 0.1m
Landing	PLACE COM NEAR $[0.7, 0.5, 0.0]^T$
Landing	BALANCE
All	RELAX arms BY 30%
Leaning	FLEX elbows BY 20°
Airborne	FLEX elbows BY 80°

7.1 User Input

Our system requires the user to break down a complex motion into phases and provide an initial controller for each phase. Determining the phases of a motion is at the user's discretion. In our experiments, we simply used the timing of contact change to determine phases. Similarly, the initial controllers do not have significant impact on the final controllers. All the initial controllers we used in our experiments were simple PD controllers with the same gains and damping coefficients. The only goal of each controller was to track a single target pose roughly defined by the user (Figure 6).

During the training process, the user needs to provide instructions iteratively to improve the motor skill of the character. However, we observed that the character could successfully learn a motor skill, even when the instructions were clearly not optimal. For example, in the scenario of training a precision jump (Table VI), the user gave repetitive commands using "MOVE downward BY" instruction to adjust the leaning angle of the character. For more complex cases, we believe that the user's prior knowledge about the motion will become an important factor. This is also true for learning motions in real world; a better coach can diagnose problems of the movement more precisely and effectively. When the user gives conflicting instructions, such as "SPEEDUP 200%" and "TRANSLATE torso backward", the optimizer will yield a solution which tries to achieve both conflicting goals, but the resulting motion could be undesirable or unpredictable.

7.2 Training Dynamic Skills

Precision Jump. A precision jump is a jumping motion that lands on a narrow target, such as the top of a wall or a rail. The precise distance and the smaller target require more accurate and coordinated control. We broke the entire sequence into four phases based on the contact state: leaning, thrusting, airborne, and landing. We trained each phase separately and applied parameterization and concatenation technique to sequence them into one controller for precision jump. The final state of the motion satisfies the balance condition, which limits the ground projection of center of mass within the suppor polygon and the velocity of center of mass to near zero.

The initial controller for each phase was a simple PD controller tracking a roughly designed pose (Figure 6). Because of the lack of control on the global state, the initial controller resulted in falling motion immediately. In all of our examples, we designed initial controllers in the same fashion.

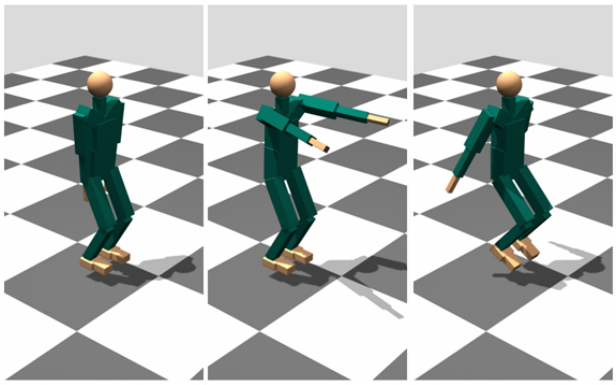

Fig. 6. We use these three target poses for all the initial controllers, except for the rolling phase of drop-and-roll.

Training all four phases took 14 instructions in total. We listed all the instructions in Table VI. For each phase, the average number of control rigs used was 3.25, which resulted in 4.25 rig parameters to optimize. The optimization time for each phase on average took two minutes. The task parameters we used for parameterizing four phases are the leaning angle, thrusting direction, and airborne traveling distance. The average time spent on parameterization of one phase is 30 minutes.

During the coaching stages, we first gave instructions to guide global motion so the character can successfully perform the jump without falling. Later, we added instructions to modify styles on the upper body. The instructions we used might not be the most effective ones because we are not experts in Parkour. For example, we used a few consecutive instructions to lower the center of mass, which could have been done in one instruction. Similarly, we instructed the character to flex the elbows in the leaning phase and later added the same style in the airborne phase. Although the total number of instructions can be reduced, our goal here is to demonstrate how this framework is used by a non-expert who tends to give imprecise and incremental instructions based on the feedback from the trainee.

Turnaround Jump. A turnaround jump requires the performer to jump in place while turning in a precise angle. Although it consists of the same four phases as a precision jump, the difference is that it involves asymmetric motion and the control of angular momentum.

In our experiment, training all phases of a turnaround jump required nine instructions. For each phase, the average number of control rigs was 3.75 and the average number of rig parameters was 4.75. The optimization time for each phase was on average one minute. We parameterized the thrusting angular momentum, which took 12 minutes to complete.

We found that the coaching skill of the human user can also improve by using this framework. Because of our previous experience in coaching a precision jump, we used fewer instructions to train a turnaround jump. We also gained better insight on developing more natural landing controller. That is, we instructed the character to raise its center of mass at landing so that the character had more room to absorb the impact.

Monkey Vault. A monkey vault is a basic Parkour movement for getting over obstacle without slowing down the motion (Figure 7). The performer approches the obstacle with squat position and uses the hands to reach for the vault. As the performer jumps in the air,

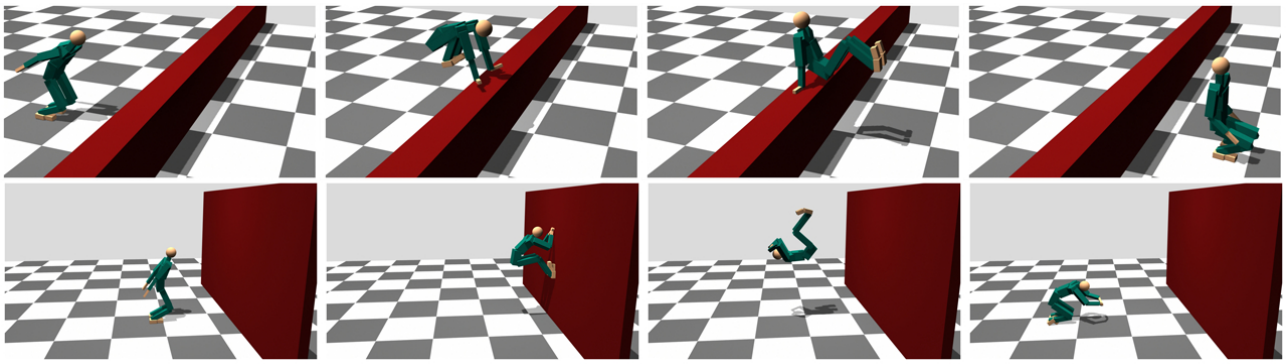

Fig. 7. Monkey vault and wall-backflip.

s/he leans forward and tucks the legs against the upper body. Since the hands are placed wider than the shoulder width, the legs can pass through in between the arms. We broke the vaulting motion into six phases: leaning, thrusting, airborne, pushing, extending, and landing.

Training all six phases of a monkey vault took 26 instructions. For each phase, the average number of control rigs was 2.33 and the average number of rig parameters was 3.66. The optimization time for each phase was on average three minutes. We parameterized the monkey vault controller by its leaning angle, thrusting direction, pushing direction, and extending velocity. The computation time for parameterization was 40 minutes for each phase.

Training a monkey vault was the most challenging task among all our experiments because our intuition of monkey vault was limited. For example, it was not clear to us when the character should accelerate or which direction the character should push. The instructions we used were more back-and-forth and repetitive due to our unfamiliarity to this motor skill.

Drop-and-roll. The purpose of rolling in Parkour is to protect joints from the landing impact. We designed a rolling controller from a standing position and later showed that it can be concatenated to a precision jump or a monkey vault to complete a drop-and-roll motion. We broke the rolling motion to three phases: leaning, thrusting, and follow-through.

Training the entire rolling motion required eight instructions. For each phase, the average number of control rigs was 2.66 and the average number of rig parameters was 4.66. The optimization time for each phase was on average 1 minute. We parameterized the rolling controller by the leaning angle and the angular momentum at the thrusting phase. The computation time for parameterization was about ten minutes for each phase.

Developing a drop-and-roll controller is relatively easy because it has only three phases and the follow-through phase is very passive. As long as the angular momentum is sufficient, the character naturally goes into rolling motion.

Wall-backflip. A wall-backflip is a combination of a wall-run and a backflip (Figure 7). The performer jumps toward the wall, kicks the wall at contact, flips backward in the air, and lands on both feet. We broke the task into six phases: leaning, thrusting, airborne, kicking, flipping, and landing.

Training all six phases require 18 instructions. For each phase, the average number of control rigs was 2.66 and the average number of rig parameters was 4.00. The optimization time for each phase was on average two minutes. We parameterized the controller by the angular momentum at the kicking phase. The computation time for parameterization was about 30 minutes for each phase.

The experience of coaching monkey vault greatly helped us to train a wall-backflip. They share similar phases, but the main difference is that the character needs to maximize the backward angular momentum at the kicking phase. Without optimization, the direction of the kicking force would have been a difficult parameter to tune because it must generate sufficient angular momentum without causing slipping contact.

7.3 Limitations

Our controllers are able to withstand some perturbations. For example, the same precision jump controller trained for landing on the floor can be applied to landing on a rail. However, most controllers become unstable when additional push forces are applied to the character. We suspect that the instability is due to the feed-forward nature of the virtual force control rig.

Using contact states to break a task into smaller phases is a good strategy for the examples we demonstrated, but it is not sufficient for more complex and timing-sensitive motor skills, such as tic-tac or wall-run. These movements switch controllers not only based on contact states, but also on character's pose, speed, or spatial relation to the environment. One possible solution is to design more sophisticated control rigs which include time-varying rig parameters.

In the absence of a running controller, we were not able to generate some motor skills which require a high initial forward momentum to carry out the motion smoothly. We believe that the monkey vault motion can be largely improved if we concatenate it with an adequate running controller.

Although the performance of CMA-C on our Parkour problems is five times faster than the standard CMA on average, the large variance in performance gain (150% to 650%) indicates that further evaluation on a broader set of benchmark problems is needed. Our toy problems do not provide comprehensive analyses on CMA-C because they made specific assumptions about the shape of the cost functions, such as asymmetric random noise or symmetric undulation. These assumptions may not reflect the true landscape of the cost functions in the Parkour problems. We conjecture that the success of CMA-C on the Parkour problems is due to the large infeasible regions and multiple local minima in the cost functions, which may cause standard CMA to converge slower.

8. CONCLUSION

We present a novel framework for controller design using human coaching and learning techniques. The framework takes in a black-box controller and improves it through an iterative learning process of coaching and practicing. The user can directly give high-level instructions to the virtual character using control rigs while the character can efficiently optimize its motor skill using a new sampling-based optimization method, CMA-C. Once a controller is developed, parameterizing it to a family of similar controllers for concatenation can be done without additional effort from the user.

In this work, we demonstrated that developing complex motor controllers does not need any motion trajectories. However, if the user wishes to utilize motion examples to train the virtual character, instead of verbal instructions, our framework can be adapted by including a control rig that modulates the reference trajectories, similar to the sampling approach proposed by Liu *et al.* [Liu et al. 2010; Liu et al. 2012].

Our current implementation requires the user to construct instructions as a script according to the template grammars. One possible future direction is to augment our framework with a natural language processor so the user can use more colloquial commands to train the character, such as "Lower your body a bit more", instead of "MOVE COM down BY 0.2m". In addition, we would like to explore Kinet-like sensors to enable the possibility of "teaching by demonstration". Therefore, one possible future direction is to augment our framework with two different types of interfaces: a natural language processor and a Kinect-like sensor. These two interfaces will allow the user to train the character by describing the motion in human language while demonstrating the movement using his/her own body.

REFERENCES

BORNO, M. A., DE LASA, M., AND HERTZMANN, A. 2013. Trajectory optimization for full-body movements with complex contacts. *IEEE Transactions on Visualization and Computer Graphics 99*.

COROS, S., BEAUDOIN, P., AND VAN DE PANNE, M. 2010. Generalized biped walking control. *ACM Trans. Graph. 29*, 130:1–130:9.

COROS, S., KARPATHY, A., JONES, B., REVERET, L., AND VAN DE PANNE, M. 2011. Locomotion skills for simulated quadrupeds. 1–11.

DA SILVA, M., DURAND, F., AND POPOVIC, J. 2009. Linear bellman combination for control of character animation. *ACM Trans. Graph 28,* 3.

FALOUTSOS, P., VAN DE PANNE, M., AND TERZOPOULOS, D. 2001. Composable controllers for physics-based character animation. In *SIGGRAPH*. 251–260.

FANG, A. C. AND POLLARD, N. S. 2003. Efficient synthesis of physically valid human motion. *ACM Trans. on Graphics (SIGGRAPH)*, 417–426.

HA, S., YE, Y., AND LIU, C. K. 2012. Falling and landing motion control for character animation. *ACM Trans. Graph 31,* 6, 155.

HANSEN, N. AND KERN, S. 2004. Evaluating the CMA evolution strategy on multimodal test functions. In *Parallel Problem Solving from Nature - PPSN VIII*. LNCS, vol. 3242. 282–291.

HODGINS, J. K., WOOTEN, W. L., BROGAN, D. C., AND O'BRIEN, J. F. 1995. Animating human athletics. In *SIGGRAPH*. 71–78.

LASA, M. D., MORDATCH, I., AND HERTZMANN, A. 2010. Feature-based locomotion controllers. *ACM Transactions on Graphics (TOG) 29*.

LEE, Y., KIM, S., AND LEE, J. 2010. Data-driven biped control. *ACM Trans. on Graphics (SIGGRAPH) 29,* 4 (July).

LIU, C. K. AND POPOVIĆ, Z. 2002. Synthesis of complex dynamic character motion from simple animations. *ACM Trans. on Graphics (SIGGRAPH) 21,* 3 (July), 408–416.

LIU, L., YIN, K., VAN DE PANNE, M., AND GUO, B. 2012. Terrain runner: control, parameterization, composition, and planning for highly dynamic motions. *ACM Trans. Graph 31,* 6, 154.

LIU, L., YIN, K., VAN DE PANNE, M., SHAO, T., AND XU, W. 2010. Sampling-based contact-rich motion control. *ACM Transactions on Graphics (TOG) 29,* 4, 128.

MORDATCH, I., DE LASA, M., AND HERTZMANN, A. 2010. Robust physics-based locomotion using low-dimensional planning. *ACM Trans. Graph. 29*, 71:1–71:8.

MUICO, U., POPOVIC, J., AND POPOVIC, Z. 2011. Composite control of physically simulated characters. *ACM Trans. Graph 30,* 3, 16.

POPOVIĆ, Z. AND WITKIN, A. 1999. Physically based motion transformation. In *SIGGRAPH*. 11–20.

RTQL8. 2012. *RTQL8, http:bitbucket.org/karenliu/rtql8.*

SAFONOVA, A., HODGINS, J. K., AND POLLARD, N. S. 2004. Synthesizing physically realistic human motion in low-dimensinal, behavior-specific spaces. *ACM Trans. on Graphics (SIGGRAPH) 23,* 3, 514–521.

SOK, K. W., KIM, M., AND LEE, J. 2007. Simulating biped behaviors from human motion data. *ACM Trans. Graph 26,* 3.

SUNADA, C., ARGAEZ, D., DUBOWSKY, S., AND MAVROIDIS, C. 1994. A coordinated jacobian transpose control for mobile multi-limbed robotic systems. In *ICRA*. 1910–1915.

WANG, J. M., FLEET, D. J., AND HERTZMANN, A. 2009. Optimizing walking controllers. *ACM Trans. Graph 28,* 5.

WANG, J. M., FLEET, D. J., AND HERTZMANN, A. 2010. Optimizing walking controllers for uncertain inputs and environments. *ACM Trans. Graph 29,* 4.

WANG, J. M., HAMNER, S. R., DELP, S. L., AND KOLTUN, V. 2012. Optimizing locomotion controllers using biologically-based actuators and objectives. *ACM Trans. Graph 31,* 4, 25.

WOOTEN, W. L. 1998. Simulation of leaping, tumbling, landing, and balancing humans. Ph.D. thesis, Georgia Institute of Technology.

WU, J.-C. AND POPOVIĆ, Z. 2010. Terrain-adaptive bipedal locomotion control. *ACM Trans. Graph. 29*, 72:1–72:10.

YIN, K., COROS, S., BEAUDOIN, P., AND VAN DE PANNE, M. 2008. Continuation methods for adapting simulated skills. *ACM Trans. Graph 27,* 3.

YIN, K., LOKEN, K., AND VAN DE PANNE, M. 2007. Simbicon: simple biped locomotion control. In *SIGGRAPH*. 105.

Received September 2008; accepted March 2009