# Curriculum Learning for Motor Skills

Andrej Karpathy and Michiel van de Panne

University of British Columbia

**Abstract.** Humans and animals acquire their wide repertoire of motor skills through an incremental learning process, during which progressively more complex skills are acquired and subsequently integrated with prior abilities. The order in which the skills are learned and the progressive manner in which they are developed play an important role in developing a final skill set. Inspired by this general idea, we develop an approach for learning motor skills based on a two-level curriculum. At the high level, the curriculum specifies an order in which different skills should be learned. At the low level, the curriculum defines a process for learning within a skill. The method is used to develop an ensemble of highly dynamic integrated motor skills for a planar articulated figure capable of doing parameterized hops, flips, rolls, and acrobatic sequences. Importantly, we demonstrate that the same curriculum can be successfully applied to significant variations of the articulated figure to yield appropriately individualized motor skill sets.

## 1 Introduction

"You have to learn to crawl before you walk" is a common maxim that summarizes the approach that humans and animals adopt when faced with the task of learning a complex motor skill. Instead of attacking a difficult learning problem in a monolithic fashion, it is nearly always beneficial to break the problem down to a sequence of manageable stages and sub-goals that are of progressively greater complexity. Humans make extensive use of this heuristic to learn many motor skills, including crawling, walking, running etc [11]. The sequenced acquisition of skills is not only a valuable heuristic in nature, but it is also an active area in machine learning and robotics, e.g., [1, 4, 6]. The defined order for learning skills and then integrating them can be thought of in terms of a *curriculum*, not unlike the training program provided an athletics coach.

In this paper we investigate a curriculum-based learning methodology for the progressive and largely autonomous development of motor skills. We propose the decomposition of skill acquisition into high-level and low-level curricula. A high-level curriculum defines an order for the acquisition of skills, while the low-level curriculum defines an *Achieve-Explore-Generalize* process for the acquisition of a given skill. A low-level curriculum allows for the exploration-based acquisition of parameterized skills, such as performing a flip of a desired height and distance at various initial speeds. In our framework, the curriculum serves as a high level learning script that our physically-simulated agents can execute to acquire

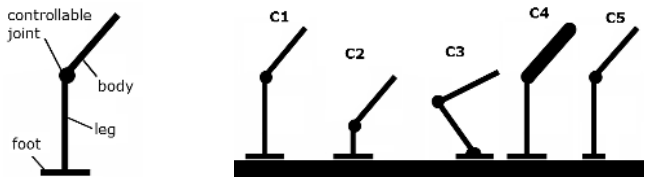



**Fig. 1. Left:** Acrobot structure. **Right:** Acrobots with different morphologies: default (C1), shortened leg (C2), movable ankle (C3), double-mass body (C4), small foot (C5)

skill-sets that are customized to their body morphology. The learning of the curriculum itself is left as an important open problem for future work.

Our curriculum learning approach is demonstrated on a planar, physically-simulated articulated figure that consists of two rigid bodies (links) connected by one actuated joint and an attached, fixed foot. Figure 1 (left) illustrates its canonical form. This is a generalization of the *Acrobot* mechanism that is commonly used as a testbed for underactuated control [3, 5, 13]. Unlike the original Acrobot, our mechanism has its lower link unconstrained and is therefore free to hop, flip, and roll. Figure 1 (right) shows a number of variations that have a diverse range of proportions and mass distributions.

Our contributions are twofold. First, our low-level curriculum structure provides a specific three-part strategy (achieve, explore, generalize) for the exploration-based development of *parameterized* motion skills. These shape the development of parameterized skills in a progressive and purposeful fashion. Our high-level curriculum sits on top of this and is based on human insight, not unlike that required by a coach when designing a training program. Second, our work serves as an important proof-of-concept for the application of curriculum-based learning to the control of *highly dynamic unstable articulated figure motion*. The high dimensional state space (8D-10D) and the highly-sensitive nature of the motions make these particularly challenging to control. To the best of our knowledge, there is little comparable prior work for developmental approaches to this class of problem. Our work thus begins to build a bridge between curriculum learning ideas and methods for controlling agile locomotion.

## 2 Related Work

Aspects of progressive learning strategies can be found in a variety of previous work. It is related to reward shaping [12, 8], macro actions [9, 10], hierarchical reinforcement learning, and continuation methods for walking skills [15]. In the context of deep learning, it has been shown that multi-stage curriculum learning strategies give rise to improved generalization and faster convergence of neural networks on vision and language tasks [2].

The sequenced acquisition of skills is an ongoing active area of research in machine learning and robotics. Asada et al. [1] develop vision-driven behaviors for a mobile robot by using a 'learning from easy missions' (LEM) strategy to

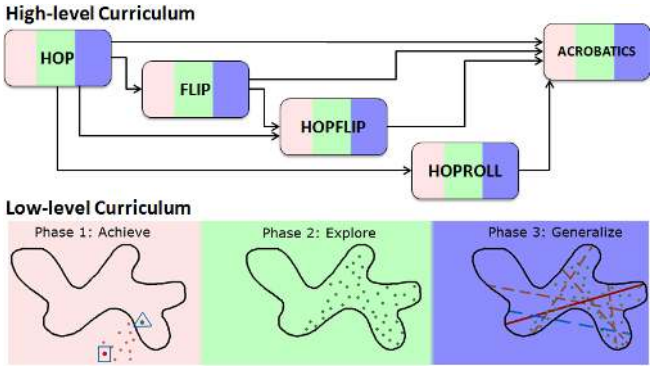



**Fig. 2. Top:** The high-level curriculum. Arrows indicate dependencies of one skill on another. **Bottom:** Each skill is learned using a three stage learning process: Achieve, Explore, and Generalize. The region bounded by the black contour contains actions that successfully accomplish the desired motion without falling.

provide fast-bootstrapping for reinforcement learning applied to mobile robot steering control. A developmental approach to hierarchical knowledge regarding sensorimotor movement is proposed by Hart in [4], along with providing an excellent overview of this area. A discrete event dynamical systems (DEDS) structure is used in combination with a language of typed control expressions in order to generate reaching, touching, and tracking behaviors for a robot arm. Konidaris and Barto [6] introduce a skill discovery method for reinforcement learning in continuous domains that constructs chains of skills leading to an end-of-task reward. This is applied to a 2D simulated pinball domain. Relatedly, the work of Neumann et al. [7] also propose the use of temporally-extended motion templates in a reinforcement learning framework. These are applied to one-link and two-link pendulum tasks. Stout and Barto [14] explore the use of a competence-based intrinsic motivation as a developmental model for skill acquisition. A grid-world proof-of-concept example is provided as an initial demonstration of these ideas.

## 3 Control Overview

The Acrobot will be trained to perform multiple skills, culminating in an acrobatics skill, as summarized by the high-level curriculum shown in Figure 2 (top). Examples of these skills are illustrated in Figure 4. The skills are structured in a high-level curriculum in order of increasing difficulty. Although some of these skills would be hard to acquire individually, the curriculum is structured in such a way that each skill is relatively easy to learn, given prior knowledge of the prerequisite skills. The curriculum thus acts to turn a challenging learning problem into a continuation problem. As we shall see, the Acrobot will first learn to hop, then to flip from rest, followed by flipping from a hop, rolling from a hop, and finally performing a flip, roll, and a flip in a fast, continuous acrobatic maneuver.

The learning of each individual skill progresses through three learning phases: Achieve, Explore, and Generalize. A schematic depiction of these phases is given in Figure 2 (bottom). Given an initial attempted action, the first phase builds on this attempt by searching for a refined action that successfully accomplishes the desired motion without falling. The second phase, Exploration, then uses this action as a seed point for exploring the outcome of many similar actions, thereby yielding many motions that may typically vary in style and distance traveled. The purpose of the third phase, Generalization, is to produce a compact model of the observations acquired during the exploration. This builds on the intuition that there is often a smooth relationship between variations in the applied actions and the resulting motion variations. It also allows for explicit parameterizations of the motions, such as a hop that is parameterized by its hopping speed, or a flip whose height can be controlled. For several of our motions, an effective parameterization can be easily created by interpolating and extrapolating between example actions.

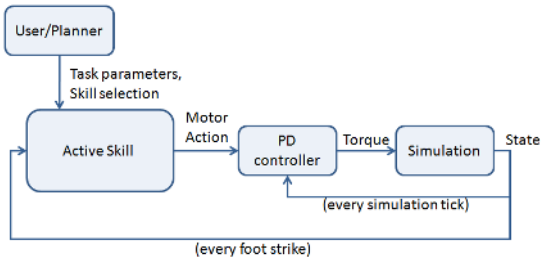
### 3.1 Control System

Figure 3 (left) shows a block diagram of the control system for the Acrobot. It moves by sequencing short, open-loop motions that we refer to as Motor Actions, or simply actions. The active skill initiates a new action every time the foot hits the ground based on skill-specific task parameters that are provided by the user, or a planner. For example, if the Hop skill is active, then a new hop action is initiated every time the foot hits the ground. Once an action is initialized, it outputs two quantities over time: the desired angle between the body and leg links ($\theta_d$) and the stiffness of the motion ($k_p$). A PD controller then calculates the torque that is to be applied on the joint to meet the desired angle. Finally, the torque is provided as input to the simulator at each time step. The simulator computes the accelerations using the equations of motion and these are then numerically integrated to update the state.

A **Motor Action** is a piecewise constant function of time $A : t \to (\theta_d, k_p)$. The number of pieces in every action is left as a design parameter, which we usually fix to be between 3 and 6 for convenience. In general, fixing the number of pieces to be $N$ allows us to think of every motor action as a $(3\times N)$-dimensional vector, because for every piece we need to specify its duration and $(\theta_d, k_p)$ over that time period. The *PD controller* computes the torque on the actuated joint using $\tau = k_p(\theta_d - \theta) - k_d\dot{\theta}$, where $\theta$ is the current angle of the joint, $\dot{\theta}$ is its instantaneous rate of change, $\theta_d$ is the desired angle, $k_p$ is the spring coefficient, and $k_d$ is the damping coefficient. We fix $k_d = \sqrt{2k_p}$ for convenience, which ensures that the system is approximately critically damped.

### 3.2 Skills

We think of a skill as a mapping $\mathbb{I} \times \mathbb{T} \to \mathbb{A}$, where $\mathbb{I}$ is a set of skill-specific parameters that parameterizes the initial conditions, $\mathbb{T}$ is a set of skill-specific task parameters, and $\mathbb{A}$ is the Motor Action space. The mapping specifies the action




| Skill | $\mathbb{I}$ | $\mathbb{T}$ |
|---|---|---|
| Hop | {} | $\{\alpha_v\}$ |
| Flip | {} | $\{\alpha_l\}$ |
| Hop-Flip | $\{\alpha_v\}$ | $\{\alpha_l, \alpha_h\}$ |
| Hop-Roll | $\{\alpha_v\}$ | $\{\alpha_l, \alpha_h\}$ |
| Acrobatics | {} | {} |

**Fig. 3.** Left: System block diagram . Right: Initial Conditions and Task Parameters sets for all skills. $\alpha_v, \alpha_l$ and $\alpha_h$ are labels for the speed, length, and height parameters, respectively.

that should be initialized by the controller in order to accomplish the goals given by the task parameters from the given initial conditions. The table in Figure 3 shows the initial condition parameterization and the task parameterization for every skill. $\mathbb{I}$ is empty for Flip, Hop, and Acrobatics skills. This indicates that these skills are initialized from a single, specific state. In our case, they all begin from the *rest state*, in which the Acrobot is in an upright position and at rest. As an exception to this rule, the actions generated by the Hop skill do not only work from the rest state, but the same actions can also be initialized repeatedly to generate a continuous hopping motion. Ideally, one could imagine learning a distinct hop-from-rest skill that transitions to a separate continuous-hopping skill, but we do not consider this extension here. Finally, the Hop-Flip and Hop-Roll skills begin from a hopping gait of some speed, $\alpha_v$, which can be reached by repeatedly executing the Hop skill. Note that we do not make explicit use of the state of the Acrobot when specifying the initial conditions. Instead, we take an embodied approach in which $\mathbb{I}$ summarizes the state. In our case, the speed parameter $\alpha_v$ used for a hop already highly constrains the set of states that the Acrobot could be in.

Once the Acrobot lands from a flip or a roll, the Hop skill is automatically initialized to revert back to a hopping motion. However, to fully specify a Hop action it is necessary to provide the speed parameter $\alpha_v$. Since not all settings may lead to a successful recovery, we will learn an additional hop recovery mapping, $R : \boldsymbol{s} \rightarrow \alpha_v$, that predicts the value of $\alpha_v$ that will most likely lead to a successful recovery from state $\boldsymbol{s}$.

### 3.3 Composite skills: Acrobatics

It is possible to naturally extend the idea of skills into a higher level of abstraction to explore more complicated, composite skills. Every skill discussed in the previous section is of the form $\mathbb{I} \times \mathbb{T} \rightarrow \mathbb{A}$, where the output is a vector that describes a Motor Action. The output of a composite skill is also a vector, but the numbers are instead interpreted as the initial condition and task parameters of other skills, which then get translated into Motor Actions accordingly.

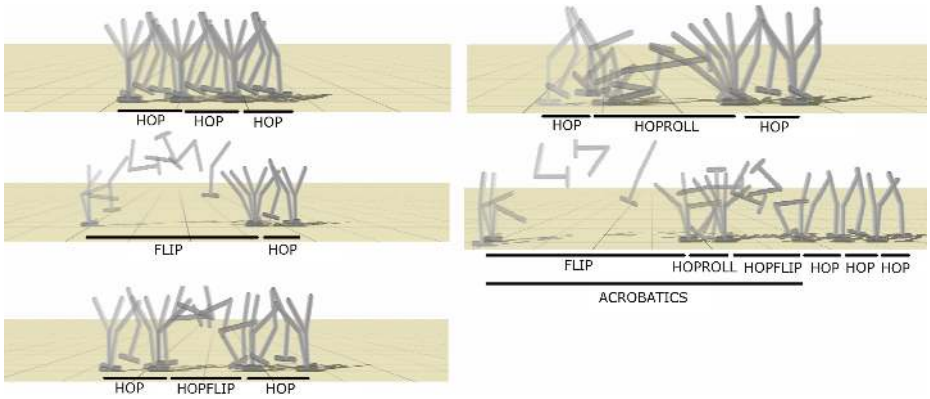



**Fig. 4.** Visualization of all skills. The Acrobot starts on the left and moves to the right.

As shown in Figure 3 (right) that the flip skill is defined as $\alpha_l \rightarrow \mathbb{A}$, and that Hop-Roll and Hop-Flip are both defined as $(\alpha_v, \alpha_l, \alpha_h) \rightarrow \mathbb{A}$. The acrobatics skill consists of one composite action $(\alpha_l^1, \alpha_v^2, \alpha_l^2, \alpha_h^2, \alpha_v^3, \alpha_l^3, \alpha_h^3)$ where $\alpha_l^1$ is used to execute the Flip, $\alpha_v^2, \alpha_l^2, \alpha_h^2$ are used for the subsequent Hop-Roll, and $\alpha_v^3, \alpha_l^3, \alpha_h^3$ are used for the Hop-Flip that immediately follows.

Note that both Hop-Roll and Hop-Flip are meant to work from a hopping gait of some speed and are therefore not generally expected to work from the initial conditions that come up during the Acrobatics sequence. Specifically, the Hop-Roll must be executed right after landing from a Flip, and the Hop-Flip right after landing from the Hop-Roll. Nonetheless, we shall show that by choosing the parameters of both skills appropriately, it is almost always possible to successfully string these skills together. In addition, making use of already existing skills will allow us to be significantly more efficient during the learning, because the learned skills effectively constrain the search to well-behaved manifolds in the Motor Action space.

## 4 High-level Curriculum

As shown in Figure 2, we designed a high-level curriculum for learning the Acrobatics skill by reasoning about the difficulty of each skill given knowledge of the skills before it. Our curriculum begins with the Hop skill, which is easy to learn from scratch due to relatively low complexity of the required motion, and the passive feedback from the interaction of the foot with the ground. In addition, the Hop skill provides a recovery mechanism that the Acrobot can use to regain balance from a wide range of landing states. Flipping from a hopping motion is not an easy task to master, especially when we expect the motion to be robust to errors in the initial conditions. A natural way to make the task easier is to first learn how to flip from a single state, which is the goal of the Flip skill. With the intuition that the action that generates a flip from the rest state should not

| Skill | Training sequence |
|---|---|
| Hop | Hop 20 times |
| Flip | Flip, 5 recovery hops |
| Hop-Flip | Hop 5 times, Hop-Flip, 5 recovery hops |
| Hop-Roll | Hop 5 times, Hop-Roll, 5 recovery hops |
| Acrobatics | Flip,Hop-Roll,Hop-Flip, 5 recovery hops |

**Table 1.** Training sequences for all skills.

be much different from the action that is required when hopping very slowly, the curriculum proceeds in learning the Hop-Flip. While learning the Hop-Flip skill, the Acrobot will learn to flip from progressively higher velocities in the exploration phase of the low-level curriculum. Next, we learn Hop-Roll directly. We did not consider learning an intermediate Roll skill first because the rolling motion is harder to accomplish from the rest state due to lack of forward momentum. Finally, the Acrobatics combines Flip, Hop-Roll and Hop-Flip into a single sequence with no hops in between. Since Hop-Roll and Hop-Flip are trained to work from a hopping motion, this builds on the intuition that the state after landing from a Flip should be approximately similar to the states encountered while hopping continuously, and that the same is true after a Hop-Roll.

## 5 Low-level Curriculum

Every skill is learned by measuring outcomes of actions through repeated *trials*. A trial is labeled as being successful if the Acrobot completes a skill-specific *training sequence* without falling. Each training sequence describes the order of skills that should be executed during every trial. All training sequences are shown in Table 1. For example, as shown in the table, the training sequence for Hop-Flip is to Hop 5 times at some speed, attempt a Hop-Flip, and then attempt 5 recovery hops on landing. If the Acrobot completes the trial, then that indicates that the particular Hop-Flip action can be successfully executed from a Hop of that speed.

More precisely, a result of a successful trial is a tuple $E = (I, A, T)$ that we refer to as an *Experience*. It encodes the observation that starting from a state summarized by initial condition parameters $I \in \mathbb{I}$ and applying Motor Action $A \in \mathbb{A}$ results in task parameter outcome $T \in \mathbb{T}$. Trials can thus be thought of as a function $\mathbb{I} \times \mathbb{A} \to \mathbb{T}$. An example of an Experience while learning Hop-Flip is the tuple $(0.3, A, (1.2, 1.5))$, where $A$ is some Motor Action. It states that when we hop with speed $\alpha_v = 0.3$ and then execute $A$ the next time we land on the ground, it will cause us to successfully flip 1.2 meters forward, and at the highest point of the motion we will be 1.5 meters off the ground.

During the execution of the learning algorithm, the Acrobot will accumulate a large database of Experiences $\mathbb{E} = \{E_i\}$ by conducting many trials. Collectively, the experiences form a tuple-based dynamics model, which is a common strategy for modeling dynamical systems. The collected experiences can later be inverted

to produce a model of the form $\mathbb{I} \times \mathbb{T} \rightarrow \mathbb{A}$, which allows the Acrobot to perform actions with specific desired effects, from given initial conditions. As discussed later, this inversion is computed using either a global linear model or via a lookup table.

**Phase 1: Achieve** The goal of phase 1 is to achieve the first successful trial. We accomplish this by running a stochastic greedy local search to find the first successful Experience $(I_0, A_0, T_0)$, starting from a user-specified seed Motor Action $A_{init}$ and Initial Condition parameters $I_0$. Ideally, $I_0$ should contain parameters that result in the easiest initial conditions. In the case of hopping, the easiest initial conditions are to hop as slowly as possible (i.e. $I_0 = \{\alpha_v = 0\}$). The seed action $A_{init}$ is only designed manually for Hop, Flip, and Hop-Roll. When learning Hop-Flip, the Acrobot will already have learned the Flip skill, and we thus use a flip action to set the $A_{init}$ in this case. Similarily for the Acrobatics skill, we use the already existing Flip, Hop-Flip and Hop-Roll skills to generate the seed action. In Figure 2, $(A_{init}, I_0)$ is drawn inside a rectangle, and $(A_0, I_0)$ is drawn inside a triangle.

A skill-specific phase 1 reward function is also assumed to be provided to help guide the search toward the first successful trial. For Hop, the function returns the number of hops that were successfuly executed before the fall. For Flip, Hop-Flip and Hop-Roll, the function returns the difference of net rotation undergone by the Acrobot from a full circle. Finally, for Acrobatics the function returns the number of skills that were successfully executed before a fall. While the existence of this function is not strictly necessary, it can help improve the performance of the search in this phase.

**Phase 2: Explore** Having acquired the first successful Experience, phase 2 of the learning algorithm incrementally grows the set $\mathbb{E} = \{E_i\}$ through a process of online, active exploration. This is done by repeatedly selecting a *promising Experience* from $\mathbb{E}$ and attempting a slight variation of it in a new trial. The variations are obtained by slightly perturbing both the initial conditions and the action that is attempted. If the trial is successful, the resulting experience is added to $\mathbb{E}$. Otherwise, results of the trial are discarded.

A crucial element of our approach is that we explicitly maintain an estimate of the *reliability* of every experience. We define the reliability of an experience $(I, A, T)$ to be $Reliability(A, I) := \frac{1}{N} \sum_{i=1}^{N} Successful(A + \Delta A, I + \Delta I)$, where $N$ is large, $\Delta A$, and $\Delta I$, are drawn from an appropriate noise distribution, and $Successful(A, I)$ returns 1 if action $A$ leads to a successful trial from initial conditions $I$, and 0 otherwise. Given an Experience $(I, A, T)$, we could compute an estimate of its reliability according to the above definition by running many Trials with slightly different actions and initial conditions. However, in order to improve the efficiency of our algorithm, we will instead approximate the reliability of every action online using $N_{tried}$ and $N_{successful}$ in Algorithm 1.

A good definition of a promising Experience is crucial to the success of the algorithm. The algorithm must ensure a proper balance of exploration and exploitation with hopes of discovering many reliable Experiences from all initial conditions that result in a wide variety of outcomes. We use simple heuristics

**Algorithm 1** Phase 2: Exploration

**Input:**
$(I_0, A_0, T_0) \leftarrow$ Initial successful Experience from phase 1
$N \leftarrow$ Number of trials to be executed

**Output:**
Successful Experience set $\mathbb{E}$

$\mathbb{E} = \{(I_0, A_0, T_0)\}$
**for** $i = 1$**to**$N$ **do**
  $(I_k, A_k, T_k) \leftarrow$ pick promising Experience from $\mathbb{E}$
  $Ntried_k \leftarrow Ntried_k + 1$
  $I_{new} \leftarrow I_k + \Delta I$
  $A_{new} \leftarrow A_k + \Delta A$
  $(T_{new}, success) \leftarrow Trial(I_{new}, A_{new})$
  **if** *success* **then**
    $\mathbb{E} \leftarrow \mathbb{E} \cup \{(I_{new}, A_{new}, T_{new})\}$
    $Nsuccessful_k \leftarrow Nsuccessful_k + 1$
  **end if**
**end for**
**return** $\mathbb{E}$

to guide the exploration with the aforementioned goals in mind. For brevity, we omit the details of these heuristics.

**Phase 3: Generalize** The goal of this last phase is to compute a compact model $\mathbb{I} \times \mathbb{T} \rightarrow \mathbb{A}$ from Experiences collected in phase 2. We consider a linear model for Hop and Flip, and a non-parametric model for Hop-Flip and Hop-Roll. In either case, the Experiences $\mathbb{E}$ that were collected in phase 2 can be discarded afterwards.

We generate linear models for the Hop and Flip skill by interpolating and extrapolating actions that were found in phase 2, as shown in Figure 2 (bottom). This approach has the advantage of being robust to outliers, which is important in this case because the output of phase 2 often contains a mix of many types of motions. We construct several candidate models by sampling two actions from the dataset, and then evaluate them all according to certain desirable criteria. The model that best meets these criteria is returned in the end.

The desirable criteria for each skill are that a model should, first, cover a large range of task parameters and second, integrate well with existing skills. For the Flip, the second requirement amounts to being able to easily transition to the Hop skill after landing. As a good quantitative correlate of this property, we record the variance in the speed of the 5 recovery hops that follow the flip. If the variance is small, it is likely that the transition was successful and the action receives a large score. For the Hop skill, it is only necessary that the skill integrates well with itself, in the sense that it should be possible to change the hopping speed without falling. This robustness toward change in speed is evaluated for each candidate model by generating a continuous hop while changing the speed parameter $\alpha_v$ randomly every hop. The model that falls the least number

of times obtains the highest score. In addition, we compute the variance in the speed of each hop while the character hops at a steady speed, as it is desirable that this variance be low. In the end, the scores of these criteria are combined in a weighted sum to compute the final score for each candidate model.

For every skill other than Hop and Flip, we construct a non-parametric table look-up model. This is done by first partitioning the volume of space $\mathbb{I} \times \mathbb{T}$ into hypercubes of some small size, and then mapping each hypercube to the most reliable Motor Action that was found in that part of $\mathbb{I} \times \mathbb{T}$ space.

Unlike other skills, we also learn a recovery model for the Hop skill, which allows us to recover from arbitrary landings. The recovery model is a mapping $R : \boldsymbol{s} \rightarrow \alpha_v$, that predicts the $\alpha_v$ of the Hop action that most likely leads toward a stable hopping motion from some state $\boldsymbol{s}$. To learn the recovery model for the Hop skill we proceed as follows: while the Acrobot is hopping using the learned Hop skill, we periodically change the speed of every hop according to $\alpha_v \leftarrow min(1, max(0, \alpha_v + N(0, 0.25)))$ with a 50% probability. On every landing, we store the Acrobot's state together with the parameter $\alpha_v$ that will be used for the next hop. If the Acrobot happens to fall during this procedure, we discard the latest 3 measurements and reset the Acrobot to rest state. The resulting database of $(\boldsymbol{s}, \alpha_v)$ forms the non-parametric recovery model for the Hop skill, which can be used to predict $\alpha_v$ through a nearest neighbor search in the state space.

## 6 Results

The simulation is implemented using Open Dynamics Engine (ODE) as a physics engine, with a time step of 0.0005s. This allows for simulation of the Acrobot 30x faster than real time and the evaluation of 10-20 trials per second. The canonical Acrobot has 0.6 $m$ body and leg links and a 0.3 $m$ foot, with masses of 5, 5, and 1 $kg$, respectively. We refer the reader to the online video[1] to view many of the results discussed in this section.

The seed Motor Actions $A_{init}$ that must be provided for Hop, Flip and Hop-Roll took less than a few minutes to create in each case. The entire curriculum was allowed to run for 300,000 trials (about 8 hours), but we found that it is possible to learn all skills in as little as 20,000 trials (about 30 minutes). As is often the case with online algorithms, the results progressively deteriorate for all skills when the algorithm is allowed to run for shorter periods of time. In our case this manifests as skills being generally less reliable, and covering smaller range of task parameters. The number of trials is usually split as 10% for Hop, 10% for Flip, 40% for Hop-Flip, 30% for Hop-Roll, and 10% for Acrobatics. These ratios roughly correspond to the difficulty of learning each skill. For the individual skills, Phase 1 of the algorithm almost always finishes in few seconds. The vast majority of the computation time is thus spent in the exploration phase. Phase 3 of the algorithm for Hop-Flip, Hop-Roll and Acrobatics does not require

[1] Associated video, submitted anonymously: http://vimeo.com/24446828

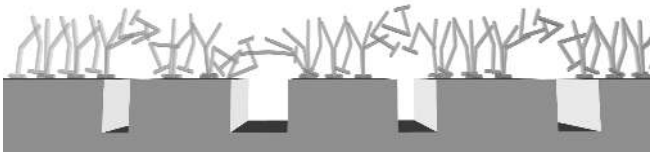


**Fig. 5.** Hop-Flip and Hop-Roll for traversing terrain, from left to right.

evaluation of trials. However, for Hop and Flip, we allocate 66% of the trials to phase 2, and 33% of the trials to phase 3.

Running the curriculum-based learning algorithm results in the following skill capabilities. The Acrobot can use the Hop skill to move at speeds between 0.5m/s and 1.9m/s. The Flip skill allows it to flip between 1.6m and 2.6m. The Hop-Flip skill allows it to flip with lengths and heights between 1.2 to 2.3m and 1.5m to 1.7m respectively from any hop speed. The Hop-Roll skill allows it roll with lengths and heights between 0.5m to 2m and 1.2m to 1.5m respectively. Chaining the Flip, Hop-Roll and Hop-Flip skills into Acrobatics was found to be relatively easy. Since each skill returns actions that already lie on reliable manifolds in the action space, we can afford to randomly sample parameters for the Flip, Hop-Flip and Hop-Roll skills to obtain successful Acrobatics trials with a high probability. To investigate this further, we randomly sampled 500 parameters for these skills and found that 10% of them lead to a successful Acrobatics trial. Therefore, only 10 trials are on average required until a successful action is found. For comparison, we tried to learn the Acrobatics skill from scratch without relying on the existing Flip, Hop-Roll and Hop-Flip skills by hand-coding an initial guess and using a greedy local search to find a successful action. Even after considering several search heuristics, the best performing ones still took on average 150 trials to find the first successful Acrobatics action.

Since our framework is largely independent of the exact proportions of the Acrobot, we successfully learn all skills for several variations (shown in Figure 1) with no parameter tuning. The single exception to this was the Hop-Roll skill of Acrobot C2, for which we had to manually adjust the initial action.

Since the skills are parameterized with respect to task parameters, they can be used by a planner for high-level tasks. In our experiments, we considered the task of jumping over gaps in the terrain (Figure 5). The planner monitors discontinuities in the ground ahead, and if a gap of some length is found, it queries the Hop-Flip or Hop-Roll skills for an action that can achieve the appropriate displacement.

## 7 Conclusions

We have presented a curriculum learning approach that progressively builds on prior abilities when learning more complex skills. We demonstrate that challenging skills, such as acrobatic sequences, can be learned efficiently in this frame-

work. Furthermore, we show that the same curriculum can be applied to varying morphologies to produce skill sets adapted to the individual body types. However, many challenges remain. Currently, the high-level curriculum itself needs to be manually specified. We do not yet address the problem of learning active feedback strategies for our skills, despite this being an important element of control. More work is needed to make more efficient use of trials and to be able to transfer skills from simulation to physical robots. Lastly, we need to develop a better understanding of which types of motion skills can benefit most from a developmental approach.

## References


1. M. Asada, S. Noda, S. Tawaratsumida, and K. Hosoda. Purposive behavior acquisition for a real robot by vision-based reinforcement learning. *Machine Learning*, 23(2):279–303, 1996.
2. Y. Bengio, J. Louradour, R. Collobert, and J. Weston. Curriculum learning. In *Proc. Intl Conf on Machine Learning*, pages 41–48. ACM, 2009.
3. G. Boone. Minimum-time control of the acrobot. In *IEEE Intl Conf on Robotics and Automation, 1997*, pages 3281 –3287, 1997.
4. S.W. Hart. *The development of hierarchical knowledge in robot systems*. PhD thesis, University of Massachusetts Amherst, 2009.
5. John Hauser and Richard M. Murray. Nonlinear controllers for non-integrable systems: the acrobot example. In *American Control Conf*, pages 669 –671, 1990.
6. G. Konidaris and A.G. Barto. Skill discovery in continuous reinforcement learning domains using skill chaining. *Advances in Neural Information Processing Systems*, 22:1015–1023, 2009.
7. G. Neumann, W. Maass, and J. Peters. Learning complex motions by sequencing simpler motion templates. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 753–760. ACM, 2009.
8. A.Y. Ng. *Shaping and policy search in reinforcement learning*. PhD thesis, University of California, Berkeley, 2003.
9. M. Pickett and A.G. Barto. Policyblocks: An algorithm for creating useful macro-actions in reinforcement learning. In *International Conference on Machine Learning*, pages 506–513, 2002.
10. M. Ponsen, M. Taylor, and K. Tuyls. Abstraction and generalization in reinforcement learning: A summary and framework. *Adaptive and Learning Agents*, pages 1–32, 2010.
11. R.A. Schmidt and T.D. Lee. *Motor control and learning: A behavioral emphasis*. Human Kinetics Publishers, 2005.
12. BF Skinner. Reinforcement today. *American Psychologist*, 13(3):94, 1958.
13. M.W. Spong. The swing up control problem for the acrobot. *Control Systems, IEEE*, 15(1):49 –55, feb 1995.
14. A. Stout and A.G. Barto. Competence progress intrinsic motivation. In *IEEE Intl Conf on Development and Learning*, pages 257–262. IEEE, 2010.
15. K.K. Yin, S. Coros, P. Beaudoin, and M. van de Panne. Continuation methods for adapting simulated skills. In *ACM SIGGRAPH 2008 papers*, pages 1–7. ACM, 2008.