Aerobatics Control of Flying Creatures via Self-Regulated Learning

JUNGDAM WON, Seoul National University, South Korea
JUNGNAM PARK, Seoul National University, South Korea
JEHEE LEE*, Seoul National University, South Korea

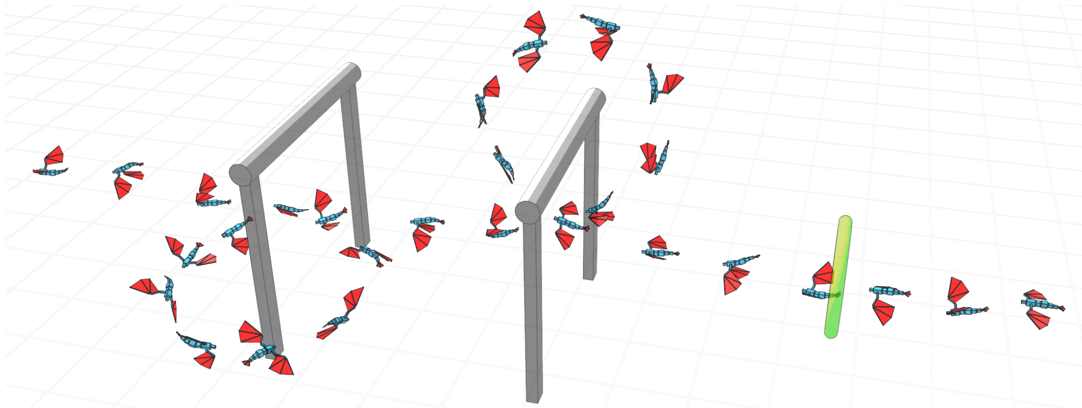

Fig. 1. The imaginary dragon learned to perform aerobatic maneuvers. The dragon is physically simulated in realtime and interactively controllable.

Flying creatures in animated films often perform highly dynamic aerobatic maneuvers, which require their extreme of exercise capacity and skillful control. Designing physics-based controllers (a.k.a., control policies) for aerobatic maneuvers is very challenging because dynamic states remain in unstable equilibrium most of the time during aerobatics. Recently, Deep Reinforcement Learning (DRL) has shown its potential in constructing physics-based controllers. In this paper, we present a new concept, *Self-Regulated Learning (SRL)*, which is combined with DRL to address the aerobatics control problem. The key idea of SRL is to allow the agent to take control over its own learning using an additional self-regulation policy. The policy allows the agent to regulate its goals according to the capability of the current control policy. The control and self-regulation policies are learned jointly along the progress of learning. Self-regulated learning can be viewed as building its own curriculum and seeking compromise on the goals. The effectiveness of our method is demonstrated with physically-simulated creatures performing aerobatic skills of sharp turning, rapid winding, rolling, soaring, and diving.

CCS Concepts: • **Computing methodologies → Animation**; *Physical simulation*; *Reinforcement learning*; *Neural networks*;

Additional Key Words and Phrases: Character Animation, Physics Simulation, Physics-based Control, Reinforcement Learning, Deep Learning, Neural Network, Flying Creature

ACM Reference Format:
Jungdam Won, Jungnam Park, and Jehee Lee. 2018. Aerobatics Control of Flying Creatures via Self-Regulated Learning. *ACM Trans. Graph.* 37, 6, Article 181 (November 2018), 10 pages. https://doi.org/10.1145/3272127.3275023

*Corresponding author

Authors´ addresses: Jungdam Won; Jungnam Park; Jehee Lee, Department of Computer Science and Engineering, Seoul National University.

© 2018 Association for Computing Machinery.
This is the author's version of the work. It is posted here for your personal use. Not for redistribution. The definitive Version of Record was published in *ACM Transactions on Graphics*, https://doi.org/10.1145/3272127.3275023.

1 INTRODUCTION

In animated films, flying creatures such as birds and dragons often perform highly dynamic flight skills, called aerobatics. The skills include rapid rotation about three major axes (pitch, yaw, roll), and a sequence of skills are performed in a consecutive manner to make dramatic effects. The creature manages to complete aerobatic skills by using its extreme of exercise capacity and endurance, which make it attractive and create tension in the audience.

Designing a physics-based controller for aerobatics is very challenging because it requires extremely skillful control. The creature remains in unstable equilibrium most of the time during aerobatics. A bit of perturbation could bring the creature to unrecoverable states. Control is even more challenging when it performs a sequence of skills continuously without delay.

Recently, reinforcement learning (RL) with deep neural networks has shown its potential in constructing physics-based controllers for character animation [Liu and Hodgins 2017; Peng et al. 2016, 2017; Won et al. 2017]. Given the current state of a character, RL determines its optimal sequence of actions that maximize the expected sum of rewards, which indicate the desirability of states and actions. Defining a reward for taking an action is the primary means by which the user can influence the control policy. The reward is a succinct description of the task. The choice of the reward also affects the performance of the controller and the progress of its learning.

We consider a simple user interface that allows a user to specify a spatial trajectory. The flying creature tracks the trajectory to perform aerobatics. Existing RL methods address similar control problems quite successfully with aerial vehicles if expert demonstrations are provided or the trajectory curve includes only mild turns [Abbeel et al. 2010, 2006; Kim et al. 2004]. However, aerobatics with bird-like articulated wings requires extreme flight maneuvers and thus poses new challenges for RL approaches.

We present a new concept, *Self-Regulated Learning (SRL)*, which is combined with deep reinforcement learning (DRL) to address the aerobatics control problem. We consider a class of problems in which the main goal can be achieved by generating a sequence of subgoals and addressing each individual subgoal sequentially. We found that subgoals naïvely generated from a user-provided trajectory are often physically unrealizable. A mechanism to systematically modulate subgoals is crucial for learning aerobatic skills. The key idea of SRL is to allow the agent to take control over its own learning using an additional self-regulation policy. The policy allows the agent to regulate subgoals according to the capability of the current control policy. The generation of subgoals is closely related to the reward system of RL. The control and self-regulation policies are learned jointly along the progress of learning. Learning self-regulation can be viewed as the process of building its own curriculum or seeking compromise on the subgoals to better achieve the main goal. SRL improves the performance of a learned control policy significantly for very challenging aerobatics control problems. We will demonstrate the effectiveness of our method with physically-simulated creatures performing aerobatic maneuvers that include a combination of repeated sharp turns, rapid winding, soaring, and diving.

2 RELATED WORK

Controller design is an essential component of creating self-actuated autonomous characters in physically based animation. While computer animation research has mainly focused on simulating biped locomotion over the past few decades [Coros et al. 2010; da Silva et al. 2008a; de Lasa et al. 2010; Lee et al. 2010, 2014; Liu et al. 2016; Mordatch et al. 2012; Peng et al. 2017; Sok et al. 2007; Wang et al. 2012; Ye and Liu 2010; Yin et al. 2007], nonhuman characters have also been studied including flying creatures [Ju et al. 2013; Won et al. 2017; Wu and Popović 2003], swimming creatures [Grzeszczuk et al. 1998; Tan et al. 2011; Tu and Terzopoulos 1994], quadrupeds [Coros et al. 2011; Zhang et al. 2018], and various imaginary creatures [Barbič et al. 2009; Coros et al. 2012; Tan et al. 2012]. A diversity of control methodologies have been explored in computer graphics, including manually-crafted feedback control laws [Ha et al. 2012; Lee et al. 2010; Liu et al. 2012; Yin et al. 2007], simplified physical models (e.g., inverted pendulums) [Kwon and Hodgins 2010, 2017; Tsai et al. 2009], and data-driven physics simulation [Ju et al. 2013; Lee et al. 2010; Sok et al. 2007].

Optimality principles played an important role of popularizing optimal control theory and nonlinear/non-convex optimization methods in character animation [Al Borno et al. 2013; Barbič et al. 2009; Mordatch et al. 2012; Wang et al. 2010, 2012; Wu and Popović 2003; Ye and Liu 2010]. We can classify control policies (a.k.a., controllers) depending on how far they look ahead into their future. Immediate control policy is a direct mapping from states to actions. The action at a moment is determined based solely on the current state of the dynamic system [Ju et al. 2013; Sok et al. 2007]. The capability to look ahead and predict the future evolution is essential for balance control and acrobatic maneuvers. Recently, model predictive control has successfully been applied to simulating human behaviors [da Silva et al. 2008b; Hämäläinen et al. 2014, 2015; Han et al. 2016, 2014]. The key concept is to predict the future evolution of the dynamic system for short time horizon and optimize its control signals. Model predictive control repeats this prediction and optimization step while receding the time horizon.

Recently, Deep Reinforcement Learning (DRL) has shown its potential in simulation and control of virtual characters. DRL for continuous control (especially actor-critic framework) has advantages of both immediate control and predictive control. The control policy (actor) is a direct mapping from states to actions, while its value function (critic) predicts future rewards for the actor. Peng et al [2016] used a mixture of actor-critic experts to learn terrain-adaptive locomotion skills of planar bipeds and quadrupeds. In their subsequent work [Peng et al. 2017], they built a hierarchical network architecture for three-dimensional bipeds. The high-level network plans footsteps for a given task, while the low-level network generates immediate action signals to accomplish the goal given by the high-level network. Recently, Peng et al [2018] learned various types of legged locomotion, spin, and kick from example motion clips. Liu et al [2017] generated a collection of control fragments for a physics-based character. Each control fragment describes a motor skill suitable at a specific state of the character. Deep Q-learning schedules the activation of control fragments to adapt to external perturbation and user interactions. Won et al [2017] learned controllers for flapping flight of winged creatures using DRL equipped with evolutionary strategies, which allow rapid exploration of unseen states and actions.

While the steady locomotion of humans and animals have been explored comprehensively, studies on highly dynamic motor skills are not abundant due to the difficulty of simulation and control. Ha and his colleagues [2014; 2012] simulated Parkour skills, such as falling safely from a high position, standing and jumping on a thin bar, and wall-flipping. Liu et al [2012] demonstrated parameterized controllers for Parkour skills, which adapt to different environment conditions. Borno et al [2013] simulated break-dancing motions including handstand, handspin, and headspin via trajectory optimization. Kwon et al [2017] developed a momentum-mapped inverted pendulum model to describe generalized balancing strategies, and demonstrated the simulation of gymnastic motions such as high-jumps, handstands, and back-flips.

Self-regulated learning was partly inspired by several ideas from machine learning literature. The idea of "changeable rewards" comes from inverse reinforcement learning [Abbeel et al. 2010; Abbeel and Ng 2004; Fu et al. 2017]. Reward shaping [Ng et al. 1999] is a technique that modifies the reward function without changing the corresponding optimal policy by a specific form of transformation. The goal of SRL is different from either inverse reinforcement learning or reward shaping. SRL assumes that the reward function provided initially is not ideal for achieving the main goal and thus allows the agent to transform the reward function adaptively.

Underlying motivation of SRL is closely related to automatic curriculum generation for RL agents. The sequence of experiences the agent encounters during learning affect not only the progress of learning but also the performance of the learned policy. Held et al [2017] proposed a Generative Adversarial Network (GAN) framework for automatic goal generation. Matiisen el al [2017] proposed a Partially Observable Markov Decision Process (POMDP) formulation for curriculum generation. Sukhbaatar et al [2017] demonstrated an intrinsic motivation approach via asymmetric self-play which uses the internal reward system for the agent. Yu et al [2018] learned locomotion skills based in DRL and curriculum learning, which introduces fictional assistive force and gradually relaxes the assistance according to a scheduled curriculum.

3 ENVIRONMENT AND LEARNING

The aerodynamics of a flying creature entails complex interactions between its skeleton and wings. In our study, we use a dragon model similar to the one presented in [Won et al. 2017] except that our model has a wider range of motion at all joints. The model has an articulated skeleton of rigid bones and thin-shells attached to the bones. The skeleton consists of a trunk, two wings, and a tail. The trunk includes four spinal segments connected by revolute joints. Each wing includes humerus (upper arms), ulna (lower arms), and manus (hands). The shoulder, elbow and wrist joints are ball-and-socket, revolute and universal, respectively. The wings are airfoil-shaped to generate aerodynamic force, which is the only source of external force that enables flight. The forces are computed by the simplified aerodynamics equation and drag and lift coefficients are manually selected similar to [Wu and Popović 2003].

3.1 Aerobatics Description

An aerobatic maneuver is described by a spatial trajectory $C(\sigma) = (R(\sigma), p(\sigma), h(\sigma))$, where $\sigma \in [0, 1]$ is a progress parameter along the trajectory, $R(\sigma) \in SO(3)$ and $p(\sigma) \in R^3$ are the desired orientation and position of the trunk (the root of the skeleton), respectively, and $h(\sigma)$ is a clearance threshold. We employ a receding target model to formulate trajectory tracking control. At every moment, the creature is provided with a target $C(\sigma^*)$ and the target is cleared if $d(R, p, \sigma^*) < h(\sigma^*)$, where R and p are the current orientation and position, respectively, of the creature's trunk. The distance is defined by

$$d(R, p, \sigma) = \|log(R^{-1}R(\sigma))\|_F^2 + w_p\|p - p(\sigma)\|^2, \tag{1}$$

where w_p normalizes the scale of position coordinates. Whenever a target is cleared, σ^* increases to suggest the next target to follow. Let σ^* be the earliest target that has not been cleared yet. The aerobatic maneuver is completed if the progress reaches the end of the trajectory, $\sigma^* = 1$.

Since the linear and angular motions in aerobatics are highly-coordinated, designing a valid, realizable trajectory is a non-trivial task. Spline interpolation of key positions and orientations often generates aerodynamically-unrealizable trajectories. Therefore, we specify only a positional trajectory $p(\sigma)$ via spline interpolation and determine the other terms automatically as follows. Let $t(\sigma) = \frac{\dot{p}(\sigma)}{\|\dot{p}(\sigma)\|}$ be the unit tangent vector and $u = [0, 1, 0]$ be the up-vector (opposite to the gravity direction). The initial orientation $R(0) = [r_x^\intercal, r_y^\intercal, r_z^\intercal] \in SO(3)$ is defined by an orthogonal frame such that $r_z = t(0)$, $r_x = \frac{u \times r_z}{\|u \times r_z\|}$, and $r_y = r_z \times r_x$. The rotation along the trajectory is

$$R(\sigma) = R(\sigma - \epsilon)U\big(t(\sigma - \epsilon), t(\sigma)\big), \tag{2}$$

where ϵ is the unit progress and $U \in SO(3)$ is the minimal rotation between two vectors.

$$U(a, b) = I + [a \times b]_\times + [a \times b]_\times^2 \frac{1 - a \cdot b}{(a \times b)^\intercal (a \times b)}. \tag{3}$$

Here, $[v]_\times$ is the skew-symmetric cross-product matrix of v. The clearance threshold $h(\sigma)$ is relaxed when the trajectory changes rapidly.

$$h(\sigma) = \bar{h}(1 + w_h\|\ddot{p}(\sigma)\|) \tag{4}$$

where $\bar{h}$ is a default threshold value and w_h adjusts the degree of relaxation. The spatial trajectory thus obtained is twist-free. Twist motions can further be synthesized over the trajectory.

3.2 Reinforcement Learning

Reinforcement learning (RL) assumes a Markov decision process $(\mathcal{S}, \mathcal{A}, \mathcal{P}(\cdot,\cdot,\cdot), \mathcal{R}(\cdot,\cdot,\cdot), \gamma)$ where $\mathcal{S}$ is a set of state, $\mathcal{A}$ is a set of actions, $\mathcal{P}(s, a, s')$ is a state transition probability from state s to state s' after taking action a, $\mathcal{R}(s, a, s')$ is an immediate scalar reward after transitioning from s to s' due to action a, and $\gamma \in [0, 1)$ is a discount factor of future rewards. The goal of RL is to find the optimal policy $\pi^* : \mathcal{S} \to \mathcal{A}$ that maximizes the expectation on cumulative rewards $\eta(\pi)$.

$$\eta(\pi) = E_{s_0, a_0, \cdots}\Big[\sum_{t=0}^{\infty} \gamma^t r_t\Big] \tag{5}$$

where $s_t \sim \mathcal{P}(s_{t-1}, a_t, s_t)$, $a_t \sim \pi(s_t)$, and $r_t = \mathcal{R}(s_{t-1}, a_t, s_t)$.

We define the reward function for the receding target model such that the receding of the target is encouraged and the deviation from the trajectory is penalized.

$$\mathcal{R}(s, a, s') = \begin{cases} \sigma^*(2 - \frac{d(R,p,\sigma^*)}{d_{max}}), & \text{if } d(R, p, \sigma^*) < h(\sigma^*) \\ 0, & \text{otherwise,} \end{cases} \tag{6}$$

where d_{max} is the predefined maximum distance value that makes the reward value positive. The reward can be thought of as the sum of progress reward 1 and target reward $1 - \frac{d(R,p,\sigma^*)}{d_{max}}$, which are both weighed by σ^*.

Given the reward function, it is straightforward to adopt a DRL algorithm to solve the problem. There are many variants of DRL algorithms, including CACLA [van Hasselt and Wiering 2007], DDPG [Lillicrap et al. 2015], Evo-CACLA [Won et al. 2017], GAE [Schulman et al. 2015], and PPO [Schulman et al. 2017]. As demonstrated in the previous study [Won et al. 2017], any of the algorithms would learn control policies successfully if the trajectory is mild and the clearance threshold is large, despite that the relaxed conditions would compromise the challenge of aerobatic maneuvers. Algorithm 1 shows the base algorithm used in our experiments. We will discuss in the next section how to modify the base algorithm to adopt self-regulated learning. Self-regulated learning is a general concept that can be incorporated into any DRL algorithm for continuous control.

The core of the algorithm is the construction of value/policy functions. We build a state-action value function and a deterministic policy function. The state-action value function Q receives a state-action pair (s, a) as input and returns the expectation on cumulative rewards. The deterministic policy π takes state s as input and generates action a. Both functions are represented as deep neural networks with parameters θ_Q, θ_π. The algorithm consists of two parts. The first part of the algorithm produces experience tuples $\{e_i = (s_{i-1}, a_i, r_i, s_i)\}$ and stores them in a replay memory B (line 2–10). Action a is chosen from the current policy and perturbed with probability ρ to explore unseen actions (line 5–6). The state transition is deterministic because forward dynamics simulation is deterministic (line 7). The second part of the algorithm updates value and policy networks (line 11–22). A mini-batch of experience tuples picked from the replay memory updates the Q network by Bellman backups (line 15–17). The policy network is updated by actions that have positive temporal difference errors (line 18–20) similar to CACLA [van Hasselt and Wiering 2007].

Algorithm 1 DRL Algorithm

$Q|_{\theta_Q}$: state-action value network
$\pi|_{\theta_\pi}$: policy network
B : experience replay memory
1: **repeat**
2: $s_0 \leftarrow$ random initial state
3: **for** $i = 1, \cdots, T$ **do**
4: $a_i \leftarrow \pi(s_{i-1})$
5: **if** unif$(0, 1) \leq \rho$ **then**
6: $a_i \leftarrow a_i + \mathcal{N}(\mathbf{0}, \Sigma)$
7: $s_i \leftarrow$ StepForward(s_{i-1}, a_i)
8: $r_i \leftarrow \mathcal{R}(s_{i-1}, a_i, s_i)$
9: $e_i \leftarrow (s_{i-1}, a_i, r_i, s_i)$
10: Store e_i in B
11: $X_Q, Y_Q \leftarrow \emptyset$
12: $X_\pi, Y_\pi \leftarrow \emptyset$
13: **for** $i = 1, \cdots, N$ **do**
14: Sample an experience tuple $e = (s, a, r, s')$ from B
15: $y \leftarrow r + \gamma Q(s', \pi(s'|\theta_\pi)|\theta_Q)$
16: $X_Q \leftarrow X_Q \cup \{(s, a)\}$
17: $Y_Q \leftarrow Y_Q \cup \{y\}$
18: **if** $y - Q(s, \pi(s|\theta_\pi)|\theta_Q) > 0$ **then**
19: $X_\pi \leftarrow X_\pi \cup \{s\}$
20: $Y_\pi \leftarrow Y_\pi \cup \{a\}$
21: Update Q by (X_Q, Y_Q)
22: Update π by (X_π, Y_π)
23: **until** no improvement on the policy

The progress of the learning algorithm depends mainly on the difficulty level of the tasks. Most of the DRL algorithms are successful with easy tasks, but they either fail to converge or converge to unsatisfactory suboptimal policies with difficult tasks. Previously, two approaches have been explored to address this type of problems. The key idea of curriculum learning is to learn easy subtasks first and then increase the level of difficulty gradually [Bengio et al. 2009]. Curriculum learning suggests that we learn easy aerobatic skills first using a collection of simple trajectories and refine the control policy gradually to learn more difficult skills step-by-step. The key component is the difficulty rating of aerobatics skills associated with spatial trajectories. We found that deciding the difficulty rating of each individual trajectory is fundamentally as difficult as the aerobatics control problem itself because we have to understand what skills are required to complete the trajectory to rate its difficulty. Recently, automatic curriculum generation methods have been studied in supervised learning [Graves et al. 2017] and reinforcement learning [Held et al. 2017; Matiisen et al. 2017; Sukhbaatar et al. 2017] to avoid the effort of manually specifying difficulty levels. However, applying those methods to our aerobatics problem is not trivial.

Alternatively, there are a class of algorithms that combine trajectory optimization with policy learning [Levine and Koltun 2014; Mordatch and Todorov 2014; Won et al. 2017]. Given a target trajectory or a sequence of sparse targets, the goal of trajectory optimization is to generate either a simulated trajectory or open-loop simulation as output. Assuming that the input target trajectory is the same, optimizing the trajectory is much easier than learning the policy from a computational point of view. Therefore, the common idea in this class of the algorithms is to solve trajectory optimization first and let the simulated output trajectory guide policy learning. This idea does not help the solution of the aerobatics problem either because even state-of-the-art trajectory optimization methods equipped with non-convex optimization and receding temporal windows often fail to converge with aerobatic maneuvers. We will discuss in the next section how this challenging problem is addressed with the aid of our self-regulated learning.

4 SELF-REGULATED LEARNING

Self-regulated learning in education refers to a way of learning that learners take control of their own learning [Ormrod 2009]. The learner achieves a goal through self-regulation, which is a recursive process of generation, evaluation, and learning [?? SRL]. *Generation* is a step that learners create a few alternatives that they can choose from. *Evaluation* is a step that judges good or bad for the alternatives. *Learning* is a final step that the learners observe the degree of achievement and confirm the success or failure of the selected alternative. For example, if two sport athletes who have different exercise ability try to learn the same skill, they first make self-determined plans based on their current ability then they practice and evaluate themselves. In the learning process, the plans and evaluations for each athlete would be different due to the discrepancy of exercise ability although they learn the same skill. The key concept of SRL is that learners can decide/regulate their plans to complete the final objective without the help of a teacher or a pre-fixed curriculum.

Aerobatics learning can benefit from this concept. What if the agent (the flying creature) can self-regulate its own learning? In the algorithm outlined in the previous section, a sequence of subgoals and their associated rewards are provided by fixed rules (i.e. fixed curriculum). Our SRL consists of two key ideas. First, the agent is allowed to regulate subgoals and their associated rewards at any time step and learn their actions accordingly. Second, self-regulation policy is also learned together with its control policy in the framework of reinforcement learning. The agent learns how to regulate

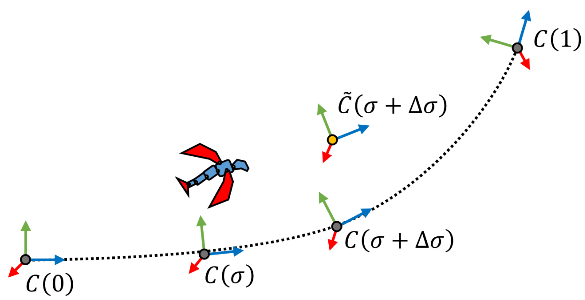

Fig. 2. The receding target on the spatial trajectory.

its reward and how to optimize its action with respect to the reward simultaneously while building its own curriculum.

4.1 Self-Regulated DRL

State $s = (s_d, \sigma, s_s)$ in our system consists of dynamic state s_d, progress parameter σ, and sensory state s_s. The dynamic state $s_d = (q, \dot{q})$ includes the generalized coordinates $q = (q_1, \cdots, q_D)$ and the generalized velocity $\dot{q} = (\dot{q}_1, \cdots, \dot{q}_D)$ of the creature's body and D is its degrees of freedom. Note that the position of the root joint is not included because the control strategy is independent of the absolute position in the world reference coordinate system. σ parameterizes the degree of completion of the trajectory tracking task. The sensory state $s_s = \left(C(\sigma), C(\sigma+\epsilon), \cdots, C(\sigma+w\epsilon)\right)$ is a part of the trajectory of window size w, where ϵ is the unit progress. $C(\sigma)$ is the subgoal the agent is tracking at σ (see Figure 2).

Action $a = (\hat{a}, \tilde{a})$ consists of dynamic action $\hat{a}$ and self-regulation $\tilde{a}$. The dynamic action $\hat{a} = (\hat{q}, \tau)$ generates joint torques for the dynamics simulation by using Proportional-Derivative (PD) servos, where $\hat{q}$ is the target pose and τ is its duration. Self-regulation $\tilde{a} = (\Delta\sigma, \Delta R, \Delta p, \Delta h)$ changes the subgoal to adjust the progress, the target orientation, position, and the clearance threshold (see Figure 2). The self-regulated subgoal $\tilde{C}(\tilde{\sigma}) = \left(\tilde{R}(\tilde{\sigma}), \tilde{p}(\tilde{\sigma}), \tilde{d}(\tilde{\sigma})\right)$ is

$$\begin{aligned} \tilde{R}(\tilde{\sigma}) &= R(\tilde{\sigma})\Delta R, \\ \tilde{p}(\tilde{\sigma}) &= p(\tilde{\sigma}) + R(\tilde{\sigma})\Delta p, \\ \tilde{h}(\tilde{\sigma}) &= h(\tilde{\sigma}) + \Delta h, \end{aligned} \tag{7}$$

where $\tilde{\sigma} = \sigma + \Delta\sigma$. Reward $\mathcal{R}(s, a, s')$ in Equation(6) is now computed with respect to the self-regulated subgoal. We also add a regularization term $(\tilde{a} - a_o)^\intercal W(\tilde{a} - a_o)$, where a_o is the default (no self-regulation) action and W is a diagonal weight matrix. In our experiments, default action $a_o = (\Delta\sigma_o, \Delta R_o, \Delta p_o, \Delta h_o)$ is defined by $\Delta R_o = I_3$, $\Delta p_o = (0,0,0)$, $\Delta h_o = 0$, and $\Delta\sigma_o$ is a positive value that matches the average flight speed. The progress of tracking is updated for the next state s' by $\tilde{\sigma}$ after taking action a. Intuitively speaking, the agent senses its body state s_d and a part of the trajectory (σ, s_s), and decides how to act and how to regulate the current subgoal $C(\sigma)$ simultaneously while avoiding excessive deviation from the input trajectory.

Incorporating self-regulation into the base DRL algorithm is straightforward. We extend the definition of actions to include self-regulation and replace line 7-8 of Algorithm 1 with self-regulated state transition and rewarding in line 2-7 of Algorithm 2. Note that dynamic action $\hat{a}$ and self-regulation $\tilde{a}$ are learned simultaneously in the single reinforcement learning framework. Dynamic action $\hat{a}$ is learned to follow the guide of self-regulation $\tilde{a}$. On the other hand, self-regulation is learned while taking the current ability (current policy) into account. Therefore, the control policy and the self-regulation policy reinforce each other to evolve together as the learning progresses.

The learning can also be interpreted in the reward point of view. The largest reward value can be achieved when the agent achieves all subgoals exactly without any modification. However, this ideal results cannot be attained if the user-provided trajectory is physically unrealizable or the maneuvers are beyond the exercise capability of the agent. In such a case, the agent with self-regulation is able to seek a point of compromise within its capability by modulating the subgoal, whereas the agent without self-regulation keeps trying to address the original subgoal. This makes a big difference when the agent performs challenging tasks such as aerobatics. The agent with self-regulation would have a better chance of completing the task successfully because the progression of learning can be facilitated by relaxed subgoals.

Algorithm 2 Step forward with self-regulation

s : the current state
$a = (\hat{a}, \tilde{a})$: the action determined by the current policy
$\tilde{a} = (\Delta\sigma, \Delta R, \Delta p, \Delta h)$: a self-regulation part of the action

1: **procedure** StepForwardWithSRL(s, a)
2: $\quad\sigma \leftarrow \sigma + \Delta\sigma$
3: $\quad\tilde{R} \leftarrow R(\sigma)\Delta R$
4: $\quad\tilde{p} \leftarrow p(\sigma) + R(\sigma)\Delta p$
5: $\quad\tilde{h} \leftarrow h(\sigma) + \Delta h$
6: $\quad s' \leftarrow$ Dynamic simulation with $\hat{a}$
7: $\quad r \leftarrow$ Compute $\mathcal{R}(s, a, s')$ with progress σ and target $(\tilde{R}, \tilde{p}, \tilde{d})$

5 RESULTS

We implemented our algorithm in Python. DART [Dart 2012] was used for the simulation of articulated rigid body dynamics, and TensorFlow [TensorFlow 2015] was used for the learning and evaluation of deep neural networks. All computations were run on CPU (Intel Xeon E5-2687W-v4) rather than GPU since dynamics simulation was a computational bottleneck. Acceleration of neural network operations on GPU does not help much.

All parameters for the dynamics simulation and neural network learning are summarized in Table 1. We used the same values for all experiments. The action values were first normalized by their min/max values and the exploration noise Σ is set to 5% of the normalized range. Starting from the initial exploration probability ρ, we linearly decreased the probability by 20% of its value until 3 million training tuples were generated. In the generation of training tuples, we re-initialize the environment whenever the task is completed,

Table 1. Simulation and learning parameters

Simulation time step	0.001
Control time step	≈ 0.2
Policy learning rate (π)	0.0001
Value learning rate (Q)	0.001
Discount factor (γ)	0.95
Exploration probability (ρ)	0.5
Exploration noise (Σ)	$0.05I$
Maximum time horizon (sec)	50
Action range (normalized)	± 10
State range (normalized)	± 10
w_p	0.005
w_h	0.001
$\bar{h}$	20.0
d_{max}	3.0
W	0.02

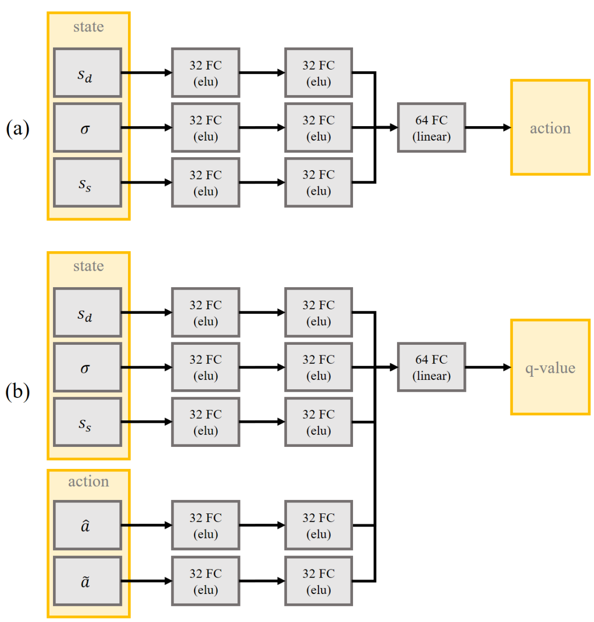

Fig. 3. The structure of deep neural networks for (a) the policy, (b) the state-action value.

the simulation reaches the maximum time horizon, or no reward is gained for a certain duration (2 seconds in our experiments).

Figure 3 illustrates deep neural networks used in our experiments. All internal layers are 32 dimensional fully connected layers with *elu* units and the final layers are 64 dimensional fully connected layers with *linear* units. Note that tracking progress σ is merely a single scalar value, but is connected forward to 32 fully connected layers. The high-dimensional inputs s_d and s_s are equally connected to 32 fully connected layers. We designed the network connectivity based on the importance of input parameters. Since σ plays an important role in learning, we separated the parameter out and made its own sub-group. Consequently, the tracking progress is weighed as much as the dynamic state and the sensory state in learning. Similarly, dynamic action $\hat{a}$ and self-regulation $\tilde{a}$ are equally weighed in our design principles because they are connected to subnetworks of the same size.

5.1 Aerobatic Maneuvers

Our self-regulated DRL learned a variety of aerobatic maneuvers ranging from simple, easy-to-learn tasks to complex, extreme tasks (see Figure 4). We categorized the tasks into beginner, intermediate, and expert levels, and learn a control policy for each individual task. The beginner level includes zero to one rapid turn. The intermediate level includes one or two rapid turns possibly about two orthogonal axes. The expert level exhibits a combination of multiple turns about all axes, soaring, diving, and rolling. The learning process took 3 to 7, 10 to 24, and 24 to 48 hours for the beginner, intermediate, and expert levels, respectively.

Beginner Level. *Straight* has no turn and thus requires only a basic locomotion skill. *X-turn* involves a 360-degree rapid turn about the X-axis (pitch direction) and the radius of the turn is only 2 times longer than the body length of the creature. The creature has to turn about the axis quickly to complete the maneuver. *Y-turn* about the vertical axis (yaw direction) is even more challenging than *X-turn*, since it requires asymmetric actions and balance maintenance about the roll direction. The radius of *Y-turn* is only 1.5 times wider than the wingspan. Smaller radius makes the maneuver even more challenging.

Intermediate Level. All maneuvers (*Double X-turn*, *Ribbon*, and *XY-turn*) in the intermediate level consist of two consecutive turns, requiring preparatory control before starting the second turn. The axes of the first and the second turn may or may not coincide with each other. The axes of *Double X-turn* are aligned, but shifted. The axes of *Ribbon* are parallel, but the trajectory winds in opposite directions. The axes of *XY-turn* are orthogonal to each other. Our learning method was able to cope with all three cases.

Expert Level. *Z-turn* involves a 360-degree turn about the Z-axis (roll direction). Although it has only one turn, it is classified as expert level because the actions are highly asymmetric and unstable. *Infinite X-turn* includes five consecutive turns that form screw-like maneuvers, gradually shifting sideways. *Zigzag* requires skillful maneuvers to change the flying direction rapidly. *Combination turn* is the longest trajectory in our experiments, consisting of three successive *Y-turns* followed by rapid descending and *X-turn*.

5.2 SRL Visualization

Figure 5 (top) shows how self-regulation actually works. The up-vectors along the trajectories are depicted for comparison. On the straight line, the self-regulated targets are almost identical to the user-provided targets (the first two targets in the figure), since the agent was able to pass the targets without the aid of self-regulation. On the curved interval, self-regulated targets incline towards the inside of the turn to guide a banked turn. Even if the input trajectory provides no information about the bank angle, self-regulation automatically discovers how much the agent has to roll to a banked

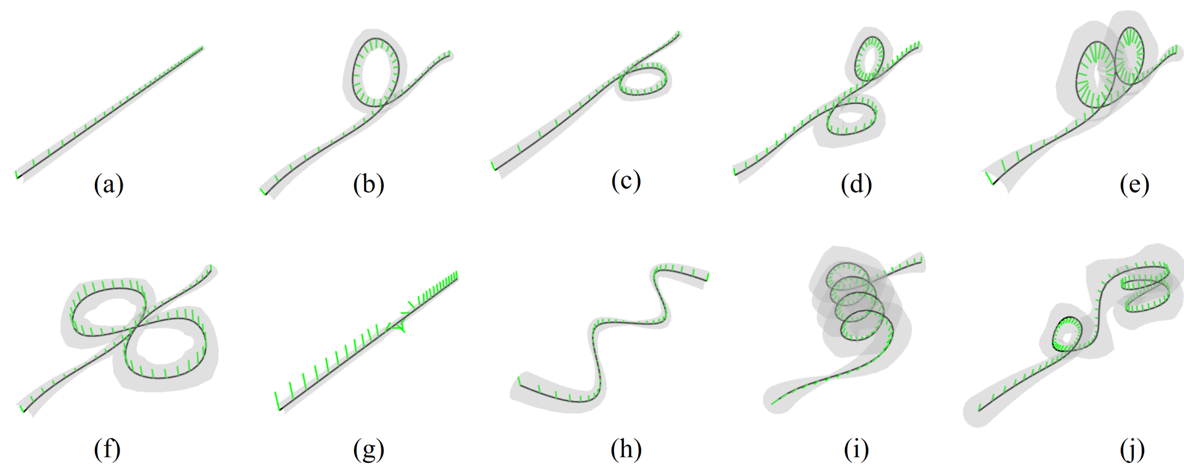

Fig. 4. User-provided spatial trajectories. The up-vectors along the trajectories are shown in green and the clearance thresholds are shown in gray. (a) *Straight*. (b) *X-turn*. (c) *Y-turn*. (d) *XY-turn* (e) *Double X-turn* (f) *Ribbon* (g) *Z-turn*. (h) *Zigzag*. (i) *Infinite X-turn*. (j) *Combination turn*.

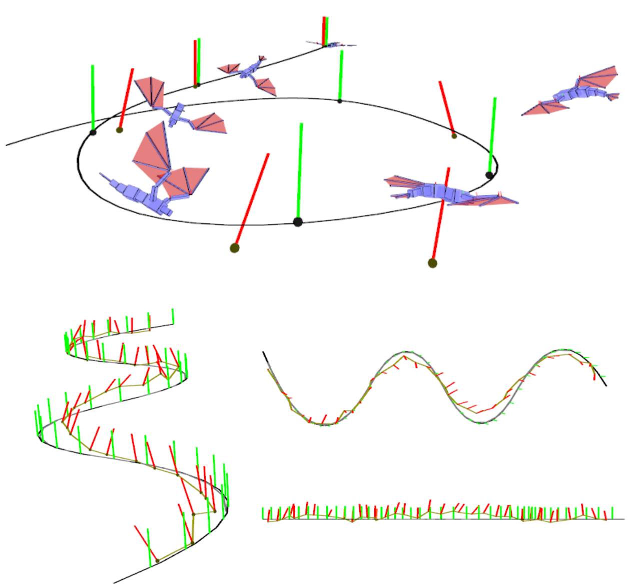

Fig. 5. The input trajectory (green) and self-regulated trajectory (red).

position. Figure 5 (bottom) shows the *Zigzag* trajectory with its self-regulation in front, top, and side views. The trajectory is physically realizable if the curvature of the trajectory is within the exercise capability of the creature, bank angles are specified appropriately, and flying speeds are specified to slow down and accelerate at corners. It is not easy for the user to specify all the details of dynamics. The figure shows that SRL relaxed the curvature of turns at sharp corners, suggested bank angles, and adjusted speeds along the trajectory (slow at corners and faster between them).

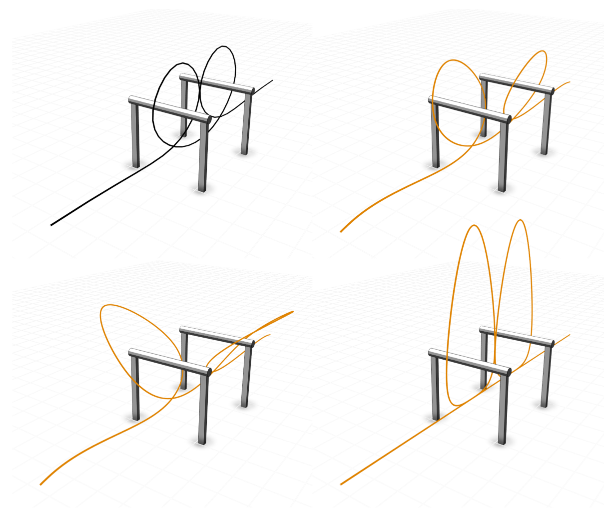

Fig. 6. The control policy learned from the black trajectory can cope with varied trajectories shown in orange. The creature was able to complete all four tasks using the same control policy.

5.3 Generalization Capability

Contrary to trajectory optimization, the RL control policy generalizes to address unseen trajectories similar to the learned trajectory. To evaluate the ability of generalization, we created three new trajectories similar to *Double X-turn* (see Figure 6). The creature was able to complete the new tasks using the policy learned from the original *Double X-turn* trajectory. This example also shows the robustness of the policy, which can withstand external perturbation to a certain extent.

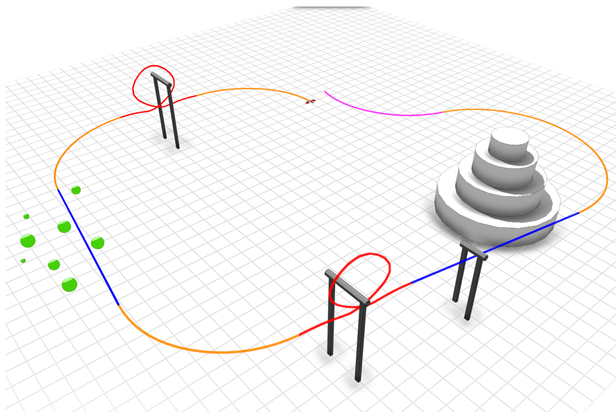

Fig. 7. Interactive animation scenario.

5.4 Interactive Animation Authoring

The user can produce animated scenes interactively by editing individual trajectories and connecting them in a sequence. Figure 7 shows an example of animation scenarios. The color indicates the type of the trajectory. Yellow, magenta, red, and blue correspond to *Left*, *Right*, *X-turn*, and *Z-turn*, respectively. The user manipulated each individual trajectory to fit into the environment (e.g., *Left* and *Right* were attained by bending *Straight*). The creature switches between control policies at the connecting points. The control policies are resilient against small perturbation, so they can handle immediate switching between policies.

5.5 Comparison

SRL vs Non-SRL. To evaluate the effectiveness of our SRL method, we compared two non-SRL versions of our algorithm with its SRL-version. In the first non-SRL algorithm (Default), self-regulated action was fixed as its default value a_o, meaning that the progress parameter increases by the default incremental progress value $\Delta\sigma_o$ and the rotation and translation parameters are fixed as ΔR_o and Δp_o, respectively. In the second non-SRL algorithm (Closest), the progress parameter is updated in a way that the closest point on the spatial trajectory from the creature is provided as a subgoal at every moment. To prevent from choosing the subgoal in reverse direction or jumping to a distant part of the trajectory when the trajectory is self-intersected, we find the subgoal in the vicinity of the current progress parameter by considering only the positive direction. As a result, the progress parameter is increased in a continuous manner, the increment could be zero if necessary. This incremental method is similar to a method in Ju *et al.* [2013].

Table 2 shows performance comparison. Note that we cannot compare the reward values of the algorithms side-by-side because SRL changes the reward system actively. Instead, we measured how closely the control policies tracked the trajectories. The user-provided trajectory and the simulated trajectory are compared through dynamic time warping (DTW). We ran each algorithm three times with different random seeds. The Default algorithm only succeeded in *Straight*, which is the most basic and easy-to-learn for all algorithms, so we did not involve it in the result. The Closest algorithm showed comparable results to our SRL algorithm for the beginner and intermediate skills, however, the SRL algorithm outperformed by large margins for the difficult skills. Note that the SRL algorithm completed all skills, the Closest algorithm was unable to complete *Z-turn*, *Infinite X-turn*, and *Combination* at all.

SRL vs Previous Work. We compared our method to Evo-CACLA by Won *et al.* [2017] with two tasks *Y-turn* and *Zigzag*. In Evo-CACLA method, if a single point is given as an input, then the policy that brings creatures to the point without losing a balance and colliding obstacles is automatically learned by using CACLA-style policy update and the evolutionary exploration. Since Evo-CACLA takes a single target position as input, we assume that the target is moving along the input trajectory at the average flight speed and the creature is controlled to track the target. The creature trained by Evo-CACLA often lagged behind or passed the target so that it often had to stray away from the trajectory to get back to the target. Evo-CACLA was unable to complete the tasks within the clearance thresholds.

SRL vs Trajectory Optimization. Trajectory optimization in our problem setting is equivalent to finding entire action variables to complete a given skill, where the dimension is usually higher than a thousand and its energy landscape is highly nonlinear. We compared our method to a window-based trajectory optimization method similar to Borno *et al.* [2013], where an entire trajectory is split into short segments (windows) and those are optimized sequentially by using CMA-ES [1996]. We used 4-16 actions as a window size. The method successfully completed *Straight*, *X-turn*, and *Double X-turn*, however, it failed for the remaining skills. One main reason for the failure is that preparatory and following motions are crucial for aerobatic skills. For example, when we have two skills in a row, we may have to be imperfect for the first skill to prepare the second skill. In window-based optimization, the action variables only in the same window are optimized simultaneously. Although this condition can be relaxed by making overlaps between windows, the effect is inherently restricted by the window size. We also tested longer window sizes, however, not only slow computation but also convergence on sub-optimal solutions were achieved.

6 DISCUSSION

We have presented a DRL-based approach to simulate and control aerobatic maneuvers of flapping-winged creatures. It allows for the demonstration of extreme motor skills, which span only a tiny bit of subspace in the extremely wide, high-dimensional action space. Self-regulated learning makes it possible to search for aerobatics skills from scratch without any supervision. The process suggested by SRL is quite similar to how people learn challenging motor skills. They set up intermediate goals, practice, evaluate the current capability, and regulate the goals repeatedly. SRL incorporated this intuitive concept into the well-established framework of DRL.

Although SRL is simple and easy-to-implement, it is surprisingly effective for a particular class of problems, for which sequential sub-goals have to be identified and addressed one-by-one to achieve the main goal. We found that a number of continuous control problems fall into this class, including locomotion of bipeds, quadrupeds, birds, fishes, and any imaginary creatures. For example, as discussed

Table 2. Performance comparison of SRL with other algorithms. Average distances between user-provided trajectories and the simulated trajectories are computed by Dynamic Time Warping. Maximum distance values are also shown in parentheses. A smaller number is better, the smallest number for each skill is marked as boldface, an asterisk symbol represents the success of the given skill.

Algorithm	X-turn	Y-turn	XY-turn	Double X-turn	Ribbon	Z-turn	Zigzag	Infinite X-turn	Combination
Default	2304.2 (12137)	1815.6 (10401)	1644.9 (13428)	14201 (79039)	8905.4 (42555)	2180.2 (11043)	1046.2 (4348.9)	36182 (107998)	48869 (250762)
Closest	**28.193*** (**132.4**)	162.10* (**461.81**)	274.48* (1266.9)	**35.891*** (**145.72**)	146.46* (739.98)	152.68 (1846.5)	175.46* (609.79)	942.81 (5653.4)	9050.0 (54705)
SRL	30.235* (177.93)	**115.89*** (516.43)	**114.77*** (**531.96**)	39.18* (232.25)	**131.47*** (**484.56**)	**67.479*** (**228.965**)	**137.70*** (**500.29**)	**136.82*** (**1456.8**)	**264.82*** (**988.96**)

by Peng *et al.* [2017], locomotion control has a hierarchical structure. The high-level control of biped locomotion plans footsteps for a given task. The footsteps serve as sequential sub-goals to be addressed in the low-level control. If the task requires Parkour-level agility in complex environments, successful learning of the low-level controller critically depends on the footstep plans. Training all levels of the hierarchy simultaneously is ideal, but end-to-end learning of high-level planning and low-level control poses a lot of challenges as noted in the previous study. SRL can help regulate the footstep plans to achieve better performance in DRL at the computational cost much cheaper than the cost of end-to-end learning.

Aerobatic maneuvers are less robust against external perturbation than normal locomotion. Extreme motor skills are *extreme* because they are at the boundary of the space of stable, successful actions and the space of unstable, failing actions. Such motor skills are quite resilient against perturbation in one direction, but could be fragile along the other direction. A general recipe for improving the robustness of control is to learn stochastic control policies by adding noise to states, actions, and environments in the training process. Stochastic learning improves robustness with increased computational costs [Wang et al. 2010].

There are also failure cases in our study. In practice, many of the spatial trajectories are aerodynamically infeasible and only a modest portion allow for the realization of aerobatic maneuvers. *Inverse X-turn* is one of the simplest examples that are aerodynamically infeasible. It is similar to *X-turn* except that its rotation axis is opposite. In *X-turn*, the creature soars up first and then goes down while making an arch. On the contrary, in *Inverse X-turn*, the creature dives first and then has to soar up while being upside down. The airfoil-shaped wings cannot generate lifting force in an inverted position. *Landing* is another example our algorithm fails to reproduce. In general, the greater angle of attack, the more lift is generated by wings. However, when the wing reaches its critical (stall) angle of attack, the wing no longer produces lift, but rather stall because of turbulence behind the wing. Birds exploit this phenomenon to rapidly decelerate and land. The simplified aerodynamics model employed in our system cannot simulate turbulent air flow. More accurate aerodynamic simulations are needed to reproduce realistic landing behavior.

Equation 6 used in this study is a mixture of continuous (dense) and discrete (sparse) reward formulation, where the switch between them is determined by the clearance threshold. The benefit of the dense reward is that it can always give feedback signals to the agent, however, sophisticated reward engineering is required and the engineering could be non-trivial in some cases (e.g. solving puzzle and maze). The pros and cons of the sparse reward are the opposite of the dense reward. Although it is known that learning by the sparse reward is much challenging than learning by the dense reward in the high-dimensional environment due to delayed reward and discontinuity, there exist cases where the sparse reward formulation works better due to the nature of the problem domain [Matiisen et al. 2017]. We tested several design choices of the reward, the current choice (a mixture of dense and discrete) with SRL performed best. We think that our SRL was able to modulate denseness/sparsity of the reward adaptively in the learning process.

There are many exciting directions to explore. We wish to explore the possibility of applying SRL to general RL problems, which do not necessarily generate sequential sub-goals in the solution process. As for flapping flight simulation and control, an interesting direction would be improving flexibility, adaptability, and controllability. We hope to be able to control the timing and speed of the action as well as its spatial trajectory. We want our creature to be able to adapt to changes in loads, winds, and all forms of perturbation. It would also be interesting to control exotic creatures with long, deformable bodies and limbs.

ACKNOWLEDGMENTS

This research was supported by the MSIP(Ministry of Science, ICT and Future Planning), Korea, under the SW STARLab support program (IITP-2017-0536-20170040) supervised by the IITP(Institute for Information communications Technology Promotion.

REFERENCES

Social Psychology, Second Edition: Handbook of Basic Principles.

Pieter Abbeel, Adam Coates, and Andrew Y. Ng. 2010. Autonomous Helicopter Aerobatics Through Apprenticeship Learning. *International Journal of Robotics Research* 29, 13 (2010), 1608–1639.

Pieter Abbeel, Adam Coates, Morgan Quigley, and Andrew Y. Ng. 2006. An Application of Reinforcement Learning to Aerobatic Helicopter Flight. In *Proceedings of the 19th International Conference on Neural Information Processing Systems (NIPS 2016).* 1–8.

Pieter Abbeel and Andrew Y. Ng. 2004. Apprenticeship Learning via Inverse Reinforcement Learning. In *Proceedings of the Twenty-first International Conference on Machine Learning (ICML 2004).*

Mazen Al Borno, Martin de Lasa, and Aaron Hertzmann. 2013. Trajectory Optimization for Full-Body Movements with Complex Contacts. *IEEE Transactions on Visualization and Computer Graphics* 19, 8 (2013).

Jernej Barbič, Marco da Silva, and Jovan Popović. 2009. Deformable Object Animation Using Reduced Optimal Control. *ACM Trans. Graph.* 28, 3 (2009).

Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. 2009. Curriculum Learning. In *Proceedings of the 26th Annual International Conference on Machine Learning (ICML 2009)*. 41–48.

M. Al Borno, M. de Lasa, and A. Hertzmann. 2013. Trajectory Optimization for Full-Body Movements with Complex Contacts. *IEEE Transactions on Visualization and Computer Graphics* 19, 8 (2013).

Stelian Coros, Philippe Beaudoin, and Michiel van de Panne. 2010. Generalized biped walking control. *ACM Trans. Graph. (SIGGRAPH 2010)* 29, 4 (2010).

Stelian Coros, Andrej Karpathy, Ben Jones, Lionel Reveret, and Michiel van de Panne. 2011. Locomotion Skills for Simulated Quadrupeds. *ACM Trans. Graph. (SIGGRPAH 2011)* 30, 4 (2011).

Stelian Coros, Sebastian Martin, Bernhard Thomaszewski, Christian Schumacher, Robert Sumner, and Markus Gross. 2012. Deformable Objects Alive! *ACM Trans. Graph. (SIGGRAPH 2012)* 31, 4 (2012).

Marco da Silva, Yeuhi Abe, and Jovan Popović. 2008a. Interactive simulation of stylized human locomotion. *ACM Trans. Graph. (SIGGRAPH 2008)* 27, 3 (2008).

Marco da Silva, Yeuhi Abe, and Jovan Popović. 2008b. Simulation of Human Motion Data Using Short-Horizon Model-Predictive Control. *Computer Graphics Forum* 27, 2 (2008).

Dart. 2012. Dart: Dynamic Animation and Robotics Toolkit. https://dartsim.github.io/. (2012).

Martin de Lasa, Igor Mordatch, and Aaron Hertzmann. 2010. Feature-based locomotion controllers. *ACM Trans. Graph. (SIGGRAPH 2010)* 29, 4 (2010).

Justin Fu, Katie Luo, and Sergey Levine. 2017. Learning Robust Rewards with Adversarial Inverse Reinforcement Learning. *CoRR* abs/1710.11248 (2017).

Alex Graves, Marc G. Bellemare, Jacob Menick, Remi Munos, and Koray Kavukcuoglu. 2017. Automated Curriculum Learning for Neural Networks. In *Proceedings of the 34th Annual International Conference on Machine Learning (ICML 2017)*. 1311–1320.

Radek Grzeszczuk, Demetri Terzopoulos, and Geoffrey E. Hinton. 1998. NeuroAnimator: Fast Neural Network Emulation and Control of Physics-based Models. In *Proceedings of International Conference on Computer Graphics and Interactive Techniques (SIGGRAPH 1998)*. 9–20.

Sehoon Ha and C. Karen Liu. 2014. Iterative Training of Dynamic Skills Inspired by Human Coaching Techniques. *ACM Trans. Graph.* 34, 1 (2014).

Sehoon Ha, Yuting Ye, and C. Karen Liu. 2012. Falling and landing motion control for character animation. *ACM Trans. Graph. (SIGGRAPH Asia 2012)* 31, 6 (2012).

Perttu Hämäläinen, Sebastian Eriksson, Esa Tanskanen, Ville Kyrki, and Jaakko Lehtinen. 2014. Online Motion Synthesis Using Sequential Monte Carlo. *ACM Trans. Graph. (SIGGRAPH 2014)* 33, 4 (2014).

Perttu Hämäläinen, Joose Rajamäki, and C. Karen Liu. 2015. Online Control of Simulated Humanoids Using Particle Belief Propagation. *ACM Trans. Graph. (SIGGRAPH 2015)* 34, 4 (2015).

Daseong Han, Haegwang Eom, Junyong Noh, and Joseph S. Shin. 2016. Data-guided Model Predictive Control Based on Smoothed Contact Dynamics. *Computer Graphics Forum* 35, 2 (2016).

Daseong Han, Junyong Noh, Xiaogang Jin, Joseph S. Shin, and Sung Yong Shin. 2014. On-line real-time physics-based predictive motion control with balance recovery. *Computer Graphics Forum* 33, 2 (2014).

N. Hansen and A. Ostermeier. 1996. Adapting arbitrary normal mutation distributions in evolution strategies: the covariance matrix adaptation. In *Proceedings of IEEE International Conference on Evolutionary Computation*. 312 –317.

David Held, Xinyang Geng, Carlos Florensa, and Pieter Abbeel. 2017. Automatic Goal Generation for Reinforcement Learning Agents. *CoRR* abs/1705.06366 (2017).

Eunjung Ju, Jungdam Won, Jehee Lee, Byungkuk Choi, Junyong Noh, and Min Gyu Choi. 2013. Data-driven Control of Flapping Flight. *ACM Trans. Graph.* 32, 5 (2013).

H. J. Kim, Michael I. Jordan, Shankar Sastry, and Andrew Y. Ng. 2004. Autonomous Helicopter Flight via Reinforcement Learning. In *Advances in Neural Information Processing Systems 16 (NIPS 2003)*. 799–806.

Taesoo Kwon and Jessica Hodgins. 2010. Control Systems for Human Running Using an Inverted Pendulum Model and a Reference Motion Capture Sequence. In *Proceedings of the 2010 ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA 2010)*.

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

Yoonsang Lee, Sungeun Kim, and Jehee Lee. 2010. Data-driven biped control. *ACM Trans. Graph. (SIGGRAPH 2010)* 29, 4 (2010).

Yoonsang Lee, Moon Seok Park, Taesoo Kwon, and Jehee Lee. 2014. Locomotion Control for Many-muscle Humanoids. *ACM Trans. Graph. (SIGGRAPH Asia 2014)* 33, 6 (2014).

Sergey Levine and Vladlen Koltun. 2014. Learning Complex Neural Network Policies with Trajectory Optimization. In *Proceedings of the 31st International Conference on Machine Learning (ICML 2014)*.

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).

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

Libin Liu, Michiel Van De Panne, and Kangkang Yin. 2016. Guided Learning of Control Graphs for Physics-Based Characters. *ACM Trans. Graph.* 35, 3 (2016).

Libin Liu, KangKang Yin, Michiel van de Panne, and Baining Guo. 2012. Terrain runner: control, parameterization, composition, and planning for highly dynamic motions. *ACM Trans. Graph. (SIGGRAPH Asia 2012)* 31, 6 (2012).

Tambet Matiisen, Avital Oliver, Taco Cohen, and John Schulman. 2017. Teacher-Student Curriculum Learning. *CoRR* abs/1707.00183 (2017).

Igor Mordatch and Emanuel Todorov. 2014. Combining the benefits of function approximation and trajectory optimization. In *In Robotics: Science and Systems (RSS 2014)*.

Igor Mordatch, Emanuel Todorov, and Zoran Popović. 2012. Discovery of complex behaviors through contact-invariant optimization. *ACM Trans. Graph. (SIGGRAPH 2012)* 29, 4 (2012).

Andrew Y. Ng, Daishi Harada, and Stuart J. Russell. 1999. Policy Invariance Under Reward Transformations: Theory and Application to Reward Shaping. In *Proceedings of the Sixteenth International Conference on Machine Learning (ICML '99)*. 278–287.

Jeanne Ellis Ormrod. 2009. *Essentials of Educational Psychology.* Pearson Education.

Xue Bin Peng, Pieter Abbeel, Sergey Levine, and Michiel van de Panne. 2018. DeepMimic: Example-Guided Deep Reinforcement Learning of Physics-Based Character Skills Paper Abstract Author Preprint Paper Video. *ACM Transactions on Graphics* 37, 4 (2018).

Xue Bin Peng, Glen Berseth, and Michiel van de Panne. 2016. Terrain-adaptive Locomotion Skills Using Deep Reinforcement Learning. *ACM Trans. Graph. (SIGGRPAH 2016)* 35, 4 (2016).

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. (SIGGRAPH 2017)* 36, 4 (2017).

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

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal Policy Optimization Algorithms. *CoRR* abs/1707.06347 (2017).

Kwang Won Sok, Manmyung Kim, and Jehee Lee. 2007. Simulating biped behaviors from human motion data. *ACM Trans. Graph. (SIGGRAPH 2007)* 26, 3 (2007).

Sainbayar Sukhbaatar, Ilya Kostrikov, Arthur Szlam, and Rob Fergus. 2017. Intrinsic Motivation and Automatic Curricula via Asymmetric Self-Play. *CoRR* abs/1703.05407 (2017).

Jie Tan, Yuting Gu, Greg Turk, and C. Karen Liu. 2011. Articulated swimming creatures. *ACM Trans. Graph. (SIGGRAPH 2011)* 30, 4 (2011).

Jie Tan, Greg Turk, and C. Karen Liu. 2012. Soft Body Locomotion. *ACM Trans. Graph. (SIGGRAPH 2012)* 31, 4 (2012).

TensorFlow. 2015. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. (2015). http://tensorflow.org/ Software available from tensorflow.org.

Yao-Yang Tsai, Wen-Chieh Lin, Kuangyou B. Cheng, Jehee Lee, and Tong-Yee Lee. 2009. Real-Time Physics-Based 3D Biped Character Animation Using an Inverted Pendulum Model. *IEEE Transactions on Visualization and Computer Graphics* 99, 2 (2009).

Xiaoyuan Tu and Demetri Terzopoulos. 1994. Artificial fishes: physics, locomotion, perception, behavior. *Proceedings SIGGRAPH '94* 28, 4 (1994).

Hado van Hasselt and Marco A. Wiering. 2007. Reinforcement Learning in Continuous Action Spaces. In *Proceedings of the 2007 IEEE Symposium on Approximate Dynamic Programming and Reinforcement Learning (ADPRL 2007)*. 272–279.

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

Jack M. Wang, Samuel. R. Hamner, Scott. L. Delp, and Vladlen. Koltun. 2012. Optimizing Locomotion Controllers Using Biologically-Based Actuators and Objectives. *ACM Transactions on Graphics (SIGGRAPH 2012)* 31, 4 (2012).

Jungdam Won, Jongho Park, Kwanyu Kim, and Jehee Lee. 2017. How to Train Your Dragon: Example-guided Control of Flapping Flight. *ACM Trans. Graph.* 36, 6 (2017).

Jia-chi Wu and Zoran Popović. 2003. Realistic modeling of bird flight animations. *ACM Trans. Graph. (SIGGRAPH 2003)* 22, 3 (2003).

Yuting Ye and C. Karen Liu. 2010. Optimal feedback control for character animation using an abstract model. *ACM Trans. Graph. (SIGGRAPH 2010)* 29, 4 (2010).

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

Wenhao Yu, Greg Turk, and C. Karen Liu. 2018. Learning Symmetry and Low-energy Locomotion Paper Abstract Author Preprint Paper Video. *ACM Transactions on Graphics* 37, 4 (2018).

He Zhang, Sebastian Starke, Taku Komura, and Jun Saito. 2018. Mode-Adaptive Neural Networks for Quadruped Motion Control. *ACM Transactions on Graphics* 37, 4 (2018).