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

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 naively 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 design 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 [Barbić 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; Barbić 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 has 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, handsprint, 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
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

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

where $wp$ normalizes the scale of position coordinates. Whenever a target is cleared, $\sigma^*$ increases to suggest the next target to follow. Let $\sigma^*$ 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 $R(0) = [x, y, z] \in SO(3)$ 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

$$R(\sigma) = R(\sigma - \epsilon)U(t(\sigma - \epsilon), t(\sigma)),$$

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

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

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

$$h(\sigma) = h(1 + w_h||\dot{p}(\sigma)||)$$

where $h$ 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 $(S, A, \mathcal{P}(\cdot | \cdot \cdot), \mathcal{R}(\cdot | \cdot \cdot), \gamma)$ where $S$ is a set of state, $A$ 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 $\pi^*: S \rightarrow A$ that maximizes the expectation on cumulative rewards $\eta(\pi)$.

$$\eta(\pi) = E_{s_0, s_1, \ldots} \sum_{t=0}^{\infty} \gamma^t r_t$$

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.

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

where $d_{\text{max}}$ 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 $1 - \frac{d(R, p, \sigma^*)}{d_{\text{max}}}$, which are both weighed by $\sigma^*$.

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], GAEM [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 \(\pi\) takes state \(s\) as input and generates action \(a\). Both functions are represented as deep neural networks with parameters \(\theta_Q, \theta_{\pi}\). 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 \(\rho\) 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

\[
\begin{align*}
& Q|\theta_Q : \text{state-action value network} \\
& \pi|\theta_{\pi} : \text{policy network} \\
& B : \text{experience replay memory} \\
& \text{repeat} \hspace{1cm} \text{for } i = 1, \cdots, T \text{ do} \\
& \hspace{1cm} a_i \leftarrow \pi(s_{i-1}) \\
& \hspace{1cm} \text{if } \text{unif}(0, 1) \leq \rho \text{ then} \\
& \hspace{1.5cm} a_i \leftarrow a_i + N(0, \Sigma) \\
& \hspace{1cm} s_i \leftarrow \text{StepForward}(s_{i-1}, a_i) \\
& \hspace{1cm} r_i \leftarrow R(s_{i-1}, a_i, s_i) \\
& \hspace{1cm} e_i \leftarrow (s_{i-1}, a_i, r_i, s_i) \\
& \hspace{1cm} \text{Store } e_i \text{ in } B \\
& \text{end} \hspace{1cm} \text{for} \\
& X_0, Y_0 \leftarrow \emptyset \\
& X_\pi, Y_\pi \leftarrow \emptyset \\
& \text{for } i = 1, \cdots, N \text{ do} \\
& \text{Sample an experience tuple } e = (s, a, r, s') \text{ from } B \\
& y \leftarrow r + \gamma Q(s', \pi(s'|\theta_{\pi})|\theta_Q) \\
& X_Q \leftarrow X_Q \cup \{(s, a)\} \\
& Y_Q \leftarrow Y_Q \cup \{y\} \\
& \text{if } y - Q(s, \pi(s|\theta_{\pi})|\theta_Q) > 0 \text{ then} \\
& X_\pi \leftarrow X_\pi \cup \{s\} \\
& Y_\pi \leftarrow Y_\pi \cup \{y\} \\
& \text{Update } Q \text{ by } (X_Q, Y_Q) \\
& \text{Update } \pi \text{ by } (X_\pi, Y_\pi) \\
& \text{until no improvement on the policy}
\end{align*}
\]

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
where \( \hat{a} = a \) from the input trajectory.

\[ C \ldots \]

\[ \text{updated for the next state } s \text{ by } W \text{ (see Figure 2). The self-regulated subgoal of the trajectory of window size } s \text{ task. The sensory state } s = (C(\sigma), (C(\sigma + e), \ldots, C(\sigma + w))) \text{ is a part of the trajectory of window size } w, \text{ where } e \text{ is the unit progress. } C(\sigma) \text{ is the subgoal the agent is tracking at } \sigma \text{ (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 \( \tau \) 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 \( \tilde{C}(\tilde{\sigma}) = (\tilde{R}(\tilde{\sigma}), \tilde{p}(\tilde{\sigma}), \tilde{h}(\tilde{\sigma})) \) is

\[
\tilde{R}(\tilde{\sigma}) = R(\tilde{\sigma}) \Delta R, \\
\tilde{p}(\tilde{\sigma}) = p(\tilde{\sigma}) + R(\tilde{\sigma}) \Delta p, \\
\tilde{h}(\tilde{\sigma}) = h(\tilde{\sigma}) + \Delta h.
\]  

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**

1. \( s \) : the current state
2. \( a = (\hat{a}, \tilde{a}) : \text{the action determined by the current policy} \)
3. \( \hat{a} = (\Delta \sigma, \Delta R, \Delta p, \Delta h) : \text{a self-regulation part of the action} \)
4. \( \hat{R} = R(\sigma) \Delta R \)
5. \( \hat{p} = p(\sigma) + R(\sigma) \Delta p \)
6. \( \hat{h} = h(\sigma) + \Delta h \)
7. \( r \) : Compute \( R(s, a, s') \) with progress \( \sigma \) and target \( (\hat{R}, \hat{p}, \hat{h}) \)

**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 \( \Sigma \) is set to 5% of the normalized range. Starting from the initial exploration probability \( \rho \), 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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simulation time step</td>
<td>0.001</td>
</tr>
<tr>
<td>Control time step</td>
<td>0.2</td>
</tr>
<tr>
<td>Policy learning rate (π)</td>
<td>0.0001</td>
</tr>
<tr>
<td>Value learning rate (Q)</td>
<td>0.001</td>
</tr>
<tr>
<td>Discount factor (γ)</td>
<td>0.95</td>
</tr>
<tr>
<td>Exploration probability (ρ)</td>
<td>0.5</td>
</tr>
<tr>
<td>Exploration noise (ε)</td>
<td>0.05</td>
</tr>
<tr>
<td>Maximum time horizon (sec)</td>
<td>50</td>
</tr>
<tr>
<td>Action range (normalized)</td>
<td>±10</td>
</tr>
<tr>
<td>State range (normalized)</td>
<td>±10</td>
</tr>
<tr>
<td>w₀</td>
<td>0.005</td>
</tr>
<tr>
<td>w₁</td>
<td>0.001</td>
</tr>
<tr>
<td>h</td>
<td>20.0</td>
</tr>
<tr>
<td>dₘₐₓ</td>
<td>3.0</td>
</tr>
<tr>
<td>W</td>
<td>0.02</td>
</tr>
</tbody>
</table>

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 $\text{elu}$ units and the final layers are 64 dimensional fully connected layers with $\text{linear}$ units. Note that tracking progress $\sigma$ is merely a single scalar value, but is connected forward to 32 fully connected layers. The high-dimensional inputs $s_q$ and $s_s$ are equally connected to 32 fully connected layers. We designed the network connectivity based on the importance of input parameters. Since $\sigma$ 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 $\tilde{a}$ and self-regulation $\hat{a}$ 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).

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.
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 $a_0$, meaning that the progress parameter increases by the default incremental progress value $\Delta \sigma_0$ and the rotation and translation parameters are fixed as $\Delta R_\theta$ and $\Delta p_\rho$, 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.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>X-turn</th>
<th>Y-turn</th>
<th>XY-turn</th>
<th>Double X-turn</th>
<th>Ribbon</th>
<th>Z-turn</th>
<th>Zigzag</th>
<th>Infinite X-turn</th>
<th>Combination</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>2304.2</td>
<td>1815.6</td>
<td>1644.9</td>
<td>14201</td>
<td>8905.4</td>
<td>2180.2</td>
<td>1046.2</td>
<td>36182</td>
<td>48869</td>
</tr>
<tr>
<td>Closest</td>
<td>28.193</td>
<td>162.31</td>
<td>274.48</td>
<td>35.891</td>
<td>146.46</td>
<td>152.68</td>
<td>175.46</td>
<td>942.81</td>
<td>9050.0</td>
</tr>
<tr>
<td>SRL</td>
<td>30.283</td>
<td>115.89</td>
<td>114.77</td>
<td>39.18</td>
<td>131.47</td>
<td>67.479</td>
<td>137.70</td>
<td>136.82</td>
<td>264.82</td>
</tr>
</tbody>
</table>

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 high-level controller critically depends on the footprint 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 footprint 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 stalls 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 [Matisen 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
