

UC Berkeley

UC Berkeley Electronic Theses and Dissertations

Title

Acquiring Motor Skills Through Motion Imitation and Reinforcement Learning

Permalink

<https://escholarship.org/uc/item/6qh84447>

Author

Peng, Xue Bin

Publication Date

2021

Peer reviewed|Thesis/dissertation

Acquiring Motor Skills Through Motion Imitation and Reinforcement Learning

by

Xue Bin Peng

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Doctor of Philosophy

in

Computer Science

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor Sergey Levine, Co-chair

Professor Pieter Abbeel, Co-chair

Assistant Professor Angjoo Kanazawa

Professor Koushil Sreenath

Fall 2021

Acquiring Motor Skills Through Motion Imitation and Reinforcement Learning

Copyright 2021
by
Xue Bin Peng

Abstract

Acquiring Motor Skills Through Motion Imitation and Reinforcement Learning

by

Xue Bin Peng

Doctor of Philosophy in Computer Science

University of California, Berkeley

Professor Sergey Levine, Co-chair

Professor Pieter Abbeel, Co-chair

Humans are capable of performing awe-inspiring feats of agility by drawing from a vast repertoire of diverse and sophisticated motor skills. This dynamism is in sharp contrast to the narrowly specialized and rigid behaviors commonly exhibited by artificial agents in both simulated and real-world domains. How can we create agents that are able to replicate the agility, versatility, and diversity of human motor behaviors? Manually constructing controllers for such motor skills often involves a lengthy and labor-intensive development process, which needs to be repeated for each skill. Reinforcement learning has the potential to automate much of this development process, but designing reward functions that elicit the desired behaviors from a learning algorithm can itself involve a laborious and skill-specific tuning process. In this thesis, we present motion imitation techniques that enable agents to learn large repertoires of highly dynamic and athletic behaviors by mimicking demonstrations. Instead of designing controllers or reward functions for each skill of interest, the agent need only be provided with a few example motion clips of the desired skill, and our framework can then synthesize a controller that closely replicates the target behavior.

We begin by presenting a motion imitation framework that enables simulated agents to imitate complex behaviors from reference motion clips, ranging from common locomotion skills such as walking and running, to more athletic behaviors such as acrobatics and martial arts. The agents learn to produce robust and life-like behaviors that are nearly indistinguishable in appearance from motions recorded from real-life actors. We then develop models that can reuse and compose skills learned through motion imitation to tackle challenging downstream tasks. In addition to developing controllers for simulated agents, our approach can also synthesize controllers for robots operating in the real world. We demonstrate the effectiveness of our approach by developing controllers for a large variety of agile locomotion skills for bipedal and quadrupedal robots.

To my family.

Contents

Contents	ii
List of Figures	v
List of Tables	xii
1 Introduction	1
1.1 Imitation Learning	3
1.2 Thesis Overview	4
2 Background	7
2.1 Reinforcement Learning	7
2.2 Motor Control	13
I Motion Imitation	18
3 Motion Imitation	19
3.1 Related Work	20
3.2 Overview	23
3.3 Policy Representation	23
3.4 Training	25
3.5 Characters	28
3.6 Tasks	29
3.7 Results	30
3.8 Discussion	38
4 Video Imitation	40
4.1 Related Work	41
4.2 Overview	44
4.3 Background	44
4.4 Pose Estimation	45
4.5 Motion Reconstruction	46

4.6	Motion Imitation with RL	47
4.7	Experimental Setup	49
4.8	Results	50
4.9	Discussion	57
5	Multiplicative Compositional Policies	59
5.1	Transfer Learning	60
5.2	Multiplicative Compositional Policies	61
5.3	Related Work	63
5.4	Experiments	64
5.5	Discussion	72
6	Adversarial Motion Priors	73
6.1	Related Work	74
6.2	Overview	77
6.3	Generative Adversarial Imitation Learning	78
6.4	Adversarial Motion Prior	79
6.5	Model Representation	82
6.6	Tasks	82
6.7	Results	85
6.8	Discussion	95
7	Recent Related Work on Motion Imitation	97
II Sim-To-Real Transfer		99
8	Dynamics Randomization	100
8.1	Related Work	101
8.2	Hindsight Experience Replay	103
8.3	Method	104
8.4	Experiments	108
8.5	Discussion	112
9	Bipedal Locomotion	113
9.1	Parameterized Control of Cassie	115
9.2	Learning Walking Control and Sim-to-Real	116
9.3	Experiments	120
9.4	Discussion	124
10	Domain Adaptation	125
10.1	Related Work	126
10.2	Overview	127

10.3 Motion Retargeting	128
10.4 Motion Imitation	129
10.5 Domain Adaptation	130
10.6 Experimental Evaluation	133
10.7 Discussion	140
11 Conclusion	141
Bibliography	144

List of Figures

2.1	Overview of the reinforcement learning framework. At each timestep t , the agent observes a state \mathbf{s}_t . The agent then queries a policy $\pi(\mathbf{a} \mathbf{s})$ for an action \mathbf{a}_t , which is applied to the environment, leading to a transition to a new state \mathbf{s}_{t+1} , as well as a reward $r_t = r(\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1})$. This process is repeated up to a time horizon T , and the objective is to learn a policy that maximizes the cumulative reward over the course of an episode.	8
2.2	Properties of our simulated model of a 3D humanoid, including the mass of each link, and the gains used for PD controllers positioned at each joint. Its body consists of 3D spherical joints and 1D revolute joints.	14
2.3	Examples of the state features used to describe the configuration of the character's body.	15
3.2	Schematic illustration of the visuomotor policy network. The heightmap H is processed by 3 convolutional layers with 16 8x8 filters, 32 4x4 filters, and 32 4x4 filters. The feature maps are then processed by 64 fully-connected units. The resulting features are concatenated with the input state s and goal g and processed by two fully-connected layer with 1024 and 512 units. The output $\mu(s)$ is produced by a layer of linear units. ReLU activations are used for all hidden layers. For tasks that do not require a heightmap, the networks consist only of layers 5-7.	24
3.3	3D simulated characters. Our framework is able to train policies for a wide range of character morphologies.	28
3.4	Characters traversing randomly generated terrains. Top-to-bottom: mixed obstacles, dense gaps, winding balance beam, stairs. The blue line traces the trajectory of the character's center-of-mass.	29
3.5	Snapshots of motions from the trained policies. Top-to-bottom: walk, run, cartwheel, dance A, backflip, frontflip, roll.	32
3.6	Simulated characters performing various skills. Our framework is able to train policies for a broad range of characters, skills, and environments.	33
3.7	Top: Spinkick policy trained to strike a target with the character's right foot. Bottom: Baseball pitch policy trained to throw a ball to a target.	34
3.8	Policy trained for the throw task without a reference motion. Instead of throwing the ball, the character learns to run towards the target.	35

3.9	Left: Original landing motion on flat terrain. Right: Policy trained to imitating landing motion while jumping down from a 2m ledge. Despite being provided only with a motion recorded on flat terrain, the policy is able to adapt the skill to jump down from a tall ledge.	36
3.10	Learning curves for policies trained with and without reference state initialization (RSI) and early termination (ET).	37
3.11	Normalized average returns of policies trained with and without reference state initialization (RSI) and early termination (ET).	37
4.2	The pipeline consists of three stages: pose estimation, motion reconstruction, and imitation. It receives as input, a video clip of an actor performing a particular skill and a simulated character model, and outputs a control policy that enables the character to reproduce the skill in simulation.	45
4.3	Comparison of the motions generated by different stages of the pipeline for Backflip A. Top-to-Bottom: Input video clip, 3D pose estimator, 2D pose estimator, simulated character.	46
4.4	Character imitating a 2-handed vault.	49
4.5	Simulated Atlas robot performing skills learned from video demonstrations. Top: Cartwheel A. Bottom: Dance.	50
4.6	Simulated characters performing skills learned from video clips. Top: Video clip. Middle: 3D pose estimator. Bottom: Simulated character.	51
4.7	Humanoid character imitating skills from video demonstrations. Top-to-Bottom: Jumping jack, kick, push.	52
4.8	Skills retargeted to different environments. Top-to-Bottom: Backflip A across slopes, cartwheel B across gaps, pushing a box downhill and uphill.	53
4.9	Learning curves comparing policies trained with fixed state initialization (FSI), reference state initialization (RSI), and adaptive state initialization (ASI). Three policies initialized with different random seeds are trained for each combination of skill and initial state distribution. Compared to its counterparts, ASI consistently improves performance and learning speed.	54
4.10	3D pose predictions with and without rotation augmentation. Top: Video. Middle: Without rotation augmentation. Bottom: With rotation augmentation. The pose estimator trained without rotation augmentation fails to correctly predict challenging poses, such as when the actor is upside-down.	55
4.11	3D pose predictions before and after motion reconstruction. Top: Video. Middle: Raw predictions from the 3D pose estimator before motion reconstruction. Bottom: After motion reconstruction. The motion reconstruction process is able to fix erroneous predictions from the 3D pose estimator by taking advantage of the information from the 2D pose estimator and temporal consistency between adjacent frames.	55

4.12	Learning curves of policies trained with ASI using different number of Gaussian components. The choice of the number of components does not appear to have a significant impact on performance.	57
4.13	Learning curves comparing policies trained with and without motion reconstruction (MR). MR improves performance for both RSI and ASI.	57
5.2	The transfer tasks pose a challenging combination of locomotion and object manipulation, such as carrying an object to a target location and dribbling a ball to a goal, which requires coordination of multiple body parts and temporally extended behaviors.	65
5.3	Schematic illustrations of the MCP architecture. The gating function receives both \mathbf{s} and \mathbf{g} as inputs, which are first encoded by separate networks, with 512 and 256 units. The resulting features are concatenated and processed with a layer of 256 units, followed by a sigmoid output layer to produce the weights $w(\mathbf{s}, \mathbf{g})$. The primitives receive only \mathbf{s} as input, which is first processed by a common network, with 512 and 256 units, before branching into separate layers of 256 units for each primitive, followed by a linear output layer that produces $\mu_i(\mathbf{s})$ and $\Sigma_i(\mathbf{s})$ for each primitive. ReLU activation is used for all hidden units.	67
5.4	Learning curves of the various models when applied to transfer tasks. MCP substantially improves learning speed and performance on challenging tasks (e.g. carry and dribble), and is the only method that succeeds on the most difficult task (Dribble: T-Rex).	69
5.5	Left: Learning curves on holdout tasks in the Ant environment. Right: Trajectories produced by models with target directions from pre-training, and target directions from the holdout set after training on transfer tasks. The latent space model is prone to overfitting to the pre-training tasks, and can struggle to adapt to the holdout tasks.	70
5.6	Trajectories of the humanoid’s root along the horizontal plane visualizing the exploration behaviors of different models. MCP and other models that are pre-trained with motion imitation produce more structured exploration behaviors.	71
5.7	Left: Weights for primitives over the course of a walk cycle. Primitives develop distinct specializations, with some primitives becoming most active during the left stance phase, and others during right stance. Right: PCA embedding of actions from each primitive exhibits distinct clusters.	72
6.2	Schematic overview of the system. Given a motion dataset defining a desired motion style for the character, the system trains a motion prior that specifies style-rewards r_t^S for the policy during training. These style-rewards are combined with task-rewards r_t^G and used to train a policy that enables a simulated character to satisfy task-specific goals \mathbf{g} , while also adopting behaviors that resemble the reference motions in the dataset.	78

6.3	The motion prior can be trained with large datasets of diverse motions, enabling simulated characters to perform complex tasks by composing a wider range of skills. Each environment is denoted by "Character: Task (Dataset)".	86
6.4	Performance of Target Heading policies trained with different datasets. Left: Learning curves comparing the normalized task returns of policies trained with a large dataset of diverse locomotion clips to policies trained with only walking or running reference motions. Three models are trained using each dataset. Right: Comparison of the target speed with the average speed achieved by the different policies. Policies trained using the larger Locomotion dataset is able to more closely follow the various target speeds by imitating different gaits.	90
6.5	Learning curves comparing the task performance of AMP to latent space models (Latent Space) and policies trained from scratch without motion data (No Data). Our method achieves comparable performance across the various tasks, while also producing higher fidelity motions.	91
6.6	Snapshots of behaviors learned by the Humanoid on the single-clip imitation tasks. AMP enables the character to closely imitate a diverse corpus of highly dynamic and acrobatic skills.	92
6.7	AMP can be used to train complex non-humanoid characters, such as a 59 DoF T-Rex and a 64 DoF dog. By providing the motion prior with different motion clips, the characters can be trained to perform various locomotion gaits, such as trotting and cantering.	93
6.8	Learning curves of various methods on the single-clip imitation tasks. We compare AMP to the motion tracking approach proposed by Peng et al. [207] (Motion Tracking), as well a version of AMP without velocity features for the discriminator (AMP - No Vel), and AMP without the gradient penalty regularizer (AMP - No GP). AMP produces results of comparable quality when compared to prior tracking-based methods, without requiring a manually designed reward function or synchronization between the policy and reference motion. Velocity features and gradient penalty are vital for effective and consistent results on challenging skills.	95
8.2	Our experiments are conducted on a 7-DOF Fetch Robotics arm. Left: Real robot. Right: Simulated MuJoCo model.	105
8.3	Schematic illustrations of the policy network (top), and value network (bottom). Features that are relevant for inferring the dynamics of the environment are processed by the recurrent branch, while the other inputs are processed by the feedforward branch.	107
8.4	LSTM policy deployed on the Fetch arm. Bottom-Right: The contact dynamics of the puck was modified by attaching a packet of chips to the bottom.	108
8.5	Joint trajectories recorded from the simulated and real robot when executing the same target trajectories. The joints correspond to the shoulder, elbow, and wrist of the Fetch arm.	109

8.6	Learning curves of different network architectures. Four policies are trained for each architecture with different random initializations. Performance is evaluated over 100 episodes in simulation with random dynamics.	111
8.7	Performance of different models when deployed on the simulated and real robot for the pushing task. Policies are trained using only data from simulation. . . .	111
9.2	Proposed learning-based walking controller. The inputs of the policy consists of desired gait parameter p^d , desired turning yaw velocity \dot{q}_ϕ^d , a reference gait g^r decoded from the desired gait parameter p^d , observed robot states $\mathbf{q}_{t-4:t}$, $\dot{\mathbf{q}}_{t-4:t}$ from time step $t-4$ to t , and past policy outputs, which consist of the desired motor positions $\hat{\mathbf{q}}_{t-4:t-1}$ from time step $t-4$ to $t-1$. The current desired motor positions $\hat{\mathbf{q}}_t$ are sent to joint-level controllers after passing through a low-pass filter (LPF).	116
9.3	Comparison between (a) proposed Curriculum (CR) and Non-Curriculum (NCR) methods and (b) proposed Non-Residual Control (NRC) and Residual Control (RC) used in previous work [296]. Our proposed method shows best overall training performance in terms of learning speed and converged rewards. The corresponding total samples for the curriculum is 6.6×10^7 , while the NCR model has full range of dynamics randomization from the start of training.	120
9.4	Comparison of proposed commands (Feasible Command Set) and achieved commands (Safe Set) between HZD-based controller [153] and proposed RL-based controller. Our RL-based controller can handle more tracking commands than the HZD-based baseline and thus results in a larger feasible command set. The safe set of RL-based policy is also larger in the sagittal walking velocity \hat{q}_x and walking height \hat{q}_z direction. The tracking performance of the RL-based policy also shows advantages as the shapes of feasible command set and safe set are closer.	122
9.5	Experiment Results. The proposed learned walking policy extensively on Cassie in real world in different scenarios. In the experiments, the policy enables the Cassie to perform various agile behaviors such as fast forward and backward walking, sideways walking, changing walking height, and turning around. Moreover, empowered by the proposed policy, the robot is able to recover from random perturbation and also able to adapt to change different ground frictions and unknown load.	123
9.6	Comparison of robustness to perturbation among 3 different methods in MuJoCo simulation. Non-Residual Controlled Gait Library (NRC+GL) trained with the gait library; Non-Residual Controlled Single Gait (NRC+SG) and Residual Controlled Single Gait (RC+SG) are trained with only one single gait. A 6 DoF force is randomly applied on the pelvis with probability $0.15\beta\%$. The <i>Normalized Return</i> is computed with the mean reward of 32 roll-outs for each model and β	124

10.2	The framework consists of three stages: motion retargeting, motion imitation, and domain adaptation. It receives as input motion data recorded from an animal, and outputs a control policy that enables a real robot to reproduce the motion.	128
10.3	Inverse-kinematics (IK) is used to retarget mocap clips recorded from a real dog (left) to the Laikago robot (right). Corresponding pairs of keypoints (red) are specified on the dog and robot's bodies, and then IK is used to compute a pose for the robot that tracks the keypoints.	129
10.4	Laikago robot performing skills learned by imitating reference motions. Top: Reference motion. Middle: Simulated robot. Bottom: Real robot.	134
10.5	Schematic illustration of the network architecture used for the adaptive policy. The encoder $E(\mathbf{z} \mu)$ receives the dynamics parameters μ as input, which are processed by two fully-connected layers with 256 and 128 ReLU units, and then mapped to a Gaussian distribution over the latent space \mathbf{Z} with mean $\mathbf{m}_E(\mu)$ and standard deviation $\Sigma_E(\mu)$. An encoding \mathbf{z} is sampled from the encoder distribution and provided to the policy $\pi(\mathbf{a} \mathbf{s}, \mathbf{g}, \mu)$ as input, along with the state \mathbf{s} and goal \mathbf{g} . The policy is modeled with two layers of 512 and 256 units, followed by an output layer which specifies the mean $\mathbf{m}_\pi(\mathbf{s}, \mathbf{g}, \mathbf{z})$ of the action distribution. The standard deviation Σ_π of the action distribution is specified by a fixed diagonal matrix. The value function $V(\mathbf{s}, \mathbf{g}, \mu)$ is modeled by a separate network with 512 and 256 hidden units.	136
10.6	Performance statistics of imitating various skills in the real world. Performance is recorded as the average normalized return between $[0, 1]$. Three policies initialized with different random seeds are trained for each combination of skill and method. The performance of each policy is evaluated over 5 episodes, for a total of 15 trials per method. The adaptive policies outperform the non-adaptive policies on most skills.	137
10.7	Comparison of the time elapsed before the robot falls when deploying various policies in the real world. The adaptive policies are often able to maintain balance longer than the other baselines policies, and tend to reach the max episode length without falling.	138
10.8	Performance of policies in 100 simulated environments with different dynamics. The y-axis represents the normalized return, and the x-axis records the portion of environments in which a policy achieves a return higher than a particular value. The adaptive policies achieve higher returns under more diverse dynamics than the non-adaptive policies.	139
10.9	Learning curves of adapting policies to different simulated environments using the learned latent space. The policies are able to adapt to new environments in a relatively small number of episodes.	139

10.10	Performance of adaptive policies trained with different coefficients β for the information penalty. "No IB" corresponds to policies trained without an information bottleneck. The dotted lines represent performance before adaptation, and the solid lines represent after adaptation. Larger values of β results in more robust but less adaptable policies, which starting with better performance before adaptation, but exhibits smaller improvements after adaptation. Vice-versa for smaller values of β	140
-------	--	-----

List of Tables

2.1	Properties of the humanoid character.	14
2.2	Torque limits of each joint.	14
3.1	Properties of the characters.	28
3.2	Performance statistics of imitating various skills. All skills are performed by the humanoid unless stated otherwise. Policies are trained only to imitate a reference motion without additional task objectives. T_{cycle} is the length of the clip. N_{samples} specifies the number of samples collected to train the final policy. NR represents the normalized return of the final policy averaged over 32 episodes, with 0 being the minimum possible return per episode, and 1 the maximum return. For cyclic skills, each episode has a horizon of 20s. For acyclic skills, the horizon is specified by T_{cycle}	32
3.3	Performance statistics of imitating motion clips while also fulfilling additional task objectives.	34
3.4	Success rate of policies trained with the imitation or task objectives disabled. Each policy is evaluated over 100 trials. Simply imitating the reference motions proves insufficient for fulfilling the task objectives. Training without a reference motion produces policies that develop awkward, but functional, strategies for satisfying the task objectives.	35
3.5	Maximum forwards and sideways push each policy can tolerate before falling. Each push is applied to the character’s pelvis for 0.2s.	38
4.1	Performance statistics of over 20 skills learned by our framework. T_{cycle} denotes the length of the clip. N_{samples} records the number of samples collected to train each policy. NR represents the average normalized return of the final policy, with 0 and 1 being the minimum and maximum possible return per episode respectively. For cyclic skills, the episode horizon is set to 20s. For acyclic skills, the horizon is determined by T_{cycle} . All statistics are recorded from the humanoid character unless stated otherwise.	52
4.2	Performance of policies trained with different initial state distributions. ASI outperforms the other methods for all skills evaluated.	54
4.3	Performance of policies with and without motion reconstruction.	56

5.1	Performance statistics of different models on transfer tasks. MCP outperforms other methods on a suite of challenging tasks with complex simulated characters.	68
6.1	AMP hyperparameters.	87
6.2	Performance statistics of combining AMP with additional task objectives. Performance is recorded as the average normalized task return, with 0 being the minimum possible return per episode and 1 being the maximum possible return. The return is averaged across 3 models initialized with different random seeds, with 32 episodes recorded per model. The motion prior can be trained with different datasets to produce policies that adopt distinct stylistic behaviors when performing a particular task.	88
6.3	Summary statistics of the different datasets used to train the motion priors. We record the total length of motion clips in each dataset, along with the number of clips, and the number of subjects (e.g. human actors) that the clips were recorded from.	89
6.4	Performance statistics of imitating individual motion clips without task objectives. "Dataset Size" records the total length of motion data used for each skill. Performance is recorded as the average pose error (in units of meters) between the time-warped trajectories from the reference motion and simulated character. The pose error is averaged across 3 models initialized with different random seeds, with 32 episodes recorded per model. Each episode has a maximum length of 20s. We compare our method (AMP) with the motion tracking approach proposed by Peng et al. [207]. AMP is able to closely imitate a diverse repertoire of complex motions, without manual reward engineering.	94
8.1	Dynamics parameters and their respective ranges.	109
8.2	Performance of the policies when deployed on the simulated and real robot. Performance in simulation is evaluated over 100 trials with randomized dynamics parameters.	110
8.3	Performance of LSTM policies on the real robot, where the policies are trained with subsets of parameters held fixed.	112
9.1	The gait library contains reference motions for gaits that satisfy a range of commands specifying different target values for forward velocity \dot{q}_x , lateral velocity \dot{q}_y , and walking height q_z .	117
9.2	Dynamics Properties and Sample Range.	119
10.1	Dynamic parameters and their respective range of values used during training and testing. A larger range of values are used during testing to evaluate the policies' ability to generalize to unfamiliar dynamics.	133
10.2	Hyper-parameters used during training in simulation with PPO.	135
10.3	Hyper-parameters used for domain adaptation with AWR in the real world.	135

10.4 Performance statistics of imitating various skills in the real world. The method that achieves the highest return for each skill on the real robot is highlighted. Our adaptive model achieves higher returns on most skills. 137

Acknowledgments

First, I would like to thank my advisors Sergey Levine and Pieter Abbeel for their support, guidance, and patience throughout my time at Berkeley. I am grateful for all that you have taught me, and for the many opportunities that you have provided me these past four years. I want to extend a big thanks to my committee members, Angjoo Kanazawa and Koushil Sreenath, who I have also been fortunate to have as collaborators on a number of projects. I am also grateful for my undergraduate and master's advisor Michiel van de Panne, who inspired me to pursue a career in research, and continuous to be a steadfast source of support.

During my PhD, I had the good fortune to work and learn from a wonderful group of collaborators: Zhongyu Li, Xuxin Cheng, Glen Berseth, Grace Zhang, Laura Smith, Ze Ma, Aviral Kumar, Sam Toyer, Michael Chang, Jitendra Malik, Marcin Andrychowicz, and Wojciech Zaremba. I am also grateful for the opportunity to intern at Google Brain, working with a talented and supportive team: Erwin Comans, Jie Tan, Tingnan Zhang, Tsang-Wei Lee, and Sehoon Ha.

To Bonny Ho, thank you for your patience, and for being an unwavering source of support and encouragement. Your gestures of kindness, both large and small, have helped carry me through the many highs and lows over the course of this journey. Somehow, you continue to find ways to make me laugh till my sides hurt.

Finally, I want to thank my parents, Fengping Lin and Jieguo Peng, and my sister, Hui Bin Peng, for their support and care throughout my life. I am grateful for all of the opportunities that you have worked so hard to provide me, often times at the cost of your own. Though it can be easily taken for granted, your quiet perseverance and tireless efforts have been an enduring source of inspiration. At the risk of being banal, none of this would have been possible without you.

Chapter 1

Introduction

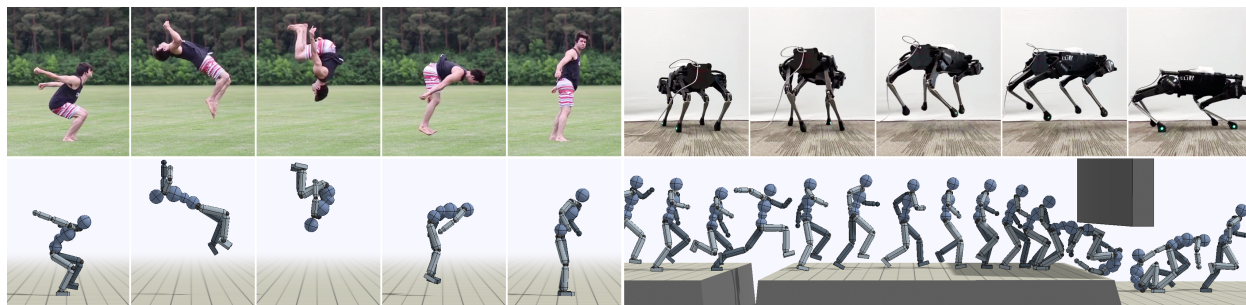


Figure 1.1: Simulated and real-world agents performing highly dynamic motor skills learned using our motion imitation techniques.

Artificial intelligence has seen remarkable breakthroughs in recent years, leading to systems that can accurately identify objects in complex scenes [135, 228, 97], synthesize high-resolution photo-realistic images [27, 120], generate functional code from natural language descriptions [29, 38], and surpass the best human players in challenging games [247, 15, 21]. While these systems are able to achieve astounding cognitive feats, the physical capabilities of artificial agents still fall well behind the athletic prowess demonstrated by humans and other animals. Humans are capable of performing awe-inspiring feats of agility in complex environments by drawing from a vast repertoire of diverse and sophisticated motor skills. This dynamism is in sharp contrast to the narrowly specialized and rigid behaviors commonly exhibited by artificial agents in both simulated and real-world domains. Developing computational models that are able to replicate these agile behaviors has the potential for far-reaching applications in a myriad of fields. Endowing robotic agents with more adept physical capabilities can drastically expand the domains in which robots can effectively operate, allowing them to venture out of the pristine environments of labs and factories, and into more complex unstructured environments of the real world. Virtual agents that are able to produce agile and life-like behaviors can offer a wealth of opportunities in computer

graphics, by automatically synthesizing naturalistic behaviors for characters without the need for artists to painstakingly author animations, or to record motions from real-life actors. These agents may also enable more interactive and immerse virtual experiences than is possible with traditional animation systems. Furthermore, developing better models of human motions can be of interest for biomechanics and physiotherapy, with potential applications in injury prevention and rehabilitation. Such models can also assist in the design and personalization of prostheses that help patients better recover their natural range of motion.

The broad potential impact of physically capable agents has led to a large body of work devoted to designing controllers that enable robots and other artificial agents to reproduce various aspect of human agility [217, 172, 221, 240, 76, 279, 302, 49, 23]. But while humans are adept at performing a wide range of skills themselves, it can be difficult to articulate the internal strategies that underlie this proficiency, and more difficult still to encode them into a controller. Designing controllers often involves a lengthy and labor-intensive development process, requiring substantial expertise of both the underlying system and the desired skills. This approach of manual controller design has produced compelling capabilities [219, 179, 105, 44, 110], but the resulting controllers are often highly specialized for a particular class of skills, and thus limited in their ability to generalize to new behaviors and scenarios. This development process is further complicated as the domain shifts to more athletic and specialized behaviors, such as acrobatics and martial arts, where human insight becomes increasingly scarce. Therefore, despite the many successes of this approach, the capabilities achieved by manually designed controllers are still far from the rich and graceful behaviors seen in their human counterparts.

To mitigate some of the challenges of manual controller engineering, optimization-based methods, such as model-predictive control [138, 250, 182, 7, 74, 54, 12] and reinforcement learning (RL) [196, 129, 264, 59, 45, 259, 90, 111], have been proposed to automate this development process. These methods synthesize controllers by optimizing the parameters of a controller against an objective function. The objective function specifies characteristics of the desired behaviors that an agent should perform, and the optimization algorithm then searches for a set of parameters that best satisfies these characteristics. Optimization-based methods therefore shift the responsibility of the designer from crafting control strategies to crafting objective functions. This in effect allows designers to specify *what* the agent should do instead of *how* to do it. Reinforcement learning in particular has been a powerful paradigm for synthesizing controllers for a large array of complex motor skills [284, 258, 149, 202, 85, 194]. However, designing effective objectives that elicit the desired behaviors from an agent can itself involve a laborious task-specific tuning process. For example, specifying an objective that produces a natural walking gait requires taking into account many nuances of human locomotion, such as energy efficiency, lateral symmetry, impact minimization, head stabilization, etc. [282, 182, 75, 42, 309, 2]. Devising these criteria again requires a great deal of domain expertise, and different behaviors often entail different objectives. Therefore, while objective functions are often considerably easier to design than control strategies, they are nonetheless task-specific and unlikely to scale to the large and diverse repertoire of skills exhibited by humans and other animals.

1.1 Imitation Learning

Though it can be difficult for humans to introspect about how they perform a particular behavior, it is often much easier for humans to simply perform that behavior. This leads one to wonder: can we build more capable agents with less effort by directly *imitating* human behaviors? Acquiring skills by imitating demonstrations can be a very effective paradigm for developing controllers for a wide array of tasks [252, 213, 188, 123, 299, 128, 160, 223], particularly for tasks where it is difficult to design suitable control strategies or objective functions. Our discussion will revolve around two class of imitation learning techniques, which we will refer to broadly as *supervised learning-based* methods and *reinforcement learning-based* methods. The primary difference between these two classes of techniques lies in how demonstration data is utilized to train a policy. Supervised learning-based methods, such as behavioral cloning, leverages demonstration data as direct supervision for training a policy [214, 236, 227, 25, 167]. This approach reduces the imitation learning problem to a supervised learning problem, where a policy is trained to directly predict the actions taken by a demonstrator in various states. Supervised learning-based methods can be highly effective in settings where a large amount of demonstration data is available and the actions taken by the demonstrator can be recorded. However, for motor control tasks, it can be extremely difficult to record the actions that a human might take in performing a given skill, such as the forces that are applied to the joints in a human body while walking. This challenge is further exacerbated when there is an embodiment mismatch between the demonstrator and the agent. In the case of a robot learning to walk by imitating a human, due to the morphological differences between the robot and the human demonstrator, the actions taken by the human will likely be ineffective when directly applied to the robot.

Reinforcement learning-based methods take an alternative approach, where instead of using demonstrations as direct supervision for training a policy, the demonstrations are used to construct an objective function that evaluates the similarity between the demonstrator’s behavior and the behavior of the agent. Given such an objective function, optimize-based methods, such as reinforcement learning or trajectory optimization, can be used to synthesize a controller that performs the desired behavior by optimizing the objective derived from the demonstration data. Reinforcement learning-based methods encompass a large class of techniques, including trajectory tracking methods [17, 243, 249, 49, 184, 161, 145], inverse-reinforcement learning [1, 319, 70], and adversarial imitation learning [104, 77, 133]. These methods have a number of advantages over supervised learning-based methods. First, the objective function can often be defined without knowledge of the demonstrator’s actions. Therefore, this approach can be applied in settings where the demonstrator’s actions cannot be easily recorded. Furthermore, the objective function can also be defined in a way that abstracts away differences between the embodiments of the demonstrator and the agent, thereby allowing agents to imitate demonstrators with significant morphological differences. This approach also tends to be much more data efficient than behavioral cloning, with techniques that can learn complex skills with as few as a single demonstration of a desired behavior. These characteristics of reinforcement learning-based methods are particularly

well-suited for motor control tasks, and therefore will be our approach of choice for much of the work in this thesis. The core challenge of reinforcement learning-based methods is devising general procedures for constructing objective functions from demonstrations that allow agents to imitate a large variety of behaviors.

1.2 Thesis Overview

The work presented in this thesis explores the use of motion imitation and reinforcement learning to develop controllers for a wide array of full-body motor skills with a diverse cast of simulated and real-world agents. We start with a review of fundamental concepts in reinforcement learning and motor control (Chapter 2). Then, the following chapters are organized into two main parts:

- **Part I - Motion Imitation (ch. 3-6):** We present motion imitation methods that allow simulated agents to imitate behaviors from reference motion clips acquired through various sources, such as motion capture (mocap) and video clips. We then develop techniques for transferring the learned skills to downstream tasks.
- **Part II - Sim-To-Real Transfer (ch. 8-10):** We explore methods for transferring controllers learned via motion imitation in simulation to robots in the real world, including techniques such as dynamics randomization and latent space adaptation.

Finally, we conclude with a discussion of the limitations of current state-of-art techniques for motion imitation and potential avenues for future work (Chapter 11).

Chapter 3: We present a versatile reinforcement learning framework for motion imitation, which lays the foundations for much of the following chapters. Our framework enables physically simulated characters to learn a large variety of highly dynamic and acrobatic skills by imitating reference motion clips, which can be provided in the form of motion capture data recorded from real-life actors or keyframed animations authored by artists. We show that the same underlying method can be used to learn a wide range of skills, ranging from everyday locomotion to challenging acrobatic stunts. Our method can also be broadly applied to character with stark morphological differences, such as bipedal humanoids and quadrupedal animals. The resulting controllers produce high quality life-like motions that are nearly indistinguishable in appearance from motion clips recorded from real-life subjects. This work was published as: Xue Bin Peng, Pieter Abbeel, Sergey Levine, and Michiel van de Panne (2018). DeepMimic: Example-Guided Deep Reinforcement Learning of Physics-Based Character Skills. ACM Transactions on Graphics (Proc. SIGGRAPH 2018) [207].

Chapter 4: Our motion imitation learning method allows simulated characters to learn a wide array of skills from motion clips. But acquiring motion clips often requires a costly

process, which typically involves either motion capture of real actors or artist generated animations. In this chapter, we present a video imitation framework that allows simulated character to learn a large and diverse corpus of skills from video clips, which can be a much more accessible source of motion data. By integrating vision-based pose estimation techniques into our previously proposed motion imitation system, our framework is able to learn complex skills from raw monocular video clips, such as those readily found on YouTube, reducing the reliance on expensive motion capture systems. This work was published as: Xue Bin Peng, Angjoo Kanazawa, Jitendra Malik, Pieter Abbeel, and Sergey Levine (2018). SFV: Reinforcement Learning of Physical Skills from Videos. ACM Transactions on Graphics (Proc. SIGGRAPH Asia 2018) [210].

Chapter 5: While motion imitation enables characters to reproduce a large corpus of skills, the resulting control policies are generally limited to performing the particular motions that they was trained to imitate. In this work, we explore transfer learning as means of reusing skills acquired through motion imitation to tackle more complex downstream tasks. We propose a method for learning and composing transferable skills using multiplicative compositional policies (MCP). By first pre-training a collection of motor primitives to imitate a corpus of motion clips, our model learns a collection of primitive skills that can be composed to produce a flexible range of behaviors. Once trained, the primitives can be applied to solve a suite of challenging mobile manipulation tasks, such as dribbling a soccer ball to a goal, and picking up an object and transporting it to a target location. This work was published as: Xue Bin Peng, Michael Chang, Grace Zhang, Pieter Abbeel, and Sergey Levine (2019). MCP: Learning Composable Hierarchical Control with Multiplicative Compositional Policies. Neural Information Processing Systems (NeurIPS 2019) [209].

Chapter 6: The previously discussed motion imitation methods all utilize some form of motion tracking, where an agent imitates a given reference motion by explicitly tracking the sequence of target poses specified by the motion clip. However, it can be challenging to apply tracking-based methods to imitate large and diverse motion datasets, which generally requires significant additional overhead to organize the motion data and select an appropriate motion clip for the agent the track in a given scenario. In this chapter, we present a alternative method based on adversarial imitation learning, which enables agents to imitate behaviors from large motion datasets by using an adversarial motion prior (AMP). The motion prior can be trained with large unstructured motion datasets, and acts as a general measure of similarity between an agent’s behaviors and behaviors depicted in the dataset. By combining the motion prior with additional task objectives, our system can train characters to perform challenging tasks, while utilizing behaviors that resemble those observed in the motion dataset. This work was published as: Xue Bin Peng, Ze Ma, Pieter Abbeel, Sergey Levine, and Angjoo Kanazawa (2021). AMP: Adversarial Motion Priors for Stylized Physics-Based Character Control. ACM Transactions on Graphics (Proc. SIGGRAPH 2021) [205].

Chapter 8: The previous chapters focus primarily on training agents in simulated domains. In this work, we take a step towards transferring controllers learned in simulation to robots operating in the real world. Our key insight is that one of the primary obstacles for sim-to-real transfer is the discrepancy between simulated and real-world dynamics. By randomizing the dynamics of the simulated environments during training, we can encourage policies to acquire robust and adaptable strategies that can cope with variations in the dynamics. This then enables the resulting control policies to operate more effectively under the dynamics of the real world. We first validate this method on transferring policies for simple non-prehensile manipulation tasks with a Fetch robot. This work was published as: Xue Bin Peng, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel (2018). Sim-to-Real Transfer of Robotic Control with Dynamics Randomization. IEEE International Conference on Robotics and Automation (ICRA 2018) [200].

Chapter 9: Next, we combine motion imitation with dynamics randomization to develop locomotion controllers for a real bipedal Cassie robot. By training controllers to imitate a motion library consisting of parameterized gaits, our system produces steerable controllers that can be interactively directed to track different walking velocities, walking heights, and turning directions. The use of dynamic randomization leads to robust control strategies that can be deployed directly on a real robot. This work was published as: Zhongyu Li, Xuxin Cheng, Xue Bin Peng, Pieter Abbeel, Sergey Levine, Glen Berseth, and Koushil Sreenath (2021). Reinforcement Learning for Robust Parameterized Locomotion Control of Bipedal Robots. IEEE International Conference on Robotics and Automation (ICRA 2021) [154].

Chapter 10: Direct sim-to-real transfer relies heavily on a model’s ability to generalize to new environments, which may not always be successful. In these cases, it can be beneficial to further adapt the behaviors of a policy using data from the real world. In this chapter, we leverage dynamics randomization to learn a latent representation of strategies that are effective for different simulated environments. Then to adapt a policy to the real world, we directly search in this latent space for a strategy that is most effective on the physical system. This approach is able to transfer a diverse repertoire of dynamic locomotion skills from simulation to a real quadruped Laikago robot. This work was published as: Xue Bin Peng, Erwin Coumans, Tingnan Zhang, Tsang-Wei Edward Lee, Jie Tan, and Sergey Levine (2020). Learning Agile Robotic Locomotion Skills by Imitating Animals. Robotics: Science and Systems (RSS 2020) [208], and was the recipient of the best paper award.

Chapter 2

Background

The work in this thesis leverages reinforcement learning to train control policies that enable simulated and real-world agents to perform complex motor skills. In this chapter, we provide a review of fundamental concepts in reinforcement learning and motor control, and introduce the notation that we will be using in the following chapters.

2.1 Reinforcement Learning

Reinforcement learning is commonly formulated as an agent interacting with an environment, modeled as a Markov decision process (MDP), with the objective of maximizing its expected return. An overview of the RL framework is available in Figure 2.1. The agent’s interactions with the environment are organized into *episodes*, where at the beginning of each episode, the agent starts in an initial state $\mathbf{s}_0 \in \mathcal{S}$, sampled according to an initial state distribution $\mathbf{s}_0 \sim p(\mathbf{s}_0)$, where \mathcal{S} denotes the state space of the MDP. At each timestep t , the agent observes the states $\mathbf{s}_t \in \mathcal{S}$ of the environment, and samples an action $\mathbf{a}_t \in \mathcal{A}$ from its policy $\mathbf{a}_t \sim \pi(\mathbf{a}_t|\mathbf{s}_t)$ in response to that state, where \mathcal{A} denotes the action space of the MDP. The agent then applies that action, which results in a new state \mathbf{s}_{t+1} , sampled according to the dynamics of the MDP $\mathbf{s}_{t+1} \sim p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$, as well as a scalar reward $r_t = r(\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1})$, reflecting the desirability of a state transition in the context of a given task. This process is repeated up to some time horizon T , which may be infinite. The agent’s interactions within an episode is recorded as a trajectory $\tau = (\mathbf{s}_0, \mathbf{a}_0, r_0, \mathbf{s}_1, \dots, \mathbf{s}_{T-1}, \mathbf{a}_{T-1}, r_{T-1}, \mathbf{s}_T)$. The agent’s objective is to find an optimal policy π^* that maximizes its expected discounted return $J(\pi)$,

$$\pi^* = \arg \max_{\pi} J(\pi) \quad (2.1)$$

$$J(\pi) = \mathbb{E}_{\tau \sim p(\tau|\pi)} \left[\sum_{t=0}^{T-1} \gamma^t r_t \right], \quad (2.2)$$

where $p(\tau|\pi) = p(\mathbf{s}_0) \prod_{t=0}^{T-1} p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t) \pi(\mathbf{a}_t|\mathbf{s}_t)$ is the distribution of trajectories induced by a policy π , and $\gamma \in [0, 1]$ is a discount factor.

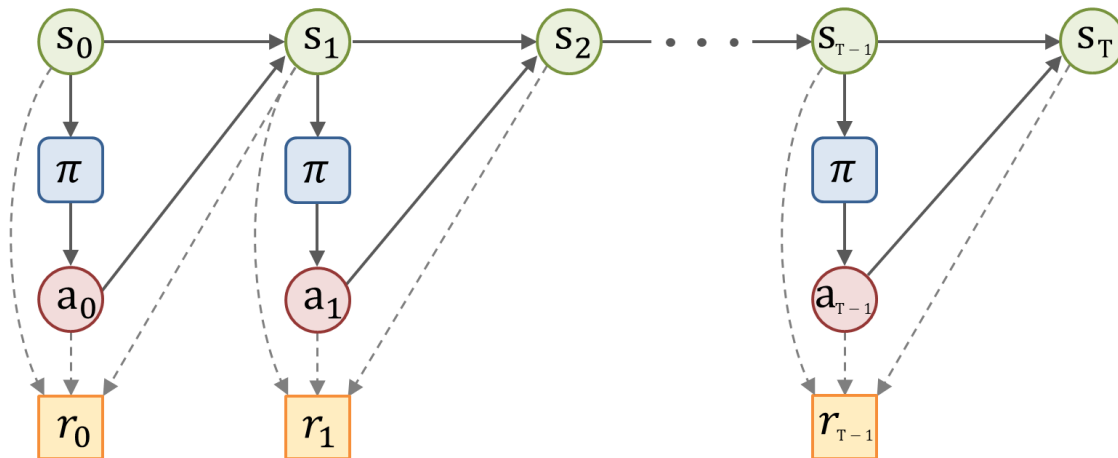


Figure 2.1: Overview of the reinforcement learning framework. At each timestep t , the agent observes a state \mathbf{s}_t . The agent then queries a policy $\pi(\mathbf{a}|\mathbf{s})$ for an action \mathbf{a}_t , which is applied to the environment, leading to a transition to a new state \mathbf{s}_{t+1} , as well as a reward $r_t = r(\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1})$. This process is repeated up to a time horizon T , and the objective is to learn a policy that maximizes the cumulative reward over the course of an episode.

Value Functions

Before we attempt to find an optimal policy, it can be useful to consider methods for estimating the performance of a given policy. Value functions are a central concept in reinforcement learning, which provide an estimate of a policy’s expected future return given a particular state and/or action [254]. The state value function of a policy π , typically denoted by $V^\pi(\mathbf{s})$, provides an estimate of agent’s expected return from following π starting at a given state \mathbf{s} ,

$$V^\pi(\mathbf{s}) = \mathbb{E}_{\tau \sim p(\tau|\pi, \mathbf{s}_0=\mathbf{s})} \left[\sum_{t=0}^{T-1} \gamma^t r_t \right], \quad (2.3)$$

where $p(\tau|\pi, \mathbf{s}_0 = \mathbf{s}) = \prod_{t=0}^{T-1} p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)\pi(\mathbf{a}_t|\mathbf{s}_t)$ represents the likelihood of a trajectory τ from following a policy π starting at state \mathbf{s} . This value can therefore be interpreted as the desirability of the agent being in a particular state. Similarly, the state-action value function $Q^\pi(\mathbf{s}, \mathbf{a})$, commonly referred to as the Q-function, estimates a policy’s expected future return from performing an action \mathbf{a} at state \mathbf{s} and then following π for all future timesteps,

$$Q^\pi(\mathbf{s}, \mathbf{a}) = \mathbb{E}_{\tau \sim p(\tau|\pi, \mathbf{s}_0=\mathbf{s}, \mathbf{a}_0=\mathbf{a})} \left[\sum_{t=0}^{T-1} \gamma^t r_t \right]. \quad (2.4)$$

Here, $p(\tau|\pi, \mathbf{s}_0 = \mathbf{s}, \mathbf{a}_0 = \mathbf{a}) = p(\mathbf{s}_1|\mathbf{s}_0, \mathbf{a}_0) \prod_{t=1}^{T-1} p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)\pi(\mathbf{a}_t|\mathbf{s}_t)$ represents the likelihood of a trajectory from performing action \mathbf{a} at state \mathbf{s} and then following π for subsequent

timesteps. Whenever convenient and without ambiguity, the value functions' dependency on π will be assumed and dropped from the notation. $V(\mathbf{s})$ will be referred to simply as the *value function*, and $Q(\mathbf{s}, \mathbf{a})$ as the *Q-function*.

The value function and Q-function of the optimal policy are commonly denoted by $V^*(\mathbf{s})$ and $Q^*(\mathbf{s}, \mathbf{a})$ respectively. Q^* is of particular interest, since an optimal policy can be recovered by selecting the action that maximizes Q^* at every state,

$$\pi^*(\mathbf{a}|\mathbf{s}) = \begin{cases} 1, & \text{if } \mathbf{a} = \arg \max_{\mathbf{a}'} Q^*(\mathbf{s}, \mathbf{a}') \\ 0, & \text{otherwise} \end{cases}. \quad (2.5)$$

The connection between the value function and Q-function can be seen through their recursive definitions,

$$Q^\pi(\mathbf{s}, \mathbf{a}) = \mathbb{E}_{\mathbf{s}' \sim p(\mathbf{s}'|\mathbf{s}, \mathbf{a})} \mathbb{E}_{\mathbf{a}' \sim \pi(\mathbf{a}'|\mathbf{s}')} [r(\mathbf{s}, \mathbf{a}, \mathbf{s}') + \gamma Q^\pi(\mathbf{s}', \mathbf{a}')] \quad (2.6)$$

$$= \mathbb{E}_{\mathbf{s}' \sim p(\mathbf{s}'|\mathbf{s}, \mathbf{a})} [r(\mathbf{s}, \mathbf{a}, \mathbf{s}') + \gamma V^\pi(\mathbf{s}')] \quad (2.7)$$

$$V^\pi(\mathbf{s}) = \mathbb{E}_{\mathbf{a} \sim \pi(\mathbf{a}|\mathbf{s})} \mathbb{E}_{\mathbf{s}' \sim p(\mathbf{s}'|\mathbf{s}, \mathbf{a})} [r(\mathbf{s}, \mathbf{a}, \mathbf{s}') + \gamma V^\pi(\mathbf{s}')] \quad (2.8)$$

$$= \mathbb{E}_{\mathbf{a} \sim \pi(\mathbf{a}|\mathbf{s})} \mathbb{E}_{\mathbf{s}' \sim p(\mathbf{s}'|\mathbf{s}, \mathbf{a})} \mathbb{E}_{\mathbf{a}' \sim \pi(\mathbf{a}'|\mathbf{s}')} [r(\mathbf{s}, \mathbf{a}, \mathbf{s}') + \gamma Q^\pi(\mathbf{s}', \mathbf{a}')]. \quad (2.9)$$

These recursive definitions can be very useful for learning the value function and Q-function of a given policy, which we will discuss below.

Policy Evaluation

Learning the value function or Q-function of π is a common subroutine in many RL algorithms. *Policy evaluation* is a dynamic programming algorithm for constructing an approximation of a policy's value function by leveraging the recursive definition in Equation 2.8 [254]. Given a dataset of transitions $\mathcal{D} = \{(\mathbf{s}_i, \mathbf{a}_i, r_i, \mathbf{s}'_i)\}$ collected by executing π in the environment, an approximation of V^π can be learned through an iterative procedure, starting with an initial guess V^0 . At each iteration k , a new approximation V^k can be constructed by optimizing the following objective,

$$V^k = \arg \min_V \mathbb{E}_{(\mathbf{s}_i, r_i, \mathbf{s}'_i) \sim \mathcal{D}} [(y_i - V(\mathbf{s}_i))^2], \quad (2.10)$$

where $y_i = r_i + \gamma V^{k-1}(\mathbf{s}'_i)$ is a target value that estimates the expected return of \mathbf{s}_i , and is computed using the value function from the previous iteration V^{k-1} . This method for computing target values is referred to as a single-step *bootstrap*, and multi-step variants of bootstrapping, such as TD(λ) [254], can also be used. It can be shown that in the tabular setting, as $k \rightarrow \infty$, V^k converges to V^π . But in practice, a reasonable approximation can be obtained within a relatively small number of iterations.

The Q-function can be learned through a similar iterative procedure, where at each iteration k , an estimate of Q^π is constructed by optimizing the following objective,

$$Q^k = \arg \min_Q \mathbb{E}_{(\mathbf{s}_i, \mathbf{a}_i, r_i, \mathbf{s}'_i, \mathbf{a}'_i) \sim \mathcal{D}} [(y_i - Q(\mathbf{s}_i, \mathbf{a}_i))^2]. \quad (2.11)$$

Again, $y_i = r_i + \gamma Q^{k-1}(\mathbf{s}'_i, \mathbf{a}'_i)$ is a target value, computed by bootstrapping with the Q-function Q^{k-1} from the previous iteration. Note that we have augmented each transition to include the action \mathbf{a}'_i taken by the agent at the successor state \mathbf{s}'_i .

In practice, when the value function is modeled using a parametric function approximator, such as a neural network, the objectives at each iteration (Equation 2.10, 2.11) can be solved using iterative methods such as gradient descent. Furthermore, instead of optimizing each objective until convergence, only a small number of update steps are required at each iteration.

Policy Gradient Methods

Now that we have reviewed some of the fundamental concepts in reinforcement learning, we will next discuss the choice of algorithms for solving RL problems. In this thesis, we will focus primarily on motor control tasks, which typically give rise to continuous action spaces. Policy gradient methods is a class of RL algorithms that is well-suited for tasks with continuous actions [253], as well as being amenable to neural network function approximators. Policy gradient algorithms iteratively update the parameters of a policy via gradient ascent using an empirical estimate of the gradient of the policy's expected return with respect to its parameters $\nabla_\pi J(\pi)$. To derive the policy gradient, we will first rewrite the expected return of a policy $J(\pi)$ (Equation 2.2) with respect to the policy's marginal state distribution $d^\pi(\mathbf{s})$ instead of its trajectory distribution $p(\tau|\pi)$,

$$J(\pi) = \mathbb{E}_{\mathbf{s} \sim d^\pi(\mathbf{s})} \mathbb{E}_{\mathbf{a} \sim \pi(\mathbf{a}|\mathbf{s})} [Q^\pi(\mathbf{s}, \mathbf{a})]. \quad (2.12)$$

Here, $d^\pi(\mathbf{s}) = \sum_{t=0}^{\infty} [\gamma^t p(\mathbf{s}_t = \mathbf{s}|\pi)]$ represents the *unnormalized* discounted state distribution induced by π , where $p(\mathbf{s}_t = \mathbf{s}|\pi)$ is the likelihood that an agents is in state \mathbf{s} at timestep t by following π . Given this formulation, the policy gradient can be derived by differentiating J with respect to the parameters of π ,

$$\nabla_\pi J(\pi) = \nabla_\pi \mathbb{E}_{\mathbf{s} \sim d^\pi(\mathbf{s})} \mathbb{E}_{\mathbf{a} \sim \pi(\mathbf{a}|\mathbf{s})} [Q^\pi(\mathbf{s}, \mathbf{a})] \quad (2.13)$$

$$= \mathbb{E}_{\mathbf{s} \sim d^\pi(\mathbf{s})} \left[\int_{\mathbf{a} \in \mathcal{A}} \nabla_\pi \pi(\mathbf{a}|\mathbf{s}) Q^\pi(\mathbf{s}, \mathbf{a}) d\mathbf{a} \right]. \quad (2.14)$$

Unfortunately, the integral over actions can be intractable in large action spaces. This calculation can be made more tractable by replacing the integral with an expectation over

actions using the score function $\nabla_{\pi} \log \pi(\mathbf{a}|\mathbf{s})$,

$$\nabla_{\pi} J(\pi) = \mathbb{E}_{\mathbf{s} \sim d^{\pi}(\mathbf{s})} \left[\int_{\mathbf{a} \in \mathcal{A}} \frac{\pi(\mathbf{a}|\mathbf{s})}{\pi(\mathbf{a}|\mathbf{s})} \nabla_{\pi} \pi(\mathbf{a}|\mathbf{s}) Q^{\pi}(\mathbf{s}, \mathbf{a}) d\mathbf{a} \right] \quad (2.15)$$

$$= \mathbb{E}_{\mathbf{s} \sim d^{\pi}(\mathbf{s})} \left[\int_{\mathbf{a} \in \mathcal{A}} \pi(\mathbf{a}|\mathbf{s}) \frac{\nabla_{\pi} \pi(\mathbf{a}|\mathbf{s})}{\pi(\mathbf{a}|\mathbf{s})} Q^{\pi}(\mathbf{s}, \mathbf{a}) d\mathbf{a} \right] \quad (2.16)$$

$$= \mathbb{E}_{\mathbf{s} \sim d^{\pi}(\mathbf{s})} \mathbb{E}_{\mathbf{a} \sim \pi(\mathbf{a}|\mathbf{s})} [\nabla_{\pi} \log \pi(\mathbf{a}|\mathbf{s}) Q^{\pi}(\mathbf{s}, \mathbf{a})]. \quad (2.17)$$

This leads to a simple procedure for computing the policy gradient. We first collect data by executing the policy in the environment. At each timestep, we compute the gradient of the log-likelihood of the action taken by the policy $\nabla_{\pi} \log \pi(\mathbf{a}|\mathbf{s})$. Then the gradients at each timestep is scaled by the Q-function, which represents the return that the agent received by taking action \mathbf{a} at state \mathbf{s} . An approximation of the policy gradient can then be computed by averaging the scaled gradients across all timesteps. While the policy gradient method is conceptually simple, in practice, many design decisions and modifications are necessary to arrive at a practical and effective algorithm. We will next review some of these design decisions.

Baselines: First, the gradient estimator in Equation 2.17 tends to have high variance, which can lead to slow and unstable learning dynamics. A common variance reduction strategy is to introduce a baseline in the form of the value function [289, 254],

$$\nabla_{\pi} J(\pi) = \mathbb{E}_{\mathbf{s} \sim d^{\pi}(\mathbf{s})} \mathbb{E}_{\mathbf{a} \sim \pi(\mathbf{a}|\mathbf{s})} [\nabla_{\pi} \log \pi(\mathbf{a}|\mathbf{s}) (Q^{\pi}(\mathbf{s}, \mathbf{a}) - V^{\pi}(\mathbf{s}))] \quad (2.18)$$

$$= \mathbb{E}_{\mathbf{s} \sim d^{\pi}(\mathbf{s})} \mathbb{E}_{\mathbf{a} \sim \pi(\mathbf{a}|\mathbf{s})} [\nabla_{\pi} \log \pi(\mathbf{a}|\mathbf{s}) A^{\pi}(\mathbf{s}, \mathbf{a})]. \quad (2.19)$$

The term $A^{\pi}(\mathbf{s}, \mathbf{a}) = Q^{\pi}(\mathbf{s}, \mathbf{a}) - V^{\pi}(\mathbf{s})$ is commonly referred as the *advantage*, which can be interpreted as how much *better than average* a given action is compared to the average return at a particular state. The update in Equation 2.19 can therefore be interpreted as increasing the likelihood of actions that do *better* than the average return at each state, and decreasing the likelihood of actions that do *worse* than average. It can be shown that the baseline does not change the policy gradient [253].

Value Estimation: Calculating the policy gradient requires estimating the policy's Q-function Q^{π} and value function V^{π} . Many different design decisions can be made when approximating these quantities. For example, data from π can be used to fit function approximators to Q^{π} and V^{π} , which can then be used to estimate the policy gradient. A common approach, which we will adopt throughout our work, is to approximate V^{π} using a function approximator, but directly estimate $Q^{\pi}(\mathbf{s}, \mathbf{a}) \approx \mathcal{R}_{\mathbf{s}, \mathbf{a}}^{\pi}$ using the empirical returns $\mathcal{R}_{\mathbf{s}, \mathbf{a}}^{\pi}$ observed from the policy. The empirical return of taking an action at a given state $\mathcal{R}_{\mathbf{s}, \mathbf{a}}^{\pi}$ can be determined by simply summing the discounted rewards along a trajectory that starts with \mathbf{s} and \mathbf{a} according to Equation 2.4.

Algorithm 1 RL Training Procedure

```

1:  $\pi \leftarrow$  initialize policy
2:  $V \leftarrow$  initialize value function

3: while not done do
4:    $\mathcal{D} \leftarrow \emptyset$  initialize dataset
5:   for trajectory  $i = 1, \dots, m$  do
6:      $\tau^i \leftarrow (\mathbf{s}_t, \mathbf{a}_t, r_t, \dots, \mathbf{s}_T)$  collect trajectory with  $\pi$ 
7:     for time step  $t = 0, \dots, T - 1$  do
8:        $y_t \leftarrow$  calculate target value from  $\tau^i$  with TD( $\lambda$ )
9:       record  $y_t$  in  $\tau^i$ 
10:       $A_t \leftarrow$  calculate advantage from  $\tau^i$  with GAE( $\lambda$ )
11:      record  $A_t$  in  $\tau^i$ 
12:    end for
13:    store  $\tau^i$  in  $\mathcal{D}$ 
14:  end for

15:  update  $V$  using samples  $\{(\mathbf{s}_i, y_i)\}$  from  $\mathcal{D}$  with Equation 2.10

16:  update  $\pi$  using samples  $\{(\mathbf{s}_i, \mathbf{a}_i, A_i)\}$  from  $\mathcal{D}$  with PPO
17: end while

```

Training

In the following chapters, our algorithm of choice for training policies will primarily be proximal policy optimization (PPO) [238], which is a policy gradient algorithm with additional modifications for improving stability and sample efficiency. Algorithm 1 summarizes the training procedure that we will be using for most of our work. The policy and value function will both be modeled with neural networks. At each update iteration, we collect a batch of trajectories $\{\tau_i\}_{i=1}^m$ from the current policy π . This data is then used to update the value function $V(\mathbf{s})$ with target values y_i computed using TD(λ) [254]. Then we estimate the advantage $A_i \approx A^\pi(\mathbf{s}_i, \mathbf{a}_i)$ for each timestep using GAE(λ) [237], which are then used to update the policy via PPO.

Goal-Conditioned Reinforcement Learning

So far, we have considered a single-task RL framework where the agent is always trying to achieve the same goals. But for many applications, it is desirable to develop agents that are able to perform a variety of different tasks. We can accommodate this multitask setting by using a goal-conditioned reinforcement learning framework, which augments the previous RL formulation with a goal $\mathbf{g} \in \mathcal{G}$ that specifies the task that an agent should perform. For

example, in a navigation task, \mathbf{g} might specify the coordinates of the target location that the agent should move to. At the start of each episode, a goal is sampled according to a task distribution $\mathbf{g} \sim p(\mathbf{g})$, and the reward function $r_t = r(\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}'_t, \mathbf{g})$ is then dependent on the selected task. The goal \mathbf{g} is also provided as an additional input to the policy $\pi(\mathbf{a}|\mathbf{s}, \mathbf{g})$, and the policy’s expected return is determined by its performance across the distribution of tasks,

$$J(\pi) = \mathbb{E}_{\mathbf{g} \sim p(\mathbf{g})} \mathbb{E}_{\tau \sim p(\tau|\pi, \mathbf{g})} \left[\sum_{t=0}^{T-1} \gamma^t r_t \right]. \quad (2.20)$$

Note, the policy’s trajectory distribution is now also dependent on the goal $p(\tau|\pi, \mathbf{g}) = p(\mathbf{s}_0) \prod_{t=0}^{T-1} p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t) \pi(\mathbf{a}_t|\mathbf{s}_t, \mathbf{g})$. Similarly, the value function $V^\pi(\mathbf{s}, \mathbf{g})$ and Q-function $Q(\mathbf{s}, \mathbf{a}, \mathbf{g})$ also receive \mathbf{g} as an additional input. As it is likely apparent by now, \mathbf{g} can be treated just as an additional observation, in the exact same fashion as \mathbf{s} . Therefore, all of the algorithms that we have discussed so far extends trivially to this goal-conditioned setting. This framework can also be easily generalized to accommodate non-stationary goals, where \mathbf{g}_t may vary over time.

2.2 Motor Control

Our work will explore applications of reinforcement learning to motor control tasks in both simulated and real world domains. However, directly applying RL to train agents in the real world raises a myriad of challenges, including sample efficiency, safety, resets, and many more. To circumvent some of these challenges, we will be training agents primarily in simulation, even when the goal is to eventually deploy the policies in the real world. In this section, we detail the design of our simulated agents, and provide a brief review of relevant concepts in motor control.

Character Model

As a didactic example, our discussion will be structured around a canonical 3D humanoid model, which will be featured frequently in the following chapters. A schematic illustration of the humanoid character is available in Figure 2.2, and summary statistics of the character model is available in Table 2.1. The character is modeled as articulated rigid bodies [63], with each link attached to its parent link via a 3 degree-of-freedom spherical joint, except for the elbows and knees, which are attached via 1 degree-of-freedom revolute joints, and the hands which are attached to the arm with a 0 degree-of-freedom fixed joint. The character’s body is composed of 15 links, with a total of 34 degrees-of-freedom. The proportions of its body are based on that of a human actor. The configuration of the character’s body can be described using its pose \mathbf{q} and velocity $\dot{\mathbf{q}}$. The pose $\mathbf{q} = (\mathbf{x}_{\text{root}}, q_{\text{root}}, q_1, q_2, \dots, q_n)$ records the global position of the root \mathbf{x}_{root} , the global rotation of the root q_{root} , and the

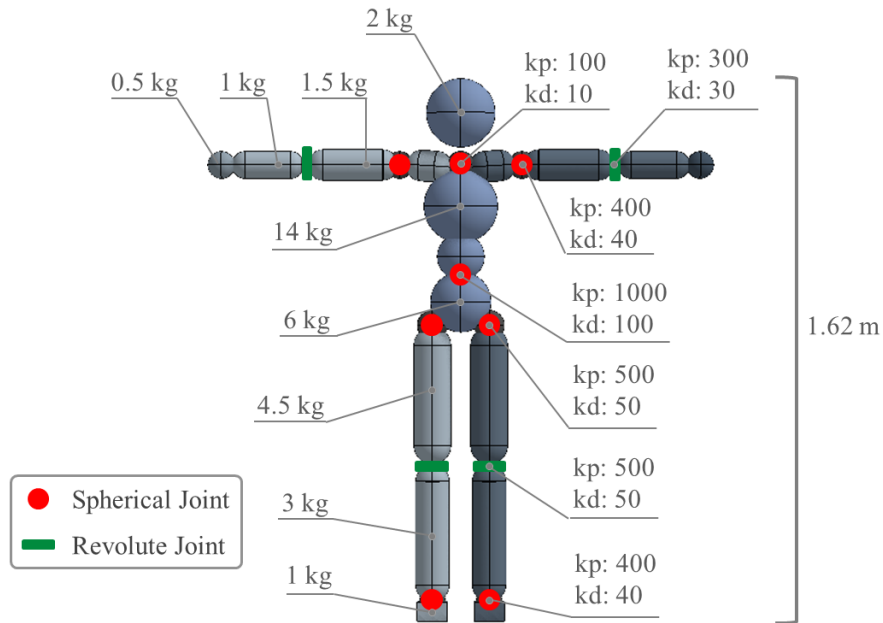


Figure 2.2: Properties of our simulated model of a 3D humanoid, including the mass of each link, and the gains used for PD controllers positioned at each joint. Its body consists of 3D spherical joints and 1D revolute joints.

Property	Value
Links	15
Joints	12
Total Mass (kg)	45
Height (m)	1.62
Degrees of Freedom	34
State Features	226
Action Parameters	28

Joint	Torque Limit
Waist	200 Nm
Neck	50 Nm
Shoulder	100 Nm
Elbow	60 Nm
Hip	200 Nm
Knee	150 Nm
Ankle	90 Nm

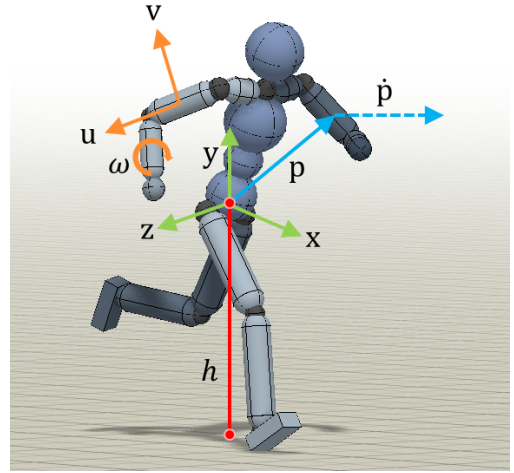
Table 2.1: Properties of the humanoid character. Table 2.2: Torque limits of each joint.

local rotations of each joint q_j expressed in the joint's local coordinate frame. Similarly, $\dot{\mathbf{q}} = (\dot{\mathbf{x}}_{\text{root}}, \omega_{\text{root}}, \omega_1, \omega_2, \dots, \omega_n)$ records the root's linear velocity $\dot{\mathbf{x}}_{\text{root}}$, the root's angular velocity ω_{root} , and the local angular velocity of each joint ω_j . The character's movements are controlled by applying torques to each joint. Table 2.2 details the maximum torque that can be applied to each joint. These torque limits are manually specified and kept fixed for all tasks. The agent's goal is therefore to learn a policy that specifies the appropriate torques for controlling the character's body to perform a desired task.

State Features

In order to control the movement of the character, the policy receives as part of its inputs, a set of state features \mathbf{s}_t that describes the configuration of the character’s body. Figure 2.3 illustrates examples of the state features, which include:

- h : Height of the root off the ground (1D).
- \mathbf{p} : Position of each link (3D).
- $\dot{\mathbf{p}}$: Linear velocity of each link (3D).
- \mathbf{u}, \mathbf{v} : Rotation of each link using a tangent-normal encoding (6D).
- ω : Angular velocity of each link (3D).



The root is designated to be the character’s pelvis. All features are recorded in the character’s local coordinate frame. The character’s local coordinate frame is defined with the origin located at the root, the x-axis oriented along the root link’s facing direction, and the y-axis aligned with the global up vector. The 3D rotation of each link is encoded using two 3D vectors corresponding to the tangent \mathbf{u} and normal \mathbf{v} of the link’s coordinate frame expressed in the character’s local coordinate frame,

Figure 2.3: Examples of the state features used to describe the configuration of the character’s body.

$$\mathbf{u} = \text{rot}(q_{lc}, [1, 0, 0]), \quad \mathbf{v} = \text{rot}(q_{lc}, [0, 1, 0]), \quad (2.21)$$

where $\text{rot}(q, \mathbf{x})$ rotates a vector \mathbf{x} by a quaternion q , and q_{lc} is a quaternion that converts from the link’s local coordinate frame to the character’s local coordinate frame. This rotation encoding provides a smooth and unique representation of a given rotation. Combined, these features result in a 226D state space for the humanoid character.

Action Parameterization and Actuation Model

At each timestep, the policy controls the character’s movements by specifying an action \mathbf{a}_t , which is used by an actuation model to calculate control forces f_j for each joint j . The choice of an action parameterization and actuation model is a crucial design decision that can have a significant impact on the learning process and the quality of the resulting motions. A simple option is to have the action $\mathbf{a} = (f_1, f_2, \dots, f_n)$ directly specify control torques for each joint. In which case, the actuation model is just the identity function. While this a commonly used action parameterization [28, 262], controlling a complex articulated system through low-level torques can pose a challenging learning problem, and often leads to visual

artifacts in the learned behaviors [100, 237, 177], such as high frequency jittering. In our past study [203], we found that higher-level control abstractions, such as PD controllers, can be much more effective for the types of full-body motion control tasks that we will explore in this thesis. Therefore, we will be using PD controllers as the actuation model of choice for much of the following chapters.

PD Controller: When using PD controllers, each action $\mathbf{a} = (\hat{q}_1, \hat{q}_2, \dots, \hat{q}_n)$ specifies target rotations \hat{q}_j for each of the character’s joints. To compute the torque for a particular joint, we will first consider the simple case of a 1D revolute joint. Given a target rotation $\hat{q} \in \mathbb{R}$, represented by a scalar rotation angle, the PD controller is modeled as an angular spring and damper system, where the torque f is computed according to

$$f = k_p(\hat{q} - q) - k_d \dot{q}. \quad (2.22)$$

Here, q denotes the current rotation of the joint, \dot{q} is its angular velocity, k_p and k_d are manually specified gain parameters. The values of the PD gains for each joint are available in Figure 2.2. This actuation model in effect applies torques to move a joint to a target rotation, while also applying damping to prevent excessively large joint velocities. In the case of 3D spherical joints, the target rotation \hat{q} can be represented by a quaternion. The torque from the PD controller can then be calculated in a similar manner,

$$f = k_p \exp_map(\hat{q} \ominus q) - k_d \dot{q}, \quad (2.23)$$

where $\dot{q} \in \mathbb{R}^3$ is the 3D angular velocity of the joint. $q_1 \ominus q_2$ denotes the quaternion difference between two rotations, which can be computed via quaternion multiplication between q_1 and the conjugate q_2' , $q_1 \ominus q_2 = q_2' q_1$. To calculate a torque from the resulting quaternion, we use $\exp_map(q)$ to convert a quaternion q into the exponential map representation of the underlying rotation [82]. For a rotation angle of θ , expressed in radians, around axis \mathbf{v} , the exponential map of this rotation is given by scaling the axis by the rotation angle $q = \theta \mathbf{v}$. Similarly, given an exponential map q , the rotation angle and rotation axis can be determined according to:

$$\mathbf{v} = \frac{q}{\|q\|_2}, \quad \theta = \|q\|_2. \quad (2.24)$$

Action Parameterization: Now that we have examined the actuation model of PD controllers for 1D revolute joints and 3D spherical joints, we will next discuss the choice of action parameterizations. Each action \mathbf{a} from the policy specifies target rotations for the character’s joints. But what representation should we use for these rotations, especially when the policies will be modeled using neural networks? For revolute joints, the target rotation can be specified by simply using a scalar value θ , representing the target rotation angle. However, the choice of representation for spherical joints is more nuanced. Since, the actuation model for spherical joints (Equation 2.23) centers around quaternions, an obvious choice might be

to also use quaternions for the action parameterization. But, a quaternion requires 4 parameters to specify a 3D rotation, the parameters need to be normalized in order to represent a proper rotation, and the quaternion representation of a particular rotation is not unique, since both q and $-q$ represent the same rotation. Therefore, quaternions are not the most compact representation for 3D rotations, and also requires additional machinery to ensure normalization and uniqueness. Euler angles is another common option, which requires only 3 parameters per rotation, and does not require normalization. Unfortunately, euler angles suffer from singularities and gimbal lock, where the joint loses a degree of freedom whenever two rotation axes align [82]. Furthermore, the euler angles representation of a rotation may also not be unique, and different settings of the components can yield the same rotation. A more in-depth review of different rotation parameterizations and their trade-offs is available in Grassia [82].

In this work, we will parameterize target rotations for spherical joints using exponential maps, which provides a compact representation using only 3 parameters. When the parameters of an exponential map is bounded $\|q\|_2 < \pi$, it provides a unique representation of a given rotation, free of singularities and gimbal lock. If q is unbounded then this representation has singularities whenever $\|q\|_2 = k\pi$ for $k = 1, 2, 3, \dots$, where all q with a norm of that is a multiple of π correspond to the same rotation. Furthermore, an unbounded exponential map leads to non-unique representations of a given rotation, where $q + k2\pi q/\|q\|_2$ for $k = 1, 2, 3, \dots$ all represent the same rotation. In practice, these problems can largely be avoided by simply bounding the policy's action space, such that all action parameters lie within $[-\pi, \pi]$.

Part I

Motion Imitation

Chapter 3

Motion Imitation

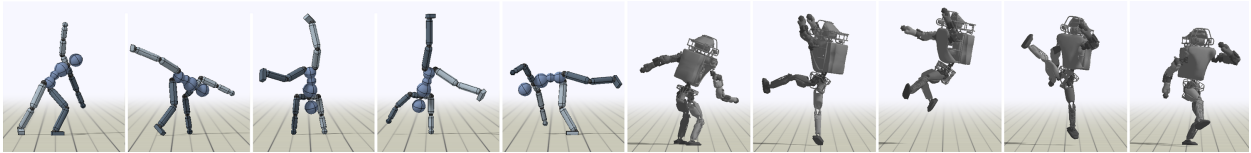


Figure 3.1: Highly dynamic skills learned by imitating reference motion capture clips using our method, executed by physically simulated characters. **Left:** Humanoid character performing a cartwheel. **Right:** Simulated Atlas robot performing a spinkick. (Video¹)

Reinforcement learning provides a powerful paradigm for synthesizing controllers for a large array of tasks, whereby an agent learns to perform various skills through trial-and-error, thus reducing the need for human insight. While deep reinforcement learning has been demonstrated to produce a range of complex behaviors in prior work [237, 57, 100], the quality of the generated motions has thus far lagged well behind state-of-the-art kinematic animation methods or manually designed controllers. In particular, controllers trained with deep RL exhibit severe (and sometimes humorous) artifacts, such as extraneous upper body motion, peculiar gaits, and unrealistic posture [99].² A natural direction to improve the quality of learned controllers is to incorporate motion capture or hand-authored animation data. In prior work, such systems have typically been designed by layering a physics-based tracking controller on top of a kinematic animation system [49, 145]. This type of approach is challenging because the kinematic animation system must produce reference motions that are feasible to track, and the resulting physics-based controller is limited in its ability to modify the motion to achieve plausible recoveries or accomplish task goals in ways that deviate substantially from the kinematic motion. Furthermore, such methods tend to be quite complex to implement.

¹ Supplementary video: <https://xbpeng.github.io/projects/DeepMimic/>

² See, for example, https://youtu.be/hx_bgoTF7bs

An ideal learning-based animation system should allow an artist or motion capture actor to supply a set of reference motions for style, and then generate goal-directed and physically realistic behavior from those reference motions. In this chapter, we take a simple approach to this problem by directly rewarding the learned controller for producing motions that resemble reference animation data, while also achieving additional task objectives. We also demonstrate three methods for constructing controllers from multiple clips: training with a multi-clip reward based on a max operator; training a policy to perform multiple diverse skills that can be triggered by the user; and sequencing multiple single-clip policies by using their value functions to estimate the feasibility of transitions.

The central contribution of this chapter is a framework for physics-based character animation that combines goal-directed reinforcement learning with data, which may be provided in the form of motion capture clips or keyframed animations. In our experiments, we demonstrate that our framework is able to produce a wide range of skills with motion quality and robustness that substantially exceed prior work. By incorporating motion capture data into a phase-aware policy, our system can produce physics-based behaviors that are nearly indistinguishable in appearance from the reference motion in the absence of perturbations, avoiding many of the artifacts exhibited by previous deep reinforcement learning algorithms, e.g., [57]. In the presence of perturbations or modifications, the motions remain natural, and the recovery strategies exhibit a high degree of robustness without the need for human engineering. We demonstrate some of the most capable physically simulated characters produced by learning-based methods. In our ablation studies, we identify two specific components of our method, reference state initialization and early termination, that are critical for achieving highly dynamic skills.

3.1 Related Work

Modeling the skilled movement of articulated figures has a long history in fields ranging from biomechanics to robotics and animation. In recent years, as machine learning algorithms for control have matured, there has also been an increase in interest in these problems from the machine learning community. Here we focus on the most closely related work in animation and RL.

Kinematic Models: Kinematic methods have been an enduring avenue of work in character animation that can be effective when large amounts of data are available. Given a dataset of motion clips, controllers can be built to select the appropriate clip to play in a given situation, e.g., [233, 144, 4]. Gaussian processes have been used to learn latent representations which can then synthesize motions at run-time [301, 148]. Extending this line of work, deep learning models, such as autoencoders and phase-functioned networks, have also been applied to develop generative models of human motion in a kinematic setting [108, 107]. Given high quality data, data-driven kinematic methods will often produce higher quality motions than most simulation-based approaches. However, their ability to

synthesize behaviors for novel situations can be limited. As tasks and environments become complex, collecting enough motion data to provide sufficient coverage of the possible behaviors quickly becomes untenable. Incorporating physics as a source of prior knowledge about how motions should change in the presence of perturbations and environmental variation provides one solution to this problem, as discussed below.

Physics-based Models: Design of controllers for simulated characters remains a challenging problem, and has often relied on human insight to implement task-specific strategies. Locomotion in particular has been the subject of considerable work, with robust controllers being developed for both human and nonhuman characters, e.g., [302, 300, 44]. Many such controllers are the products of an underlying simplified model and an optimization process, where a compact set of parameters are tuned in order to achieve the desired behaviors [284, 5, 88]. Dynamics-aware optimization methods based on quadratic programming have also been applied to develop locomotion controllers [49, 145, 146]. While model-based methods have been shown to be effective for a variety of skills, they tend to struggle with more dynamics motions that require long-term planning, as well as contact-rich motions. Trajectory optimization has been explored for synthesizing physically-plausible motions for a variety of tasks and characters [182, 281]. These methods synthesize motions over an extended time horizon using an offline optimization process, where the equations of motion are enforced as constraints. Recent work has extended such techniques into online model-predictive control methods [91, 261], although they remain limited in both motion quality and capacity for long-term planning. The principal advantage of our method over the above approaches is that of generality. We demonstrate that a single model-free framework is capable of a wider range of motion skills (from walks to highly dynamic kicks and flips) and an ability to sequence these; the ability to combine motion imitation and task-related demands; compact and fast-to-compute control policies; and the ability to leverage rich high-dimensional state and environment descriptions.

Reinforcement Learning: Many of the optimization techniques used to develop controllers for simulated characters are based on reinforcement learning. Value iteration methods have been used to develop kinematic controllers to sequence motion clips in the context of a given task [144, 148]. Similar approaches have been explored for simulated characters [45, 201]. More recently, the introduction of deep neural network models for RL has given rise to simulated agents that can perform a diverse array of challenging tasks [57, 28, 202, 159, 223, 265]. Policy gradient methods have emerged as the algorithms of choice for many continuous control problems [254, 239, 238]. Although RL algorithms have been capable of synthesizing controllers using minimal task-specific control structures, the resulting behaviors generally appear less natural than their more manually engineered counterparts [237, 176]. Part of the challenge stems from the difficulty in specifying reward functions for natural movement, particularly in the absence of biomechanical models and objectives that can be used to achieve natural simulated locomotion [284, 146]. Naïve objectives for torque-actuated locomotion,

such as forward progress or maintaining a desired velocity, often produce gaits that exhibit extraneous motion of the limbs, asymmetric gaits, and other objectionable artifacts. To mitigate these artifacts, additional objectives such as effort or impact penalties have been used to discourage these undesirable behaviors. Crafting such objective functions requires a substantial degree of human insight, and often yields only modest improvements. Alternatively, recent RL methods based on the imitation of motion capture, such as GAIL [104], address the challenge of designing a reward function by using data to induce an objective. While this has been shown to improve the quality of the generated motions, current results still do not compare favorably to standard methods in computer animation [176]. The DeepLoco system [206] takes an approach similar to the one we use here, namely adding an imitation term to the reward function, although with significant limitations. It uses fixed initial states and is thus not capable of highly dynamic motions; it is demonstrated only on locomotion tasks defined by foot-placement goals computed by a high-level controller; and it is applied to a single armless biped model. Lastly, the multi-clip demonstration involves a hand-crafted procedure for selecting suitable target clips for turning motions.

Motion Imitation: Imitation of reference motions has a long history in computer animation. An early instantiation of this idea was in bipedal locomotion with planar characters [244, 249], using controllers tuned through policy search. Model-based methods for tracking reference motions have also been demonstrated for locomotion with 3D humanoid characters [302, 185, 145]. Reference motions have also been used to shape the reward function for deep RL to produce more natural locomotion gaits [203, 206] and for flapping flight [293]. In our work, we demonstrate the capability to perform a significantly broader range of difficult motions: highly dynamic spins, kicks, and flips with intermittent ground contact, and we show that reference-state initialization and early termination are critical to their success. We also explore several options for multi-clip integration and skill sequencing.

The work most reminiscent of ours in terms of capabilities is the Sampling-Based Controller (SAMCON) [161, 160]. An impressive array of skills has been reproduced by SAMCON, and to the best of our knowledge, SAMCON has been the only system to demonstrate such a diverse corpus of highly dynamic and acrobatic motions with simulated characters. However, the system is complex, having many components and iterative steps, and requires defining a low dimensional state representation for the synthesized linear feedback structures. The resulting controllers excel at mimicking the original reference motions, but it is not clear how to extend the method for task objectives, particularly if they involve significant sensory input. A more recent variation introduces deep Q-learning to train a high-level policy that selects from a precomputed collection of SAMCON control fragments [159]. This provides flexibility in the order of execution of the control fragments, and is demonstrated to be capable of challenging non-terminating tasks, such as balancing on a bongo-board and walking on a ball. In this work, we propose an alternative framework using deep RL, that is conceptually much simpler than SAMCON, but is nonetheless able to learn highly dynamic and acrobatic skills, including those having task objectives and multiple clips.

3.2 Overview

Our system receives as input a character model, a corresponding set of kinematic reference motions, and a task defined by a reward function. It then synthesizes a controller that enables the character to imitate the reference motions, while also satisfying task objectives, such as striking a target or running in a desired direction over irregular terrain. Each reference motion is represented as a sequence of target poses $\{\hat{\mathbf{q}}_t\}$. A control policy $\pi(\mathbf{a}_t|\mathbf{s}_t, \mathbf{g}_t)$ maps the state of the character \mathbf{s}_t , a task-specific goal \mathbf{g}_t to an action \mathbf{a}_t , which is then used to compute torques to be applied to each of the character’s joints. Each action specifies target angles for proportional-derivative (PD) controllers that then produce the final torques applied at the joints. The reference motions are used to define an imitation reward $r^I(\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1})$, and the goal defines a task-specific reward $r^G(\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1}, \mathbf{g}_t)$. The final result of our system is a policy that enables a simulated character to imitate the behaviours from the reference motions while also fulfilling the specified task objectives. The policies are modeled using neural networks and trained using the proximal policy optimization algorithm [238].

3.3 Policy Representation

Given a reference motion clip, represented by a sequence of target poses $\{\hat{\mathbf{q}}_t\}$, the goal of the policy is to reproduce the desired motion in a physically simulated environment, while also satisfying additional task objectives. Since a reference motion only provides kinematic information in the form of target poses, the policy is responsible for determining which actions should be applied at each time step in order to realize the desired trajectory.

States and Actions

The state \mathbf{s} uses a similar set of features as those detailed in Chapter 2 to describe the configuration of the character’s body. Since the target poses from the reference motions vary with time, a phase variable $\phi \in [0, 1]$ is also included among the state features. $\phi = 0$ denotes the start of a motion, and $\phi = 1$ denotes the end. For cyclic motions, ϕ is reset to 0 after the end of each cycle. Policies trained to achieve additional task objectives, such as walking in a particular direction or hitting a target, are also provided with a goal \mathbf{g} . Specific goal representations used in the experiments are discussed in section 3.6. The policy is queried at $30Hz$, and the actions \mathbf{a} from the policy specify target rotations for PD controllers positioned at each joint.

Network

Each policy π is represented by a neural network that maps a given state \mathbf{s} and goal \mathbf{g} to a distribution over action $\pi(\mathbf{a}|\mathbf{s}, \mathbf{g})$. The action distribution is modeled as a Gaussian, with a state dependent mean $\mu(\mathbf{s})$ specified by the network, and a fixed diagonal covariance matrix

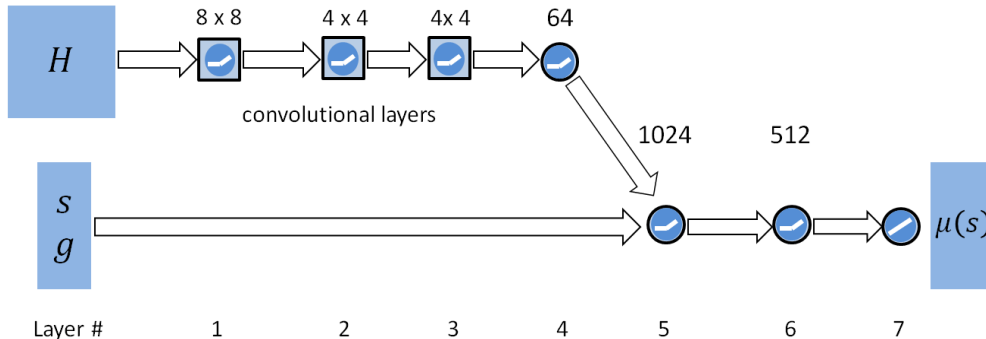


Figure 3.2: Schematic illustration of the visuomotor policy network. The heightmap H is processed by 3 convolutional layers with 16 8×8 filters, 32 4×4 filters, and 32 4×4 filters. The feature maps are then processed by 64 fully-connected units. The resulting features are concatenated with the input state s and goal g and processed by two fully-connected layers with 1024 and 512 units. The output $\mu(s)$ is produced by a layer of linear units. ReLU activations are used for all hidden layers. For tasks that do not require a heightmap, the networks consist only of layers 5-7.

Σ that is treated as a hyperparameter of the algorithm:

$$\pi(\mathbf{a}|\mathbf{s}) = \mathcal{N}(\mu(\mathbf{s}), \Sigma). \quad (3.1)$$

The inputs are processed by two fully-connected layers with 1024, and 512 units each, followed by a linear output layer. ReLU activations are used for all hidden units. The value function is modeled by a similar network, with exception of the output layer, which consists of a single linear unit.

For vision-based tasks, discussed in section 3.6, the inputs are augmented with a heightmap H of the surrounding terrain, sampled on a uniform grid around the character. The policy and value networks are augmented accordingly with convolutional layers to process the heightmap. A schematic illustration of this visuomotor policy network is shown in Figure 3.2. The heightmap is first processed by a series of convolutional layers, followed by a fully-connected layer. The resulting features are then concatenated with the input state \mathbf{s} and goal \mathbf{g} , and processed by a similar fully-connected network as the one used for tasks that do not require vision.

Reward

The reward r_t at each step t consists of two terms that encourage the character to match the reference motion while also satisfying additional task objectives:

$$r_t = \omega^I r_t^I + \omega^G r_t^G. \quad (3.2)$$

Here, r_t^I and r_t^G represent the imitation and task objectives, with ω^I and ω^G being their respective weights. The task objective r_t^G incentivizes the character to fulfill task-specific objectives, the details of which will be discussed in the following section. The imitation objective r_t^I encourages the character to follow a given reference motion $\{\hat{\mathbf{q}}_t\}$. It is further decomposed into terms that reward the character for matching certain characteristics of the reference motion, such as joint orientations and velocities, as follows:

$$\begin{aligned} r_t^I &= w^p r_t^p + w^v r_t^v + w^e r_t^e + w^c r_t^c \\ w^p &= 0.65, \quad w^v = 0.1, \quad w^e = 0.15, \quad w^c = 0.1 \end{aligned} \quad (3.3)$$

The pose reward r_t^p encourages the character to match the joint orientations of the reference motion at each step, and is computed as the difference between the joint orientation quaternions of the simulated character and those of the reference motion. In the equation below, q_t^j and \hat{q}_t^j represent the orientations of the j th joint from the simulated character and reference motion respectively, $q_1 \ominus q_2$ denotes the quaternion difference, and $\|q\|$ computes the scalar rotation of a quaternion about its axis in radians:

$$r_t^p = \exp \left[-2 \left(\sum_j \|\hat{q}_t^j \ominus q_t^j\|^2 \right) \right]. \quad (3.4)$$

The velocity reward r_t^v is computed from the difference of local joint velocities, with \dot{q}_t^j being the angular velocity of the j th joint. The target velocity \hat{q}_t^j is computed from the data via finite difference.

$$r_t^v = \exp \left[-0.1 \left(\sum_j \|\hat{q}_t^j - \dot{q}_t^j\|^2 \right) \right]. \quad (3.5)$$

The end-effector reward r_t^e encourages the character’s hands and feet to match the positions from the reference motion. Here, \mathbf{p}_t^e denotes the 3D world position in meters of end-effector $e \in [\text{left foot, right foot, left hand, right hand}]$:

$$r_t^e = \exp \left[-40 \left(\sum_e \|\hat{\mathbf{p}}_t^e - \mathbf{p}_t^e\|^2 \right) \right]. \quad (3.6)$$

Finally, r_t^c penalizes deviations in the character’s center-of-mass \mathbf{p}_t^c from that of the reference motion $\hat{\mathbf{p}}_t^c$:

$$r_t^c = \exp \left[-10 \left(\|\hat{\mathbf{p}}_t^c - \mathbf{p}_t^c\|^2 \right) \right]. \quad (3.7)$$

3.4 Training

Our policies are trained with PPO using the clipped surrogate objective [238]. We maintain two networks, one for the policy $\pi(\mathbf{a}|\mathbf{s}, \mathbf{g})$ and another for the value function $V(\mathbf{s}, \mathbf{g})$. Training

proceeds episodically, where at the start of each episode, an initial state \mathbf{s}_0 is sampled uniformly from the reference motion (section 3.4), and rollouts are generated by sampling actions from the policy at every step. Each episode is simulated to a fixed time horizon or until a termination condition has been triggered (section 3.4). Once a batch of data has been collected, the policy and value function are updated using PPO according to Algorithm 1.

One of the persistent challenges in RL is the problem of exploration. Since most formulations assume an unknown MDP, the agent is required to use its interactions with the environment to infer the structure of the MDP and discover high value states that it should endeavor to reach. A number of algorithmic improvements have been proposed to improve exploration, such as using metrics for novelty or information gain [19, 109, 71]. However, less attention has been placed on the structure of the episodes during training and their potential as a mechanism to guide exploration. In the following sections, we consider two design decisions, the initial state distribution and the termination condition, which have often been treated as fixed properties of a given RL problem. We will show that appropriate choices are crucial for allowing our method to learn challenging skills such as highly-dynamic kicks, spins, and flips. With common default choices, such as a fixed initial state and fixed-length episodes, we find that imitation of these difficult motions is often unsuccessful.

Initial State Distribution

The initial state distribution $p(\mathbf{s}_0)$ determines the states in which an agent begins each episode. A common choice for $p(\mathbf{s}_0)$ is to always place the agent in a fixed state. However, consider the task of imitating a desired motion. A simple strategy is to initialize the character to the starting state of the motion, and allow it to proceed towards the end of the motion over the course of an episode. With this design, the policy must learn the motion in a sequential manner, by first learning the early phases of the motion, and then incrementally progressing towards the later phases. Before mastering the earlier phases, little progress can be made on the later phases. This can be problematic for motions such as backflips, where learning the landing is a prerequisite for the character to receive a high return from the jump itself. If the policy cannot land successfully, jumping will actually result in *worse* returns. Another disadvantage of a fixed initial state is the resulting exploration challenge. The policy only receives reward retrospectively, once it has visited a state. Therefore, until a high-reward state has been visited, the policy has no way of learning that this state is favorable. Both disadvantages can be mitigated by modifying the initial state distribution.

For many RL tasks, a fixed initial state can be more convenient, since it can be challenging to initialize the agent in other states (e.g., physical robots) or obtain a richer initial state distribution. For motion imitation tasks, however, the reference motion provides a rich and informative state distribution, that can be leveraged to guide the agent during training. At the start of each episode, a state can be sampled from the reference motion, and used to initialize the state of the agent. We will refer to this strategy as *reference state initialization* (RSI). Similar strategies have been previously used for planar bipedal walking [244] and manipulation [186, 223]. By sampling initial states from the reference motion, the

agent encounters desirable states along the motion, even before the policy has acquired the proficiency needed to reach those states. For example, consider the challenge of learning to perform a backflip. With a fixed initial state, in order for the character to discover that performing a full rotation mid-air will result in high returns, it must first learn to perform a carefully coordinated jump. However, for the character to be motivated to perform such a jump, it must be aware that the jump will lead to states that yield higher rewards. Since the motion is highly sensitive to the initial conditions at takeoff, many strategies will result in failure. Thus the agent is unlikely to encounter states from a successful flip, and never discover such high reward states. With RSI, the agent immediately encounters such promising states during the early stages of training. Instead of accessing information from the reference motion only through the reward function, RSI can be interpreted as an additional channel through which the agent can access information from the reference motion in the form of a more informative initial state distribution.

Early Termination

For cyclic skills, the task can be modeled as an infinite horizon MDP. But during training, each episode is simulated for a finite horizon. An episode terminates either after a fixed period of time, or when certain termination conditions have been triggered. For locomotion, a common condition for early termination (ET) is the detection of a fall, characterized by the character’s torso making contact with the ground [202] or certain links falling below a height threshold [100]. While these strategies are prevalent, they are often mentioned in passing and their impact on performance has not been well evaluated. In this work, we will use a similar termination condition as [202], where an episode is terminated whenever certain links, such as the torso or head, makes contact with the ground. Once early termination has been triggered, the character is left with zero reward for the remainder of the episode. This instantiation of early termination provides another means of shaping the reward function to discourage undesirable behaviors. Another advantages of early termination is that it can function as a curating mechanism that biases the data distribution in favor of samples that may be more relevant for a task. In the case of skills such as walking and flips, once the character has fallen, it can be challenging for it to recover and return to its nominal trajectory. Without early termination, data collected during the early stages of training will be dominated by samples of the character struggling on the ground in vain, and much of the capacity of the network will be devoted to modeling such futile states. This phenomena is analogous to the class imbalance problem encountered by other methodologies such as supervised learning. By terminating the episode whenever such failure states are encountered, this imbalance can be mitigated.

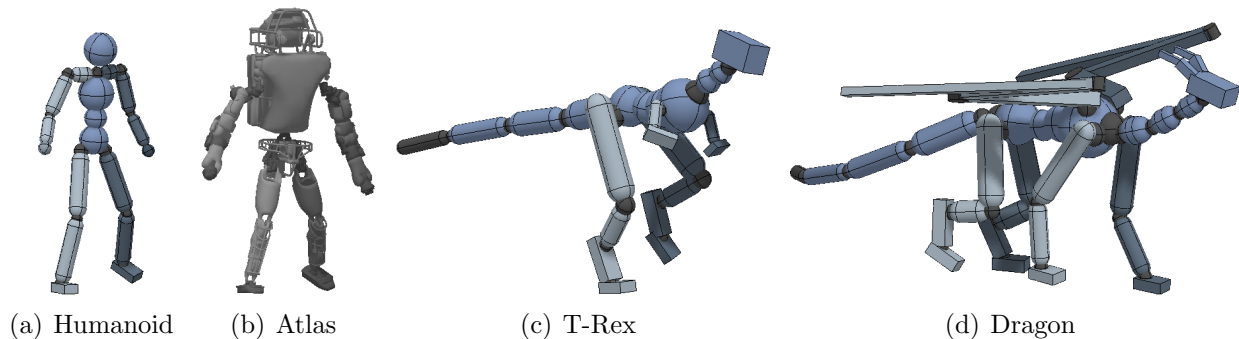


Figure 3.3: 3D simulated characters. Our framework is able to train policies for a wide range of character morphologies.

Property	Humanoid	Atlas	T-Rex	Dragon
Links	13	12	20	32
Total Mass (kg)	45	169.8	54.5	72.5
Height (m)	1.62	1.82	1.66	1.83
Degrees of Freedom	34	31	55	79
State Features	227	212	355	482
Action Parameters	28	25	49	73

Table 3.1: Properties of the characters.

3.5 Characters

Our characters include a 3D humanoid, an Atlas robot model, a T-Rex, and a dragon. Illustrations of the characters are available in Figure 3.3, and Table 3.1 details the properties of each character. Both the humanoid and Atlas share similar body structures, but their morphology (e.g., mass distribution) and actuators (e.g., PD gains and torque limits) differ significantly, with the Atlas being almost four times the mass of the humanoid. The T-Rex and dragon provide examples of learning behaviors for characters from keyframed animation when no mocap data is available, and illustrate that our method can be readily applied to non-bipedal characters. The humanoid character has a 227D state space and a 28D action space. Our most complex character, the dragon, has a 482D state space and 73D action space. Compared to standard continuous control benchmarks for RL [28], which typically have action spaces varying between 3D to 17D, our characters have significantly higher-dimensional action spaces.

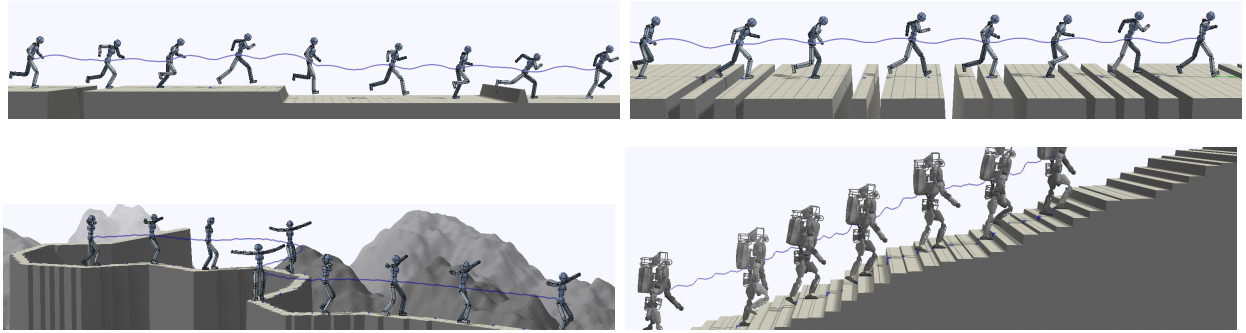


Figure 3.4: Characters traversing randomly generated terrains. **Top-to-bottom:** mixed obstacles, dense gaps, winding balance beam, stairs. The blue line traces the trajectory of the character’s center-of-mass.

3.6 Tasks

In addition to imitating a set of motion clips, the policies can be trained to perform a variety of tasks while preserving the style prescribed by the reference motions. The task-specific behaviors are encoded into the task objective r_t^G . We describe the tasks evaluated in our experiments below.

Target Heading: Steerable controllers can be trained by introducing an objective that encourages the character to travel in a target direction \mathbf{d}_t^* , represented as a 2D unit vector in the horizontal plane. The reward for this task is given by

$$r_t^G = \exp \left[-2.5 \max(0, \mathbf{v}^* - \mathbf{v}_t^T \mathbf{d}_t^*)^2 \right], \quad (3.8)$$

where \mathbf{v}^* specifies the desired speed along the target direction \mathbf{d}_t^* , and \mathbf{v}_t represents the center-of-mass velocity of the simulated character. The objective therefore penalizes the character for traveling slower than the desired speed along the target direction, but does not penalize it for exceeding the requested speed. The target direction is provided as the input goal $\mathbf{g}_t = \mathbf{d}_t^*$ to the policy. During training, the target direction is randomly varied throughout an episode. At runtime, \mathbf{d}_t^* can be manually specified to steer the character.

Strike: In this task, the character’s goal is to strike a randomly placed spherical target using specific links, such as the feet. The reward is given by

$$r_t^G = \begin{cases} 1, & \text{target has been hit} \\ \exp[-4\|\mathbf{p}_t^{\text{tar}} - \mathbf{p}_t^e\|^2], & \text{otherwise} \end{cases} \quad (3.9)$$

$\mathbf{p}_t^{\text{tar}}$ specifies the location of the target, and \mathbf{p}_t^e represents the position of the link used to hit the target. The target is marked as being hit if the center of the link is within 0.2m of

the target location. The goal $\mathbf{g}_t = (\mathbf{p}_t^{\text{tar}}, h)$ consists of the target location $\mathbf{p}_t^{\text{tar}}$ and a binary variable h that indicates if the target has been hit in a previous time step. As we are using feedforward networks for all policies, h acts as a memory for the state of the target. The target is randomly placed within a distance of $[0.6, 0.8]m$ from the character, the height is sampled randomly between between $[0.8, 1.25]m$, and the initial direction from the character to the target varies by $2rad$. The target location and h are reset at the start of each cycle. The memory state h can be removed by training a recurrent policy, but our simple solution avoids the complexities of training recurrent networks while still attaining good performance.

Throw: This task is a variant of the strike task, where instead of hitting a target with one of the character’s links, the character is tasked with throwing a ball to the target. At the start of an episode, the ball is attached to the character’s hand via a spherical joint. The joint is released at a fixed point in time during the episode. The goal \mathbf{g}_t and reward r_t^G is the same as the strike task, but the character state s_t is augmented with the position, rotation, linear and angular velocity of the ball. The distance of the target varies between $[2.5, 3.5]m$, with height between $[1, 1.25]m$, and direction direction between $[0.7, 0.9]rad$.

Terrain Traversal: In this task, the character is trained to traverse obstacle-filled environments. The goal \mathbf{g}_t and task-objective r_t^G are similar to those of the target heading task, except the target heading is fixed along the direction of forward progress.

We consider four environments consisting of mixed obstacles, dense gaps, a winding balance beam, and stairs. Figure 3.4 illustrates examples of the environments. The mixed obstacles environment is composed of gap, step, and wall obstacles similar to those presented in [202]. Each gap has a width between $[0.2, 1]m$, the height of each wall varies between $[0.25, 0.4]m$, and each step has a height between $[0.35, -0.35]m$. The obstacles are interleaved with flat stretches of terrain between $[5, 8]m$ in length. The next environment consists of sequences of densely packed gaps, where each sequence consists of 1 to 4 gaps. The gaps are $[0.1, 0.3]m$ in width, with $[0.2, 0.4]m$ of separation between adjacent gaps. Sequences of gaps are separate by $[1, 2]m$ of flat terrain. The winding balance beam environment sports a narrow winding path carved into irregular terrain. The width of the path is approximately $0.4m$. Finally, we constructed a stairs environment, where the character is to climb up irregular steps with height varying between $[0.01, 0.2]m$ and a depth of $0.28m$. Since the mixed obstacles and dense gaps environments follow a linear layout, the heightmaps are represented by a 1D heightfield with 100 samples spanning $10m$. In the winding balance beam environment, a 32×32 heightmap is used, covering a $3.5 \times 3.5m$ area.

3.7 Results

The motions from the trained policies are best seen in the supplemental videos¹. Snapshots of the skills performed by the simulated characters are available in Figure 3.5, 3.6, and 3.7. The policies are executed at 30Hz. Physics simulation is performed at 1.2kHz using the

Bullet physics engine [30]. All neural networks are built and trained using TensorFlow. The characters’ motions are driven by torques computed using stable PD controllers [257]. Policy updates are performed after a batch of $m = 4096$ samples has been collected. Minibatches of size $n = 256$ are then sampled from the data for each gradient step. A discount factor $\gamma = 0.95$ is used for all motions. $\lambda = 0.95$ is used for both TD(λ) and GAE(λ). The PPO likelihood ratio clipping threshold is set to $\epsilon = 0.2$. A stepsize of $\alpha_v = 10^{-2}$ is used for the value function. A policy step size of $\alpha_\pi = 5 \times 10^{-5}$ is used for the humanoid and Atlas, and $\alpha_\pi = 2 \times 10^{-5}$ for the dragon and T-Rex. Once gradients have been computed, the network parameters are updated using stochastic gradient descent with momentum 0.9. Humanoid policies for imitating individual skills typically require about 60 million samples to train, requiring about 2 days on an 8-core machine.

Results for the humanoid are demonstrated for a large collection of locomotion, acrobatic, and martial arts skills, while the results for the dragon and T-Rex are demonstrated for locomotion. Each skill is learned from approximately 0.5-5s of mocap data collected from <http://mocap.cs.cmu.edu> and <http://mocap.cs.sfu.ca>. For characters such as the T-Rex and dragon, where mocap data is not available, we demonstrate that our framework is also capable of learning skills from artist-authored keyframes. Before being used for training, the clips are manually processed and retargeted to their respective characters. A comprehensive list of the learned skills and performance statistics is available in Table 3.2. Each environment is denoted by “Character: Skill - Task”. The task is left unspecified for policies that are trained solely to imitate a reference motion without additional task objectives. Performance is measured by the average return normalized by the minimum and maximum possible return per episode. Note that the maximum return may not be achievable. For example, for the throwing task, the maximum return requires moving the ball instantaneously to the target. When evaluating the performance of the policies, early termination is applied and the state of the character at the start of each episode is initialized via RSI. The weights for the imitation and task objectives are set to $\omega^I = 0.7$ and $\omega^G = 0.3$ for all tasks. More details regarding hyperparameter settings are available in the supplemental material.

By encouraging the policies to imitate motion capture data from human subjects, our system is able to learn policies for a rich repertoire of skills. For locomotion skills such as walking and running, our policies produce natural gaits that avoid many of the artifacts exhibited by previous deep RL methods [237, 176]. The humanoid is able to learn a variety of acrobatic skills with long flight phases, such as backflips and spinkicks, which are particularly challenging since the character needs to learn to coordinate its motion in mid-air. The system is also able to reproduce contact-rich motions, such as crawling and rolling, as well as motions that require coordinated interaction with the environment, such as the vaulting skills shown in Figure 3.6. The learned policies are robust to significant external perturbation and generate plausible recovery behaviors. The policies trained for the T-Rex and dragon demonstrate that the system can also learn from artist generated keyframes when mocap is not available and scale to much more complex characters than those that have been demonstrated by previous work using deep RL.

Skill	T_{cycle} (s)	N_{samples} (10^6)	NR
Backflip	1.75	72	0.729
Balance Beam	0.73	96	0.783
Baseball Pitch	2.47	57	0.785
Cartwheel	2.72	51	0.804
Crawl	2.93	68	0.932
Dance A	1.62	67	0.863
Dance B	2.53	79	0.822
Frontflip	1.65	81	0.485
Getup Face-Down	3.28	49	0.885
Getup Face-Up	4.02	66	0.838
Headspin	1.92	112	0.640
Jog	0.80	51	0.951
Jump	1.77	86	0.947
Kick	1.53	50	0.854
Landing	2.83	66	0.590
Punch	2.13	60	0.812
Roll	2.02	81	0.735
Run	0.80	53	0.951
Sideflip	2.44	64	0.805
Spin	4.42	191	0.664
Spinkick	1.28	67	0.748
Vault 1-Handed	1.53	41	0.695
Vault 2-Handed	1.90	87	0.757
Walk	1.26	61	0.985
Atlas: Backflip	1.75	63	0.630
Atlas: Run	0.80	48	0.846
Atlas: Spinkick	1.28	66	0.477
Atlas: Walk	1.26	44	0.988
T-Rex: Walk	2.00	140	0.979
Dragon: Walk	1.50	139	0.990

Table 3.2: Performance statistics of imitating various skills. All skills are performed by the humanoid unless stated otherwise. Policies are trained only to imitate a reference motion without additional task objectives. T_{cycle} is the length of the clip. N_{samples} specifies the number of samples collected to train the final policy. NR represents the normalized return of the final policy averaged over 32 episodes, with 0 being the minimum possible return per episode, and 1 the maximum return. For cyclic skills, each episode has a horizon of 20s. For acyclic skills, the horizon is specified by T_{cycle} .

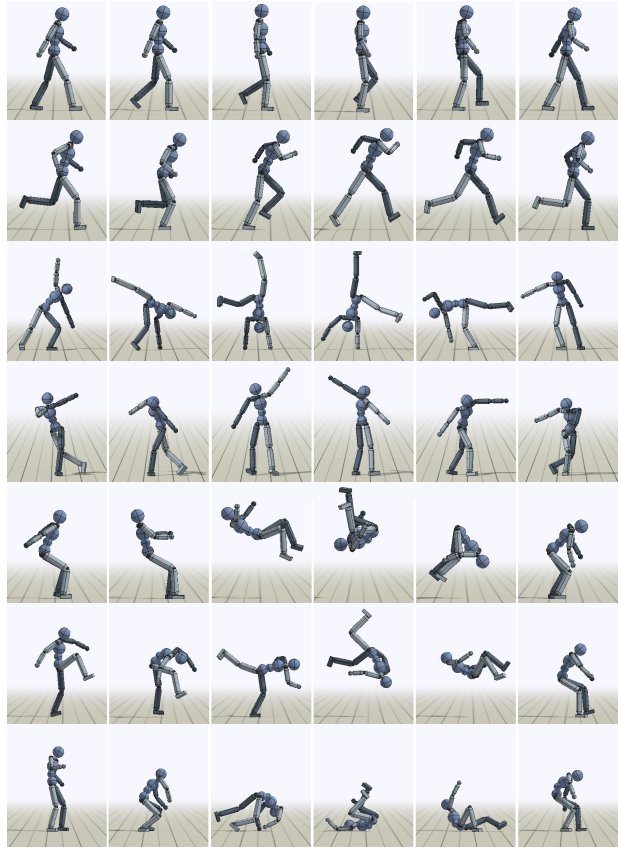


Figure 3.5: Snapshots of motions from the trained policies. **Top-to-bottom:** walk, run, cartwheel, dance A, backflip, frontflip, roll.

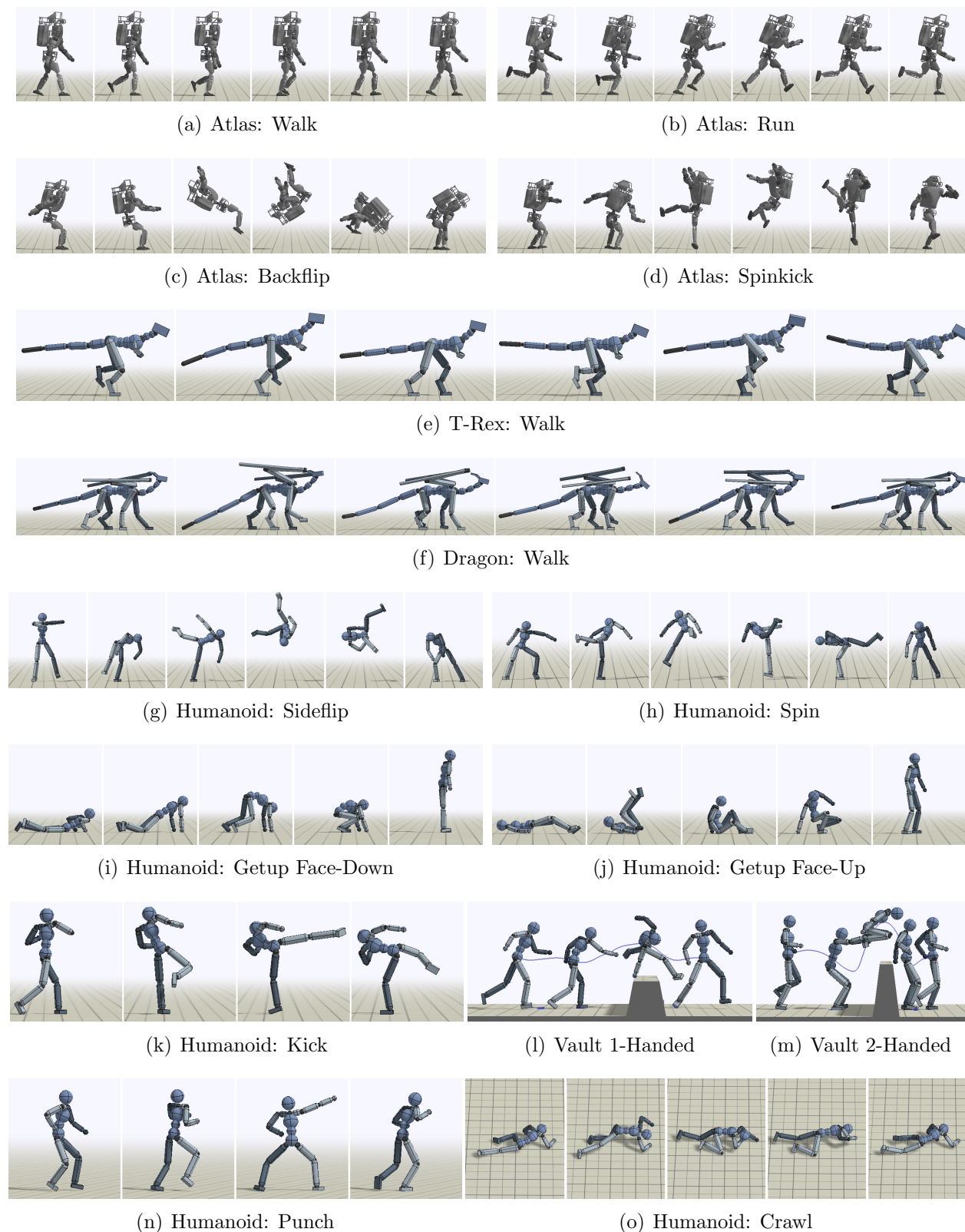
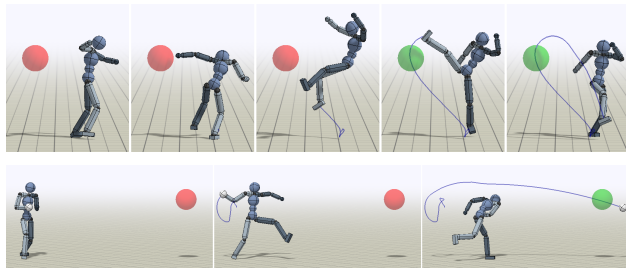


Figure 3.6: Simulated characters performing various skills. Our framework is able to train policies for a broad range of characters, skills, and environments.



The figure consists of two rows of motion clips. The top row shows a character performing a spinkick, with a red ball on the left and a green ball on the right. The bottom row shows a character performing a baseball pitch, with a red ball on the left and a green ball on the right. The clips are arranged in a grid format.

Environment	$N_{samples}$ (10^6)	NR
Humanoid: Walk - Target Heading	85	0.911
Humanoid: Jog - Target Heading	108	0.876
Humanoid: Run - Target Heading	40	0.637
Humanoid: Spinkick - Strike	85	0.601
Humanoid: Baseball Pitch - Throw	221	0.675
Humanoid: Run - Mixed Obstacles	466	0.285
Humanoid: Run - Dense Gaps	265	0.650
Humanoid: Winding Balance Beam	124	0.439
Atlas: Walk - Stairs	174	0.808

Figure 3.7: **Top:** Spinkick policy trained to strike a target with the character’s right foot. **Bottom:** Baseball pitch policy trained to throw a ball to a target.

Tasks

In addition to imitating reference motions, the policies can also adapt the motions as needed to satisfy additional task objectives, such as following a target heading and throwing a ball to a randomly placed target. Performance statistics for each task are available in Table 3.3. To investigate the extent to which the motions are adapted for a particular task, we compared the performance of policies trained to optimize both the imitation objective r^I and the task objective r^G to policies trained only with the imitation objective. Table 3.4 summarizes the success rates of the different policies. For the throwing task, the policy trained with both objectives is able to hit the target with a success rate of 75%, while the policy trained only to imitate the baseball pitch motion is successful only in 5% of the trials. Similarly, for the strike task, the policy trained with both objectives successfully hits 99% of the targets, while the policy trained only to imitate the reference motion has a success rate of 19%. These results suggest that simply imitating the reference motions is not sufficient to successfully perform the tasks. The policies trained with the task objective are able to deviate from the original reference motion and developing additional strategies to satisfy their respective goals. We further trained policies to optimize only the task objective, without imitating a reference motion. The resulting policies are able to fulfill the task objectives, but without a reference motion, the policies develop unnatural behaviors, similar to those produced by prior deep RL methods. For the throw task, instead of throwing the ball, the policy adopts an awkward but functional strategy of running towards the target with the ball. Figure 3.8 illustrates this behavior.

Retargeting

Due to modeling discrepancies between simulation and the real world, the dynamics under which a motion capture clip was recorded can differ dramatically from the dynamics of the

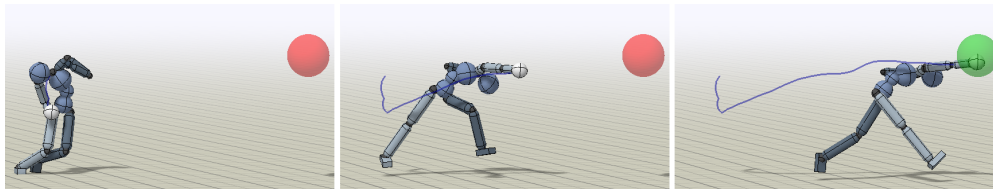


Figure 3.8: Policy trained for the throw task without a reference motion. Instead of throwing the ball, the character learns to run towards the target.

Environment	$r^I + r^G$	r^I	r^G
Humanoid: Strike - Spinkick	99%	19%	55%
Humanoid: Baseball Pitch - Throw	75%	5%	93%

Table 3.4: Success rate of policies trained with the imitation or task objectives disabled. Each policy is evaluated over 100 trials. Simply imitating the reference motions proves insufficient for fulfilling the task objectives. Training without a reference motion produces policies that develop awkward, but functional, strategies for satisfying the task objectives.

simulated environments. Furthermore, keyframed motions may not be physically correct at all. To evaluate our framework’s robustness to these discrepancies, we trained policies to perform similar skills with different character models, environments, and physics.

Character Retargeting: To demonstrate the system’s capabilities in retargeting motions to different characters, we trained policies for walking, running, backflips and spinkicks on a simulated model of the Atlas robot from <http://www.mujoco.org/forum/index.php?resources/atlas-v5.16/>. The Atlas has a total mass of $169.8kg$, almost four times the mass of the humanoid, as well as a different mass distribution. The serial 1D revolute joints in the original Atlas model are aggregated into 3D spherical joints to facilitate simpler retargeting of the reference motions. To retarget the motion clips, we simply copied the local joint rotations from the humanoid to the Atlas, without any further modification. New policies are then trained for the Atlas to imitate the retargeted clips. Despite the starkly different character morphologies, our system is able to train policies that successfully reproduce the various skills with the Atlas model. Performance statistics of the Atlas policies are available in Table 3.2. The performance achieved by the Atlas policies are comparable to those achieved by the humanoid. Qualitatively, the motions generated by the Atlas character closely resemble the reference clips. To further highlight the differences in the dynamics of the characters, we evaluated the performance of directly applying policies trained for the humanoid to the Atlas. The humanoid policies, when applied to the Atlas, fail to reproduce any of the skills, achieving a normalized return of 0.013 and 0.014 for the run and backflip, compared to 0.846 and 0.630 achieved by policies that were trained specifically for the Atlas but using the same reference clips.

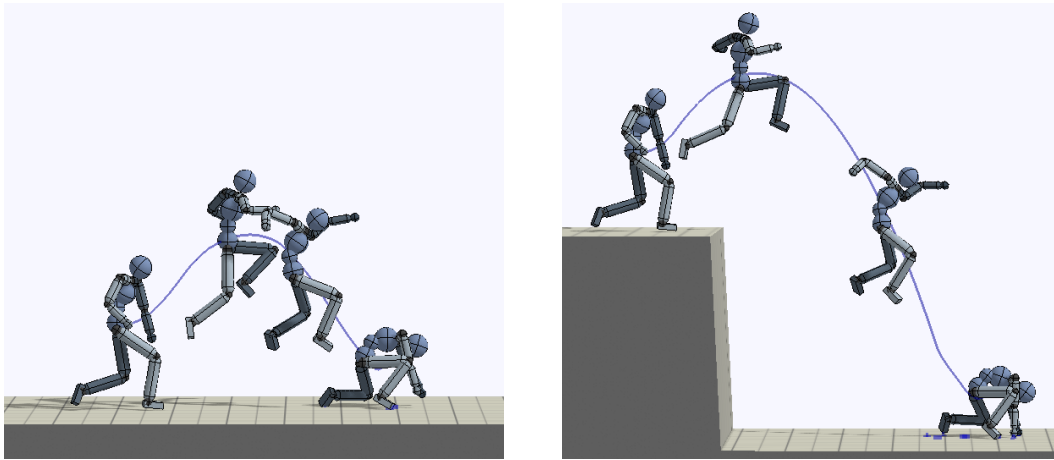


Figure 3.9: **Left:** Original landing motion on flat terrain. **Right:** Policy trained to imitating landing motion while jumping down from a $2m$ ledge. Despite being provided only with a motion recorded on flat terrain, the policy is able to adapt the skill to jump down from a tall ledge.

Environment Retargeting: While most of the reference motions were recorded on flat terrain, we show that the policies can be trained to adapt the motions to irregular environments. First, we consider the case of retargeting a landing motion, where the original reference motion is of a human subject jumping and landing on flat ground. From this reference motion, we trained a character to reproduce the motion while jumping down from a $2m$ ledge. Figure 3.9 illustrates the motion from the final policy. The system was able to adapt the reference motion to a new environment that is significantly different from that of the original clip.

Next, we explore vision-based locomotion in more complex procedurally generated environments. By augmenting the networks with a heightmap input, we are able to train the humanoid to run across terrains consisting of random obstacles. Examples of the environments are available in Figure 3.4. Over the course of training, the policies are able to adapt a single clip of a forward running motion into a variety of strategies for traversing across the different classes of obstacles. Furthermore, by training a policy to imitate a balance beam walk, the character learns to follow a narrow winding path using only a heightmap for pathfinding. The balance beam policy was trained with only a forward walking clip, but is nonetheless able to develop turning motions to follow the winding path. In addition to the humanoid, we also trained a policy for the Atlas to climb up stairs with irregular step heights. The policy was able to adapt the original walking clip on flat terrain to climb the steps, although the resulting motion still exhibits an awkward gait. We suspect that the problem is partly related to the walking reference motion being ill-suited for the stairs environment.

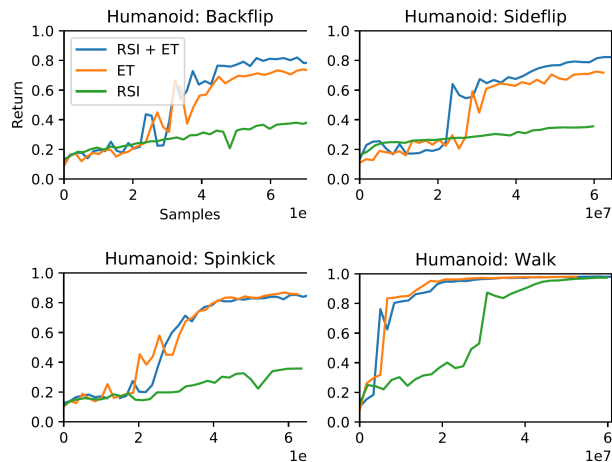


Figure 3.10: Learning curves for policies trained with and without reference state initialization (RSI) and early termination (ET).

Skill	RSI + ET	ET	RSI
Backflip	0.791	0.730	0.379
Sideflip	0.823	0.717	0.355
Spinkick	0.848	0.858	0.358
Walk	0.980	0.981	0.974

Figure 3.11: Normalized average returns of policies trained with and without reference state initialization (RSI) and early termination (ET).

Physics Retargeting: To further evaluate the framework’s robustness to discrepancies between the dynamics of the motion capture data and simulation, we trained policies to perform a spinkick and cartwheel under moon gravity ($1.622m/s^2$). Despite the difference in gravity, the policies were able to adapt the motions to the new dynamics, achieving a return of 0.792 for the spinkick and 0.688 for the cartwheel.

Ablations

To evaluate the impact of our design decisions, we compare our full method against alternative training schemes that disable some of the components. We found that the reference state initialization and early termination are two of the most important components of our training procedure. The comparisons include training with reference state initialization and with a fixed initial state, as well as training with early termination and without early termination. Without early termination, each episode is simulated for the full 20s. Figure 3.10 compares the learning curves with the different configurations and Table 3.11 summarizes the performance of the final policies. During evaluation, early termination is disabled and the character is initialized to a fixed state at the start of each episode. Due to the time needed to train each policy, the majority of performance statistics are collected from one run of the training process. However, we have observed that the behaviors are consistent across multiple runs. Early termination proves to be crucial for reproducing many of the skills. By heavily penalizing the character for making undesirable contacts with the ground, early termination helps to eliminate local optima, such as those where the character falls and mimes the motions as it lies on the ground. RSI also appears vital for more dynamic skills that have significant flight phases, such as the backflip. While the policies trained without RSI appear to achieve a similar return to those trained with RSI, an inspection

Skill	Forward (N)	Sideway (N)
Backflip	440	100
Cartwheel	200	470
Run	720	300
Spinkick	690	600
Walk	240	300

Table 3.5: Maximum forwards and sideways push each policy can tolerate before falling. Each push is applied to the character’s pelvis for $0.2s$.

of the resulting motions show that, without RSI, the character often fails to reproduce the desired behaviours. For the backflip, without RSI, the policy never learns to perform a full mid-air flip. Instead, it performs a small backwards hop while remaining upright.

Robustness

To determine the policies’ robustness to external perturbations, we subjected the trained policies to perturbation forces and recorded the maximum force the character can tolerate before falling. The perturbation forces are applied halfway through a motion cycle to the character’s pelvis for $0.2s$. The magnitude of the force is increased by $10N$ until the character falls. This procedure is repeated for forces applied along the forward direction (x-axis) and sideway direction (z-axis). Table 3.5 summarizes the results from the experiments on the humanoid. The learned policies show comparable-or-better robustness than figures reported for SAMCON [160]. The run policy is able to recover from $720N \times 0.2s$ forward pushes, while the spin-kick policy is able to survive $600N \times 0.2s$ perturbations in both directions. Note that no external perturbations are applied during the training process; we suspect that the policies’ robustness is in large part due to the exploration noise applied by the stochastic policy used during training.

3.8 Discussion

In this chapter, we presented a data-driven deep reinforcement learning framework for training control policies for simulated characters. We show that our method can replicate a wide range of challenging skills. The resulting policies are highly robust and produce natural motions that are nearly indistinguishable from the original motion capture data in the absence of perturbations. Our framework is also able to retarget skills to a variety of characters, environments, and tasks.

Although our experiments illustrate the flexibility of this approach, there are still numerous limitations to be addressed in future work. First, our policies require a phase variable to be synchronized with the reference motion, which advances linearly with time. This limits the ability of the policy to adjust the timing of the motion, and lifting this limitation could

produce more natural and flexible perturbation recoveries. The learning process itself is also quite time consuming, often requiring several days per skill, and is performed independently for each policy. Although we use the same imitation reward across all motions, this is still currently based on a manually defined state-similarity metric. The weighting of the reward terms also needs to be defined with some care. It would also be interesting to understand the learned control strategies and compare them to the equivalent human strategies.

Chapter 4

Video Imitation

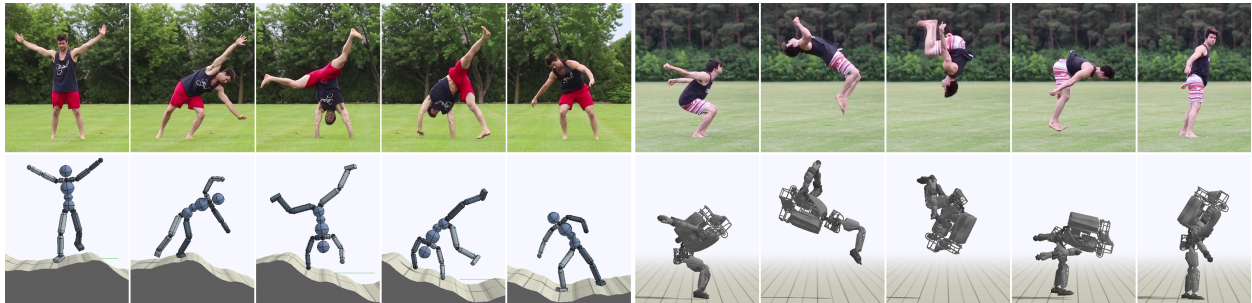


Figure 4.1: Simulated characters performing highly dynamic skills learned by imitating video clips of human demonstrations. **Left:** Humanoid performing cartwheel B on irregular terrain. **Right:** Backflip A retargeted to a simulated Atlas robot. (Video¹)

Motion capture (mocap) is one of the most popular methods for acquiring motion data, and is the crucial source of reference motions for our work in Chapter 3. However, the acquisition of mocap data can pose major hurdles for practitioners, often requiring heavily instrumented environments and actors. The infrastructure required to procure such data can be prohibitive, and some activities remain exceedingly difficult to motion capture, such as large-scale outdoor sports. A more abundant and flexible source of motion data is monocular video. A staggering 300 hours of video is uploaded to YouTube every minute [16]. Searching and querying video sources on the web can quickly yield a large number of clips for any desired activity or behavior. However, it is a daunting challenge to extract the necessary motion information from monocular video frames, and the quality of the motions generated by previous methods still falls well behind the best mocap-based animation systems [278].

In this chapter, we propose a method for acquiring dynamic character controllers directly from monocular video through a combination of pose estimation and deep reinforcement learning. Recent advances with deep learning techniques have produced breakthrough

¹ Supplementary video: <https://xbpeng.github.io/projects/SFV/>

results for vision-based 3D pose estimation from monocular images [119]. However, pose estimation alone is not yet sufficient to produce high-fidelity and physically plausible motions: frequent errors and physical inconsistencies in the estimated poses accumulate and result in unnatural character behaviors. Motion imitation with reinforcement learning provides a powerful tool for acquiring skills from videos while remaining robust to such imperfections. By reproducing the skill in a physical simulation, the learning process can refine imperfect and noisy pose sequences, compensate for missing frames, and take into account the physical constraints of the character and environment. By bringing together deep pose estimation and reinforcement learning, we propose a framework that enables simulated characters to learn a diverse collection of dynamic and acrobatic skills directly from video demonstrations.

The primary contribution of this chapter is a system for learning character controllers from video clips that integrates pose estimation and reinforcement learning. To make this possible, we introduce a number of extensions to both the pose tracking system and the reinforcement learning algorithm. We propose a motion reconstruction method that improves the quality of reference motions to be more amenable for imitation by a simulated character. We further introduce a novel reinforcement learning method that incorporates *adaptive* state initialization, where the initial state distribution is dynamically updated to facilitate long-horizon performance in reproducing a desired motion. We find that this approach for dynamic curriculum generation substantially outperforms standard methods when learning from lower-fidelity reference motions constructed from video tracking sequences. Our framework is able to reproduce a significantly larger repertoire of skills and higher fidelity motions from videos than has been demonstrated by prior methods. The effectiveness of our framework is evaluated on a large set of challenging skills including dances, acrobatics, and martial arts. Our system is also able to retarget video demonstrations to widely different morphologies and environments. Figure 1 illustrates examples of the skills learned by our framework. While our framework is able to reproduce a substantially larger corpus of skills than previous methods, there remains a large variety of video clips that our system is not yet able to imitate. We include a discussion of these challenges and other limitations that arise from the various design decisions.

4.1 Related Work

Our work lies at the intersection of pose estimation and physics-based character animation. The end goal of our system is to produce robust and naturalistic controllers that enable virtual characters to perform complex skills in physically simulated environments. Facets of this problem have been studied in a large body of prior work, from techniques that have sought to produce realistic skills from first principles (i.e. physics and biomechanics) [44, 284, 281], to methods that incorporate reference motion data into the controller construction process [246, 144, 161]. These techniques can synthesize motions kinematically [144, 148, 107] or as the product of dynamic control in a physics simulation [146, 75]. Most data-driven methods, save for a few exceptions, are based on motion capture data, which often requires

costly instrumentation and pre-processing [108]. Raw video offers a potentially more accessible and abundant alternative source of motion data. While there has been much progress in the computer vision community in predicting human poses from monocular images or videos, integrating pose predictions from video with data-driven character animation still presents a number of challenges. Pose estimators can generally produce reasonable predictions of an actor’s motion, but they do not benefit from the manual cleanup and accurate tracking enjoyed by professionally recorded mocap data. Prior methods that learn from motion data often assume accurate reference motions as a vital component in the learning process. For example, during training, [207] reinitializes the character state to frames sampled from the reference motion. The effectiveness of these strategies tend to deteriorate in the presence of low-fidelity reference motions.

Reinforcement Learning: Many methods for acquiring character controllers utilize reinforcement learning [45, 283, 144, 148, 201]. The use of deep neural network models for RL has been demonstrated for a diverse array of challenging skills [57, 28, 202, 159, 223, 265]. While deep RL methods have been effective for motion control tasks, the policies are prone to developing unnatural behaviours, such as awkward postures, overly energetic movements, and asymmetric gaits [237, 176]. In order to mitigate these artifacts, additional auxiliary objectives such as symmetry, effort minimization, or impact penalties have been incorporated into the objective to discourage unnatural behaviors [309]. Designing effective objectives can require substantial human insight and may nonetheless fall short of eliminating undesirable behaviours. An alternative for encouraging more natural motions is to incorporate high-fidelity biomechanical models [284, 75, 146]. However, these models can be challenging to build, difficult to control, and may still result in unnatural behaviours. In light of these challenges, data-driven RL methods that utilize reference motion data have been proposed as an alternative [293, 207]. Reference motion clips can be incorporated via a motion imitation objective that incentivizes the policy to produce behaviours that resemble the reference motions. In this chapter, we explore methods for extending motion imitation with RL to accommodate low-fidelity reference motions extracted from videos, and introduce a novel adaptive state initialization technique that makes this practical even for highly dynamic and acrobatic movements.

Monocular Human Pose Estimation: While mocap remains the most popular source of demonstrations, it typically requires significant instrumentation, which limits its accessibility. Practitioners therefore often turn to public databases to satisfy their mocap needs [43, 242]. Unfortunately, the volume of publicly available mocap data is severely limited compared to datasets in other disciplines, such as ImageNet [53]. Alternatively, video clips are an abundant and accessible source of motion data. While recovering motion from raw video has been a long standing challenge [139, 26], recently deep learning approaches have made rapid progress in this area.

Performance of 2D pose estimation improved rapidly after Toshev and Szegedy [273] introduced a deep learning approach for predicting the 2D coordinates of joints directly from images. This is followed by methods that predict joint locations as a spatial heat map [271, 288, 191]. In this work we build upon the recent OpenPose framework [33], which extends previous methods for real-time multi-person 2D pose estimation. Monocular 3D pose estimation is an even more challenging problem due to depth ambiguity, which traditional methods resolve with strong priors [263, 315, 24]. The introduction of large-scale mocap datasets [113] with ground truth 3D joint locations allowed for the development of deep learning based methods that directly estimate 3D joint locations from images [316, 173, 199]. However, mocap datasets are typically captured in heavily instrumented environments, and models trained on these datasets alone do not generalize well to the complexity of images of humans *in the wild*. Therefore, recent methods focus on weakly supervised techniques, where a model may also be trained on images without ground truth 3D pose [226, 318]. Note that most approaches only estimate the 3D joint *locations* and not the 3D rotations of a kinematic tree, which is necessary to serve as reference for our RL algorithm. Methods that predict joint locations require additional post-processing to recover the joint rotations through inverse kinematics [173]. Only a handful of techniques directly estimate the 3D human pose as 3D joint rotations [317, 275, 119]. Although there are methods that utilize video sequences as input [266], most state-of-the-art approaches predict the pose independently for each video frame. Recently Xu et al. [298] propose a method that recovers a temporally consistent trajectory from monocular video by an additional optimization step in the 3D pose space. However, their method requires a pre-acquired template mesh of the actor and hence cannot be applied to legacy videos, such as those available from YouTube. In this work we build on the recent work of Kanazawa et al. [119], which is a weakly-supervised deep learning framework that trains a model to directly predict the 3D pose, as joint rotations, from a single image. A more detailed discussion is available in Section 4.3.

Video Imitation: The problem of learning controllers from monocular video has received modest attention from the computer graphics community. The work most related to ours is the previous effort by Vondrak et al. [278], which demonstrated learning bipedal controllers for walking, jumping, and handsprings from videos. The controllers were represented as a finite-state machines (FSM), where the structure of the FSM and the parameters at each state were learned through an incremental optimization process. Manually-crafted balance strategies and inverse-dynamics models were incorporated into the control structure within each state of the FSM. To imitate the motion of the actor in a video, the controllers were trained by optimizing a 2D silhouette likelihood computed between the actor and simulated character. To resolve depth ambiguity, they incorporated a task-specific pose prior computed from mocap data. While the system was able to synthesize controllers for a number of skills from video demonstrations, the resulting motions can appear robotic and the use of a silhouette likelihood can neglect a significant amount of task-relevant information in the video. Furthermore, the task-pose priors require access to mocap clips that are similar to the

skills being learned. If such data is already available, it might be advantageous to imitate the mocap clips instead. Similarly, Coros et al. [46] utilized video clips of canine motions to train quadruped controllers, where the reference motions were extracted via manually annotating gait graphs and marker locations.

In this work, we take advantage of state-of-the-art 3D pose estimation techniques to extract full-body 3D reference motions from video, which resolves much of the depth ambiguity inherent in monocular images and improves the motion quality of the learned controllers. Deep RL enables the use of simple but general control structures that can be applied to a substantially wider range of skills, including locomotion, acrobatics, martial arts, and dancing. To the best of our knowledge, the only prior work that has demonstrated learning full-body controllers from monocular video is the work by Vondrak et al. [278]. Although incorporating reinforcement learning to imitate video demonstrations is conceptually natural, in practice it presents a number of challenges arising from nonphysical behaviours and other artifacts due to inaccurate pose estimation.

4.2 Overview

Our framework receives as input a video clip and a simulated character model. It then synthesizes a controller that enables a physically simulated character to perform the skill demonstrated by the actor in the video. The resulting policies are robust to significant perturbations, can be retargeted to different characters and environments, and are usable in interactive settings. The learning process is divided into three stages: pose estimation, motion reconstruction, and motion imitation. A schematic illustration of the framework is available in Figure 4.2. The input video is first processed by the pose estimation stage, where a learned 2D and 3D pose estimators are applied to extract the pose of the actor in each frame. Next, the set of predicted poses proceeds to the motion reconstruction stage, where a reference motion trajectory $\{\mathbf{q}_t^*\}$ is optimized such that it is consistent with both the 2D and 3D pose predictions, while also enforcing temporal-consistency between frames and mitigating other artifacts present in the original set of predicted poses. The reference motion is then utilized in the motion imitation stage, where a control policy π is trained to enable the character to reproduce the reference motion in a physically simulated environment. The pose estimator is trained with a weakly-supervised learning approach, and the control policy is trained with reinforcement learning using a motion imitation objective.

4.3 Background

Pose estimation: Our approach builds upon the recent 2D and 3D pose estimators, OpenPose [288] and Human Mesh Recovery (HMR) [119] respectively. OpenPose performs both detection and 2D pose estimation of humans from a single image. It outputs the 2D pose as joint locations $\mathbf{x}_j \in \mathbb{R}^2$ in the image coordinate space, as well as a confidence score for

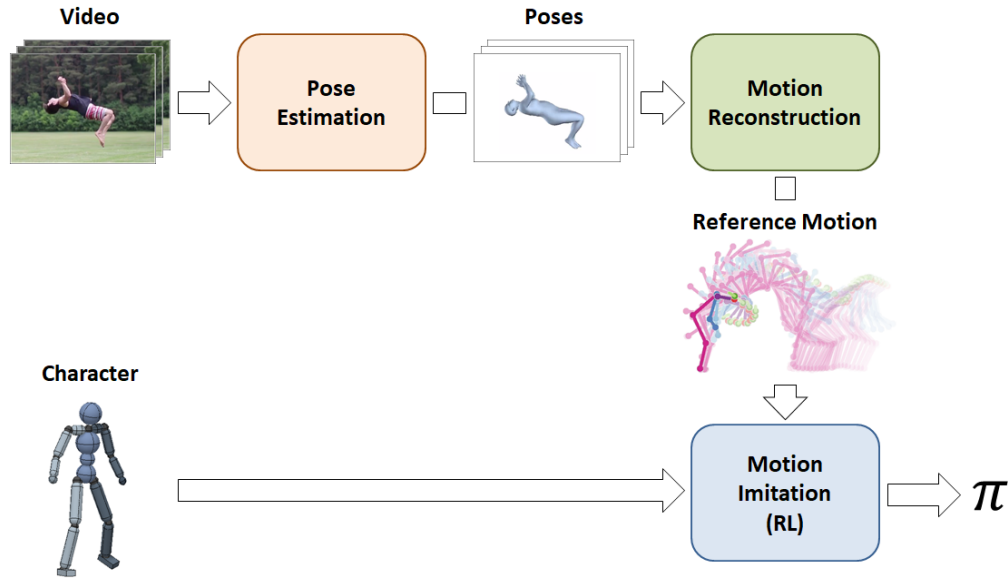


Figure 4.2: The pipeline consists of three stages: pose estimation, motion reconstruction, and imitation. It receives as input, a video clip of an actor performing a particular skill and a simulated character model, and outputs a control policy that enables the character to reproduce the skill in simulation.

each joint $c_j \in \mathbb{R}$. HMR is a recent approach that directly predicts the 3D pose and shape of a human model [163], along with the camera configuration from an image of a localized person. The predicted 3D pose $\mathbf{q} = \{\mathbf{q}_j\}$ is parameterized by the local rotation of each joint \mathbf{q}_j , represented in axis-angle form with respect to the parent link’s coordinate frame. The world transformation of the root, designated to be the pelvis, is obtained using the predicted weak-perspective camera Π . The 3D pose is predicted by first encoding an image I into a $2048D$ latent space $\mathbf{z} = f(I)$ via a learned encoder f . The latent features are then decoded by a learned decoder $\mathbf{q}(\mathbf{z})$ to produce the pose. HMR uses a weakly-supervised adversarial framework that allows the model to be trained on images with only 2D pose annotations, *without* any ground truth 3D labels. Therefore, it can be trained on datasets of in-the-wild images, such as COCO [156], and sports datasets [118], which is vital for learning acrobatic skills from video clips.

4.4 Pose Estimation

Given a video clip, the role of the pose estimation stage is to predict the pose of the actor in each frame. Towards this goal, there are two main challenges for our task. First, the acrobatic skills that we wish to imitate exhibit challenging poses that vary significantly from the distribution of common poses available in most datasets. Second, poses are predicted independently for each frame, and therefore may not be temporally consistent, especially for

dynamic motions. We address these challenges by leveraging an ensemble of pose estimators and a simple but effective data augmentation technique that substantially improve the quality of the predictions.

One of the challenges of tracking acrobatic movements is that they tend to exhibit complex poses with wildly varying body orientations (e.g. flips and spins). These poses are typically underrepresented in existing datasets, which are dominated by everyday images of humans in upright orientations. Thus, off-the-shelf pose estimators struggle to predict the poses in these videos. To compensate for this discrepancy, we augment the standard datasets with rotated versions of the existing images, where the rotations are sampled uniformly between $[0, 2\pi]$. We found that training the pose estimators with this augmented dataset, substantially improves performance for acrobatic poses. Once trained, both estimators are applied independently to every frame to extract a 2D pose trajectory $\{\hat{\mathbf{x}}_t\}$ and 3D pose trajectory $\{\hat{\mathbf{q}}_t\}$. Note that the 2D pose \hat{x}_t consists only of the 2D screen coordinates of the actor’s joints, but tends to be more accurate than the 3D predictions. Examples of the predictions from the different pose estimators are shown in Figure 4.3. The independent predictions from the pose estimators are then consolidated in the motion reconstruction stage to produce the final reference motion.

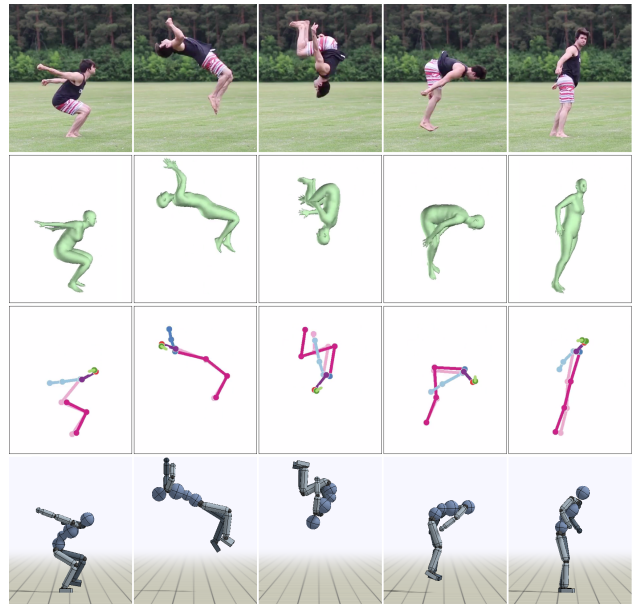


Figure 4.3: Comparison of the motions generated by different stages of the pipeline for Backflip A. **Top-to-Bottom:** Input video clip, 3D pose estimator, 2D pose estimator, simulated character.

4.5 Motion Reconstruction

Since poses are predicted independently for every frame in the pose estimation stage, simply sequencing the poses into a trajectory tends to produce motions that exhibit artifacts due to inconsistent predictions across adjacent frames (see supplementary video¹). The motion artifacts often manifest as nonphysical behaviours in the reference motion, such as high-frequency jitter and sudden changes in pose. These artifacts can hinder the simulated character’s ability to reproduce the intended motion. The role of the motion reconstruction stage is to take advantage of the predictions from the two pose estimators to reconstruct a new kinematic trajectory that reconciles the individual predictions and mitigates artifacts, such that the resulting reference motion is more amenable for imitation.

Specifically, given the predictions from the 2D and 3D pose estimators, we optimize a 3D

pose trajectory that consolidates their predictions while also enforcing temporal consistency between adjacent frames. Instead of directly optimizing in the 3D pose space, we take advantage of the encoder-decoder structure of the 3D pose estimator and optimize the 3D pose trajectory in the latent pose space \mathbf{z}_t , which captures the manifold of 3D human poses [119]. The final 3D reference motion is constructed by optimizing a trajectory $Z = \{\mathbf{z}_t\}$ in the latent space to minimize the reconstruction loss l_{rec} :

$$\begin{aligned} l_{\text{rec}}(Z) &= w_{2\text{D}}l_{2\text{D}}(Z) + w_{3\text{D}}l_{3\text{D}}(Z) + w_{\text{sm}}l_{\text{sm}}(Z) \\ w_{2\text{D}} &= 10, \quad w_{3\text{D}} = 100, \quad w_{\text{sm}} = 25, \end{aligned} \quad (4.1)$$

The 2D consistency loss $l_{2\text{D}}$ minimizes the reprojection error between the predicted 2D joint locations and the 2D projections of the corresponding joints from the pose specified by z_t ,

$$l_{2\text{D}} = \sum_t \sum_j c_{t,j} \|\hat{\mathbf{x}}_{t,j} - \Pi[F_j(\mathbf{q}(\mathbf{z}_t))]\|_1, \quad (4.2)$$

where $\hat{x}_{t,j}$ is the predicted 2D location of the j th joint, $c_{t,j}$ is the confidence of the prediction, and $F_j[\cdot]$ is the forward kinematics function that computes the 3D position of joint j given the 3D pose. $q(z_t)$ represents the pose decoded from z_t , and $\Pi[\cdot]$ is the weak-perspective projection that transforms 3D positions to 2D screen coordinates.

The 3D consistency loss $l_{3\text{D}}$ encourages the optimized trajectory to stay close to the initial 3D prediction $\hat{\mathbf{q}}_t$:

$$l_{3\text{D}} = \sum_t w_t \text{dist}(\hat{\mathbf{q}}_t, \mathbf{q}(\mathbf{z}_t)), \quad (4.3)$$

where $\text{dist}(\cdot, \cdot)$ measures the distance between two rotations by the angle of the difference rotation. $w_t = \exp(-\delta_t)$ estimates the confidence of the initial 3D prediction using the difference between the initial 2D and 3D predictions, computed via the reprojection error $\delta_t = \sum_j c_{t,j} \|\hat{\mathbf{x}}_{t,j} - \Pi F_j(\hat{\mathbf{q}}_t)\|_2$. This ensures that initial 3D poses that are consistent with the 2D predictions are preserved, while inconsistent poses are adjusted through the other terms in the loss.

Finally, the smoothness loss l_{sm} encourages smoothness of the 3D joint positions between adjacent frames

$$l_{\text{sm}} = \sum_t \sum_j \|F_j(\mathbf{q}(\mathbf{z}_t)) - F_j(\mathbf{q}(\mathbf{z}_{t+1}))\|_2^2. \quad (4.4)$$

After the optimization process, we obtain the final 3D reference motion $\{\mathbf{q}_t^*\} = \{\mathbf{q}(\mathbf{z}_t^*)\}$.

4.6 Motion Imitation with RL

Once the reference motion has been reconstructed, it progresses to the motion imitation stage, where the goal is to learn a policy π that enables the character to reproduce the

demonstrated skill in simulation. The reference motion extracted by the previous stages is used to define an imitation objective, and a policy is then trained through reinforcement learning to imitate the given motion. The policy is modeled as a feedforward network that receives as input the current state \mathbf{s} and outputs an action distribution $\pi(\mathbf{a}|\mathbf{s})$. To train the policy, we propose a variant of proximal policy optimization (PPO) [238] augmented with an adaptive initial state distribution, as described below.

Initial State Distribution

The initial state distribution $p(\mathbf{s}_0)$ determines the states at which an agent starts each episode. Careful choice of $p(\mathbf{s}_0)$ can have a significant impact on the performance of the learned policy, as it can mitigate the challenges of exploration inherent in RL. An effective initial state distribution should expose the agent to promising states that are likely to maximize its expected return, thereby reducing the need for the agent to discover such states on its own. As we showed in Chapter 3, sampling initial states from the reference motion can be highly effective for reproducing dynamic motions. But the effectiveness of this strategy depends heavily on the quality of the reference motion. Due to artifacts from the pose estimator and modeling discrepancies between the simulated character and real-world actor, states sampled from the reference motion may not be ideal for reproducing the entirety of the motion. For example, a common artifact present in motion data recorded from real-world actors, be it through motion capture or vision-based pose estimation, is high-frequency jittering, which can manifest as initial states with large joint velocities. Naively initializing the agent to such states will then require the agent to recover from the artifacts of the reference motion. These artifacts can be substantially more pronounced in reference motions reconstructed from video. Though the motion reconstruction stage is able to mitigate many of these artifacts, some errors may still persist.

While a myriad of post-processing techniques can be applied to mitigate artifacts in the reference motion, we can instead reduce the dependency on the quality of the reference motion by learning an initial state distribution with the specific purpose of aiding the character in learning an effective controller. This can be formulated as a cooperative multi-agent reinforcement learning problem where the first agent, defined by the policy $\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$, controls the movement of the character, and the second agent $\rho_\omega(\mathbf{s}_0)$ proposes the initial states at which the character should start each episode. Both agents cooperate in order to maximize the multi-agent objective:

$$\begin{aligned} J(\theta, \omega) &= \mathbb{E}_{\tau \sim p_{\theta, \omega}(\tau)} \left[\sum_{t=0}^T \gamma^t r_t \right] \\ &= \int_{\tau} \left(\rho_\omega(\mathbf{s}_0) \prod_{t=0}^{T-1} p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t) \pi_\theta(\mathbf{a}_t|\mathbf{s}_t) \right) \left(\sum_{t=0}^T \gamma^t r_t \right) d\tau. \end{aligned} \quad (4.5)$$

Note that, since the reward requires tracking the entire reference motion (as discussed in the

next section), the initial state distribution cannot obtain the maximum return by “cheating” and providing excessively easy initializations. The maximum return is attained when the initial state distribution is close to the reference trajectory, but does not initialize the character in states from which recovery is impossible, as might be the case with erroneous states due to tracking error. The policy gradient of the initial state distribution $\rho_\omega(s_0)$ can be estimated according to:

$$\nabla_\omega J(\theta, \omega) = \mathbb{E}_{\tau \sim p_{\theta, \omega}(\tau)} \left[\nabla_\omega \log(\rho_\omega(s_0)) \sum_{t=0}^T \gamma^t r_t \right]. \quad (4.6)$$

Similar to the standard policy gradient for π , the gradient of the initial state distribution can be interpreted as increasing the likelihood of initial states that result in high returns. Unlike the standard policy gradient, which is calculated at every time step, the gradient of the initial state distribution is calculated only at the first time step. The discount factor captures the intuition that the effects of the initial state attenuates as the episode progresses. We will refer to this strategy of learning the initial state distribution as adaptive state initialization (ASI). Learning an initial state distribution can also be interpreted as a form of automatic curriculum generation, since $\rho_\omega(s_0)$ is incentivized to propose states that enable the character to closely follow the reference motion while avoiding states that may be too challenging for the current policy.

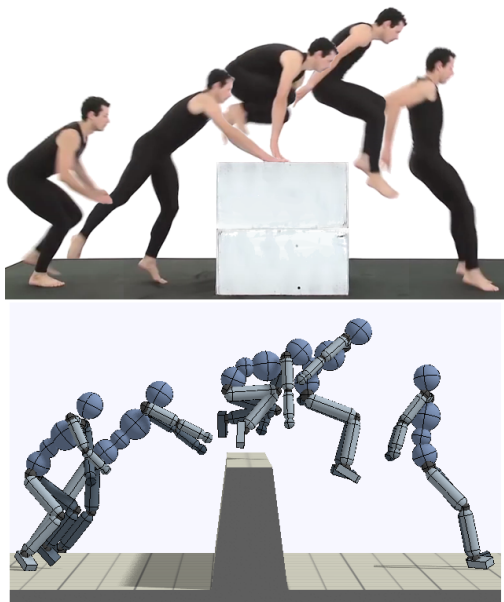


Figure 4.4: Character imitating a 2-handed vault.

4.7 Experimental Setup

Once a reference motion has been reconstructed from a video clip, training proceeds in a similar manner as Chapter 3, using the same reward function and network architecture. Our framework will be demonstrated with a humanoid character and a simulated Atlas robot.

Initial State Distribution: At the start of each episode, the character is initialized to a state \mathbf{s}_0 sampled from the initial state distribution $\rho_\omega(\mathbf{s}_0)$. When applying adaptive state initialization, $\rho_\omega(\mathbf{s}_0)$ is represented with a parametric model composed of independent Gaussian distributions over the character state. The Gaussian components are positioned at uniform points along the phase of the motion. To sample from this distribution, we first partition the state features $\mathbf{s} = [\hat{\mathbf{s}}, \phi]$, where ϕ is the phase variable and $\hat{\mathbf{s}}$ represents the other features. The distribution $\rho_\omega(\mathbf{s})$ is then factorized according to:

$$\rho_\omega(\mathbf{s}) = p_\omega(\hat{\mathbf{s}}|\phi)p(\phi), \quad (4.7)$$

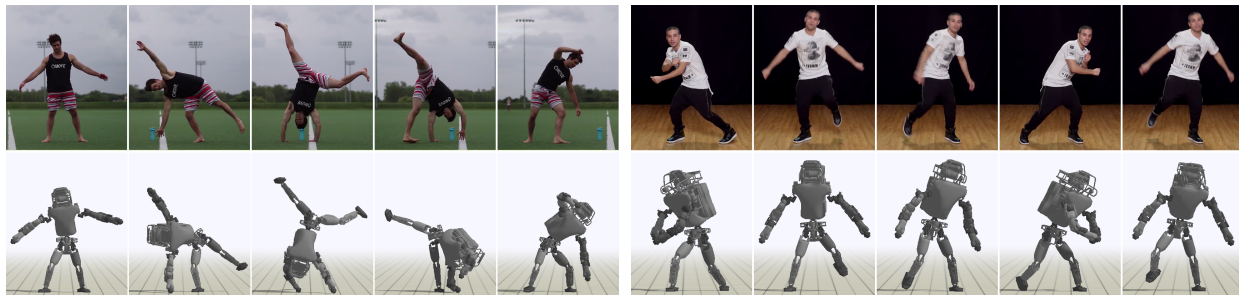


Figure 4.5: Simulated Atlas robot performing skills learned from video demonstrations. **Top:** Cartwheel A. **Bottom:** Dance.

with $p(\phi)$ being a uniform distribution over discrete phase values $[\phi^0, \phi^1, \dots, \phi^{k-1}]$. Each phase-conditioned state distribution $p_\omega(\hat{\mathbf{s}}|\phi^i)$, corresponding to ϕ^i , is modeled as a Gaussian $\mathcal{N}(\mu^i, \Sigma^i)$, with mean μ^i and diagonal covariance matrix Σ^i . The parameters of the initial state distribution consists of the parameters for each Gaussian component $\omega = \{\mu^i, \Sigma^i\}_{i=0}^{k-1}$. Both the mean and covariance matrix of each component are learned using policy gradients. When sampling an initial state, a phase value is first sampled from the discrete distribution $p(\phi)$. Next, $\hat{\mathbf{s}}$ is sampled from $p_\omega(\hat{\mathbf{s}}|\phi)$, and the combined features constitute the initial state.

4.8 Results

Motions from the trained policies can be seen in the supplementary video¹. Figures 4.4, 4.6, and 4.7 compare snapshots of the simulated characters with the original video clips. All video clips were collected from YouTube, and depict human actors performing various acrobatic stunts (e.g. flips and cartwheels) and locomotion skills (walking and running). The clips were selected such that the camera is primarily stationary over the course of the motion, and only a single actor is present in the scene. Each clip is trimmed to contain only the relevant portions of their respective motion, and depicts one demonstration of a particular skill.

Table 4.1 summarizes the performance of the final policies. Performance is recorded as the average normalized return over multiple episodes. As it is challenging to directly quantify the difference between the motion of the actor in the video and the simulated character, performance is evaluated with respect to the reconstructed reference motion. Since the reference motions recovered from the video clips may not be physically correct, a maximum return of 1 may not be achievable. Nonetheless, given a single video demonstration of each skill, the policies are able to reproduce a large variety of challenging skills ranging from contact-rich motions, such as rolling, to motions with significant flight phases, such as flips and spins. Policies can also be trained to perform skills that require more coordinated interactions with the environment, such as vaulting and pushing a large object.

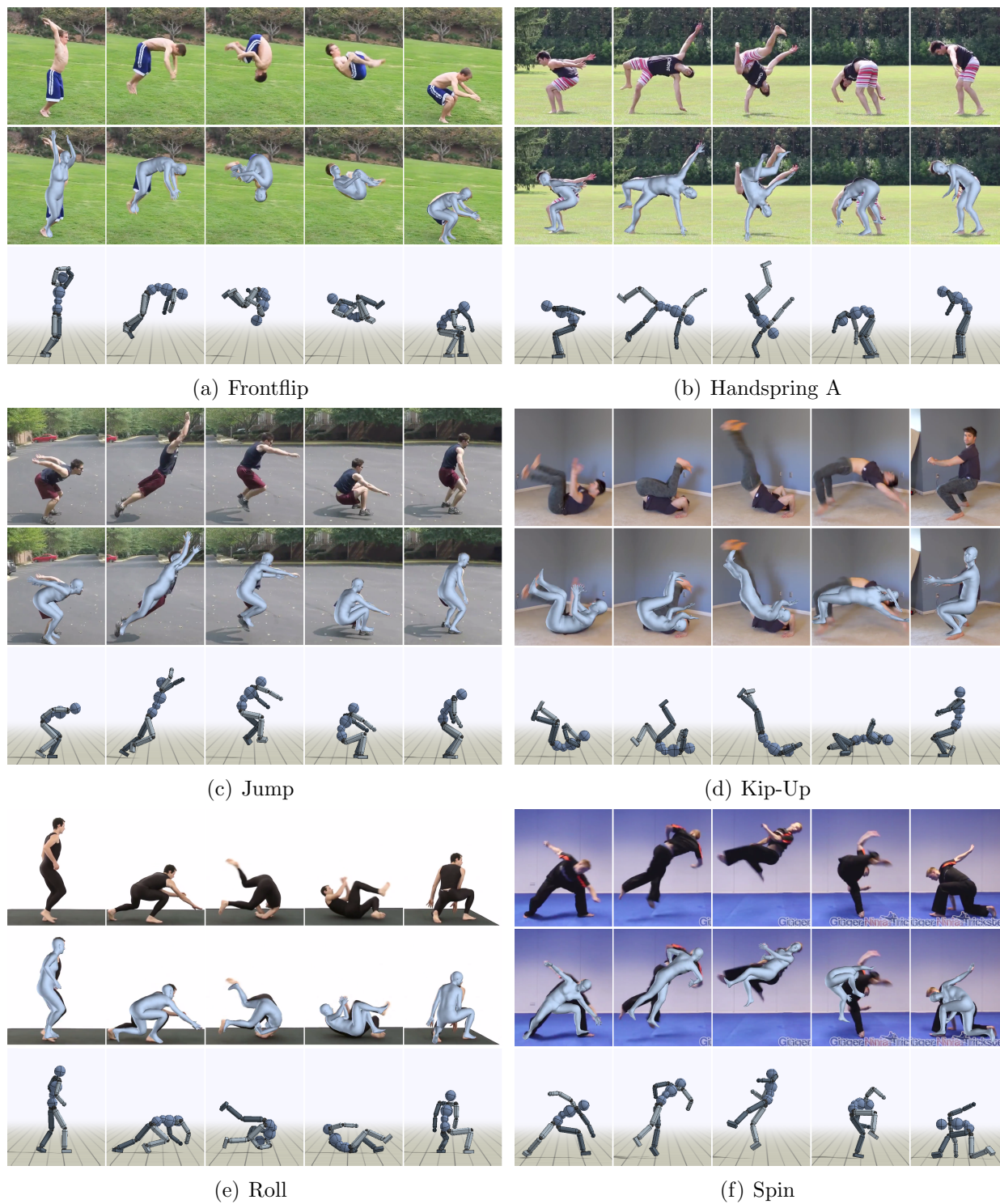


Figure 4.6: Simulated characters performing skills learned from video clips. **Top:** Video clip. **Middle:** 3D pose estimator. **Bottom:** Simulated character.

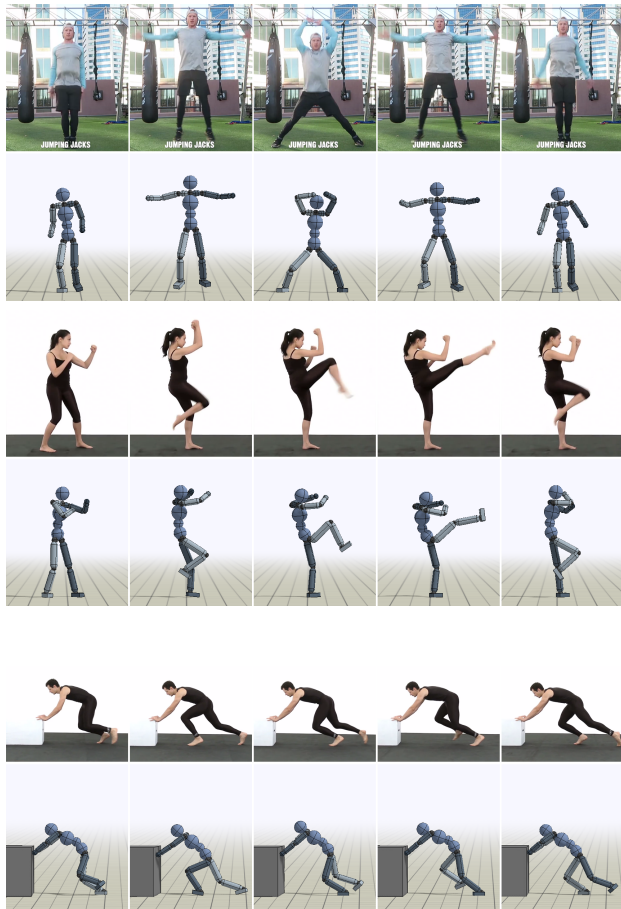


Figure 4.7: Humanoid character imitating skills from video demonstrations. **Top-to-Bottom:** Jumping jack, kick, push.

Retargeting: One of the advantages of physics-based character animation is its ability to synthesize behaviours for novel situations that are not present in the original data. In addition to reproducing the various skills, our framework is also able to retarget the skills to characters and environments that differ substantially from what is presented in the video demonstrations. Since the same simulated character is trained to imitate motions from different human actors, the morphology of the character tends to differ drastically from that of the actor. To demonstrate the system’s robustness to morphological discrepancies, we also trained a simulated Atlas robot to imitate a variety of video clips. The proportions of the Atlas’ limbs differ significantly from

Skill	T_{cycle} (s)	$N_{samples}$ (10^6)	NR
Backflip A	2.13	146	0.741
Backflip B	1.87	198	0.653
Cartwheel A	2.97	136	0.824
Cartwheel B	2.63	147	0.732
Dance	2.20	257	0.631
Frontflip	1.57	126	0.708
Gangnam Style	1.03	97	0.657
Handspring A	1.83	155	0.696
Handspring B	1.47	311	0.578
Jump	2.40	167	0.653
Jumping Jack	0.97	122	0.893
Kick	1.27	158	0.761
Kip-Up	1.87	123	0.788
Punch	1.17	115	0.831
Push	1.10	225	0.487
Roll	2.07	122	0.603
Run	0.73	126	0.878
Spin	1.07	146	0.779
Spinkick	1.87	196	0.747
Vault	1.43	107	0.730
Walk	0.87	122	0.932
Atlas: Backflip A	2.13	177	0.318
Atlas: Cartwheel A	2.97	174	0.456
Atlas: Dance	2.20	141	0.324
Atlas: Handspring A	1.83	115	0.360
Atlas: Jump	2.40	134	0.508
Atlas: Run	0.73	130	0.881
Atlas: Vault	1.43	112	0.752
Atlas: Walk	0.87	172	0.926

Table 4.1: Performance statistics of over 20 skills learned by our framework. T_{cycle} denotes the length of the clip. $N_{samples}$ records the number of samples collected to train each policy. NR represents the average normalized return of the final policy, with 0 and 1 being the minimum and maximum possible return per episode respectively. For cyclic skills, the episode horizon is set to 20s. For acyclic skills, the horizon is determined by T_{cycle} . All statistics are recorded from the humanoid character unless stated otherwise.

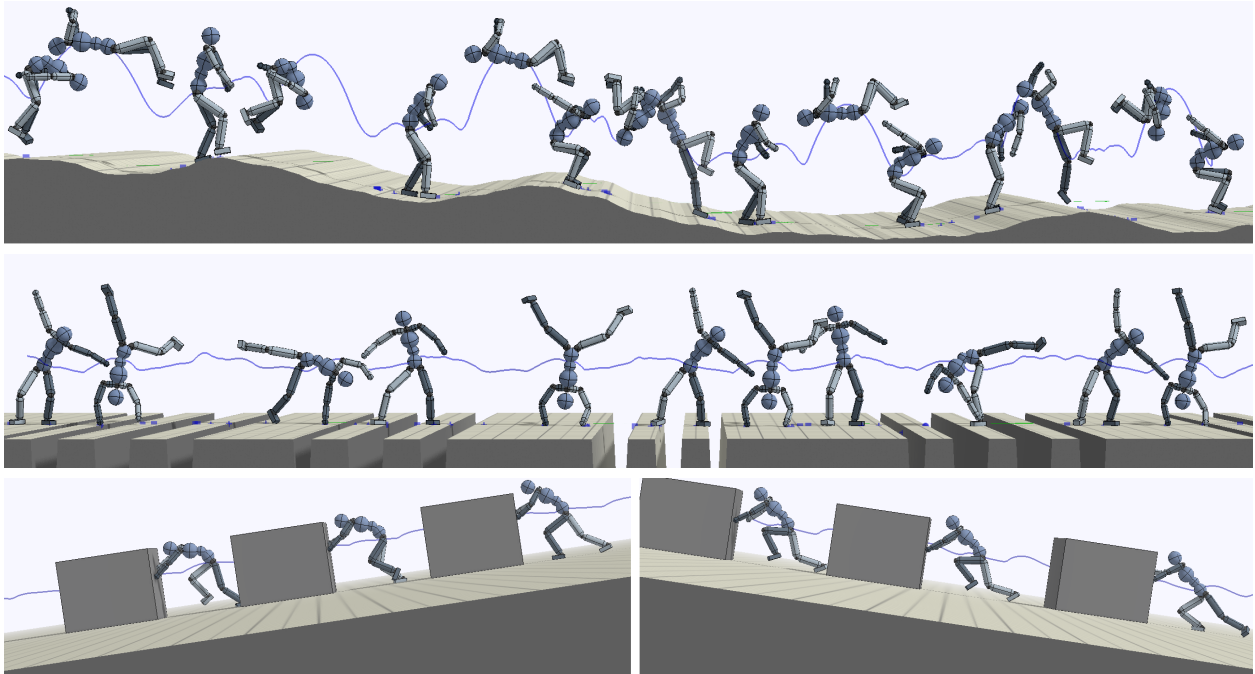


Figure 4.8: Skills retargeted to different environments. **Top-to-Bottom:** Backflip A across slopes, cartwheel B across gaps, pushing a box downhill and uphill.

normal human proportions, and with a weight of $169.5kg$, it is considerably heavier than the average human. Despite these drastic differences in morphology, our framework is able to learn policies that enable the Atlas to reproduce a diverse set of challenging skills. Table 4.1 summarizes the performance of the Atlas policies, and Figure 4.5 illustrates snapshots of the simulated motions.

In addition to retargeting to difference morphologies, the skills can also be adapted to different environments. While the video demonstrations were recorded on flat terrain, our framework is able to train policies to perform the skills on irregular terrain. Figure 4.8 highlights some of the skills that were adapted to environments composed of randomly generated slopes or gaps. The pushing skill can also be retargeted to push a $50kg$ box uphill and downhill with a slope of 15% . To enable the policies to perceive their environment, we follow the architecture used by Peng et al. [207], where a heightmap of the surrounding terrain is included in the input state, and the networks are augmented with corresponding convolutional layers to process the heightmap. Given a single demonstration of an actor performing a backflip on flat terrain, the policy is able to develop strategies for performing a backflip on randomly varying slopes. Similarly, the cartwheel policy learns to carefully coordinate the placement of the hands and feet to avoid falling into the gaps.

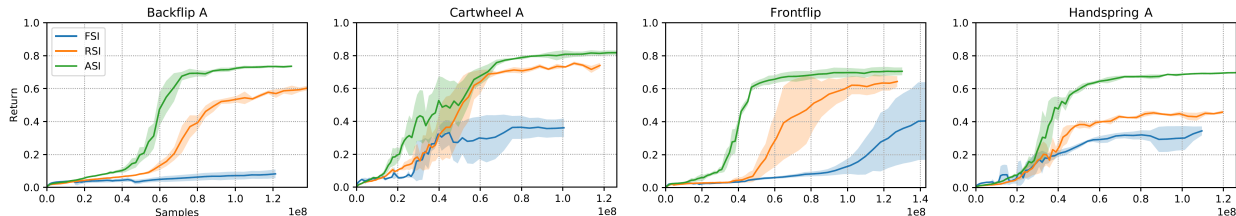


Figure 4.9: Learning curves comparing policies trained with fixed state initialization (FSI), reference state initialization (RSI), and adaptive state initialization (ASI). Three policies initialized with different random seeds are trained for each combination of skill and initial state distribution. Compared to its counterparts, ASI consistently improves performance and learning speed.

Initial State Distribution: To evaluate the impact of adaptive state initialization (ASI), we compare the performance of policies trained with ASI to those trained with fixed state initialization (FSI) and reference state initialization (RSI). In the case of fixed state initialization, the character is always initialized to the same pose at the start of the motion. With reference state initialization, initial states are sampled randomly from the reference motion as proposed by Peng et al. [207]. For ASI, the initial state distribution is modeled as a collection of $k = 10$ independent Gaussian distributions positioned at uniformly spaced phase values. The mean of each Gaussian is initialized to the state at the corresponding phase of the reference motion, and the diagonal covariance matrix is initialized with the sample covariance of the states from the entire reference motion. Both the mean and covariance matrix of each distribution are then learned through the training process, while the corresponding phase for each distribution is kept fixed. Figure 4.9 compares the learning curves using the three different methods and Table 4.2 compares the performance of the final policies. Each result is averaged over three independent runs with different random seeds.

Skill	FSI	RSI	ASI
Backflip A	0.086	0.602	0.741
Cartwheel A	0.362	0.738	0.824
Frontflip	0.435	0.658	0.708
Handspring A	0.358	0.464	0.696

Table 4.2: Performance of policies trained with different initial state distributions. ASI outperforms the other methods for all skills evaluated.

Overall, the behaviour of the learning algorithm appears consistent across multiple runs. Policies trained with ASI consistently outperform their counterparts, converging to the highest return between the different methods. For more challenging skills, such as the backflip and frontflip, ASI also shows notable improvements in learning speed. Policies trained with FSI struggles to reproduce any of the skills. Furthermore, we evaluate the sensitivity of ASI to different numbers of Gaussian components. Policies were trained using $k = 5, 10, 20$ components and their corresponding learning curves are available in Figure 4.12. Using different numbers of components does not seem to have a significant impact on the performance of ASI. Qualitatively, the resulting motions also appear similar.

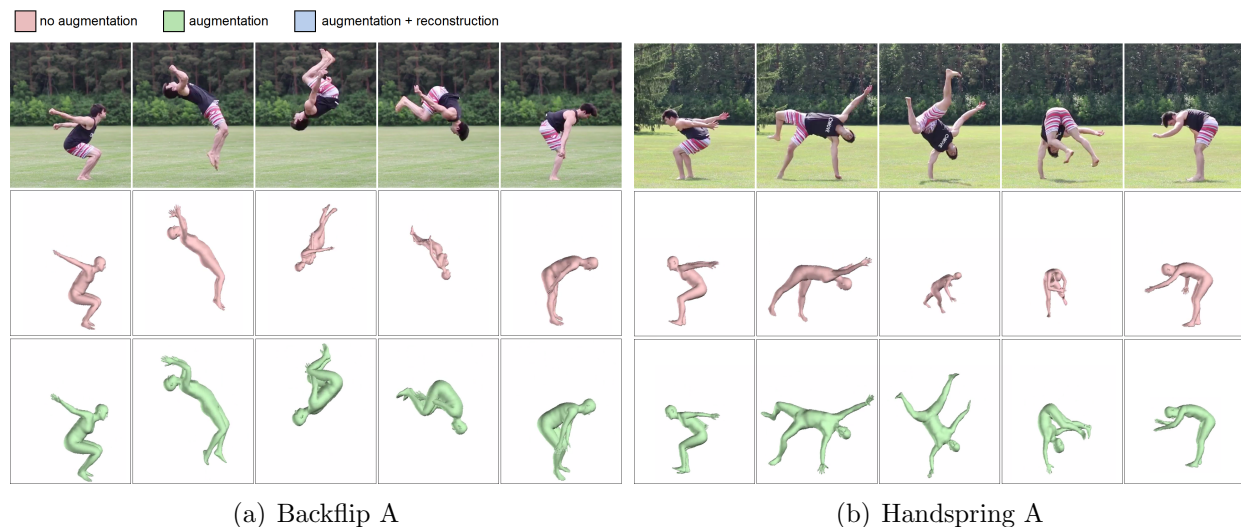


Figure 4.10: 3D pose predictions with and without rotation augmentation. **Top:** Video. **Middle:** Without rotation augmentation. **Bottom:** With rotation augmentation. The pose estimator trained without rotation augmentation fails to correctly predict challenging poses, such as when the actor is upside-down.

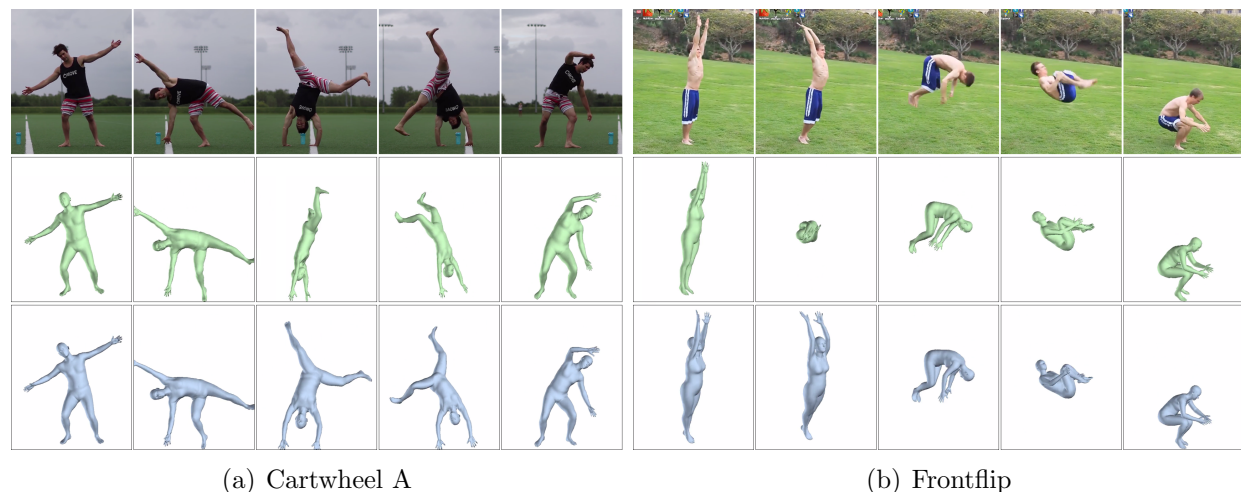


Figure 4.11: 3D pose predictions before and after motion reconstruction. **Top:** Video. **Middle:** Raw predictions from the 3D pose estimator before motion reconstruction. **Bottom:** After motion reconstruction. The motion reconstruction process is able to fix erroneous predictions from the 3D pose estimator by taking advantage of the information from the 2D pose estimator and temporal consistency between adjacent frames.

Skill	RSI	RSI + MR	ASI	ASI + MR
Frontflip	0.404	0.658	0.403	0.708
Handspring A	0.391	0.464	0.631	0.696

Table 4.3: Performance of policies with and without motion reconstruction.

Reference Motion: A policy’s ability to reproduce a video demonstration relies on the quality of the reconstructed reference motion. Here, we investigate the effects of rotation augmentation and motion reconstruction on the resulting reference motions. Most existing datasets of human poses are biased heavily towards upright poses. However, an actor’s orientation can vary more drastically when performing highly dynamic and acrobatic skills, e.g. upside-down poses during a flip. Rotation augmentation significantly improves predictions for these less common poses. Figure 4.10 compares the predictions from pose estimators trained with and without rotation augmentation. We found that this step is vital for accurate predictions of more extreme poses, such as those present in the backflip and handspring. Without augmentation, both pose estimators consistently fail to predict upside-down poses.

Next, we evaluate the effects of the motion reconstruction stage in producing reference motions that can be better reproduced by a simulated character. Policies that are trained to imitate the optimized reference motions generated by motion reconstruction (MR), are compared to policies trained without MR, where the poses from the 3D pose estimator are directly used as the reference motion. Figure 4.11 compares the motions before and after MR. While the 3D pose estimator occasionally produces erroneous predictions, the MR process is able to correct these errors by taking advantage of the predictions from the 2D estimator and enforcing temporal consistency between adjacent frames. Learning curves comparing policies trained using reference motions before and after motion reconstruction are available in Figure 4.13, and Table 4.3 summarizes the performance of the final policies. For each type of reference motion, we also compared policies trained with either RSI or ASI. Overall, imitating reference motions generated by the motion reconstruction processes improves performance and learning speed for the different skills. The improvements due to MR appears more pronounced when policies are trained with RSI. Since the initial states are sampled directly from the reference motion, performance is more susceptible to artifacts present in the reference motion. MR also shows consistent improvement across multiple training runs when using ASI. Note that since the reward reflects similarly to the reference motion, and not the original video, a higher return does not necessarily imply better reproduction of the video demonstration. Instead, the higher return with MR indicates that the simulated character is able to better reproduce the reference motions produced by MR than the raw predictions from the pose estimator. Thus, the results suggest that by enforcing temporal consistency and mitigating artifacts due to inaccurate pose predictions, motion reconstruction is able to generate reference motions that are more amenable to being mimicked by a simulated character.

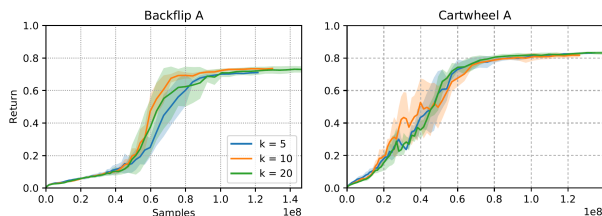


Figure 4.12: Learning curves of policies trained with ASI using different number of Gaussian components. The choice of the number of components does not appear to have a significant impact on performance.

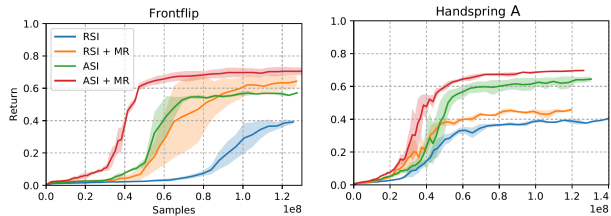


Figure 4.13: Learning curves comparing policies trained with and without motion reconstruction (MR). MR improves performance for both RSI and ASI.

4.9 Discussion

In this chapter, we presented a framework for learning full-body motion skills from monocular video demonstrations. Our method is able to reproduce a diverse set of highly dynamic and acrobatic skills with simulated humanoid characters. We proposed a data augmentation technique that improves the performance of pose estimators for challenging acrobatic motions, and a motion reconstruction method that leverages an ensemble of pose estimators to produce higher-fidelity reference motions. Our adaptive state initialization method substantially improves the performance of the motion imitation process when imitating low-fidelity reference motions. Our framework is also able to retarget skills to characters and environments that differ drastically from those present in the original video clips.

While our framework is able to imitate a diverse collection of video clips, it does have a number of limitations. Since the success of the motion imitation stage depends on the accuracy of the reconstructed motion, when the pose estimators are not able to correctly predict an actor’s pose, the resulting policy will fail to reproduce the behavior. Examples include the kip-up, where the reconstructed motions did not accurately capture the motion of the actor’s arms, and the spinkick, where the pose estimator did not capture the extension of the actor’s leg during the kick. Furthermore, our characters still sometimes exhibit artifacts such as peculiar postures and stiff movements. Fast dance steps, such as those exhibited in the Gangnam Style clip, remains challenging for the system, and we have yet to be able to train policies that can closely reproduce such nimble motions. Due to difficulties in estimating the global translation of the character’s root, our results have primarily been limited to video clips with minimal camera motion.

Nonetheless, we believe this work opens many exciting directions for future exploration. Our experiments suggest that learning highly-dynamic skills from video demonstrations is achievable by building on state-of-the-art techniques from computer vision and reinforcement learning. An advantage of our modular design is that new advances relevant to the various stages of the pipeline can be readily incorporated to improve the overall effectiveness of the framework. However, an exciting direction for future work is to investigate methods for

more end-to-end learning from visual demonstrations, for example taking inspiration from Sermanet et al. [241] and Yu et al. [305], which may reduce the dependence on accurate pose estimators. Another exciting direction is to capitalize on our method’s ability to learn from video clips and focus on large, outdoor activities, as well as motions of nonhuman animals that are conventionally very difficult, if not impossible, to mocap.

Chapter 5

Multiplicative Compositional Policies

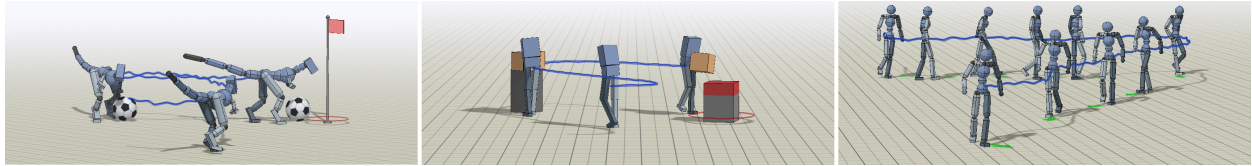


Figure 5.1: Multiplicative compositional policies enable physically simulated characters to perform challenging tasks by reusing skills learned through motion imitation. **Left:** Simulated T-Rex dribbling a soccer ball to a target. **Middle:** Biped picking up and carrying a box to a goal location. **Right:** Humanoid walking along a target heading direction. (Video¹)

In the previous chapters, we have primarily used reinforcement learning to train policies for each task from scratch. While *tabula rasa* learning can achieve state-of-the-art performance on a broad range of tasks [180, 28, 85, 248, 194], this approach can incur significant drawbacks in terms of sample efficiency and limits the complexity of skills that an agent can acquire. The ability to transfer and re-purpose skills learned from prior experiences to new domains is a hallmark of intelligent agents. Transferable skills can enable agents to solve tasks that would otherwise be prohibitively challenging to learn from scratch, by leveraging prior experiences to provide structured exploration and more effective representations. However, learning versatile and reusable skills that can be applied to a diverse set of tasks remains a challenging problem, particularly when controlling systems with large numbers of degrees-of-freedom.

In this chapter, we propose multiplicative compositional policies (MCP), a method for learning reusable motor primitives that can be composed to produce a continuous spectrum of skills. Once learned, the primitives can be transferred to new tasks and combined to yield different behaviors as necessary in the target domain. Standard hierarchical models [255, 61] often activate only a single primitive at each time step, which can limit the diversity

¹ Supplementary video: <https://xbpeng.github.io/projects/MCP/>

of behaviors that can be produced by the agent. MCP composes primitives through a multiplicative model that enables multiple primitives to be activated at a given time step, thereby providing the agent a more flexible range of skills. Our method can therefore be viewed as providing a means of composing skills in space, while standard hierarchical models compose skills in time by temporally sequencing the set of available skills. MCP can also be interpreted as a variant of latent space models, where the latent encoding specifies a particular composition of a discrete set of primitives.

The primary contribution of this chapter is a method for learning and composing transferable skills using multiplicative compositional policies. By pre-training the primitives to imitate a corpus of different motion clips, our method learns a set of primitives that can be composed to produce a flexible range of behaviors. While conceptually simple, MCP is able to solve a suite of challenging mobile manipulation tasks with complex simulated characters, significantly outperforming prior methods as task complexity grows. Our analysis shows that the primitives discover specializations that are reminiscent of previous manually-designed control structures, and produce coherent exploration strategies that are vital for high-dimensional long-horizon tasks. In our experiments, MCP substantially outperforms prior methods for skill transfer, with our method being the only approach that learns a successful policy on the most challenging task in our benchmark.

5.1 Transfer Learning

We consider a multi-task RL framework for transfer learning, consisting of a set of pre-training tasks and transfer tasks. An agent is trained from scratch on the pre-training tasks, but it may then apply any skills learned during pre-training to the subsequent transfer tasks. The objective then is to leverage the pre-training tasks to acquire a set of reusable skills that enables the agent to be more effective at the later transfer tasks. Each task is represented by a state space $\mathbf{s}_t \in \mathcal{S}$, an action space $\mathbf{a}_t \in \mathcal{A}$, a dynamics model $\mathbf{s}_{t+1} \sim p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$, a goal space $\mathbf{g} \in \mathcal{G}$, a goal distribution $\mathbf{g} \sim p(\mathbf{g})$, and a reward function $r_t = r(\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1}, \mathbf{g})$. The goal specifies task specific features, such as a motion clip to imitate, or the target location an object should be placed. All tasks share a common state space, action space, and dynamics model. However, the goal space, goal distribution, and reward function may differ between pre-training and transfer tasks. For each task, the agent’s objective is to learn an optimal policy π^* that maximizes its expected return. Successful transfer cannot be expected for unrelated tasks. Therefore, we consider the setting where the pre-training tasks encourage the agent to learn relevant skills for the subsequent transfer tasks, but may not necessarily cover the full range of skills required to be effective at the transfer tasks.

Hierarchical policies are a common model for reusing and composing previously learned skills. One approach for constructing a hierarchical policy is by using a mixture-of-experts model [114, 189, 267, 94, 202], where the composite policy’s action distribution $\pi(\mathbf{a}|\mathbf{s}, \mathbf{g})$ is represented by a weighted sum of distributions from a set of primitives $\pi_i(\mathbf{a}|\mathbf{s}, \mathbf{g})$ (i.e. low-level policies). A gating function determines the weights $w_i(\mathbf{s}, \mathbf{g})$ that specify the probability

of activating each primitive for a given \mathbf{s} and \mathbf{g} ,

$$\pi(\mathbf{a}|\mathbf{s}, \mathbf{g}) = \sum_{i=1}^k w_i(\mathbf{s}, \mathbf{g})\pi_i(\mathbf{a}|\mathbf{s}, \mathbf{g}), \quad \sum_{i=1}^k w_i(\mathbf{s}, \mathbf{g}) = 1, \quad w_i(\mathbf{s}, \mathbf{g}) \geq 0. \quad (5.1)$$

Here, k denotes the number of primitives. We will refer to this method of composing primitives as an *additive model*. To sample from the composite policy, a primitive π_i is first selected according to w , then an action is sampled from the primitive’s distribution. Therefore, a limitation of the additive model is that only one primitive can be active at a particular time step. While complex behaviors can be produced by sequencing the various primitives in time, the action taken at each time step remains restricted to the behavior prescribed by a single primitive. Selecting from a discrete set of primitive skills can be effective for simple systems with a small number of actuated degrees-of-freedom, where an agent is only required to perform a small number of subtasks at the same time. But as the complexity of the system grows, an agent might need to perform more and more subtasks *simultaneously*. For example, a person can walk, speak, and carry an object all at the same time. Furthermore, these subtasks can be combined in any number of ways to produce a staggering array of diverse behaviors. This combinatorial explosion can be prohibitively challenging to model with policies that activate only one primitive at a time.

5.2 Multiplicative Compositional Policies

In this work, we propose multiplicative compositional policies (MCP), a method for composing primitives that addresses this combinatorial explosion by explicitly factoring the agent’s behavior – not with respect to time, but with respect to the action space. Our model enables the agent to activate multiple primitives simultaneously, with each primitive specializing in different behaviors that can be composed to produce a continuous spectrum of skills. Our probabilistic formulation accomplishes this by treating each primitive as a distribution over actions, and the composite policy is obtained by a multiplicative composition of these distributions,

$$\pi(\mathbf{a}|\mathbf{s}, \mathbf{g}) = \frac{1}{Z(\mathbf{s}, \mathbf{g})} \prod_{i=1}^k \pi_i(\mathbf{a}|\mathbf{s}, \mathbf{g})^{w_i(\mathbf{s}, \mathbf{g})}, \quad w_i(\mathbf{s}, \mathbf{g}) \geq 0. \quad (5.2)$$

Unlike an additive model, which activates only a single primitive per time step, the *multiplicative model* allows multiple primitives to be activated simultaneously. The gating function specifies the weights $w_i(\mathbf{s}, \mathbf{g})$ that determine the influence of each primitive on the composite action distribution, with a larger weight corresponding to a larger influence. The weights need not be normalized, but in the following experiments, the weights will be bounded $w_i(\mathbf{s}, \mathbf{g}) \in [0, 1]$. $Z(\mathbf{s}, \mathbf{g})$ is the partition function that ensures the composite distribution is normalized. While the additive model directly samples actions from the selected primitive’s distribution, the multiplicative model first combines the primitives, and then samples actions from the resulting distribution.

Gaussian Primitives

Gaussian policies are a staple for continuous control tasks, and modeling multiplicative primitives using Gaussian policies provides a particularly convenient form for the composite policy. Each primitive $\pi_i(\mathbf{a}|\mathbf{s}, \mathbf{g}) = \mathcal{N}(\mu_i(\mathbf{s}, \mathbf{g}), \Sigma_i(\mathbf{s}, \mathbf{g}))$ will be modeled by a Gaussian with mean $\mu_i(\mathbf{s}, \mathbf{g})$ and diagonal covariance matrix $\Sigma_i(\mathbf{s}, \mathbf{g}) = \text{diag}(\sigma_i^1(\mathbf{s}, \mathbf{g}), \sigma_i^2(\mathbf{s}, \mathbf{g}), \dots, \sigma_i^{|\mathcal{A}|})$, where $\sigma_i^j(\mathbf{s}, \mathbf{g})$ denotes the variance of the j th action parameter from primitive i , and $|\mathcal{A}|$ represents the dimensionality of the action space. A multiplicative composition of Gaussian primitives yields yet another Gaussian policy $\pi(\mathbf{a}|\mathbf{s}, \mathbf{g}) = \mathcal{N}(\mu(\mathbf{s}, \mathbf{g}), \Sigma(\mathbf{s}, \mathbf{g}))$. Since the primitives model each action parameter with an independent Gaussian, the action parameters of the composite policy π will also assume the form of independent Gaussians with component-wise mean $\mu^j(\mathbf{s}, \mathbf{g})$ and variance $\sigma^j(\mathbf{s}, \mathbf{g})$,

$$\mu^j(\mathbf{s}, \mathbf{g}) = \frac{1}{\sum_{l=1}^k \frac{w_l(\mathbf{s}, \mathbf{g})}{\sigma_l^j(\mathbf{s}, \mathbf{g})}} \sum_{i=1}^k \frac{w_i(\mathbf{s}, \mathbf{g})}{\sigma_i^j(\mathbf{s}, \mathbf{g})} \mu_i^j(\mathbf{s}, \mathbf{g}), \quad \sigma^j(\mathbf{s}, \mathbf{g}) = \left(\sum_{i=1}^k \frac{w_i(\mathbf{s}, \mathbf{g})}{\sigma_i^j(\mathbf{s}, \mathbf{g})} \right)^{-1}. \quad (5.3)$$

Note that while $w_i(\mathbf{s}, \mathbf{g})$ determines a primitive’s overall influence on the composite distribution, each primitive can also independently adjust its influence per action parameter through $\sigma_i^j(\mathbf{s}, \mathbf{g})$. Once the parameters of the composite distribution have been determined, π can be treated as a regular Gaussian policy, and trained end-to-end using standard automatic differentiation tools.

Pre-Training and Transfer

The primitives are learned through a set of pre-training tasks. The same set of primitives is responsible for solving all pre-training tasks, which results in a collection of primitives that captures the range of behaviors needed for the set of tasks. Note, the primitives are not manually assigned to particular tasks. Instead, the primitives are trained jointly in an end-to-end fashion and the specializations emerge automatically from the learning process. Algorithm 2 illustrates the overall training process. $J_{pre}(\pi_{1:k}, w)$ denotes the objective for the pre-training tasks for a given set of primitives $\pi_{1:k}$ and gating function w , and $J_{tra}(\pi_{1:k}, \omega)$ denotes the objective for the transfer tasks. When transferring primitives to a new task, the parameters of the primitives are kept fixed, while a new policy is trained to specify weights for composing the primitives. Therefore, the primitives can be viewed as a set of nonlinear basis functions that defines a new action space for use in subsequent tasks. During pre-training, in order to force the primitives to specialize in distinct skills, we use an asymmetric model, where only the gating function $w_i(\mathbf{s}, \mathbf{g})$ observes the goal \mathbf{g} , and the primitives have access only to the state \mathbf{s} ,

$$\pi(\mathbf{a}|\mathbf{s}, \mathbf{g}) = \frac{1}{Z(\mathbf{s}, \mathbf{g})} \prod_{i=1}^k \pi_i(\mathbf{a}|\mathbf{s})^{w_i(\mathbf{s}, \mathbf{g})}, \quad \pi_i(\mathbf{a}|\mathbf{s}) = \mathcal{N}(\mu_i(\mathbf{s}), \Sigma_i(\mathbf{s})). \quad (5.4)$$

Algorithm 2 MCP Pre-Training and Transfer

- 1: Pre-training:
 - 2: $\pi_i \leftarrow$ random parameters for $i = 1, \dots, k$
 - 3: $w \leftarrow$ random parameters
 - 4: $\pi_{1:k}^*, w^* = \arg \max_{\pi_{1:k}, w} J_{pre}(\pi_{1:k}, w)$
 - 5: Transfer:
 - 6: $\omega \leftarrow$ random parameters
 - 7: $\omega^* = \arg \max_{\omega} J_{tra}(\pi_{1:k}^*, \omega)$
-

This asymmetric model prevents the degeneracy of a single primitive becoming responsible for all goals, and instead encourages the primitives to learn distinct skills that can then be composed by the gating function as needed for a given goal. Furthermore, since the primitives depend only on the state, they can be conveniently transferred to new tasks that share similar state spaces but may have different goal spaces. When transferring the primitives to new tasks, the parameters of the primitives $\pi_i(\mathbf{a}|\mathbf{s})$ are kept fixed to prevent catastrophic forgetting, and a new gating function $\omega(\mathbf{w}|\mathbf{s}, \mathbf{g})$ is trained to specify the weights $\mathbf{w} = (w_1, w_2, \dots)$ for composing the primitives.

5.3 Related Work

Learning reusable representations that are transferable across multiple tasks has a long history in machine learning [268, 34, 13, 197, 225]. Finetuning remains a popular transfer learning technique when using neural network, where a model is first trained on a source domain, and then the learned features are reused in a target domain by finetuning via backpropagation [103, 56]. One of the drawbacks of this procedure is catastrophic forgetting, as backpropagation is prone to destroying previously learned features before the model is able to utilize them in the target domain [229, 126, 230].

Hierarchical Policies: A popular method for combining and reusing skills is by constructing hierarchical policies, where a collection of low-level controllers, which we will refer to as primitives, are integrated together with the aid of a gating function that selects a suitable primitive for a given scenario [255, 18, 94]. A common approach for building hierarchical policies is to first train a collection of primitives through a set of pre-training tasks, which encourages each primitive to specialize in distinct skills [45, 202, 159, 69, 175]. Once trained, the primitives can be integrated into a hierarchical policy and transferred to new tasks. End-to-end methods have also been proposed for training hierarchical policies [52, 18, 134, 277]. However, since standard hierarchical policies only activate one primitive at a time, it is not as amenable for composition or interpolation of multiple primitives in order to produce new skills.

Latent Space Models: Our work falls under a class of methods that we will refer to broadly as latent space models. These methods specify controls through a latent representation that is then mapped to the controls (i.e. actions) of the underlying system [130]. Similar to hierarchical models, a latent representation can first be learned using a set of pre-training tasks, before transferring to downstream tasks [89, 100]. But unlike a standard hierarchical model, which activates a single primitive at a time, continuous latent variables can be used to enable more flexible interpolation of skills in the latent space. Various diversity-promoting pre-training techniques have been proposed for encouraging the latent space to model semantically distinct behaviors [67, 60, 95]. Demonstrations can also be incorporated during pre-training to acquire more complex skills [177]. In this work, we present a method for modeling latent skill representations as a composition of multiplicative primitives. We show that the additional structure introduced by the primitives enables our agents to tackle complex continuous control tasks, achieving competitive performance when compared to previous models, and significantly outperforming prior methods as task complexity grows.

5.4 Experiments

We evaluate the effectiveness of our method on controlling complex simulated characters, with large numbers of degrees-of-freedom (DoFs), to perform challenging long-horizon tasks. The tasks vary from simple locomotion tasks to difficult mobile manipulation tasks. The characters include a simple 14 DoF ant, a 23 DoF biped, a more complex 34 DoF humanoid, and a 55 DoF T-Rex. Examples of transfer tasks are shown in Figure 5.2. Our experiments aim to study MCP’s performance on complex temporally extended tasks, and examine the behaviors learned by the primitives. We also evaluate our method comparatively to determine the value of multiplicative primitives as compared to more standard additive mixture models, as well as to prior methods based on options and latent space embeddings. Behaviors learned by the policies are best seen in the supplementary video¹.

Experimental Setup

Pre-Training Tasks: The pre-training tasks in our experiments consist of motion imitation tasks, where the objective is for the character to mimic a corpus of different reference motions. Each reference motion specifies a sequence of target states $\{\hat{\mathbf{s}}_0, \hat{\mathbf{s}}_1, \dots, \hat{\mathbf{s}}_T\}$ that the character should track at each time step. We use a motion imitation approach following Peng et al. [207]. But instead of training separate policies for each motion, a single policy, composed of multiple primitives, is trained to imitate a variety of motion clips. To imitate multiple motions, the goal $\mathbf{g}_t = (\hat{\mathbf{s}}_{t+1}, \hat{\mathbf{s}}_{t+2})$ provides the policy with target states for the next two time steps. A reference motion is selected randomly at the start of each episode. To encourage the primitives to learn to transition between different skills, the reference motion is also switched randomly to another motion within each episode. The corpus of motion clips is comprised of different walking and turning motions.

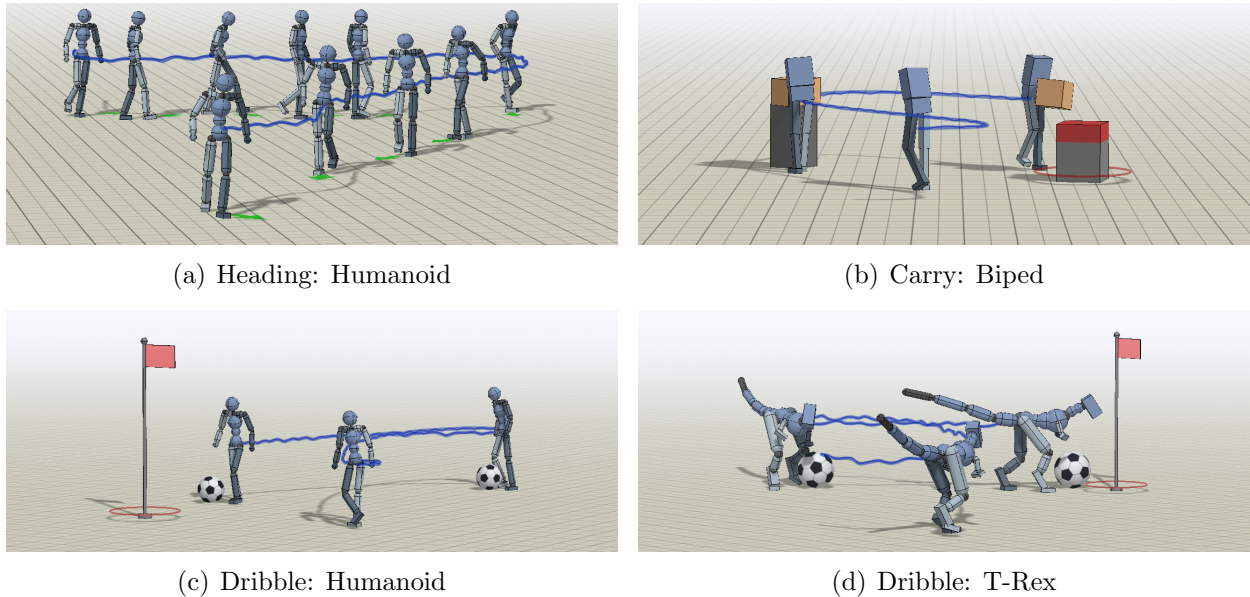


Figure 5.2: The transfer tasks pose a challenging combination of locomotion and object manipulation, such as carrying an object to a target location and dribbling a ball to a goal, which requires coordination of multiple body parts and temporally extended behaviors.

Transfer Tasks: We evaluate our method on a set of challenging continuous control tasks, involving locomotion and object manipulation using the various characters.

Heading: First we consider a simple heading task, where the objective is for the character to move along a target heading direction $\hat{\theta}_t$. The heading is changed every timestep by applying a random perturbation $\hat{\theta}_t = \hat{\theta}_{t-1} + \nabla\theta_t$ sampled from a uniform distribution $\nabla\theta_t \sim \text{Uniform}(-0.15\text{rad}, 0.15\text{rad})$. The goal $\mathbf{g}_t = (\cos(\hat{\theta}_t), -\sin(\hat{\theta}_t))$ encodes the heading as a unit vector along the horizontal plane. The reward r_t encourages the character to follow the target heading, and is computed according to

$$r_t = \exp\left(-4 \left(\hat{\mathbf{u}} \cdot \mathbf{v}_{\text{com}} - \hat{v}\right)^2\right). \quad (5.5)$$

Here, (\cdot) denotes the dot product, \mathbf{v}_{com} represents the character’s center-of-mass (COM) velocity along the horizontal plane, $\hat{v} = 1\text{m/s}$ represents the target speed that the character should travel in along the target direction $\hat{\mathbf{u}} = (\cos(\hat{\theta}_t), -\sin(\hat{\theta}_t))$.

Carry: To evaluate our method’s performance on long horizon tasks, we consider a mobile manipulation task, where the objective is to move a box from a source location to a target location. The task can be decomposed into a sequence of subtasks, where the character must first pickup the box from the source location, before carrying it to the target location, and placing it on the table. To enable the character to carry the box, when the character makes

contact with the box at the source location with a specific link (e.g. torso), a virtual joint is created that attaches the box to the character. Once the box has been placed at the target location, the joint is detached. The box has a mass of 5kg and is initialized to a random source location at a distance of [0m, 10m] from the character. The target is initialized to a distance of [0m, 10m] from the source. Depending on the initial configuration, the task may require thousands of time steps to complete. The goal $\mathbf{g}_t = (\mathbf{x}_{\text{tar}}, q_{\text{tar}}, \mathbf{x}_{\text{src}}, q_{\text{src}}, \mathbf{x}_b, q_b, \mathbf{v}_b, \omega_b)$ encodes the target table’s position \mathbf{x}_{tar} and orientation q_{tar} , the source table’s position \mathbf{x}_{src} and orientation q_{src} , and box’s position \mathbf{x}_b , orientation q_b , linear velocity \mathbf{v}_b , and angular velocity ω_b . The reward function consists of terms that encourage the character to move towards the box, as well as to move the box towards the target,

$$r_t = w^{\text{cv}} r_t^{\text{cv}} + w^{\text{cp}} r_t^{\text{cp}} + w^{\text{bv}} r_t^{\text{bv}} + w^{\text{bp}} r_t^{\text{bp}}, \quad (5.6)$$

r_t^{cv} encourages the character to move towards the box, while r_t^{cp} encourages the character to stay near the box,

$$r_t^{\text{cv}} = \exp(-1.5 \min(0, \mathbf{u}_b \cdot \mathbf{v}_{\text{com}} - \hat{v})^2) \quad (5.7)$$

$$r_t^{\text{cp}} = \exp(-0.25 \|\mathbf{x}_{\text{com}} - \mathbf{x}_b\|^2). \quad (5.8)$$

\mathbf{u}_b is a unit vector pointing in the direction of the box with respect to the character’s COM, \mathbf{v}_{com} is the COM velocity of the character, $\hat{v} = 1\text{m/s}$ is the target speed, \mathbf{x}_{com} is the COM position, and \mathbf{x}_b is the box’s position. All quantities are expressed on the horizontal plane. Similarly, r_t^{bv} and r_t^{bp} encourages the character to move the box towards the target,

$$r_t^{\text{bv}} = \exp(-1 \min(0, \mathbf{u}_{\text{tar}} \cdot \mathbf{v}_b - \hat{v})^2) \quad (5.9)$$

$$r_t^{\text{bp}} = \exp(-0.5 \|\mathbf{x}_b - \mathbf{x}_{\text{tar}}\|^2). \quad (5.10)$$

\mathbf{u}_{tar} represents the unit vector pointing in the direction of the target with respect to the box, \mathbf{v}_b is the velocity of the box, and \mathbf{x}_{tar} is the target location. The weights for the reward terms are specified according to $(w^{\text{cv}}, w^{\text{cp}}, w^{\text{bv}}, w^{\text{bp}}) = (0.1, 0.2, 0.3, 0.4)$.

Dribble: This task poses a challenging combination of locomotion and object manipulation, where the objective is to move a soccer ball to a target location. Since the policy does not have direct control over the ball, it must rely on complex contact dynamics in order to manipulate the movement of the ball while also maintaining balance. The ball is randomly initialized at a distance of [0m, 10m] from the character, and the target is initialized to a distance of [0m, 10m] from the ball. The goal $\mathbf{g}_t = (\mathbf{x}_{\text{tar}}, \mathbf{x}_b, q_b, \mathbf{v}_b, \omega_b)$ encodes the target location \mathbf{x}_{tar} , and ball’s position \mathbf{x}_b , orientation q_b , linear velocity \mathbf{v}_b , and angular velocity ω_b . The reward function for this task follows a similar structure as the reward for the carry task, consisting of terms that encourage the character to move towards the ball, as well as to move the ball towards the target,

$$r_t = w^{\text{cv}} r_t^{\text{cv}} + w^{\text{cp}} r_t^{\text{cp}} + w^{\text{bv}} r_t^{\text{bv}} + w^{\text{bp}} r_t^{\text{bp}}, \quad (5.11)$$

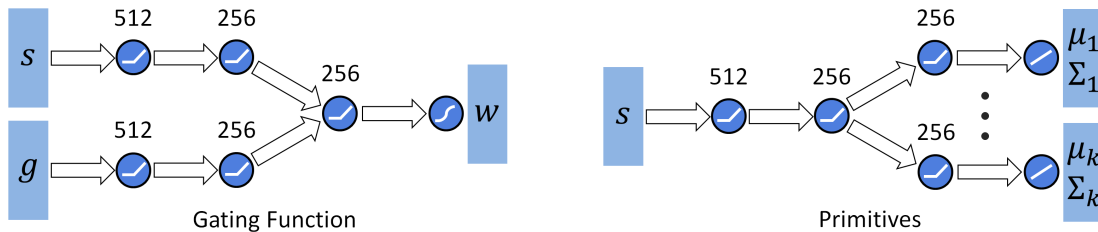


Figure 5.3: Schematic illustrations of the MCP architecture. The gating function receives both \mathbf{s} and \mathbf{g} as inputs, which are first encoded by separate networks, with 512 and 256 units. The resulting features are concatenated and processed with a layer of 256 units, followed by a sigmoid output layer to produce the weights $w(\mathbf{s}, \mathbf{g})$. The primitives receive only \mathbf{s} as input, which is first processed by a common network, with 512 and 256 units, before branching into separate layers of 256 units for each primitive, followed by a linear output layer that produces $\mu_i(\mathbf{s})$ and $\Sigma_i(\mathbf{s})$ for each primitive. ReLU activation is used for all hidden units.

r_t^{cv} encourages the character to move towards the ball, while r_t^{cp} encourages the character to stay near the ball,

$$r_t^{\text{cv}} = \exp(-1.5 \min(0, \mathbf{u}_b \cdot \mathbf{v}_{\text{com}} - \hat{v})^2) \quad (5.12)$$

$$r_t^{\text{cp}} = \exp(-0.5 \|\mathbf{x}_{\text{com}} - \mathbf{x}_b\|^2). \quad (5.13)$$

\mathbf{u}_b represents the unit vector pointing in the direction of the ball with respect to the character’s COM, \mathbf{v}_{com} is the character’s COM velocity, $\hat{v} = 1\text{m/s}$ is the target speed, \mathbf{x}_{com} is the COM position, and \mathbf{x}_b is the ball’s position. Similarly, r_t^{bv} and r_t^{bp} encourages the character to move the ball towards the target,

$$r_t^{\text{bv}} = \exp(-1 \min(0, \mathbf{u}_{\text{tar}} \cdot \mathbf{v}_b - \hat{v})^2) \quad (5.14)$$

$$r_t^{\text{bp}} = \exp(-0.5 \|\mathbf{x}_b - \mathbf{x}_{\text{tar}}\|^2). \quad (5.15)$$

\mathbf{u}_{tar} represents the unit vector pointing in the direction of the target with respect to the ball, \mathbf{v}_b is the velocity of the ball, and \mathbf{x}_{tar} is the target location. The weights for the reward terms are specified according to $(w^{\text{cv}}, w^{\text{cp}}, w^{\text{bv}}, w^{\text{bp}}) = (0.1, 0.1, 0.3, 0.5)$.

Model Representation: All experiments use a similar network architecture for the policy, as illustrated in Figure 5.3. Each policy is composed of $k = 8$ primitives. The gating function and primitives are modeled by separate networks that output $w(\mathbf{s}, \mathbf{g})$, $\mu_{i:k}(\mathbf{s})$, and $\Sigma_{i:k}(\mathbf{s})$, which are then composed according to Equation 5.2 to produce the composite policy. The state describes the configuration of the character’s body, using a similar set of features as those described in Chapter 2. Actions from the policy specify target rotations for PD controllers positioned at each joint. Target rotations for 3D spherical joints are parameterized using exponential maps. The policies operate at 30Hz and are trained using proximal policy optimization (PPO) [238].

Environment	Scratch	Finetune	Hierarchical	Option-Critic	MOE	Latent Space	MCP (Ours)
Heading: Biped	0.927 ± 0.032	0.970 ± 0.002	0.834 ± 0.001	0.952 ± 0.012	0.918 ± 0.002	0.970 ± 0.001	0.976 ± 0.002
Heading: Humanoid	0.965 ± 0.010	0.975 ± 0.008	0.681 ± 0.006	0.958 ± 0.001	0.857 ± 0.018	0.969 ± 0.002	0.970 ± 0.003
Heading: T-Rex	0.840 ± 0.003	0.953 ± 0.004	–	0.830 ± 0.004	0.672 ± 0.011	0.686 ± 0.003	0.932 ± 0.007
Carry: Biped	0.027 ± 0.035	0.324 ± 0.014	0.001 ± 0.002	0.346 ± 0.011	0.013 ± 0.013	0.456 ± 0.031	0.575 ± 0.032
Dribble: Biped	0.072 ± 0.012	0.651 ± 0.025	0.546 ± 0.024	0.046 ± 0.008	0.073 ± 0.021	0.768 ± 0.012	0.782 ± 0.008
Dribble: Humanoid	0.076 ± 0.024	0.598 ± 0.030	0.198 ± 0.002	0.058 ± 0.007	0.043 ± 0.021	0.751 ± 0.006	0.805 ± 0.006
Dribble: T-Rex	0.065 ± 0.032	0.074 ± 0.011	–	0.098 ± 0.013	0.070 ± 0.017	0.115 ± 0.013	0.781 ± 0.021
Holdout: Ant	0.951 ± 0.093	0.885 ± 0.062	–	–	–	0.745 ± 0.060	0.812 ± 0.030

Table 5.1: Performance statistics of different models on transfer tasks. MCP outperforms other methods on a suite of challenging tasks with complex simulated characters.

Reference Motions: During pre-training, the primitives are trained by imitating a corpus of reference motions. The biped and humanoid share the same set of reference motions, consisting of mocap clips of walking and turning motions collected from a publicly available database [242]. In total, 230 seconds of motion data is used to train the biped and humanoid. To retarget the humanoid reference motions to the biped, we simply removed extraneous joints in the upper body (e.g. arms and head). The reference motions for the T-Rex consist of artist generated keyframe animations. Due to the cost of manually authored animations, the T-Rex is trained with substantially less motion data than the other characters. In total, 11 seconds of motion data is used to train the T-Rex. The T-Rex motions include 1 forward walk, 2 left turns, and 2 right turns. Despite having access to only a small corpus of reference motions, MCP is nonetheless able to learn a flexible set of primitives that enables the complex T-Rex character to perform challenging tasks.

Comparisons

We compare MCP to a number of prior methods, including a baseline model trained from scratch for each transfer task, and a model first pre-trained to imitate a reference motion before being finetuned on the transfer tasks. To evaluate the effects of being able to activate and compose multiple primitives simultaneously, we compare MCP to models that activate only one primitive at a time, including a hierarchical model that sequences a set of pre-trained skills [159, 175], an option-critic model [18], and a mixture-of-experts model (MOE) analogous to Equation 5.1. Finally, we also include comparisons to a continuous latent space model with an architecture similar to Hausman et al. [95] and Merel et al. [177]. All models, except for the scratch model, are pre-trained with motion imitation [207]. Figure 5.4 illustrates learning curves for the various methods on the transfer tasks and Table 5.1 summarizes their performance. Each environment is denoted by "Task: Character". Performance is recorded as the average normalized return across approximately 100 episodes, with 0 being the minimum possible return per episode and 1 being the maximum. Three models initialized with different random seeds are trained for each environment and method.

Our experiments show that MCP performs well across the suite of tasks. For simple tasks such as heading, all models show similar performance. But as task complexity in-

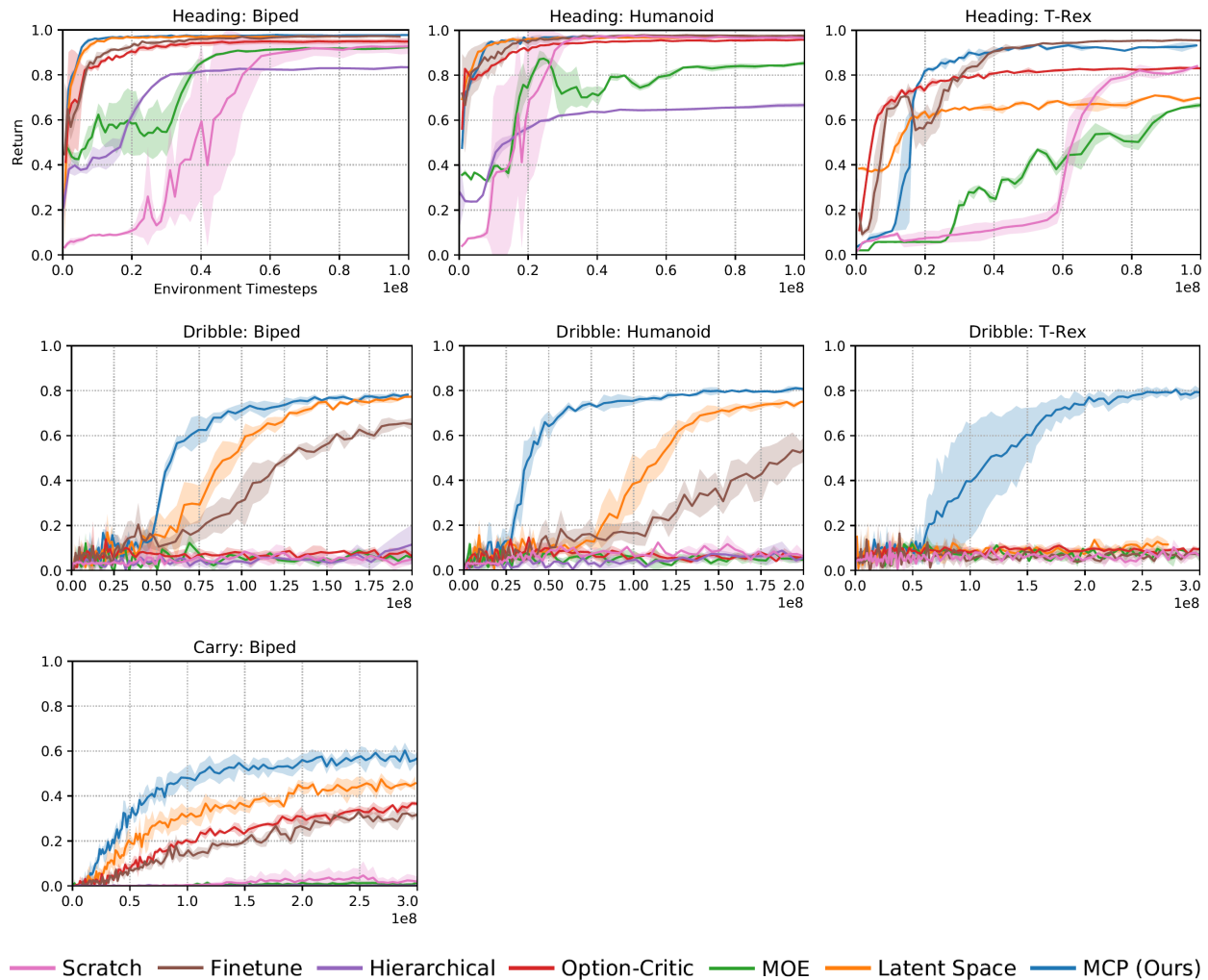


Figure 5.4: Learning curves of the various models when applied to transfer tasks. MCP substantially improves learning speed and performance on challenging tasks (e.g. carry and dribble), and is the only method that succeeds on the most difficult task (Dribble: T-Rex).

creases, MCP exhibits significant improvements to learning speed and asymptotic performance. Training from scratch is effective for the simple heading task, but is unable to solve the more challenging carry and dribble tasks. Finetuning proved to be a strong baseline, but struggles with the more complex morphologies. With higher dimensional action spaces, independent action noise is less likely to produce useful behaviors. Models that activate only a single primitive at a time, such as the hierarchical model, option-critic model, and MOE model, tend to converge to lower asymptotic performance due to their limited expressivity. MOE is analogous to MCP where only a single primitive is active at a time. Despite using a similar number of primitives as MCP, being able to activate only one primitive per time

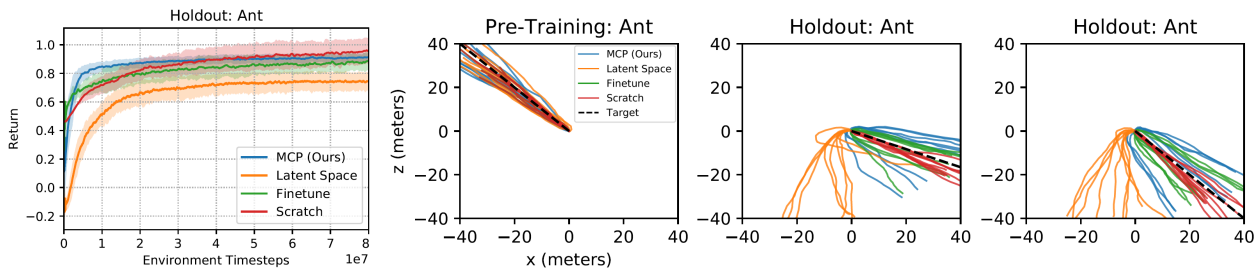


Figure 5.5: **Left:** Learning curves on holdout tasks in the Ant environment. **Right:** Trajectories produced by models with target directions from pre-training, and target directions from the holdout set after training on transfer tasks. The latent space model is prone to overfitting to the pre-training tasks, and can struggle to adapt to the holdout tasks.

step limits the variety of behaviors that can be produced by MOE. This suggests that the flexibility of MCP to compose multiple primitives is vital for more sophisticated tasks. The latent space model shows strong performance on most tasks. But when applied to characters with more complex morphologies, such as the humanoid and T-Rex, MCP consistently outperforms the latent space model, with MCP being the only model that solves the dribbling task with the T-Rex.

We hypothesize that the performance difference between MCP and the latent space model may be due to the process through which a latent code \mathbf{w} is mapped to an action for the underlying system. With the latent space model, the pre-trained policy $\pi(\mathbf{a}|\mathbf{s}, \mathbf{w})$ acts as a decoder that maps \mathbf{w} to a distribution over actions. We have observed that this decoder has a tendency to overfit to the pre-training behaviors, and can therefore limit the variety of behaviors that can be deployed on the transfer tasks. In the case of MCP, if σ_i^j is the same across all primitives, then we can roughly view \mathbf{w} as specifying a convex combination of the primitive means $\mu_{i:k}$. Therefore, $\mu_{1:k}$ forms a convex hull in the original action space, and the transfer policy $\omega(\mathbf{w}|\mathbf{s}, \mathbf{g})$ can select any action within this set. As such, MCP may provide the transfer policy with a more flexible range of skills than the latent space model. To test this hypothesis, we evaluate the different models on transferring to out-of-distribution tasks using a simple setup. The environment is a variant of the standard Gym Ant environment [28], where the agent’s objective is to run along a target direction $\hat{\theta}$. During pre-training, the policies are trained with directions $\hat{\theta} \in [0, 3/2\pi]$. During transfer, the directions are sampled from a holdout set $\hat{\theta} \in [3/2\pi, 2\pi]$. Figure 5.5 illustrates the learning curves on the transfer task, along with the trajectories produced by the models when commanded to follow different target directions from the pre-training and transfer tasks. Indeed we see that the latent space model is prone to overfitting to the directions from pre-training, and struggles to adapt to the holdout directions. MCP provides the transfer policy sufficient flexibility to adapt quickly to the transfer tasks. The scratch and finetune models also perform well on the transfer tasks, since they operate directly on the underlying action space.

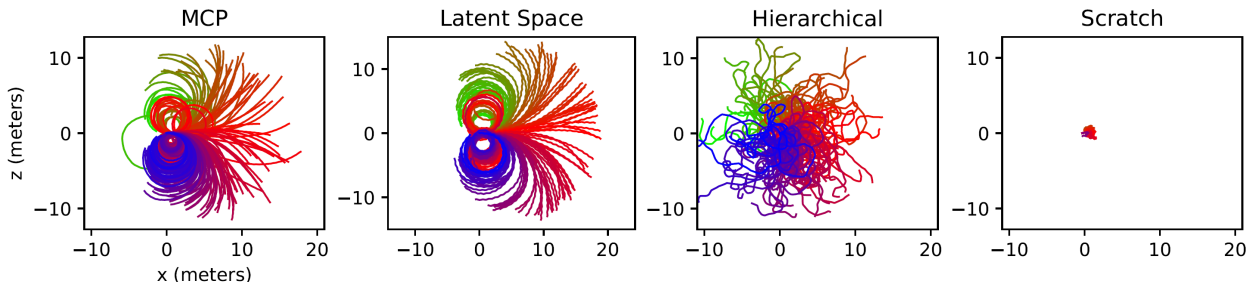


Figure 5.6: Trajectories of the humanoid’s root along the horizontal plane visualizing the exploration behaviors of different models. MCP and other models that are pre-trained with motion imitation produce more structured exploration behaviors.

Exploration Behaviors

To analyze the exploration behaviors produced by the primitives, we visualize the trajectories obtained by random combinations of the primitives, where the weights are sampled from a Gaussian and held fixed over the course of a trajectory. Figure 5.6 illustrates the trajectories of the humanoid’s root produced by various models. Similar to MCP, the trajectories from the latent space model are also produced by sampling w from a Gaussian. The trajectories from the hierarchical model are generated by randomly sequencing the set of primitives. The model trained from scratch simply applies Gaussian noise to the actions, which leads to a fall after only few time steps. Models that are pre-trained with motion imitation produce more structured behaviors that travel in different directions.

Primitive Specializations

To analyze the specializations of the primitives, we record the weight of each primitive over the course of a walk cycle. Figure 5.7 illustrates the weights during pre-training, when the humanoid is trained to imitate walking motions. The activations of the primitives show a strong correlation to the phase of a walk cycle, with primitive 1 becoming most active during left stance and becoming less active during right stance, while primitive 2 exhibits precisely the opposite behavior. The primitives appear to have developed a decomposition of a walking gait that is commonly incorporated into the design of locomotion controllers [302]. Furthermore, these specializations consistently appear across multiple training runs. Next, we visualize the actions proposed by each primitive. Figure 5.7 shows a PCA embedding of the mean action from each primitive. The actions from each primitive form distinct clusters, which suggests that the primitives are indeed specializing in different behaviors.

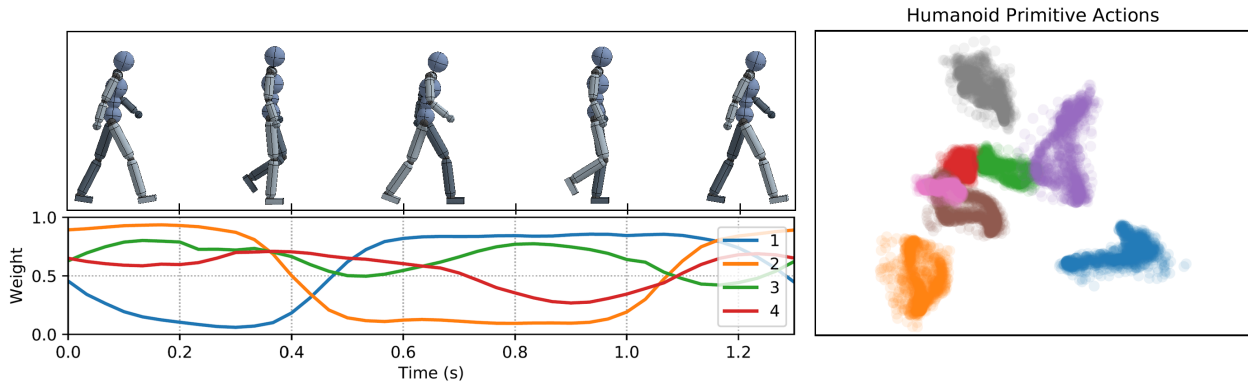


Figure 5.7: **Left:** Weights for primitives over the course of a walk cycle. Primitives develop distinct specializations, with some primitives becoming most active during the left stance phase, and others during right stance. **Right:** PCA embedding of actions from each primitive exhibits distinct clusters.

5.5 Discussion

In this chapter, we presented multiplicative compositional policies (MCP), a method for learning and composing skills using multiplicative primitives. Despite its simplicity, our method is able to learn sophisticated behaviors that can be transferred to solve challenging continuous control tasks with complex simulated agents. Once trained, the primitives form a new action space that enables more structured exploration and provides the agent with the flexibility to combine the primitives in novel ways in order to elicit new behaviors for a task. Our experiments show that MCP can be effective for long horizon tasks and outperforms prior methods as task complexity grows. While MCP provides a form of spatial abstraction, we believe that incorporating temporal abstractions is an important direction. During pre-training, some care is required to select an expressive corpus of reference motions. In future work, we wish to investigate methods for recovering sophisticated primitive skills without this supervision.

Chapter 6

Adversarial Motion Priors

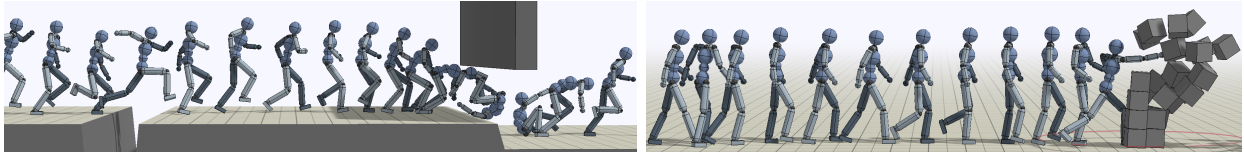


Figure 6.1: Our framework enables physically simulated character to solve challenging tasks while adopting stylistic behaviors specified by unstructured motion data. **Left:** A character learns to traverse an obstacles course using a variety of locomotion skills. **Right:** A character learns to walk to and punch a target. (Video¹)

In the previous chapters, we showed that motion imitation can be a highly effective approach for training agents to perform a wide array of sophisticated skills. The primary motion imitation method we have utilized thus far is *motion tracking*, where an agent imitates a motion by explicitly tracking the sequence of target poses specified by a reference motion. This is accomplished through a tracking objective that encourages an agent to minimize the difference between the pose of the character’s body and the pose specified by the reference motion at every time step. These tracking-based methods can produce high-quality motions for a large repertoire skills. But extending these techniques to effectively leverage large unstructured motion datasets remains challenging, since a suitable motion clip needs to be selected for the character to track at each time step. This selection process is typically performed by a motion planner, which generates reference trajectories for solving a particular task [206, 20, 198]. However, constructing an effective motion planner can itself be a challenging endeavour, and entails significant overhead to annotate and organize the motion clips for a desired task. For many applications, it is not imperative to exactly track a particular reference motion. Since a dataset typically provides only a limited collection of example motions, a character will inevitably need to deviate from the reference motions in

¹ Supplementary video: <https://xbpeng.github.io/projects/AMP/>

order to effectively perform a given task. Therefore, the intent is often not for the character to closely track a particular motion, but to adopt general behavioral characteristics depicted in the dataset. We refer to these behavioral characteristics as a *style*.

In this chapter, we aim to develop a system where users can specify high-level task objectives for a character to perform, while the low-level *style* of a character’s movements can be controlled through examples provided in the form of unstructured motion clips. To control the style of a character’s motions, we propose adversarial motion priors (AMP), a method for imitating behaviors from raw motion clips without requiring any task-specific annotations or organization of the dataset. Given a set of reference motions that constitutes a desired motion style, the motion prior is modeled as an adversarial discriminator, trained to differentiate between behaviors depicted in the dataset from those produced by the character. The motion prior therefore acts as a general measure of similarity between the motions produced by a character and the motions in the dataset. By incorporating the motion prior in a goal-conditioned reinforcement learning framework, our system is able to train physically simulated characters to perform challenging tasks with natural and life-like behaviors. Composition of diverse behaviors emerges automatically from the motion prior, without the need for a motion planner or other mechanism for selecting *which* clip to imitate.

The central contribution of this chapter is an adversarial learning approach for physics-based character animation that combines goal-conditioned reinforcement with an adversarial motion prior, which enables the *style* of a character’s movements to be controlled via example motion clips, while the *task* is specified through a simple reward function. We present one of the first adversarial learning systems that is able to produce high-quality full-body motions for physically simulated characters. By combining the motion prior with additional task objectives, our system provides a convenient interface through which users can specify high-level directions for controlling a character’s behaviors. These task objectives allow our characters to acquire more complex skills than those demonstrated in the original motion clips. While our system is built on well-known adversarial imitation learning techniques, we propose a number of important design decisions that lead to substantially higher quality results than those achieved by prior work, enabling our characters to learn highly dynamic and diverse motor skills from unstructured motion data.

6.1 Related Work

Developing systems that can synthesize natural motions for virtual characters is one of the fundamental challenges of computer animation. These procedural animation techniques can be broadly categorized as *kinematic methods* and *physics-based methods*. Kinematic methods generally do not explicitly utilize the equations of motion for motion synthesis. Instead, these methods often leverage datasets of motion clips to generate motions for a character [140, 144]. Given a motion dataset, controllers can be constructed to select an appropriate motion clip to play back for a particular scenario [233, 274, 4]. Data-driven methods using generative models, such as Gaussian processes [301, 148] and neural networks

[157, 106, 313], have also been applied to synthesize motions online. When provided with sufficiently large and high-quality datasets, kinematic methods are able to produce realistic motions for a large variety of sophisticated skills [144, 151, 4, 142, 251]. However, their ability to synthesize motions for novel situations can be limited by the availability of data. For complex tasks and environments, it can be difficult to collect a sufficient amount of data to cover all possible behaviors that a character may need to perform. This is particularly challenging for nonhuman and fictional creatures, where motion data can be scarce. In this work, we combine data-driven techniques with physics-based animation methods to develop characters that produce realistic and responsive behaviors to novel tasks and environments.

Physics-Based Methods: Physics-based methods address some of the limitations of kinematic methods by synthesizing motions from first principles. These methods typically leverage a physics simulation, or more general knowledge of the equations of motion, to generate motions for a character [220, 281]. Optimization techniques, such as trajectory optimization and reinforcement learning, play a pivotal role in many physics-based methods, where controllers that drive a character’s motions are produced by optimizing an objective function [196, 182, 258]. While these methods are able to synthesize physically plausible motions for novel scenarios, even in the absence of motion data, designing effective objectives that lead to natural behaviors can be exceptionally difficult. Heuristics derived from prior knowledge of the characteristics of natural motions are commonly included into the objective function, such as symmetry, stability, effort minimization, and many more [282, 182, 308]. Simulating more biologically accurate actuators can also improve motion quality [284, 75, 116], but may nonetheless yield unnatural behaviors.

Imitation Learning: The challenges of designing objective functions that lead to natural motions have spurred the adoption of data-driven physics-based animation techniques [320, 244, 49, 145, 137], which utilizes reference motion data to improve motion quality. Reference motions are typically incorporated through an imitation objective that encourages a character to imitate motions in the dataset. The imitation objective is commonly implemented as a tracking objective, which attempts to minimize the pose error between the simulated character and target poses from a reference motion [249, 145, 161, 160, 207]. Since the pose error is generally computed with respect to a single target pose at a time, some care is required to select an appropriate target pose from the dataset. A simple strategy is to synchronize the simulated character with a given reference motion using a phase variable [207, 210, 143], which is provided as an additional input to the controller. The target pose at each time step can then be conveniently determined by selecting the target pose according to the phase. This strategy has been effective for imitating individual motion clips, but it can be difficult to scale to datasets containing multiple disparate motions, as it may not be possible to synchronize and align multiple reference motions according to a single-phase variable. Recent methods have extended these tracking-based techniques to larger motion datasets by explicitly providing target poses from the reference motion that is being tracked

as inputs to the controller [39, 20, 198, 290]. This then allows a controller to imitate different motions depending on the input target poses. However, selecting the appropriate motion for a character to imitate in a given scenario can still entail significant algorithmic overhead. These methods often require a high-level motion planner that selects which motion clip the character should imitate for a given task [206, 20, 198]. The character’s performance on a particular task can therefore be constrained by the performance of the motion planner.

Another major limitation of tracking-based imitation techniques is the need for a pose error metric when computing the tracking objective [244, 161, 207]. These error metrics are often manually-designed, and it can be challenging to construct and tune a common metric that is effective across all skills that a character is to imitate. Adversarial imitation learning provides an appealing alternative [1, 319, 104], where instead of using a manually-designed imitation objective, these algorithms train an adversarial discriminator to differentiate between behaviors generated by an agent from behaviors depicted in the demonstration data (e.g. reference motions). The discriminator then serves as the objective function for training a control policy to imitate the demonstrations. While these methods have shown promising results for motion imitation tasks [176, 287], adversarial learning algorithms can be notoriously unstable and the resulting motion quality still falls well behind what has been achieved with state-of-the-art tracking-based techniques. Peng et al. [211] was able to produce substantially more realistic motions by regularizing the discriminator with an information bottleneck. However, their method still requires a phase variable to synchronize the policy and discriminator with the reference motion. Therefore, their results are limited to imitating a single motion per policy, and thus not suitable for learning from large diverse motion datasets. In this work, we propose an adversarial method for learning general motion priors from large unstructured datasets that contain diverse motion clips. Our approach does not necessitate any synchronization between the policy and reference motion. Furthermore, our approach does not require a motion planner, or any task-specific annotation and segmentation of the motion clips [206, 198, 20]. Instead, composition of multiple motions in furtherance of a task objective emerges automatically through the motion prior. We also present a number of design decisions for stabilizing the adversarial training process, leading to consistent and high-quality results.

Latent Space Models: Latent space models can also act as a form of motion prior that leads to more life-like behaviors. These models specify controls through a learned latent representation, which is then mapped to controls for the underlying system [31, 100, 67, 95]. The latent representation is typically learned through a pre-training phase using supervised learning or reinforcement learning techniques to encode a diverse range of behaviors into a latent representation. Once trained, this latent representation can be used to build a control hierarchy, where the latent space model acts as a low-level controller, and a separate high-level controller is trained to specify controls via the latent space [67, 89, 167]. For motion control of simulated characters, the latent representation can be trained to encode behaviors from reference motion clips, which then constrains the behavior of a character to

be similar to those observed in the motion data, therefore leading to more natural behaviors for downstream tasks [177, 209]. However, since the realism of the character’s motions is enforced implicitly through the latent representation, rather than explicitly through an objective function, it is still possible for the high-level controller to specify latent encodings that produce unnatural behaviors [209, 174]. Luo et al. [166] proposed an adversarial domain confusion loss to prevent the high-level controller from specifying encodings that are different from those observed during pre-training. However, since this adversarial objective is applied in the latent space, rather than on the actual motions produced by the character, the model is nonetheless prone to generating unnatural behaviors. Our proposed motion prior directly enforces similarity between the motions produced by the character and those in the reference motion dataset, which enables our method to produce higher fidelity motions than what has been demonstrated by latent space models. Our motion prior also does not require a separate pre-training phase, and instead, can be trained jointly with the policy.

6.2 Overview

Given a dataset of reference motions and a task objective defined by a reward function, our system synthesizes a control policy that enables a character to achieve the task objective in a physically simulated environment, while utilizing behaviors that resemble the motions in the dataset. Crucially, the character’s behaviors need not exactly match any specific motion in the dataset, instead its movements need only to adopt more general characteristics exhibited by the corpus of reference motions. These reference motions collectively provide an example-based definition of a behavioral *style*. By providing the system with different motion datasets, the character can then be trained to perform a task in a variety of distinct styles.

Figure 6.2 provides a schematic overview of the system. The motion dataset \mathcal{M} consists of a collection of reference motions, where each motion $\mathbf{m}^i = \{\hat{\mathbf{q}}_t^i\}$ is represented as a sequence of poses $\hat{\mathbf{q}}_t^i$. The motion clips may be collected from the mocap of real-life actors or from artist-authored keyframe animations. Unlike previous frameworks, our system can be applied directly on raw motion data, without requiring task-specific annotations or segmentation of a clip into individual skills. The motion of the simulated character is controlled by a policy $\pi(\mathbf{a}_t|\mathbf{s}_t, \mathbf{g})$ that maps the state of the character \mathbf{s}_t and a given goal \mathbf{g} to a distribution over actions \mathbf{a}_t . The actions from the policy specify target positions for proportional-derivative (PD) controllers positioned at each of the character’s joints, which in turn produce control forces that drive the motion of the character. The goal \mathbf{g} specifies a task reward function $r_t^G = r^G(\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1}, \mathbf{g})$, which defines high-level objectives for the character to satisfy (e.g. walking in a target direction or punching a target). The style objective $r_t^S = r^S(\mathbf{s}_t, \mathbf{s}_{t+1})$ is specified by an adversarial discriminator, trained to differentiate between motions depicted in the dataset from motions produced by the character. The style objective therefore acts as a task-agnostic motion prior that provides an a-priori estimate of the naturalness or style of a given motion, independent of a specific task. The style objective then encourages the policy to produce motions that resemble behaviors depicted in the dataset.

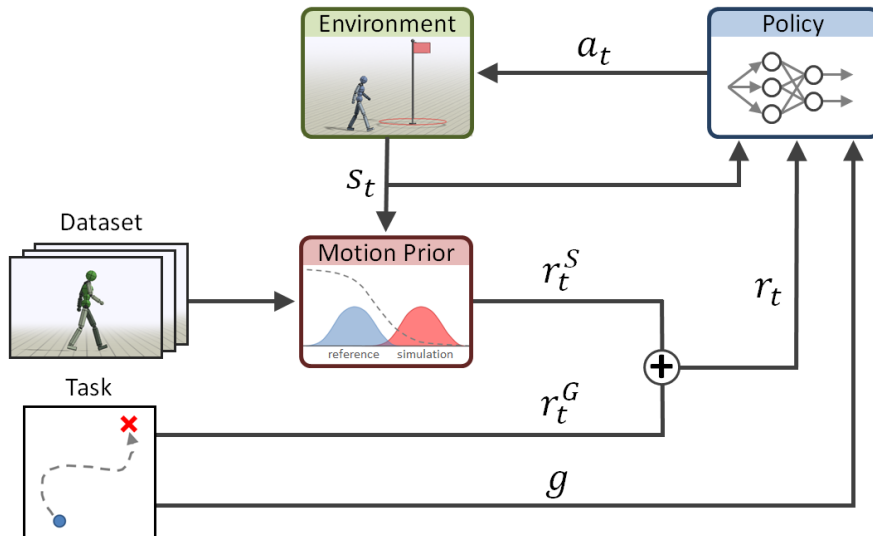


Figure 6.2: Schematic overview of the system. Given a motion dataset defining a desired motion style for the character, the system trains a motion prior that specifies style-rewards r_t^S for the policy during training. These style-rewards are combined with task-rewards r_t^G and used to train a policy that enables a simulated character to satisfy task-specific goals \mathbf{g} , while also adopting behaviors that resemble the reference motions in the dataset.

6.3 Generative Adversarial Imitation Learning

Our system combines techniques from goal-conditioned reinforcement learning and generative adversarial imitation learning to train control policies that enable simulated characters to perform challenging tasks in a desired behavioral style. Generative adversarial imitation learning (GAIL) [104] adapts techniques developed for generative adversarial networks (GAN) [80] to the domain of imitation learning. In the interest of brevity, we exclude the goal \mathbf{g} from the notation, but the following discussion readily generalizes to goal-conditioned settings. Given a dataset of demonstrations $\mathcal{M} = \{(\mathbf{s}_i, \mathbf{a}_i)\}$, containing states \mathbf{s}_i and actions \mathbf{a}_i recorded from an unknown demonstration policy, the objective is to train a policy $\pi(\mathbf{a}|\mathbf{s})$ that imitates the behaviors of the demonstrator. Behavioral cloning can be used to directly fit a policy to map from states observed in \mathcal{M} to their corresponding actions using supervised learning [214, 25]. However, if only a small amount of demonstrations are available, then behavioral cloning techniques are prone to drift [227]. Furthermore, behavioral cloning is not directly applicable in settings where the demonstration actions are not observable (e.g. reference motion data).

GAIL addresses some of the limitations of behavioral cloning by learning an objective function that measures the similarity between the policy and the demonstrations, and then updating π via reinforcement learning to optimize the learned objective. The objective is modeled as a discriminator $D(\mathbf{s}, \mathbf{a})$, trained to predict whether a given state \mathbf{s} and action \mathbf{a}

is sampled from the demonstrations \mathcal{M} or generated by running the policy π ,

$$\arg \min_D -\mathbb{E}_{d^{\mathcal{M}}(\mathbf{s}, \mathbf{a})} [\log (D(\mathbf{s}, \mathbf{a}))] - \mathbb{E}_{d^{\pi}(\mathbf{s}, \mathbf{a})} [\log (1 - D(\mathbf{s}, \mathbf{a}))]. \quad (6.1)$$

$d^{\mathcal{M}}(\mathbf{s}, \mathbf{a})$ and $d^{\pi}(\mathbf{s}, \mathbf{a})$ denote the state-action distribution of the dataset and policy respectively. The policy is then trained using RL with rewards specified by,

$$r_t = -\log (1 - D(\mathbf{s}_t, \mathbf{a}_t)). \quad (6.2)$$

This adversarial training procedure can be interpreted as training a policy to produce states and actions that appear to the discriminator as being indistinguishable from the demonstrations. It can be shown that this objective minimizes the Jensen-Shannon divergence between $d^{\mathcal{M}}(\mathbf{s}, \mathbf{a})$ and $d^{\pi}(\mathbf{s}, \mathbf{a})$ [193, 122].

6.4 Adversarial Motion Prior

In this work, we consider reward functions that consist of two components specifying: 1) *what* task a character should perform, and 2) *how* the character should go about performing that task,

$$r(\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1}, \mathbf{g}) = w^G r^G(\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1}, \mathbf{g}) + w^S r^S(\mathbf{s}_t, \mathbf{s}_{t+1}). \quad (6.3)$$

The *what* is represented by a task-specific reward $r^G(\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1}, \mathbf{g})$, which defines high-level objectives that a character should satisfy (e.g. moving to a target location). The *how* is represented through a learned task-agnostic style-reward $r^S(\mathbf{s}_t, \mathbf{s}_{t+1})$, which specifies low-level details of the behaviors that the character should adopt when performing the task (e.g., walking vs. running to a target). The two reward terms are combined linearly with weights w^G and w^S . The task-reward r^G can be relatively intuitive and simple to design. However, it can be exceptionally difficult to design a style-reward r^S that leads a character to learn naturalistic behaviors, or behaviors that conform to a particular style. Learning effective style objectives will therefore be the primary focus of this work.

We propose to model the style-reward with a learned discriminator, which we refer to as an adversarial motion prior (AMP), by analogy to the adversarial pose priors that were previously proposed for vision-based pose estimation tasks [119]. Unlike standard tracking objectives, which measure pose similarity with respect to a specific reference motion, the motion prior returns a general score indicating the similarity of the character’s motion to the motions depicted in the dataset, without explicitly comparing to a particular motion clip. Given a motion dataset, the motion prior is trained using the GAIL framework to predict whether a state transition $(\mathbf{s}_t, \mathbf{s}_{t+1})$ is a *real* sample from the dataset or a *fake* sample produced by the character. The motion prior is independent of the task-specific goal \mathbf{g} , therefore a single motion prior can be applied to multiple tasks, and different motion priors can be applied to train policies that perform the same task but in different styles. By combining GAIL with additional task objectives, our approach decouples task specification

from style specification, thereby enabling our characters to perform tasks that may not be depicted in the original demonstrations. However, adversarial RL techniques are known to be highly unstable. In the following sections, we discuss a number of design decisions to stabilize the training process and produce higher fidelity results.

Imitation from Observations

The original formulation of GAIL requires access to the demonstrator’s actions [104]. However, when the demonstrations are provided in the form of motion clips, the actions taken by the demonstrator are unknown, and only states are observed in the data. To extend GAIL to settings with state-only demonstrations, the discriminator can be trained on state transitions $D(\mathbf{s}, \mathbf{s}')$ instead of state-action pairs $D(\mathbf{s}, \mathbf{a})$ [272],

$$\arg \min_D - \mathbb{E}_{d^{\mathcal{M}}(\mathbf{s}, \mathbf{s}')} [\log (D(\mathbf{s}, \mathbf{s}'))] - \mathbb{E}_{d^{\pi}(\mathbf{s}, \mathbf{s}')} [\log (1 - D(\mathbf{s}, \mathbf{s}'))]. \quad (6.4)$$

$d^{\mathcal{M}}(\mathbf{s}, \mathbf{s}')$ and $d^{\pi}(\mathbf{s}, \mathbf{s}')$ denote the likelihood of observing a state transition from \mathbf{s} to \mathbf{s}' in the dataset \mathcal{M} and by following policy π respectively. Note that if the demonstrator is different from the agent (e.g. a human actor), the observed state transitions may not be physically consistent for the agent, and therefore impossible for the agent to perfectly reproduce. Despite this discrepancy, we show that the discriminator still provides an effective objective for imitating a wide range of behaviors.

Least-Squares Discriminator

The standard GAN objective detailed in Equation 6.4 typically uses a sigmoid cross-entropy loss function. However, this loss tends to lead to optimization challenges due to vanishing gradients as the sigmoid function saturates, which can hamper training of the policy [14]. A myriad of techniques have been proposed to address these optimization challenges [216, 235, 121, 14, 22, 127, 86, 178]. In this work, we adopt the loss function proposed for least-squares GAN (LSGAN) [170], which has demonstrated more stable training and higher quality results for image synthesis tasks. The following objective is used to train the discriminator,

$$\arg \min_D \mathbb{E}_{d^{\mathcal{M}}(\mathbf{s}, \mathbf{s}')} \left[(D(\mathbf{s}, \mathbf{s}') - 1)^2 \right] + \mathbb{E}_{d^{\pi}(\mathbf{s}, \mathbf{s}')} \left[(D(\mathbf{s}, \mathbf{s}') + 1)^2 \right]. \quad (6.5)$$

The discriminator is trained by solving a least-squares regression problem to predict a score of 1 for samples from the dataset and -1 for samples recorded from the policy. The reward function for training the policy is then given by

$$r(\mathbf{s}_t, \mathbf{s}_{t+1}) = \max [0, 1 - 0.25(D(\mathbf{s}_t, \mathbf{s}_{t+1}) - 1)^2]. \quad (6.6)$$

The additional offset, scaling, and clipping are applied to bound the reward between $[0, 1]$, as is common practice in previous RL frameworks [202, 207, 262]. Mao et al. [170] showed that this least-squares objective minimizes the Pearson χ^2 divergence between $d^{\mathcal{M}}(\mathbf{s}, \mathbf{s}')$ and $d^{\pi}(\mathbf{s}, \mathbf{s}')$.

Discriminator Observations

Since the discriminator specifies rewards for training the policy, selecting an appropriate set of features for use by the discriminator when making its predictions is vital to provide the policy with effective feedback. As such, before a state transition is provided as input to the discriminator, we first apply an observation map $\Phi(\mathbf{s}_t)$ that extracts a set of features relevant for determining the characteristics of a given motion. The resulting features are then used as inputs to the discriminator $D(\Phi(\mathbf{s}), \Phi(\mathbf{s}'))$. The set of features include:

- Linear velocity of the root (3D).
- Angular velocity of the root (3D).
- Local rotation of each joint using the tangent-normal encoding (6D).
- Local velocity of each joint (3D).
- 3D positions of the end-effectors (e.g. hands and feet) (3D).

Similar to the state features, all features in the discriminator’s observation are also recorded in the character’s local coordinate frame. This set of observation features is selected to provide a compact representation of the motion across a single state transition. The observations also do not include any task-specific features, thus enabling the motion prior to be trained without requiring task-specific annotation of the reference motions, and allowing motion priors trained with the same dataset to be used for different tasks.

Gradient Penalty

The interplay between the discriminator and generator in a GAN often results in unstable training dynamics. One source of instability is due to function approximation errors in the discriminator, where the discriminator may assign nonzero gradients on the manifold of real data samples [178]. These nonzero gradients can cause the generator to *overshoot* and move off the data manifold, instead of converging to the manifold, leading to oscillations and instability during training. To mitigate this phenomenon, a gradient penalty can be applied to penalize nonzero gradients on samples from the dataset [127, 86, 178]. We incorporate this technique to improve training stability. The discriminator objective is then given by:

$$\begin{aligned} \arg \min_D \quad & \mathbb{E}_{d^{\mathcal{M}}(\mathbf{s}, \mathbf{s}')} \left[(D(\Phi(\mathbf{s}), \Phi(\mathbf{s}')) - 1)^2 \right] + \mathbb{E}_{d^{\pi}(\mathbf{s}, \mathbf{s}')} \left[(D(\Phi(\mathbf{s}), \Phi(\mathbf{s}')) + 1)^2 \right] \\ & + \frac{w^{\text{gp}}}{2} \mathbb{E}_{d^{\mathcal{M}}(\mathbf{s}, \mathbf{s}')} \left[\left\| \nabla_{\phi} D(\phi) \Big|_{\phi=(\Phi(\mathbf{s}), \Phi(\mathbf{s}'))} \right\|^2 \right], \end{aligned} \quad (6.7)$$

where w^{gp} is a manually-specified coefficient. Note, the gradient penalty is calculated with respect to the observation features $\phi = (\Phi(\mathbf{s}), \Phi(\mathbf{s}'))$, not the full set of state features $(\mathbf{s}, \mathbf{s}')$. As we show in our experiments, the gradient penalty is crucial for stable training and effective performance.

6.5 Model Representation

Given a high-level task objective and a dataset of reference motions, the agent is responsible for learning a control policy that fulfills the task objectives, while utilizing behaviors that resemble the motions depicted in the dataset. In this section, we detail the design of various components of the learning framework.

Network Architecture

Similar to previous chapters, each policy π is modeled by a neural network that maps a given state \mathbf{s}_t and goal \mathbf{g} to a Gaussian distribution over actions $\pi(\mathbf{a}_t|\mathbf{s}_t, \mathbf{g}) = \mathcal{N}(\mu(\mathbf{s}_t, \mathbf{g}), \Sigma)$, with an input-dependent mean $\mu(\mathbf{s}_t, \mathbf{g})$ and a fixed diagonal covariance matrix Σ . The mean is specified by a fully-connected network with two hidden layers, consisting of 1024 and 512 ReLU [187], followed by a linear output layer. The values of the covariance matrix $\Sigma = \text{diag}(\sigma_1, \sigma_2, \dots)$ are manually-specified and kept fixed over the course of training. The value function $V(\mathbf{s}_t, \mathbf{g})$ and discriminator $D(\mathbf{s}_t, \mathbf{s}_{t+1})$ are modeled by separate networks with a similar architecture as the policy.

Training

Our policies are trained using a combination of GAIL [104] and proximal-policy optimization (PPO) [238]. Algorithm 3 provides an overview of the training process. At each time step t , the agent receives a task-reward $r_t^G = r^G(\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1}, \mathbf{g})$ from the environment, it then queries the motion prior for a style-reward $r_t^S = r^S(\mathbf{s}_t, \mathbf{s}_{t+1})$, computed according to Equation 6.6. The two rewards are combined according to Equation 6.3 to yield the reward for the particular timestep. Reference state initialization is applied by initializing the character to states sampled randomly from all motion clips in the dataset.

Once a batch of data has been collected with the policy, the recorded trajectories are used to update the policy and value function. The policy and value function are updated using the same procedure as Chapter 3. Each trajectory recorded from the policy is also stored in a replay buffer \mathcal{B} , containing trajectories from past training iterations. The discriminator is updated according to Equation 6.7 using minibatches of transitions $(\mathbf{s}, \mathbf{s}')$ sampled from the reference motion data set \mathcal{M} and transitions from the replay buffer \mathcal{B} . The replay buffer helps to stabilize training by preventing the discriminator from overfitting to the most recent batch of trajectories from the policy.

6.6 Tasks

To evaluate AMP’s effectiveness for controlling the style of a character’s motions, we apply our framework to train complex 3D simulated characters to perform various motion control

Algorithm 3 Training with AMP

```

1: input  $\mathcal{M}$ : dataset of reference motions
2:  $D \leftarrow$  initialize discriminator
3:  $\pi \leftarrow$  initialize policy
4:  $V \leftarrow$  initialize value function
5:  $\mathcal{B} \leftarrow \emptyset$  initialize reply buffer

6: while not done do
7:   for trajectory  $i = 1, \dots, m$  do
8:      $\tau^i \leftarrow \{(\mathbf{s}_t, \mathbf{a}_t, r_t^G)_{t=0}^{T-1}, \mathbf{s}_T^G, \mathbf{g}\}$  collect trajectory with  $\pi$ 
9:     for time step  $t = 0, \dots, T - 1$  do
10:       $d_t \leftarrow D(\Phi(\mathbf{s}_t), \Phi(\mathbf{s}_{t+1}))$ 
11:       $r_t^S \leftarrow$  calculate style reward according to Equation 6.6 using  $d_t$ 
12:       $r_t \leftarrow w^G r_t^G + w^S r_t^S$ 
13:      record  $r_t$  in  $\tau^i$ 
14:     end for
15:     store  $\tau^i$  in  $\mathcal{B}$ 
16:   end for

17:   for update step  $= 1, \dots, n$  do
18:      $b^{\mathcal{M}} \leftarrow$  sample batch of  $K$  transitions  $\{(\mathbf{s}_j, \mathbf{s}'_j)\}_{j=1}^K$  from  $\mathcal{M}$ 
19:      $b^{\pi} \leftarrow$  sample batch of  $K$  transitions  $\{(\mathbf{s}_j, \mathbf{s}'_j)\}_{j=1}^K$  from  $\mathcal{B}$ 
20:     update  $D$  according to Equation 6.7 using  $b^{\mathcal{M}}$  and  $b^{\pi}$ 
21:   end for

22:   update  $V$  and  $\pi$  using data from trajectories  $\{\tau^i\}_{i=1}^m$ 
23: end while

```

tasks using different motion styles. The characters include a 34 DoF humanoid, a 59 DoF T-Rex, and a 64 DoF dog.

Target Heading: In this task, the character’s objective is to move along a target heading direction \mathbf{d}^* at a target speed v^* . The goal for the policy is specified as $\mathbf{g}_t = (\tilde{\mathbf{d}}_t^*, v^*)$, with $\tilde{\mathbf{d}}_t^*$ being the target direction in the character’s local coordinate frame. The task-reward is calculated according to:

$$r_t^G = \exp\left(-0.25(v^* - \mathbf{d}^* \cdot \dot{\mathbf{x}}_t^{\text{com}})^2\right), \quad (6.8)$$

where $\dot{\mathbf{x}}_t^{\text{com}}$ is the center-of-mass velocity of the character at time step t , and the target speed is selected randomly between $v^* \in [1, 5]$ m/s. For slower moving styles, such as Zombie and Stealthy, the target speed is fixed at 1m/s.

Target Location: In this task, the character’s objective is to move to a target location \mathbf{x}^* . The goal $\mathbf{g}_t = \tilde{\mathbf{x}}_t^*$ records the target location in the character’s local coordinate frame. The task-reward is given by:

$$r_t^G = 0.7 \exp(-0.5 \|\mathbf{x}^* - \mathbf{x}_t^{\text{root}}\|^2) + 0.3 \exp(-(\max(0, v^* - \mathbf{d}_t^* \cdot \dot{\mathbf{x}}_t^{\text{com}}))^2). \quad (6.9)$$

Here, $v^* = 1\text{m/s}$ specifies a minimum target speed at which the character should move towards the target, and the character will not be penalized for moving faster than this threshold. \mathbf{d}_t^* is a unit vector on the horizontal plane that points from the character’s root to the target.

Dribbling: To evaluate our system on more complex object manipulation tasks, we train policies for a dribbling task, where the character’s objective is to dribble a soccer ball to a target location. The reward function is given by:

$$r_t^G = 0.1r_t^{\text{cv}} + 0.1r_t^{\text{cp}} + 0.3r_t^{\text{bv}} + 0.5r_t^{\text{bp}} \quad (6.10)$$

$$r_t^{\text{cv}} = \exp\left(-1.5 \max\left(0, v^* - \mathbf{d}_t^{\text{ball}} \cdot \dot{\mathbf{x}}_t^{\text{com}}\right)^2\right) \quad (6.11)$$

$$r_t^{\text{cp}} = \exp\left(-0.5 \|\mathbf{x}_t^{\text{ball}} - \mathbf{x}_t^{\text{com}}\|^2\right) \quad (6.12)$$

$$r_t^{\text{bv}} = \exp\left(-\max\left(0, v^* - \mathbf{d}_t^* \cdot \dot{\mathbf{x}}_t^{\text{ball}}\right)^2\right) \quad (6.13)$$

$$r_t^{\text{bp}} = \exp\left(-0.5 \|\mathbf{x}_t^* - \mathbf{x}_t^{\text{com}}\|^2\right). \quad (6.14)$$

r_t^{cv} and r_t^{cp} encourages the character to move towards and stay near the ball, where $\mathbf{x}_t^{\text{ball}}$ and $\dot{\mathbf{x}}_t^{\text{ball}}$ represent the position and velocity of the ball, $\mathbf{d}_t^{\text{ball}}$ is a unit vector pointing from the character to the ball, and $v^* = 1\text{m/s}$ is the target velocity at which the character should move towards the ball. Similarly, r_t^{bv} and r_t^{bp} encourages the character to move the ball to the target location, with \mathbf{d}_t^* denoting a unit vector pointing from the ball to the target. The goal $\mathbf{g}_t = \tilde{\mathbf{x}}_t^*$ records the relative position of the target location with respect to the character. The state \mathbf{s}_t is augmented with additional features that describe the state of the ball, including the position $\tilde{\mathbf{x}}_t^{\text{ball}}$, orientation $\tilde{\mathbf{q}}_t^{\text{ball}}$, linear velocity $\tilde{\dot{\mathbf{x}}}_t^{\text{ball}}$, and angular velocity $\tilde{\dot{\mathbf{q}}}_t^{\text{ball}}$ of the ball in the character’s local coordinate frame.

Strike: To demonstrate AMP’s ability to compose diverse behaviors, we consider a task where the character’s objective is to strike a target using a designated end-effector (e.g. hands). The target may be located at various distances from the character. Therefore, the character must first move close to the target before striking it. These distinct phases entail different optimal behaviors, and thus require the policy to compose and transition between the appropriate skills. The goal $\mathbf{g}_t = (\tilde{\mathbf{x}}_t^*, h_t)$ records the location of the target $\tilde{\mathbf{x}}_t^*$ in the character’s local coordinate frame, along with an indicator variable h_t that specifies if the

target has already been hit. The task-reward is partitioned into three phases:

$$r_t^G = \begin{cases} 1, & \text{target has been hit} \\ 0.3 r_t^{\text{near}} + 0.3, & \|\mathbf{x}^* - \mathbf{x}_t^{\text{root}}\| < 1.375\text{m} . \\ 0.3 r_t^{\text{far}}, & \text{otherwise} \end{cases} \quad (6.15)$$

If the character is far from the target \mathbf{x}^* , r_t^{far} encourages the character to move to the target using a similar reward function as the Target Location task (Equation 6.9). Once the character is within a given distance of the target, r_t^{near} encourages the character to strike the target with a particular end-effector,

$$r_t^{\text{near}} = 0.2 \exp(-2\|\mathbf{x}^* - \mathbf{x}_t^{\text{eff}}\|^2) + 0.8 \text{clip}\left(\frac{2}{3}\mathbf{d}_t^* \cdot \dot{\mathbf{x}}_t^{\text{eff}}, 0, 1\right),$$

where $\mathbf{x}_t^{\text{eff}}$ and $\dot{\mathbf{x}}_t^{\text{eff}}$ denote the position and velocity of the end-effector, and \mathbf{d}_t^* is a unit vector pointing from the character’s root to the target. After striking the target, the character receives a constant reward of 1 for the remaining time steps.

Obstacles: Finally, we consider tasks that involve visual perception and interaction with more complex environments, where the character’s objective is to traverse an obstacle-filled terrain, while maintaining a target speed. Policies are trained for two types of environments: 1) An environment containing a combination of obstacles include gaps, steps, and overhead obstructions that the character must duck under. 2) An environment containing narrow stepping stones that requires more precise contact planning. Examples of the environments are available in Figure 6.1 and 6.3. The task-reward is the same as the one used for the Target Heading task (Equation 6.8), except the target heading is fixed along the direction of forward progress. In order for the policy to perceive the upcoming obstacles, the state is augmented with a 1D height-field of the upcoming terrain. The height-field records the height of the terrain at 100 sample locations, uniformly spanning 10m ahead of the character.

6.7 Results

We evaluate our framework’s effectiveness on a suite of challenging motion control tasks with complex simulated characters. First, we demonstrate that our approach can readily scale to large unstructured datasets containing diverse motion clips, which then enables our characters to perform challenging tasks in a natural and life-like manner by imitating behaviors from the dataset. The characters automatically learn to compose and generalize different skills from the motion data in order to fulfill high-level task objectives, without requiring mechanisms for explicit motion selection. We then evaluate AMP on a single-clip imitation task, and show that our method is able to closely imitate a diverse corpus of dynamic and acrobatic skills, producing motions that are nearly indistinguishable from reference motions recorded from human actors. Behaviors learned by the characters can be viewed in the supplementary video¹.

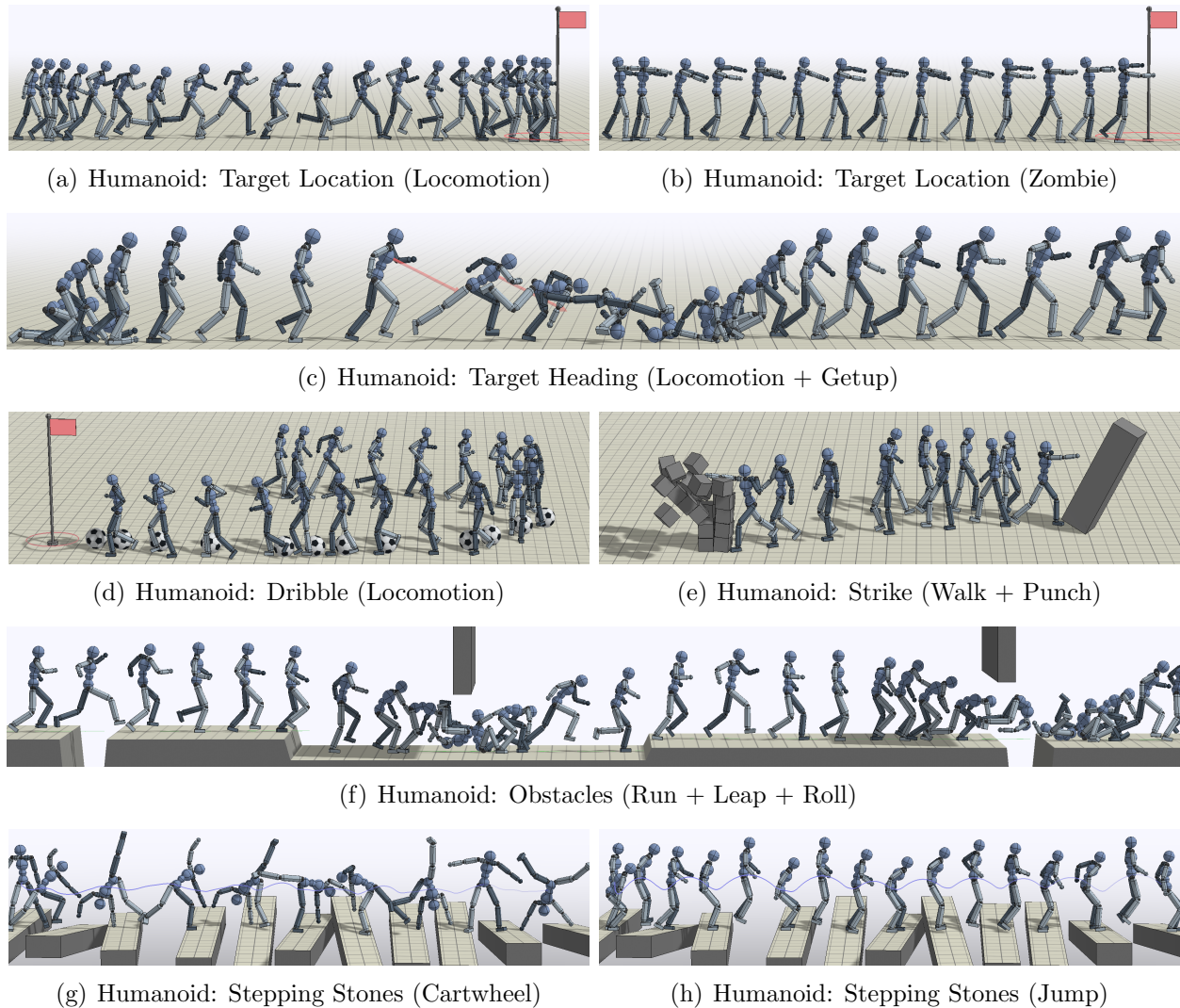


Figure 6.3: The motion prior can be trained with large datasets of diverse motions, enabling simulated characters to perform complex tasks by composing a wider range of skills. Each environment is denoted by "Character: Task (Dataset)".

Experimental Setup

All environments are simulated using the Bullet physics engine [30], with a simulation frequency of 1.2kHz. The policy is queried at 30Hz, and each action specifies target positions for PD controllers positioned at the character's joints. All neural networks are implemented using Tensorflow [171]. Reference motion clips are collected from a combination of public mocap libraries, custom recorded mocap clips, and artist-authored keyframe animations [43, 242, 313]. Hyperparameter settings used in the AMP experiments are available in Table 6.1. The gradient penalty coefficient is set to $w^{\text{gp}} = 10$. For single-clip imitation tasks, we found

Parameter	Value
w^G Task-Reward Weight	0.5
w^S Style-Reward Weight	0.5
w^{gp} Gradient Penalty	10
Samples Per Update Iteration	4096
Batch Size	256
K Discriminator Batch Size	256
π Policy Stepsize (Single-Clip Imitation)	2×10^{-6}
π Policy Stepsize (Tasks)	4×10^{-6}
V Value Stepsize (Single-Clip Imitation)	10^{-4}
V Value Stepsize (Tasks)	2×10^{-5}
D Discriminator Stepsize	10^{-5}
\mathcal{B} Discriminator Replay Buffer Size	10^5
γ Discount (Single-Clip Imitation)	0.95
γ Discount (Tasks)	0.99
SGD Momentum	0.9
GAE(λ)	0.95
TD(λ)	0.95
PPO Clip Threshold	0.2

Table 6.1: AMP hyperparameters.

that a smaller discount factor $\gamma = 0.95$ allows the character to more closely imitate a given reference motion. A larger discount factor $\gamma = 0.99$ is used for experiments that include additional task objective, since these tasks may require longer horizon planning, such as *Dribble* and *Strike*. Depending on the task and character, each policy is trained with 100-300 million samples, requiring about 30-140 hours on 16 CPU cores.

Tasks

In this section, we demonstrate AMP’s effectiveness for controlling the style of a character’s motions as it performs other high-level tasks. The weights for the task-reward and style-reward are set to $w^G = 0.5$ and $w^S = 0.5$ for all tasks. The character can be trained to perform tasks in a variety of distinct styles by providing the motion prior with different datasets. Figure 6.3 illustrates behaviors learned by the Humanoid on various tasks. Table 6.2 records the performance of the policies with respect to the normalized task return, and summary statistics of the different datasets used to train the motion priors are available in Table 6.3. AMP can accommodate large unstructured datasets, with the largest dataset containing 56 clips from 8 different human actors, for a total of 434s of motion data. In the case of the *Target Heading* task, a motion prior trained using a locomotion dataset, containing walking, running, and jogging motions, leads to a policy that executes different locomotion gaits depending on the target speed. Transitions between various gaits emerge

Character	Task	Dataset	Task Return
Humanoid	Target Heading	Locomotion	0.90 ± 0.01
		Walk	0.46 ± 0.01
		Run	0.63 ± 0.01
		Stealthy	0.89 ± 0.02
		Zombie	0.94 ± 0.00
	Target Location	Locomotion	0.63 ± 0.01
		Zombie	0.50 ± 0.00
	Obstacles	Run + Leap + Roll	0.27 ± 0.10
	Stepping Stones	Cartwheel	0.43 ± 0.03
		Jump	0.56 ± 0.12
	Dribble	Locomotion	0.78 ± 0.05
		Zombie	0.60 ± 0.04
	Strike	Walk + Punch	0.73 ± 0.02
T-Rex	Target Location	Locomotion	0.36 ± 0.03

Table 6.2: Performance statistics of combining AMP with additional task objectives. Performance is recorded as the average normalized task return, with 0 being the minimum possible return per episode and 1 being the maximum possible return. The return is averaged across 3 models initialized with different random seeds, with 32 episodes recorded per model. The motion prior can be trained with different datasets to produce policies that adopt distinct stylistic behaviors when performing a particular task.

automatically through the motion prior, with the character adopting walking gaits at slow speeds ($\sim 1\text{m/s}$), switching to jogging gaits at faster speeds ($\sim 2.5\text{m/s}$), and breaking into a fast run as the target speed approaches ($\sim 4.5\text{m/s}$). The motion prior also leads to other human-like strategies, such as banking into turns, and slowing down before large changes in direction. The policies develop similar behaviors for the *Target Location* task. When the target is near the character, the policy adopts slower walking gaits. But when the target is further away, the character automatically transitions into a run.

These intricate behaviors arise automatically from the motion prior, without requiring a motion planner to explicitly select which motion the character should execute in a given scenario, such as those used in prior systems [206, 20, 166]. In addition to standard locomotion gaits, the motion prior can also be trained for more stylistic behaviors, such as walking like a shambling zombie or walking in a stealthy manner. Our framework enables the character to acquire these distinct styles by simply providing the motion prior with different unstructured motion datasets.

To determine whether the transitions between distinct gaits are a product of the motion prior or a result of the task objective, we train policies to perform the *Target Heading* task using limited datasets containing only walking or running data. Figure 6.4 compares the

Character	Dataset	Size (s)	Clips	Subjects
Humanoid	Cartwheel	13.6	3	1
	Jump	28.6	10	4
	Locomotion	434.1	56	8
	Run	204.4	47	3
	Run + Leap + Roll	22.1	10	7
	Stealthy	136.5	3	1
	Walk	229.6	9	5
	Walk + Punch	247.8	15	9
	Zombie	18.3	1	1
T-Rex	Locomotion	10.5	5	1

Table 6.3: Summary statistics of the different datasets used to train the motion priors. We record the total length of motion clips in each dataset, along with the number of clips, and the number of subjects (e.g. human actors) that the clips were recorded from.

performance of policies trained with these different datasets. Policies trained with only walking motions learn to perform only walking gaits, and do not show any transitions to faster running gaits even at faster target speeds. As a result, these policies are not able to achieve the faster target speeds. Similarly, policies trained with only running motions are not able to match slower target speeds. Training the motion prior with a diverse dataset results in more flexible and optimal policies that are able to achieve a wider range of target speeds. This indicates that the diversity of behaviors exhibited by our policies can in large part be attributed to the motion prior, and is not solely a result of the task objective.

To further illustrate AMP’s ability to compose disparate skills, we introduce additional reference motions into the dataset for getting up from the ground in various configurations. These additional motion clips then enable our character to recover from a fall and continue to perform a given task (Figure 6.3(c)). The policy also discovers novel recovery behaviors that are not present in the dataset. When the character falls forward, it tucks its body into a roll during the fall in order to more quickly transition into a getup behavior. While this particular behavior is not present in the motion clips, the policy is able to generalize behaviors observed in the dataset to produce novel and naturalistic strategies for new scenarios.

For the *Strike* task (Figure 6.1), the motion prior is trained using a collection of walking motion clips and punching motion clips. The resulting policy learns to walk to the target when it is far away, and then transition to a punching motion once it is within range to hit the target. Note that the motion clips in the dataset contain strictly walking-only motions or punching-only motion, and none of the clips show an actor walking to and punching a target. Instead, the policy learns to temporally sequence these different behaviors in order to fulfill the high-level task objectives. Again, this composition of different skills emerges automatically from the motion prior, without requiring a motion planner or other mechanisms for motion selection.

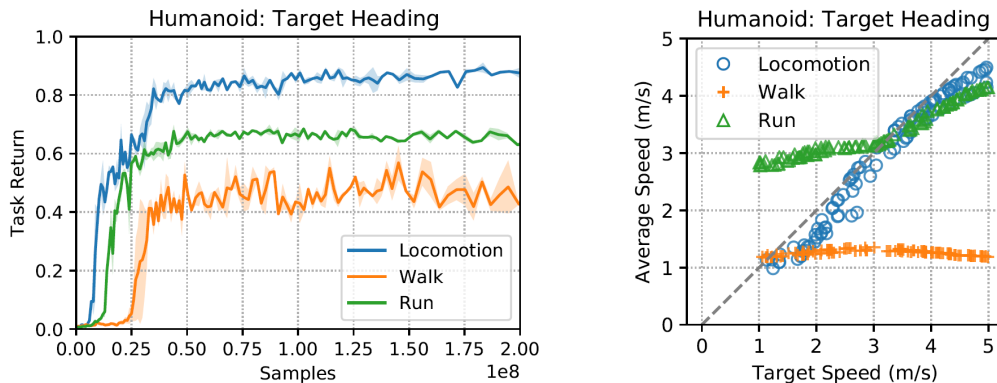


Figure 6.4: Performance of Target Heading policies trained with different datasets. **Left:** Learning curves comparing the normalized task returns of policies trained with a large dataset of diverse locomotion clips to policies trained with only walking or running reference motions. Three models are trained using each dataset. **Right:** Comparison of the target speed with the average speed achieved by the different policies. Policies trained using the larger Locomotion dataset is able to more closely follow the various target speeds by imitating different gaits.

Finally, our system can also train visuomotor policies for traversing obstacle-filled environments. By providing the motion prior with a collection of locomotion clips and rolling clips, the character learns to utilize these diverse behaviors to traverse the different obstacles. The character learns to leap over obstacles such as gaps. But as it approaches the overhead obstructions, the character transitions into a rolling behavior in order to pass underneath the obstacles. Previous systems that have demonstrated similar composition of diverse maneuvers for clearing obstacle have typically required a separate motion planner or manual annotations [162, 198]. Our approach provides a unified framework where the same underlying algorithm is able to learn how to perform the various skills and which skill to execute in a given scenario. Furthermore, the character can also be trained to traverse obstacles in distinct styles by providing the motion prior with different motion clips, such as jumping or cartwheeling across stepping stones (Figure 6.3).

Comparisons

An alternative approach for learning a motion prior from unstructured motion data is to build a latent space model [100, 167, 209, 174]. Unlike AMP, which encourages a character to adopt a desired motion style directly through an optimization objective, a latent space model enforces a particular motion style indirectly, by using a latent representation to constrain the policy’s actions to those that produce motions of the desired style. To compare AMP to these latent space models, we first pre-train a low-level controller using a motion tracking objective to imitate the same set of reference motions that are used to train the motion prior.

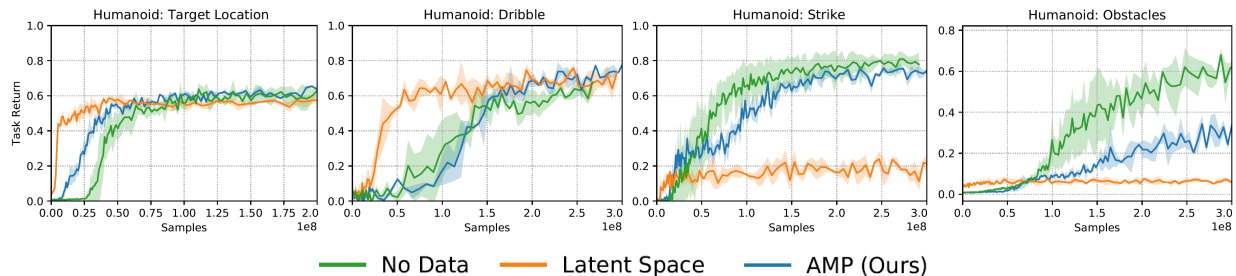


Figure 6.5: Learning curves comparing the task performance of AMP to latent space models (Latent Space) and policies trained from scratch without motion data (No Data). Our method achieves comparable performance across the various tasks, while also producing higher fidelity motions.

The learned low-level controller is then used to train separate high-level controllers for each downstream task. Note that reference motions are used only during pre-training, and the high-level controllers are trained to optimize only the task objectives.

A qualitative comparison of the behaviors learned using AMP and the latent space model is available in the supplementary video¹. Figure 6.5 compares the task performance of the different models, along with a baseline model trained from scratch for each task without leveraging any motion data. Both AMP and the latent space models are able to produce substantially more life-like behaviors than the baseline models. For the latent space models, since the low-level and high-level controllers are trained separately, it is possible for the distribution of encodings specified by the high-level controller to be different than the distribution of encodings observed by the low-level controller during pre-training [166]. This in turn can result in unnatural motions that deviate from the behaviors depicted in the original dataset. AMP enforces a motion style directly through the reward function, and is therefore able to better mitigate some of these artifacts. The more structured exploration behaviors from the latent space model enable the policies to solve downstream tasks more quickly. However, the pre-training stage used to construct the low-level controller can itself be sample intensive. In our experiments, the low-level controllers are trained using 300 million samples before being transferred to downstream tasks. With AMP, no such pre-training is necessary, and the motion prior can be trained jointly with the policy.

Single-Clip Imitation

Although our goal is to train characters with large motion datasets, to evaluate the effectiveness of our framework for imitating behaviors from motion clips, we consider a single-clip imitation task. In this setting, the character’s objective is to imitate a single motion clip at a time, without additional task objectives. Therefore, the policy is trained solely to maximize the style-reward r_t^S from the motion prior. Unlike previous motion tracking methods, our approach does not require a manually designed tracking objective or a phase-based synchro-

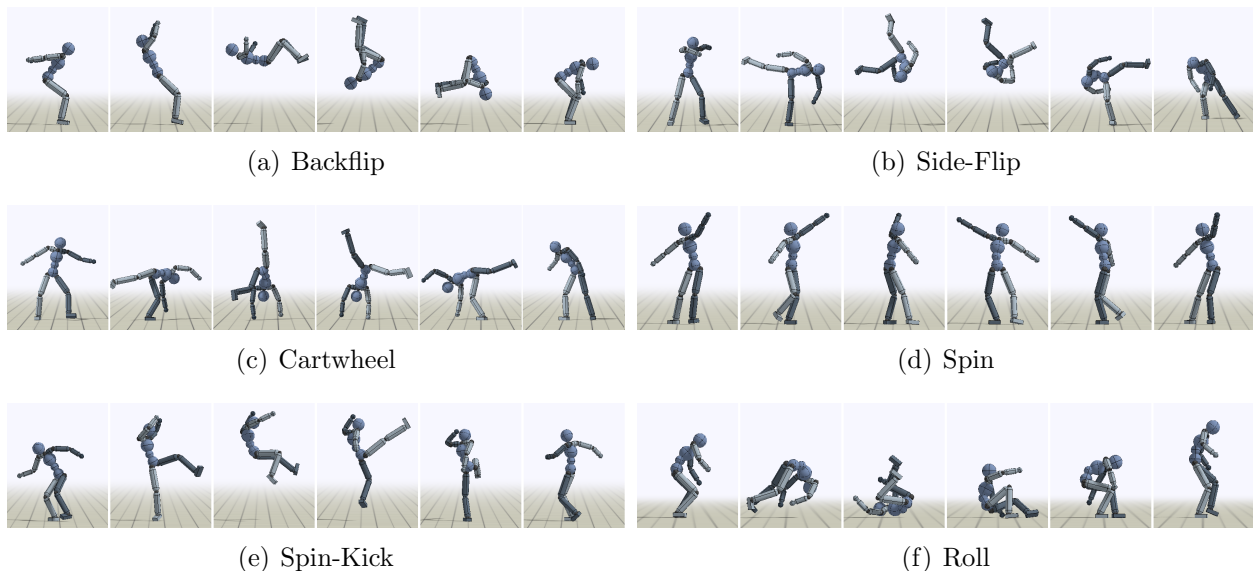


Figure 6.6: Snapshots of behaviors learned by the Humanoid on the single-clip imitation tasks. AMP enables the character to closely imitate a diverse corpus of highly dynamic and acrobatic skills.

nization of the reference motion and the policy Peng et al. [207]. Table 6.4 summarizes the performance of policies trained using AMP to imitate a diverse corpus of motions. Figure 6.6 and 6.7 illustrate examples of motions learned by the characters. Performance is evaluated using the average pose error, where the pose error e_t^{pose} at each time step t is computed between the pose of the simulated character and the reference motion using the relative positions of each joint with respect to the root (in units of meters),

$$e_t^{\text{pose}} = \frac{1}{N^{\text{joint}}} \sum_{j \in \text{joints}} \|(\mathbf{x}_t^j - \mathbf{x}_t^{\text{root}}) - (\hat{\mathbf{x}}_t^j - \hat{\mathbf{x}}_t^{\text{root}})\|_2. \quad (6.16)$$

\mathbf{x}_t^j and $\hat{\mathbf{x}}_t^j$ denote the 3D Cartesian position of joint j from the simulated character and the reference motion, and N^{joint} is the total number of joints in the character’s body. This method of evaluating motion similarity has previously been reported to better conform to human perception of motion similarity [93, 260]. Since AMP does not use a phase variable to synchronize the policy with the reference motion, the motions may progress at different rates, resulting in de-synchronization that can lead to large pose errors even when the overall motions are similar. To better evaluate the similarity of the motions, we first apply dynamic time warping (DTW) to align the reference motion with the motion of the simulated character [234], before computing the pose error between the two aligned motions. DTW is applied using Equation 6.16 as the cost function.

AMP is able to closely imitate a large variety of highly dynamic skills, while also avoiding many of the visual artifacts exhibited by prior adversarial motion imitation systems [176,

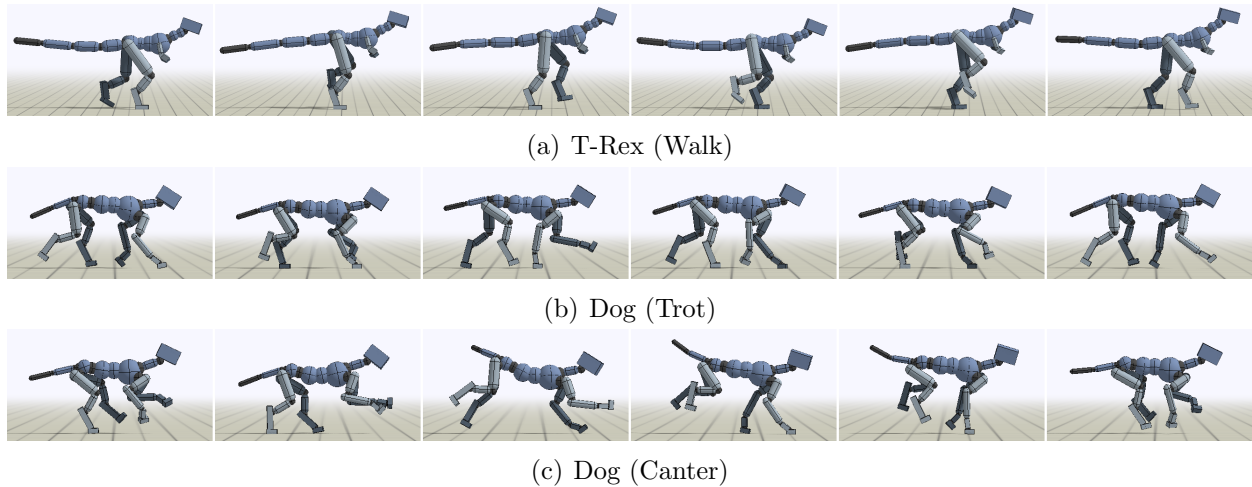


Figure 6.7: AMP can be used to train complex non-humanoid characters, such as a 59 DoF T-Rex and a 64 DoF dog. By providing the motion prior with different motion clips, the characters can be trained to perform various locomotion gaits, such as trotting and cantering.

287]. We compare the performance of our system to results produced by the motion tracking approach from Peng et al. [207], which uses a manually designed reward function and requires synchronization of the policy with a reference motion via a phase variable. Figure 6.8 compares the learning curves of the different methods. Since the tracking-based policies are synchronized with their respective reference motions, they are generally able to learn faster and achieve lower errors than policies trained with AMP. Nonetheless, our method is able to produce results of comparable quality without the need to manually design or tune reward functions for different motions. However, for some motions, such as the Front-Flip, AMP is prone to converging to locally optimal behaviors, where instead of performing a flip, the character learns to simply shuffle forwards in order to avoid falling. Tracking-based methods can mitigate these local optima by terminating an episode early if the character’s pose deviates too far from the reference motion [207, 290]. However, this strategy is not directly applicable to AMP, since the policy is not synchronized with the reference motion. But as shown in the previous sections, this lack of synchronization is precisely what allows AMP to easily leverage large datasets of diverse motion clips to solve more complex tasks.

Ablations

Our system is able to produce substantially higher fidelity motions than prior adversarial learning frameworks for physics-based character control [176, 287]. In this section, we identify critical design decisions that lead to more stable training and higher quality results. Figure 6.8 compares learning curves of policies trained on the single-clip imitation tasks with different components of the system disabled. Gradient penalty proves to be the most vital component. Models trained without this regularization tend to exhibit large performance

Character	Motion	Dataset Size	Motion Tracking	AMP (Ours)
Humanoid	Back-Flip	1.75s	0.076 ± 0.021	0.150 ± 0.028
	Cartwheel	2.72s	0.039 ± 0.011	0.067 ± 0.014
	Crawl	2.93s	0.044 ± 0.001	0.049 ± 0.007
	Dance	1.62s	0.038 ± 0.001	0.055 ± 0.015
	Front-Flip	1.65s	0.278 ± 0.040	0.425 ± 0.010
	Jog	0.83s	0.029 ± 0.001	0.056 ± 0.001
	Jump	1.77s	0.033 ± 0.001	0.083 ± 0.022
	Roll	2.02s	0.072 ± 0.018	0.088 ± 0.008
	Run	0.80s	0.028 ± 0.002	0.075 ± 0.015
	Spin	1.00s	0.063 ± 0.022	0.047 ± 0.002
	Side-Flip	2.44s	0.191 ± 0.043	0.124 ± 0.012
	Spin-Kick	1.28s	0.042 ± 0.001	0.058 ± 0.012
	Walk	1.30s	0.018 ± 0.005	0.030 ± 0.001
	Zombie	1.68s	0.049 ± 0.013	0.058 ± 0.014
T-Rex	Turn	2.13s	0.098 ± 0.011	0.284 ± 0.023
	Walk	2.00s	0.069 ± 0.005	0.096 ± 0.027
Dog	Canter	0.45s	0.026 ± 0.002	0.034 ± 0.002
	Pace	0.63s	0.020 ± 0.001	0.024 ± 0.003
	Spin	0.73s	0.026 ± 0.002	0.086 ± 0.008
	Trot	0.52s	0.019 ± 0.001	0.026 ± 0.001

Table 6.4: Performance statistics of imitating individual motion clips without task objectives. "Dataset Size" records the total length of motion data used for each skill. Performance is recorded as the average pose error (in units of meters) between the time-warped trajectories from the reference motion and simulated character. The pose error is averaged across 3 models initialized with different random seeds, with 32 episodes recorded per model. Each episode has a maximum length of 20s. We compare our method (AMP) with the motion tracking approach proposed by Peng et al. [207]. AMP is able to closely imitate a diverse repertoire of complex motions, without manual reward engineering.

fluctuations over the course of the training, and lead to noticeable visual artifacts in the final policies, as shown in the supplementary video. The addition of the gradient penalty not only improves stability during training, but also leads to substantially faster learning across a large set of skills. The inclusion of velocity features in the discriminator's observations is also an important component for imitating some motions. In principle, including consecutive poses as input to the discriminator should provide some information that can be used to infer the joint velocities. But we found that this was insufficient for some motions, such as rolling. As shown in the supplementary video, in the absence of velocity features, the character is prone to converging to a strategy of holding a fixed pose on the ground, instead of performing a roll. The additional velocity features mitigate these undesirable behaviors.

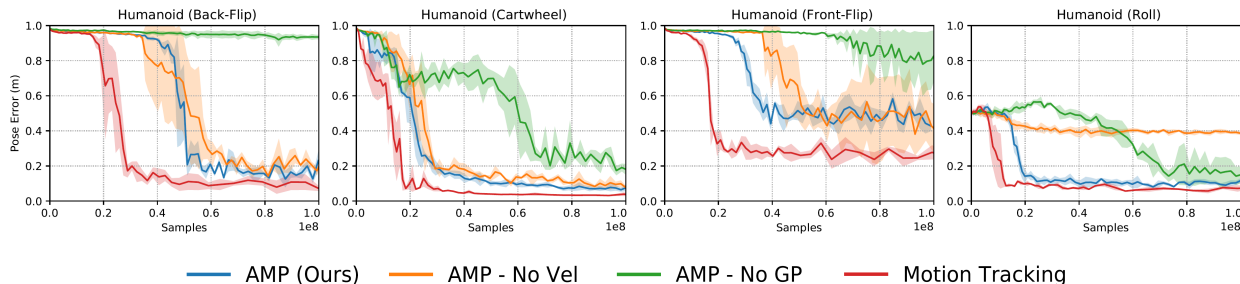


Figure 6.8: Learning curves of various methods on the single-clip imitation tasks. We compare AMP to the motion tracking approach proposed by Peng et al. [207] (Motion Tracking), as well a version of AMP without velocity features for the discriminator (AMP - No Vel), and AMP without the gradient penalty regularizer (AMP - No GP). AMP produces results of comparable quality when compared to prior tracking-based methods, without requiring a manually designed reward function or synchronization between the policy and reference motion. Velocity features and gradient penalty are vital for effective and consistent results on challenging skills.

6.8 Discussion

In this chapter, we presented an adversarial learning system for physics-based character animation that enables characters to imitate diverse behaviors from large unstructured datasets, without the need for motion planners or other mechanisms for clip selection. Our system allows users to specify high-level task objectives for controlling a character’s behaviors, while the more granular low-level style of a character’s motions can be controlled using a learned motion prior. Composition of disparate skills in furtherance of a task objective emerges automatically from the motion prior. The motion prior also enables our characters to closely imitate a rich repertoire of highly dynamic skills, and produces results that are on par with tracking-based techniques, without requiring manual reward engineering or synchronization between the controller and the reference motions.

Our system demonstrates that adversarial imitation learning techniques can indeed produce high fidelity motions for complex skills. However, like many other GAN-based techniques, AMP is susceptible to mode collapse. When provided with a large dataset of diverse motion clips, the policy is prone to imitating only a small subset of the example behaviors, ignoring other behaviors that may ultimately be more optimal for a given task. The motion priors in our experiments are also trained from scratch for each policy. But since the motion prior is largely task-agnostic, it should in principle be possible to transfer and reuse motion priors for different policies and tasks. Exploring techniques for developing general and transferable motion priors may lead to modular objective functions that can be conveniently incorporated into downstream tasks, without requiring retraining for each new task. While the motion prior does not require direct access to task-specific information, the data used to train the motion prior is generated by policies trained to perform a particular task. This

may introduce some task dependencies into the motion prior, which can hinder its ability to be transferred to other tasks. Training motion priors using data generated from larger and more diverse repertoires of tasks may help to facilitate transferring the learned motion priors to new tasks. Our experiments also focus primarily on tasks that involve temporal composition of different skills, which require the character to perform different behaviors at different points in time. However, spatial composition might also be vital for some tasks that require a character to perform multiple skills simultaneously. Developing motion priors that are more amenable to spatial composition of disparate skills may lead to more flexible and sophisticated behaviors. Despite these limitations, we hope this work provides a useful tool that enables physically simulated characters to take advantage of the large motion datasets that have been so effective for kinematic animation techniques, and open exciting directions for future exploration in data-driven physics-based character animation.

Chapter 7

Recent Related Work on Motion Imitation

The motion imitation approach presented in Chapter 3 can be effective for learning a wide range of skills. But the breadth of behaviors that can be modeled by an individual policy can be fairly limited, and our experiments in that chapter have primarily trained separate policies for each motion clip. Since the publication of that work, several subsequent works have extended this approach to train general policies that can imitate large libraries of diverse motion clips [39, 290]. Wang et al. [285] trained a single model to imitate thousands of motion clips by leveraging a fast GPU simulator [169], along with a number of careful design decisions, to drastically increase the number of samples that can be used to train a policy. The resulting model showed strong zero-shot generalization performance when used to imitating new motion clips. Fussell, Bergamin, and Holden [72] showed that model-based RL can also be applied to imitate large motion datasets through a similar tracking-based approach. By learning a differential dynamics model, their approach can produce lower variance gradient estimates, which empirically lead to higher quality motions and allow the system to imitate more intricate behaviors, such as dancing.

Our simulated characters have primarily been controlled using simplified actuation models (e.g. PD controllers), which do not capture the complex nonlinear dynamics of biological actuators (e.g. muscles). Lee et al. [143] applied a similar motion imitation technique to control characters with complex musculoskeletal systems. Their use of biologically-inspired actuators led to more naturalistic behaviors, particularly when the character is subjected to perturbations and novel scenarios that deviate from the context of the original reference motion data. Rather than training policies for controlling a specific character, Won and Lee [292] showed that by randomizing the morphology of the character during training, a more general policy can be trained to control characters with large morphological variations.

Motion imitation has also been used as an effective pre-training task for learning reusable motor skills in order to solve more challenging downstream tasks. Park et al. [198] and Bergamin et al. [20] combined general motion tracking policies with kinematic motion planners to control simulated characters to perform high-level tasks, such as navigation and

obstacle traversal. Merel et al. [174] used motion imitation to learn latent variable models of motor skills, which can then be used by high-level policies to solve challenging locomotion and manipulation tasks. Luo et al. [166] combined the MCP architecture from Chapter 5 with a domain confusion loss to prevent unnatural behaviors when training models to perform new tasks without motion data. Won, Gopinath, and Hodgins [291] used a similar hierarchical architecture to train policies for two-player sports, such as boxing and fencing.

One of the advantages of imitating skills from video clips is the ability to learn behaviors that are difficult to record using motion capture. Yu, Park, and Lee [304] used a similar video imitation framework as the one presented in Chapter 4 to learn figure skating skills from video clips. Figure skating motions are particularly difficult to record using mocap, since the routines often span large areas. Similar video imitation techniques have also been applied to learn dexterous manipulation skills from video clips [11]. In addition to leveraging vision-based pose estimation techniques to train simulated characters, physically simulated characters can also be used as priors for improving the physical realism of vision-based pose predictors [312].

Part II

Sim-To-Real Transfer

Chapter 8

Dynamics Randomization

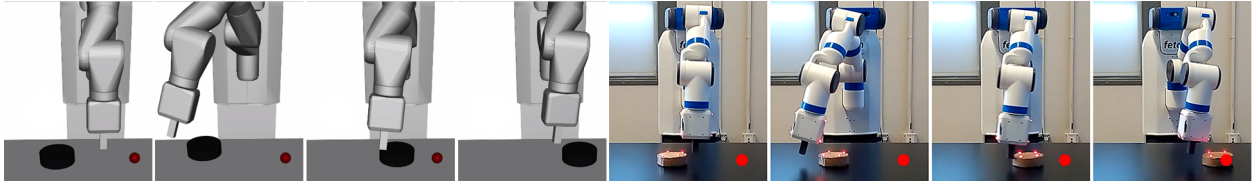


Figure 8.1: A recurrent neural network policy trained for a pushing task in simulation is deployed directly on a Fetch Robotics arm. The red marker indicates the target location for the puck. (Video¹)

In the previous part of this thesis, we showed that reinforcement learning can be a powerful framework for developing agents that are able to perform a diverse array of sophisticated skills in simulation. Unfortunately, many of the capabilities demonstrated by simulated agents have often not been realized by their physical counterparts. In the second part of this thesis, we will explore sim-to-real techniques for transferring policies trained in simulation to the real world. One of the obstacles for translating some of the successes of reinforcement learning in simulation to real-world domains is that many of the modern RL algorithms, which have spurred recent breakthroughs, require large amounts of data to learn effective policies, therefore often precluding their direct application to physical systems. In addition to sample complexity, deploying RL algorithms in the real world also raises a number of safety concerns both for the agent and its surroundings. Since exploration is a key component of the learning process, an agent can at times perform actions that endanger itself or its environment. Therefore, training agents in simulation is a promising approach that circumvents some of these obstacles. However, transferring policies from simulation to the real world entails challenges in bridging the "reality gap", the mismatch between the simulated and real world environments. Narrowing this gap has been a subject of intense interest

¹ Supplementary video: <https://xbpeng.github.io/projects/SimToReal/>

in robotics, as it offers the potential of applying powerful algorithms that have so far been relegated to simulated domains.

While significant efforts have been devoted to building higher fidelity simulators, we show that dynamics randomization using low fidelity simulations can also be an effective approach to develop policies that can be transferred directly to the real world. In this chapter, we provide a proof-of-concept for the effectiveness of this approach on an object pushing task, where a policy trained exclusively in simulation is able to successfully perform the task with a real robot without additional training on the physical system. Then in the following chapters, we will extend this technique to transfer more complex locomotion behaviors from simulation to robots operating in the real world.

8.1 Related Work

Recent years have seen the application of deep reinforcement learning to a growing repertoire of control problems. The framework has enabled simulated agents to develop highly dynamic motor skills [202, 206, 159, 99]. But due to the high sample complexity of RL algorithms and other physical limitations, many of the capabilities demonstrated in simulation have yet to be replicated in the physical world. Guided Policy Search (GPS) [147] represents one of the few algorithms capable of training policies directly on a real robot. By leveraging trajectory optimization with learned linear dynamics models, the method is able to develop complex manipulation skills with relatively few interactions with the environment. The method has also been extended to learning vision-based manipulation policies [149]. Researchers have also explored parallelizing training across multiple robots [150]. Nonetheless, successful examples of training policies directly on physical robots have so far been demonstrated only on relatively restrictive domains.

Domain Adaptation

The problem of transferring control policies from simulation to the real world can be viewed as an instance of domain adaptation, where a model trained in a source domain is transferred to a new target domain. One of the key assumptions behind these methods is that the different domains share common characteristics such that representations and behaviours learned in one will prove useful for the other. Learning invariant features has emerged as a promising approach of taking advantage of these commonalities [276, 73]. Tzeng et al. [276] and Gupta et al. [87] explored using pairwise constraints to encourage networks to learn similar embeddings for samples from different domains that are labeled as being similar. Daftry, Bagnell, and Hebert [50] applied a similar approach to transfer policies for controlling aerial vehicles to different environments and vehicle models. In the context of RL, adversarial losses have been used to transfer policies between different simulated domains, by encouraging agents to adopt similar behaviours across the various environments [294]. Alternatively, progressive networks have also been used to transfer policies for a robotic

arm from simulation to the real world [230]. By reusing features learned in simulation, their method was able to significantly reduce the amount of data needed from the physical system. Christiano et al. [40] transferred policies from simulation to a real robot by training an inverse-dynamics model from real world data. While promising, these methods nonetheless still require data from the target domain during training.

Domain Randomization

Domain randomization is a complementary class of techniques for adaptation that is particularly well suited for simulation. With domain randomization, discrepancies between the source and target domains are modeled as variability in the source domain. Randomization in the visual domain has been used to directly transfer vision-based policies from simulation to the real world without requiring real images during training [232, 269]. Sadeghi and Levine [232] trained vision-based controllers for a quadrotor using only synthetically rendered scenes, and Tobin et al. [269] demonstrated transferring image-based object detectors. Unlike previous methods, which sought to bridge the reality gap with high fidelity rendering [115], their systems used only low fidelity rendering and modeled differences in visual appearance by randomizing scene properties such as lighting, textures, and camera placement. In addition to randomizing the visual features of a simulation, randomized dynamics have also been used to develop controllers that are robust to uncertainty in the dynamics of the system. Mordatch, Lowrey, and Todorov [181] used a trajectory optimizer to plan across an ensemble of dynamics models, to produce robust trajectories that are then executed on a real robot. Their method allowed a Darwin robot to perform a variety of locomotion skills. But due to the cost of the trajectory optimization step, the planning is performed offline. Other methods have also been proposed to develop robust policies through adversarial training schemes [222, 212]. Yu, Liu, and Turk [307] trained a system identification module to explicitly predict parameters of interest, such as mass and friction. The predicted parameters are then provided as input to a policy to compute the appropriate controls. While the results are encouraging, these methods have so far only been demonstrated on transfer between different simulators.

The work most reminiscent to our proposed method is that of Antonova et al. [10], where randomized dynamics was used to transfer manipulation policies from simulation to the real world. By randomizing physical parameters such as friction and latency, they were able to train policies in simulation for pivoting objects held by a gripper, and later transfer the policies directly to a Baxter robot without requiring additional fine-tuning on the physical system. However their policies were modeled using memoryless feedforward networks, and while the policies developed robust strategies, the lack of internal state limits the feedforward policies' ability to adapt to mismatch between the simulated and real environment. We show that memory-based policies are able to cope with greater variability during training and also better generalize to the dynamics of the real world. Unlike previous methods which often require meticulous calibration of the simulation to closely conform to the physical system, our policies are able to adapt to significant calibration error.

Non-Prehensile Manipulation

Pushing, a form of non-prehensile manipulation, is an effective strategy for positioning and orienting objects that are too large or heavy to be grasped [303]. Though pushing has attracted much interest from the robotics community [168, 55, 62], it remains a challenging skill for robots to adopt. Part of the difficulty stems from accurately modeling the complex contact dynamics between surfaces. Characteristics such as friction can vary significantly across the surface of an object, and the resulting motions can be highly sensitive to the initial configuration of the contact surfaces [303]. Models have been proposed to facilitate planning algorithms [168, 6, 55], but they tend to rely on simplifying assumptions that are often violated in practice. More recently, deep learning methods have been applied to train predictive models for pushing [66]. While data-driven methods overcome some of the modeling challenges faced by previous frameworks, they require a large corpus of real world data during training. Such a dataset can be costly to collect, and may become prohibitive for more complex tasks. Ignasi Clavera and Abbeel [112] demonstrated transferring pushing policies trained in simulation to a real PR2. Their approach took advantage of shaped reward functions and careful calibration to ensure that the behaviour of the simulation conforms to that of the physical system. In contrast, we will show that adaptive policies can be trained exclusively in simulation and using only sparse rewards. The resulting policies are able to accommodate large calibration errors when deployed on a real robot and also generalize to variability in the dynamics of the physical system.

8.2 Hindsight Experience Replay

During training, RL algorithms often benefit from carefully shaped reward functions that help guide the agent towards fulfilling the overall objective of a task. But designing a reward function can be challenging for more complex tasks, and may bias the policy towards adopting less optimal behaviours. An alternative is to use a binary reward $r(\mathbf{s}, \mathbf{g})$ that only indicates if a goal is satisfied in a given state,

$$r(\mathbf{s}, \mathbf{g}) = \begin{cases} 0, & \text{if } \mathbf{g} \text{ is satisfied in } \mathbf{s} \\ -1, & \text{otherwise} \end{cases} \quad (8.1)$$

Learning from a sparse binary reward is known to be challenging for most modern RL algorithms. We will therefore leverage a recent innovation, Hindsight Experience Relay (HER) [9], to train policies using sparse rewards. Consider an episode with trajectory $\tau \in (\mathbf{s}_0, \mathbf{a}_0, \dots, \mathbf{a}_{T-1}, \mathbf{s}_T)$, where the goal \mathbf{g} was not satisfied over the course the trajectory. Since the goal was not satisfied, the reward will be -1 at every timestep, therefore providing the agent with little information on how to adjust its actions to procure more rewards. But suppose that we are provided with a mapping $m : \mathcal{S} \rightarrow \mathcal{G}$, that maps a state to the corresponding goal satisfied in the given state. For example, $m(\mathbf{s}_T) = \mathbf{g}'$ represents the goal that is satisfied in the final state of the trajectory. Once a new goal has been determined,

rewards can be recomputed for the original trajectory under the new goal \mathbf{g}' . While the trajectory was unsuccessful under the original goal, it becomes a successful trajectory under the new goal. Therefore, the rewards computed with respect to \mathbf{g}' will not be -1 for every timestep. By replaying past experiences with HER, the agent can be trained with more successful examples than is available in the original recorded trajectories. So far, we have only considered replaying goals from the final state of a trajectory. But HER is also amenable to other replay strategies, and we refer interested readers to the original paper [9] for more details.

8.3 Method

Our objective is to train policies that can perform a task under the dynamics of the real world $p^*(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$. Since sampling from the real world dynamics can be prohibitive, we instead train a policy using an approximate dynamics model $\hat{p}(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t) \approx p^*(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$ that is easier to sample from. For all of our experiments, \hat{p} assumes the form of a physics simulation. Due to modeling and other forms of calibration error, behaviours that successfully accomplish a task in simulation may not be successful once deployed in the real world. Furthermore, it has been observed that DeepRL policies are prone to exploiting idiosyncrasies of the simulator to realize behaviours that are infeasible in the real world [155, 99]. Therefore, instead of training a policy under one particular dynamics model, we train a policy that can perform a task under a variety of different dynamics models. First we introduce a set of dynamics parameters μ that parameterizes the dynamics of the simulation $\hat{p}(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t, \mu)$. The objective is then modified to maximize the expected return across a distribution of dynamics models ρ_μ ,

$$\mathbb{E}_{\mu \sim \rho_\mu} \left[\mathbb{E}_{\tau \sim p(\tau|\pi, \mu)} \left[\sum_{t=0}^{T-1} r(\mathbf{s}_t, \mathbf{a}_t) \right] \right] \quad (8.2)$$

By training policies to adapt to variability in the dynamics of the environment, the resulting policy might then better generalize to the dynamics of real world.

Tasks

Our experiments are conducted on a puck pushing task using a 7-DOF Fetch Robotics arm. Images of the real robot and simulated model is available in Figure 8.2. The goal g for each episode specifies a random target position on the table that the puck should be moved to. The reward is binary with $r_t = 0$ if the puck is within a given distance of the target, and $r_t = -1$ otherwise. At the start of each episode, the arm is initialized to a default pose and the initial location of the puck is randomly placed within a fixed bound on the table.

State and Action

The state is represented using the joint positions and velocities of the arm, the position of the gripper, as well as the puck’s position, orientation, linear and angular velocities. The combined features result in a 52D state space. Actions from the policy specify target joint angles for a position controller. Target angles are specified as relative offsets from the current joint rotations. This yields a 7D action space.

Dynamics Randomization

During training, rollouts are organized into episodes of a fixed length. At the start of each episode, a random set of dynamics parameters μ are sampled according to ρ_μ and held fixed for the duration of the episode. The parameters which we randomize include:

- Mass of each link in the robot’s body
- Damping of each joint
- Mass, friction, and damping of the puck
- Height of the table
- Gains for the position controller
- Timestep between actions
- Observation noise

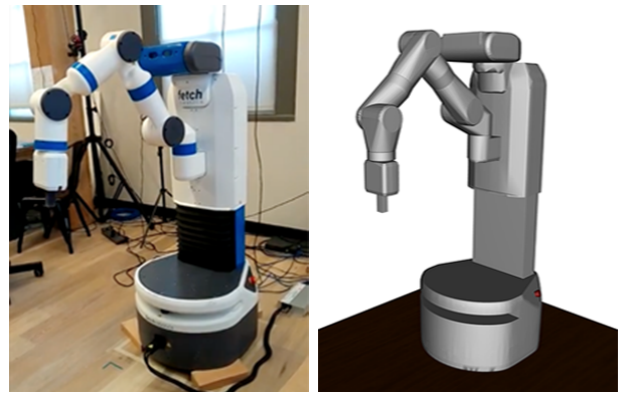


Figure 8.2: Our experiments are conducted on a 7-DOF Fetch Robotics arm. **Left:** Real robot. **Right:** Simulated MuJoCo model.

which results in a total of 95 randomized parameters. The timestep between actions specifies the amount of time an action is applied before the policy is queried again to sample a new action. This serves as a simple model of the latency exhibited by the physical controller. The observation noise models uncertainty in the sensors and is implemented as independent Gaussian noise applied to each state feature. While parameters such as mass and damping are constant over the course of an episode, the action timestep and the observation noise varies randomly each timestep.

Adaptive Policy

Manipulation tasks, such as pushing, have a strong dependency on the physical properties of the system (e.g. mass, friction, and characteristics of the actuators). In order to determine the appropriate actions, a policy requires some means of inferring the underlying

Algorithm 4 Dynamics Randomization with HER and RDPG

```

1:  $\theta \leftarrow$  random weights
2:  $\varphi \leftarrow$  random weights
3: while not done do
4:    $\mathbf{g} \sim \rho_{\mathbf{g}}$  sample goal
5:    $\mu \sim \rho_{\mu}$  sample dynamics
6:   Generate rollout  $\tau = (\mathbf{s}_0, \mathbf{a}_0, \dots, \mathbf{s}_T)$  with dynamics  $\mu$ 
7:   for each  $\mathbf{s}_t, \mathbf{a}_t$  in  $\tau$  do
8:      $r_t \leftarrow r(\mathbf{s}_t, \mathbf{g})$ 
9:   end for
10:  Store  $(\tau, \{r_t\}, \mathbf{g}, \mu)$  in  $M$ 
11:  Sample episode  $(\tau, \{r_t\}, \mathbf{g}, \mu)$  from  $M$ 
12:  with probability  $k$ 
13:     $\mathbf{g} \leftarrow$  replay new goal with HER
14:     $r_t \leftarrow r(\mathbf{s}_t, \mathbf{g})$  for each  $t$ 
15:  endwith

16: for each  $t$  do
17:   Compute memories  $z_t$  and  $y_t$ 
18:    $\hat{\mathbf{a}}_{t+1} \leftarrow \pi_{\theta}(\mathbf{s}_{t+1}, \mathbf{z}_{t+1}, \mathbf{g})$ 
19:    $\hat{\mathbf{a}}_t \leftarrow \pi_{\theta}(\mathbf{s}_t, \mathbf{z}_t, \mathbf{g})$ 
20:    $q_t \leftarrow r_t + \gamma Q_{\varphi}(\mathbf{s}_{t+1}, \hat{\mathbf{a}}_{t+1}, \mathbf{y}_{t+1}, \mathbf{g}, \mu)$ 
21:    $\Delta q_t \leftarrow q_t - Q_{\varphi}(\mathbf{s}_t, \mathbf{a}_t, \mathbf{y}_t, \mathbf{g}, \mu)$ 
22: end for
23:  $\nabla_{\varphi} = \frac{1}{T} \sum_t \Delta q_t \frac{\partial Q_{\varphi}(\mathbf{s}_t, \mathbf{a}_t, \mathbf{y}_t, \mathbf{g}, \mu)}{\partial \varphi}$ 
24:  $\nabla_{\theta} = \frac{1}{T} \sum_t \frac{\partial Q_{\varphi}(\mathbf{s}_t, \hat{\mathbf{a}}_t, \mathbf{y}_t, \mathbf{g}, \mu)}{\partial \mathbf{a}} \frac{\partial \hat{\mathbf{a}}_t}{\partial \theta}$ 
25: Update value function and policy with  $\nabla_{\theta}$  and  $\nabla_{\varphi}$ 
26: end while

```

dynamics of its environment. While the dynamics parameters are readily available in simulation, the same does not hold once a policy has been deployed in the real world. In the absence of direct knowledge of the parameters, the dynamics can be inferred from a history of past states and actions. System identification using a history of past trajectories has been previously explored by Yu, Liu, and Turk [307]. Their system incorporates an online system identification module $\phi(\mathbf{s}_t, \mathbf{h}_t) = \hat{\mu}$, which utilizes a history of past states and actions $\mathbf{h}_t = [\mathbf{a}_{t-1}, \mathbf{s}_{t-1}, \mathbf{a}_{t-2}, \mathbf{s}_{t-2}, \dots]$ to predict the dynamics parameters μ . The predicted parameters are then used as inputs to a universal policy that samples an action according to the current state and inferred dynamics $\pi(\mathbf{a}_t | \mathbf{s}_t, \hat{\mu})$. However, this decomposition requires identifying the dynamics parameters of interest to be predicted at runtime, which may be difficult for more complex systems. Constructing such a set of parameters necessarily requires some structural assumptions about the dynamics of a system, which may not hold

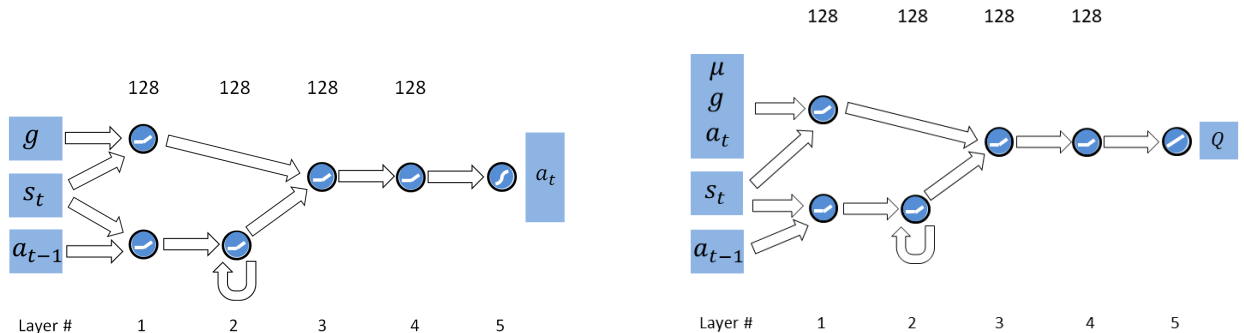


Figure 8.3: Schematic illustrations of the policy network (**top**), and value network (**bottom**). Features that are relevant for inferring the dynamics of the environment are processed by the recurrent branch, while the other inputs are processed by the feedforward branch.

in the real world. Alternatively, SysID can be implicitly embedded into a policy by using a recurrent model $\pi(\mathbf{a}_t | \mathbf{s}_t, \mathbf{z}_t, \mathbf{g})$, where the internal memory $\mathbf{z}_t = z(\mathbf{h}_t)$ acts as a summary of past states and actions, thereby providing a mechanism with which the policy can use to infer the dynamics of the system. This model can then be trained end-to-end and the representation of the internal memory can be learned without requiring manual identification of a set of dynamics parameters to be inferred at runtime.

Recurrent Deterministic Policy Gradient

Since HER augments the original training data recorded from rollouts of the policy with additional data generated from replayed goals, it requires off-policy learning. Deep Deterministic Policy Gradient (DDPG) [155] is a popular off-policy algorithm for continuous control. Its extension to recurrent policies, Recurrent Deterministic Policy Gradient (RDPG) [101], provides a method to train recurrent policies with off-policy data. To apply RDPG, we denote a deterministic policy as $\pi(\mathbf{s}_t, \mathbf{z}_t, \mathbf{g}) = \mathbf{a}_t$. In addition to the policy, we will also model a recurrent universal value function as $Q(\mathbf{s}_t, \mathbf{a}_t, \mathbf{y}_t, \mathbf{g}, \mu)$, where $\mathbf{y}_t = y(\mathbf{h}_t)$ is the value function’s internal memory. Since the value function is used only during training and the dynamics parameters μ of the simulator are known, μ is provided as an additional input to the value function but not to the policy. We will refer to a value function with knowledge of the dynamics parameters as an *omniscient critic*. This follows the approach of [68, 164], where additional information is provided to the value function during training in order to reduce the variance of the policy gradients and allow the value function to provide more meaningful feedback for improving the policy.

Algorithm 4 summarizes the training procedure, where M represents a replay buffer [155], and θ and φ are the parameters for the policy and value function respectively. We also incorporate target networks [155], but they are excluded for brevity.

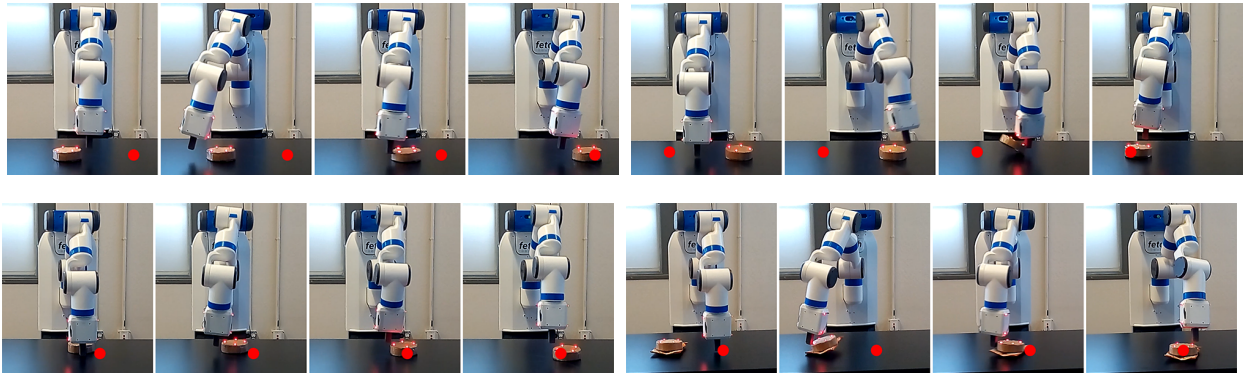


Figure 8.4: LSTM policy deployed on the Fetch arm. **Bottom-Right:** The contact dynamics of the puck was modified by attaching a packet of chips to the bottom.

Network Architecture

A schematic illustrations of the policy and value networks are available in Figure 8.3. The inputs to the network consist of the current state \mathbf{s}_t and previous action \mathbf{a}_{t-1} , and the internal memory is updated incrementally at every step. Each network consists of a feedforward branch and recurrent branch, with the latter being tasked with inferring the dynamics from past observations. The internal memory is modeled using a layer of LSTM units and is provided only with information required to infer the dynamics (e.g. \mathbf{s}_t and \mathbf{a}_{t-1}). The recurrent branch consists of an embedding layer of 128 fully-connected units followed by 128 LSTM units. The goal \mathbf{g} does not hold any information regarding the dynamics of the system, and is therefore processed only by the feedforward branch. Furthermore, since the current state \mathbf{s}_t is of particular importance for determining the appropriate action for the current timestep, a copy is also provided as input to the feedforward branch. This presents subsequent layers with more direct access to the current state, without requiring information to filter through the LSTM. The features computed by both branches are then concatenated and processed by 2 additional fully-connected layers of 128 units each. The value network $Q(\mathbf{s}_t, \mathbf{a}_t, \mathbf{a}_{t-1}, \mathbf{g}, \mu)$ follows a similar architecture, with the query action a_t and parameters μ being processed by the feedforward branch. ReLU activations are used after each hidden layer (apart from the LSTM). The output layer of Q consists of linear units, while π consists of tanh output units scaled to span the bounds of each action parameter.

8.4 Experiments

Results are best seen in the supplemental video¹. Snapshots of policies deployed on the real robot are available in Figure 8.4. All simulations are performed using the MuJoCo physics engine [270] with a simulation timestep of 0.002s. 20 simulation timesteps are performed for every control timestep. Each episode consists of 100 control timestep, corresponding

Parameter	Range
Link Mass	$[0.25, 4] \times$ default mass of each link
Joint Damping	$[0.2, 20] \times$ default damping of each joint
Puck Mass	$[0.1, 0.4] kg$
Puck Friction	$[0.1, 5]$
Puck Damping	$[0.01, 0.2] Ns/m$
Table Height	$[0.73, 0.77] m$
Controller Gains	$[0.5, 2] \times$ default gains
Action Timestep λ	$[125, 1000] s^{-1}$

Table 8.1: Dynamics parameters and their respective ranges.

to approximately 4 seconds per episode, but may vary as a result of the random timesteps between actions. Table 8.1 details the range of values for each dynamics parameter. At the start of each episode, a new set of parameters μ is sampled by drawing values for each parameter from their respective range. Parameters such as mass, damping, friction, and controller gains are logarithmically sampled, while other parameters are uniformly sampled. The timestep Δt between actions varies every step according to $\Delta t \sim \Delta t_0 + \text{Exp}(\lambda)$, where $\Delta t_0 = 0.04s$ is the default control timestep, and $\text{Exp}(\lambda)$ is an exponential distribution with rate parameter λ . While Δt varies every timestep, λ is fixed within each episode. In addition to randomizing the physical properties of the simulated environment, we also simulate sensor noise by applying gaussian noise to the observed state features at every step. The noise has a mean of zero and a standard deviation of 5% of the running standard deviation of each feature. Gaussian action exploration noise is added at every step with a standard deviation of $0.01rad$.

The real puck has a mass of approximately $0.2kg$ and a radius of $0.065m$. The goal is considered satisfied if the puck is within $0.07m$ of the target. The location of the puck is tracked using the PhaseSpace mocap system. When evaluating performance on the physical system, each episode consists of 200 timesteps. Little calibration

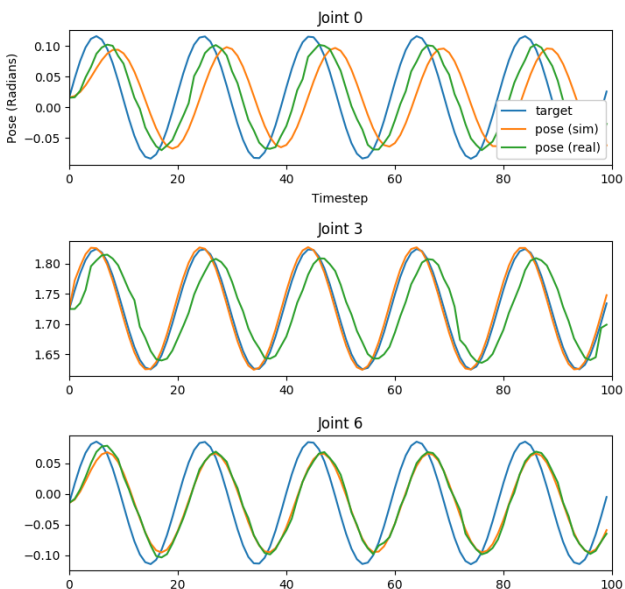


Figure 8.5: Joint trajectories recorded from the simulated and real robot when executing the same target trajectories. The joints correspond to the shoulder, elbow, and wrist of the Fetch arm.

Model	Success (Sim)	Success (Real)	Trials (Real)
LSTM	0.91 ± 0.03	0.89 ± 0.06	28
FF no Rand	0.51 ± 0.05	0.0 ± 0.0	10
FF	0.83 ± 0.04	0.67 ± 0.14	12
FF + Hist	0.87 ± 0.03	0.70 ± 0.10	20

Table 8.2: Performance of the policies when deployed on the simulated and real robot. Performance in simulation is evaluated over 100 trials with randomized dynamics parameters.

was performed to ensure that the behaviour of the simulation closely conforms to that of the real robot. While more extensive calibration will likely improve performance, we show that our policy is nonetheless able to adapt to the physical system despite poor calibration. To illustrate the discrepancies between the dynamics of the real world and simulation we executed the same target trajectory on the real and simulated robot, and recorded the resulting joint trajectories. Figure 8.5 illustrates the recorded trajectories. Given the same target trajectory, the pose trajectories of the simulated and real robot differ significantly, with varying degrees of mismatch across joints.

During training, parameter updates are performed using the ADAM optimizer [124] with a stepsize of 5×10^{-4} for both the policy and value function. Updates are performed using batches of 128 episodes with 100 steps per episode. New goals are sampled using HER with a probability of $k = 0.8$. Each policy is trained for approximately 8000 update iterations using about 100 million samples, which requires approximately 8 hours to simulate on a 100 core cluster.

Comparison of Architectures

To evaluate the impact of different architectural choices, we compared policies modeled using different architectures and tested their performance in simulation and on the real robot. The first is an LSTM policy following the architecture illustrated in Figure 8.3. Next we consider a memoryless feedforward network (FF) that receives only the current state s_t and goal g as input. As a baseline, we also trained a memoryless feedforward network without randomization (FF no Rand), then evaluated the performance with randomization. To provide the feedforward network with more information to infer the dynamics, we augmented the inputs with a history of the 8 previously observed states and actions (FF + Hist). The success rate is determined as the portion of episodes where the goal is fulfilled at the end of the episode. In simulation, performance of each policy is evaluated over 100 episodes, with randomized dynamics parameters for each episode. Learning curves comparing the performance of different model architectures in simulation are available in Figure 8.6. Four policies initialized with different random seeds are trained for each architecture. The LSTM learns faster while also converging to a higher success rate than the feedforward models. The

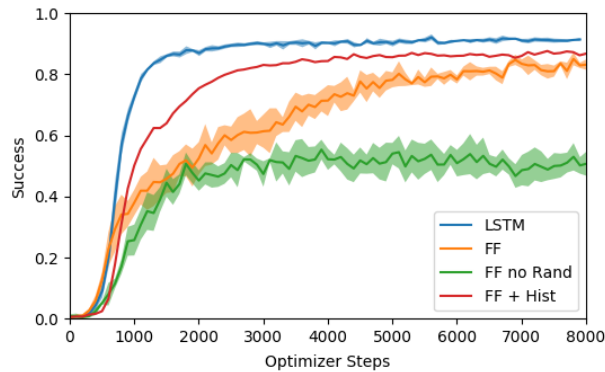


Figure 8.6: Learning curves of different network architectures. Four policies are trained for each architecture with different random initializations. Performance is evaluated over 100 episodes in simulation with random dynamics.

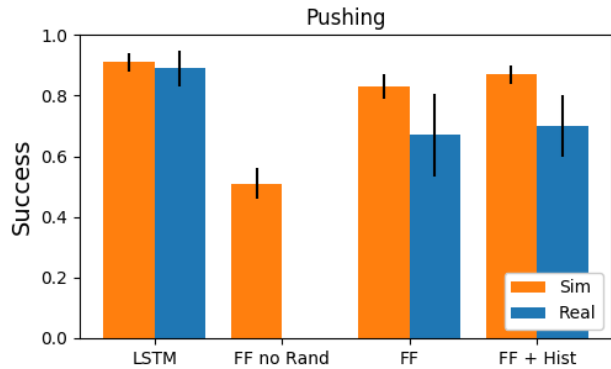


Figure 8.7: Performance of different models when deployed on the simulated and real robot for the pushing task. Policies are trained using only data from simulation.

feedforward network trained without randomization is unable to cope with unfamiliar dynamics during evaluation. While training a memoryless policy with randomization improves robustness to random dynamics, it is still unable to perform the task consistently.

Next, we evaluate the performance of the different models when deployed on the real Fetch arm. Figure 8.7 compares the performance of the final policies when deployed in simulation and the real world. Table 8.2 summarizes the performance of the models. The target and initial location of the puck is randomly placed within a $0.3m \times 0.3m$ bound. While the performance of LSTM and FF + Hist policies are comparable in simulation, the LSTM is able to better generalize to the dynamics of the physical system. The feedforward network trained without randomization is unable to perform the task under the real world dynamics.

Ablation

To evaluate the effects of randomizing the various dynamics parameters, we trained policies with subsets of the parameters held fixed. A complete list of the dynamics parameters are available in Table 8.1. The configurations we consider include training with a fixed timestep between actions, training without observation noise, or with fixed mass for each link. Table 8.3 summarizes the performance of the resulting policies when deployed on the real robot. Disabling randomization of the action timestep, observation noise, link mass, and friction impairs the policies' ability to adapt to the physical environment. Policies trained without randomizing the action timestep and observation noise show particularly noticeable drops in performance. This suggests that coping with the latency of the controller and sensor noise are important factors in adapting to the physical system.

Model	Success	Trials
all	0.89 ± 0.06	28
fixed action timestep	0.29 ± 0.11	17
no observation noise	0.25 ± 0.12	12
fixed link mass	0.64 ± 0.10	22
fixed puck friction	0.48 ± 0.10	27

Table 8.3: Performance of LSTM policies on the real robot, where the policies are trained with subsets of parameters held fixed.

Robustness

To evaluate the robustness of the LSTM policy to different dynamics when deployed on the real robot, we experimented with changing the contact dynamics of the physical system by attaching a packet of chips to the bottom of the puck. The texture of the bag reduces the friction between the puck and the table, while the contents of the bag further alters the contact dynamics. Nonetheless, the LSTM policy achieves a success rate of 0.91 ± 0.04 , which is comparable to the success rate without the attachment 0.89 ± 0.06 . The policy also develops clever strategies to make fine adjustments to position the puck over the target. One such strategy involves pressing on one side of the puck in order to partially unpend it before sliding it to the target. Other strategies including manipulating the puck from the top or sides depending on the required adjustments, and correcting for case where the puck overshoots the target. These behaviours emerged naturally from the learning process using only a sparse binary reward.

8.5 Discussion

In this chapter, we demonstrated the use of dynamics randomization to train recurrent policies that are capable of adapting to unfamiliar dynamics at run-time. Training policies with randomized dynamics in simulation enables the resulting policies to be deployed directly on a physical robot despite poor calibrations. By training exclusively in simulation, we are able to leverage simulators to generate a large volume of training data, thereby enabling us to use powerful RL techniques that are not yet feasible to apply directly on a physical system. Our experiments with a real world pushing tasks showed comparable performance to simulation and the ability to adapt to changes in contact dynamics. We also evaluated the importance of design decisions pertaining to choices of architecture and parameters which to randomize during training. In the next chapters, we will extend this approach to transfer policies for more challenging locomotion tasks to legged robots in the real world.

Chapter 9

Bipedal Locomotion

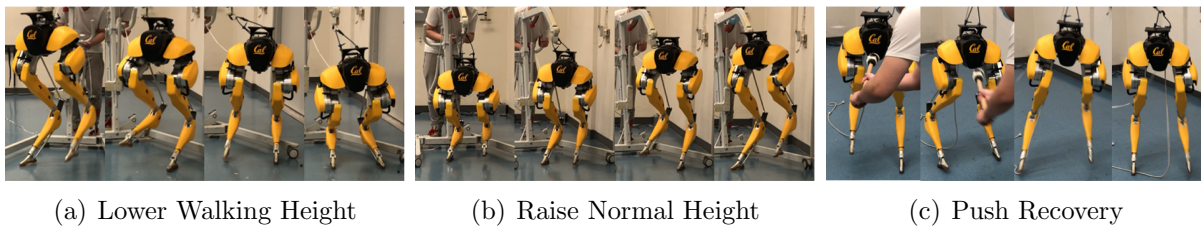


Figure 9.1: Our system is able to learn robust parameterized locomotion controllers for a bipedal Cassie robot. The controller can vary parameters such as walking velocity and height, while also being robust to significant external perturbations. (Video¹)

In this chapter, we combine the dynamics randomization techniques discussed in Chapter 8 with the motion imitation method from Chapter 3 to develop locomotion controllers for a bipedal Cassie robot². Many environments, particularly those designed for humans, are more accessible by legged systems. However, designing locomotion controllers for bipedal systems involves many challenges due to the high degrees-of-freedom (DoFs), hybrid non-linear dynamics, and persistent but hard-to-model ground impacts. Classical model-based methods [280, 131, 84] for stabilizing and controlling bipedal systems tend to require careful modeling and usually lack the ability to adapt to changes in the environment. In this work, we present a system that addresses these challenges of bipedal locomotion by using reinforcement learning. To develop policies that are capable of performing diverse gaits, we train the policies to imitate motion clips from a gait library containing diverse walking motions. The gait library is constructed using Hybrid Zero Dynamics (HZD) [84], but our system can also easily accommodate motions from other sources, such as mocap. This diverse dataset allows us to train parameterized policies, whose behaviors can be controlled using low-dimensional gait parameters.

¹ Supplementary video: https://xbpeng.github.io/projects/Cassie_Walking/

² This work was led by Zhongyu Li.

Due to the size and instability of real bipedal robots, it is particularly dangerous to perform RL directly on the physical system. We instead leverage techniques from sim-to-real transfer to train policies in a simulated environment, which are then evaluated in another higher fidelity simulator, before finally being deployed on the real robot. Domain randomization is incorporated during training in simulation to develop robust policies that can successfully transfer to the real world. The resulting policies can then be deployed on the real Cassie robot to perform a repertoire of different locomotion behaviors in challenging scenarios, as shown in Fig. 9.1.

The primary contribution of this chapter is the development of a reinforcement learning based framework that produces diverse and robust walking controllers on the Cassie robot. We develop an end-to-end versatile walking policy that combines a HZD-based gait library with reinforcement learning to enable a bipedal robot to walk while following commands that specify frontal and lateral walking speeds, walking height, and turning yaw rate. The proposed learning-based walking policy notably expands the feasible command set and safe set over prior model-based controllers, and improves stability during gait transitions compared to a HZD-based baseline walking controller. The learned policies are robust to modelling error, perturbations, and environmental changes. This robustness emerges from our training strategy, which trains a policy to imitate a collection of diverse gaits, and also incorporates domain randomization. The learned policy can be directly transferred to other simulators, such as a more accurate simulator implemented with SimMechanics, as well as to a real robot. Our policies are able to reliably follow commands in indoor and outdoor environments. The policies demonstrate agile recoveries from random perturbations, while also being robust to malfunctioning motors, changes of ground friction, and carrying unknown payloads.

Related Work

Traditional approaches for locomotion of bipedal robots are typically based on notions of gait stability, such as the ZMP criterion [280] and capturability [131], simplified models [136, 215, 297], and constrained optimization methods [132, 64, 51]. These methods have been shown to be effective for controlling various humanoid robots with flat feet, but the resulting motion tends to be slow and conservative. Hybrid Zero Dynamics (HZD) [84, 102, 48, 192, 79, 153] is another control technique for generating stable periodic walking gaits based on input-output linearization. Our work, which is based on reinforcement learning, is not constrained by the requirement of a precise model and stabilization to a periodic orbit, as is the case for HZD, which enables our method to produce more diverse behaviors.

RL-based Control for Legged Robots: Reinforcement learning for legged locomotion has shown promising results in acquiring locomotion skills in simulation [46, 206, 207] and in the real world [129, 111, 208]. Data-driven methods provide a general framework that enables legged robots to perform a rich variety of behaviors by introducing reference motion terms into the learning process [224, 207, 208]. However, most previous RL-based work

are deployed on either multi-legged systems [152, 208, 141] or on low-dimensional bipedal robots [183, 311], where learned motions are typically quasi-static.

More recently, RL has been applied to learn agile walking skills for Cassie. Model-based RL in [36, 35] attains a velocity regulating adaptive walking controller on Cassie in simulation. In [295, 296], reference motions combined with model-free residual learning [117] are used to learn walking policies that are able to reliably track given planar velocity commands on Cassie in the real world. Residual control structure used in the policy can speed up training, but the resulting policy can only apply limited corrections to the underlying reference trajectory. Moreover, a model-based walking controller on Cassie is still needed to provide reference motions recorded from control outputs. This limits the accessibility to the reference motions and therefore reduces the diversity of learned behaviors. In addition, most of the previous learning-based walking policies on Cassie do not show significant improvement over traditional model-based controllers. Also, they lack the ability to change the walking height and turning yaw, which increases the complexity of controller design but enables the robot to travel in narrow environments. In our work, we show a clear improvement over model-based methods by examining the tracking performance and robustness over a range of gait parameters.

Simulation to Real World Transfer: Sim-to-real transfer is an attractive approach for developing policies, which takes advantage of fast simulations as a safe and inexpensive source of data. Model-based methods require careful system identification to bridge the reality gap [79, 153]. Randomizing the system properties in the source domain in order to cover the uncertainty in the target domain has allowed for solutions that use low-fidelity simulations for learning-based methods [232, 269, 200, 311, 310, 208, 245]. In this paper, we adopt domain randomization to overcome the sim-to-real gap, without the need for any additional training on the robot.

9.1 Parameterized Control of Cassie

In this section, we present the Cassie robot, which is the platform for our experiments and introduce a HZD-based gait library of versatile walking behaviors for the Cassie.

Cassie Robot Model

Cassie is a person-sized bipedal robot with 20 DoFs, as shown in Fig. 9.1 and explained in [153, Sec. II]. There are 10 actuated 1D revolute joints, which include abduction, rotation, hip pitch, knee, and toe motors. There are also four passive joints that correspond to the shin and tarsus joints. Its floating base pelvis has 3 transitional DoFs (sagittal, lateral, transverse) and 3 rotational DoFs (roll, pitch, yaw).

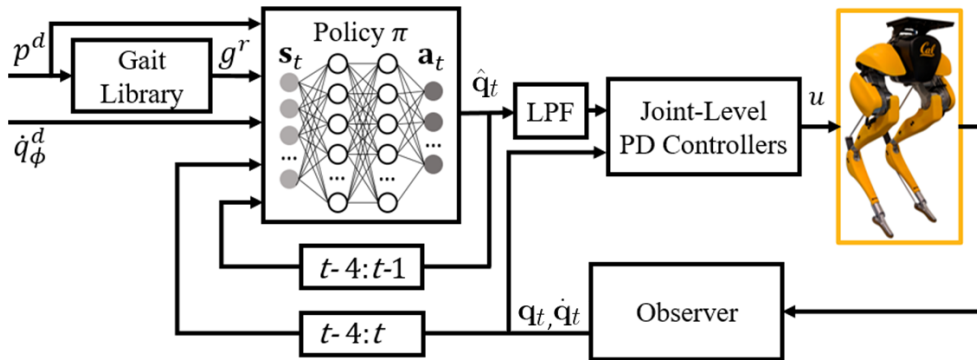


Figure 9.2: Proposed learning-based walking controller. The inputs of the policy consists of desired gait parameter p^d , desired turning yaw velocity \dot{q}_ϕ^d , a reference gait g^r decoded from the desired gait parameter p^d , observed robot states $\mathbf{a}_{t-4:t}$, $\hat{\mathbf{q}}_{t-4:t}$ from time step $t-4$ to t , and past policy outputs, which consist of the desired motor positions $\hat{\mathbf{q}}_{t-4:t-1}$ from time step $t-4$ to $t-1$. The current desired motor positions $\hat{\mathbf{q}}_t$ are sent to joint-level controllers after passing through a low-pass filter (LPF).

Gait Library and Parameterized Control

To create a controller that can be directed online to perform and transition between different motions, we parameterize the input to the system using a set of *gait parameter* p that specifies the desired *gait*. A *gait* g is a set of periodic joint trajectories that encode a locomotion behavior [96]. In this work, we use 5th order Bézier curves α to represent smooth profiles for the 10 actuated joints. The Bézier curves are normalized by 1 step period \bar{t} . The gaits designed in this paper consist of 2 steps, referred to as *right stance* and *left stance*, and transitions between the steps are triggered by a foot *impact* on the ground. The gait parameters chosen in this work are forward velocity \dot{q}_x , lateral velocity \dot{q}_y and walking height q_z , *i.e.*, $p = [\dot{q}_x \ \dot{q}_y \ q_z]^T \in \mathbb{R}^3$. A *gait library* $\mathcal{G} = \{g_i(\alpha, \bar{t})\}$ is constructed by indexing the i^{th} gait g_i with its gait parameter p_i . The optimization program for constructing the HZD-based gait library is formulated in CFROST [102] and the resulting gaits are described in Tab. 9.1 [153]. The gait library is later combined with an online regulator to implement a parameterized walking controller in [153, Sec. IV].

9.2 Learning Walking Control and Sim-to-Real

Having an optimized gait library is not enough to control bipedal robots without online feedback. We will next combine the pre-computed HZD-based gait library with reinforcement learning to develop a versatile locomotion policy π for the Cassie. Our policies are trained using a simulation environment built with the MuJoCo simulator [270, 3]. This section details the design of the simulation environment.

Parameter	Values
\dot{q}_x	$[-1, -0.8, \dots, 0.8, 1.0]$
\dot{q}_y	$[-0.3, -0.24, \dots, 0.24, 0.3]$
q_z	$[0.65, 0.685, \dots, 0.965, 1.0]$
number of gaits	$11 \times 11 \times 11 = 1331$

Table 9.1: The gait library contains reference motions for gaits that satisfy a range of commands specifying different target values for forward velocity \dot{q}_x , lateral velocity \dot{q}_y , and walking height q_z .

Action Space

The action $\mathbf{a}_t = \hat{\mathbf{q}}$ specifies target positions for the 10 motors on Cassie. In order to obtain a smoother motion, the target positions are first passed through a low-pass filter [208], as shown in Fig. 9.2, before being applied to the motors. A joint-level PD controller generates torque u for each motor on Cassie based on the filtered targets.

State Space

The state $\mathbf{s}_t = (\mathbf{q}_{t-4:t}, \dot{\mathbf{q}}_{t-4:t}, \mathbf{a}_{t-4:t-1})$ at time t consists of three components. The first component $\mathbf{q}_{t-4:t}$ consists of features that describe the pose of the robot's body in the past 5 time steps. The features include the local rotations of each joint, as well as the rotation of the pelvis. Note, that \mathbf{q} does not include the translation of the pelvis, since that can be difficult to measure in the real world without external instrumentation. Similarly, the second component $\dot{\mathbf{q}}_{t-4:t}$ consists of the joint velocities in the past 5 time steps. The final component $\mathbf{a}_{t-4:t-1}$ records the actions taken by the policy in the past 4 time steps. This history of past observations and actions provides the policy with some information to infer the system dynamics.

Goal

To train a policy to produce a desired reference motion, target frames from the reference motion are provided to the policy as input via a time-dependent goal \mathbf{g}_t . The user command c is used to operate the robot online and it is defined as $c = [p^d \ \dot{q}_\phi^d]$ which includes desired gait parameters p^d and desired turning yaw velocity \dot{q}_ϕ^d . Given a desired gait parameter p^d , a reference gait g^r is constructed by interpolating the parameterized gait library with respect to p^d as explained in Sec. 9.1. The goal is then specified by $\mathbf{g}_t = (c(t), \hat{\mathbf{q}}_t, \hat{\mathbf{q}}_t, \hat{\mathbf{q}}_{t+1}, \hat{\mathbf{q}}_{t+1}, \hat{\mathbf{q}}_{t+4}, \hat{\mathbf{q}}_{t+4}, \hat{\mathbf{q}}_{t+7}, \hat{\mathbf{q}}_{t+7})$, which includes the current user commands $c(t)$, and the target motor positions $\hat{\mathbf{q}}_t$ and velocities $\hat{\dot{\mathbf{q}}}_t$ for current and future time steps.

Reward Function

The reward function is designed to encourage the agent to satisfy the given command while reproducing the corresponding reference motion from the gait library. The reward at each time step t is given by:

$$r_t = 0.3r_t^m + 0.24r_t^p + 0.15r_t^{\dot{p}} + 0.13r_t^r + 0.06r_t^{\dot{r}} + 0.06r_t^u + 0.06r_t^f \quad (9.1)$$

The motor reward r_t^m encourages the policy to minimize the discrepancies between the motor positions \mathbf{q}_t and the target positions $\hat{\mathbf{q}}_t$ from the reference motion:

$$r_t^m = \exp[-\rho_m \|\hat{\mathbf{q}}_t - \mathbf{q}_t\|_2^2]. \quad (9.2)$$

where ρ_m is a scaling factor for the error term. The reward terms r_t^p , $r_t^{\dot{p}}$ and $r_t^{\dot{r}}$ follow the same formulation as equation 9.2 and encourage the agent to track reference pelvis translational position, translational velocity and rotational velocity in robot local frame, respectively. The pelvis rotation reward r_t^r encourages the robot to reduce the difference between the rotation of its pelvis q_t^r and the reference rotation \hat{q}_t^r , and it is formulated by $r_t^r = \exp[-\rho_r \|\hat{q}_t^r \ominus q_t^r\|_2^2]$ where \ominus denotes the geodesic distance between two rotation angles. The torque reward $r_t^u = \exp[-\rho_u \|u\|_2^2]$ encourages the robot to reduce energy consumption. Lastly, the ground reaction force reward $r_t^f = \exp[-\rho_f \|f\|_2^2]$ helps to minimize the vertical contact forces f . The weights of the reward terms are specified manually.

The desired roll and pitch velocity are always set to 0 to stabilize the pelvis, while the desired yaw velocity is specified by the user command $c(t)$. Note that the reference motion from the gait library does not specify the turning yaw. Therefore, including non-zero desired turning yaw in the reward can encourage the agent to develop turning behaviors, which are not provided in the gait library.

Domain Randomization

In order to improve the robustness of the policy and bridge the gap between the simulation and the real world, the dynamics of the environment is randomized during training in simulation. The randomization regiment is designed to address three major sources of uncertainty in the environment: 1) modelling error of the robot and the environment, 2) sensor noise, and 3) communication delay between the policy and the joint-level controller. These dynamics properties are parameterized as $\boldsymbol{\mu}$, with values varying between the ranges specified in Tab. 9.2.

Network Architecture

The policy is modeled by a neural network with 2 hidden layers of 512 *tanh* units. The value function is modeled by a separate network with a similar architecture. For the policy network $\pi(\mathbf{a}_t | \mathbf{s}_t, \mathbf{g}_t)$, the inputs include the state \mathbf{s}_t and goal \mathbf{g}_t . The policy network uses

Parameter	Range	Unit
Link Mass	$[0.75, 1.15] \times \text{default}$	kg
Link Mass Center	$[0.75, 1.15] \times \text{default}$	m
Joint Damping	$[0.75, 1.15] \times \text{default}$	Nms/rad
Ground Friction Ratio	$[0.5, 3.0]$	1
Motor Rotation Noise	$[-0.1, 0.1]$	rad
Motor Angle Velocity Noise	$[-0.1, 0.1]$	rad/s
Accelerometer Noise	$[-0.4, 0.4]$	m/s^2
Gyro Rotation Noise	$[-0.1, 0.1]$	rad
Gyro Angle Velocity Noise	$[-0.1, 0.1]$	rad/s
Communication Delay	$[0, 0.03]$	sec

Table 9.2: Dynamics Properties and Sample Range.

\tanh as the activation function for the output layer. The output of the policy network is a 10 dimensional vector representing a Gaussian action distribution $\mathcal{N}(\mathbf{m}_\pi(\mathbf{a}|\mathbf{s}, \mathbf{g}), \Sigma_\pi)$ with a learned mean $\mathbf{m}_\pi(\mathbf{a}|\mathbf{s})$ and fixed standard deviation $\Sigma_\pi = 0.1I$. The action \mathbf{a} (*i.e.*, desired motor positions) is sampled from this output distribution. The value network outputs a scalar value $V(\mathbf{s}, \mathbf{g}, \boldsymbol{\mu})$ representing the expected return of the policy given state \mathbf{s} , goal \mathbf{g} , and the dynamics parameters $\boldsymbol{\mu}$ used in simulation. As in Chapter 8, we use an asymmetry architecture where the value function is provided with access to the ground-truth dynamics parameters, which is only accessible in simulation.

Training Setup

The policy operates at 30 Hz, while the joint-level PD controller illustrated in Fig. 9.2 runs at 2000 Hz. The maximum number of time steps for each episode is designated to be $T=2500$, corresponding to approximately 83 s. In each episode, a new command $c(t) = [\dot{q}_x^d \ \dot{q}_y^d \ \dot{q}_z^d \ \dot{q}_\phi^d]$ is uniformly sampled every 8 s, and remains unchanged during the 8 s window. The command range is from $[-2, -0.8, 0.65, -\pi/6]$ to $[2, 0.8, 1, \pi/6]$. The first command in each episode is always set to a random walking forward velocity with 0 yaw velocity at a normal height above 0.9 m. Note that the range of training commands is larger compared to the gait parameter provided in Tab. 9.1. In this way, the agent is able to learn to follow a command that is out-of-range of gait parameters and thus learns behaviors beyond what the gait library can provide. An episode ends when the maximum number of time steps is reached, or early termination conditions have been triggered. Early termination is triggered if the height of the pelvis drops below 0.55 m, and if the tarsus joints hit the ground.

Dynamics randomization presented in Sec. 9.2 is introduced gradually over the course of training through a curriculum. The curriculum helps to prevent the policy from adopting excessively conservative sub-optimal behaviors. For example, if training starts with the full range of randomizations detailed in Table 9.2, the policy is prone to adopting strategies that

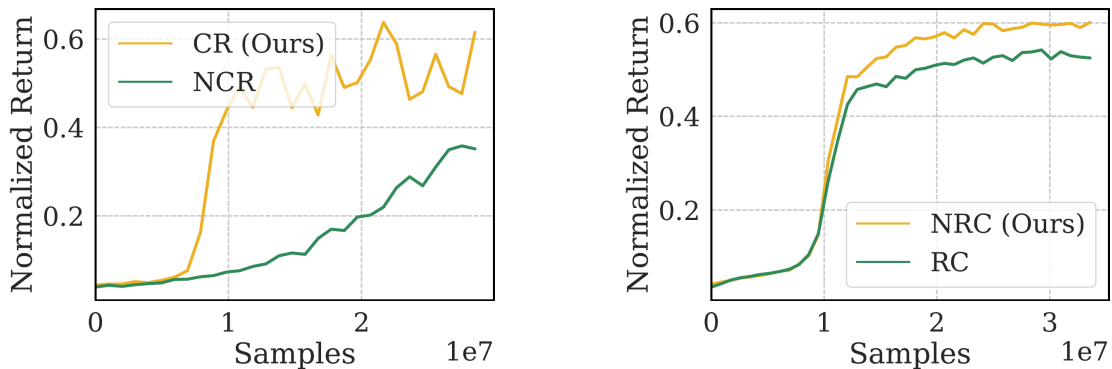


Figure 9.3: Comparison between (a) proposed Curriculum (CR) and Non-Curriculum (NCR) methods and (b) proposed Non-Residual Control (NRC) and Residual Control (RC) used in previous work [296]. Our proposed method shows best overall training performance in terms of learning speed and converged rewards. The corresponding total samples for the curriculum is 6.6×10^7 , while the NCR model has full range of dynamics randomization from the start of training.

prevent the robot from falling by simply standing in-place. In a highly dynamic environment, standing can be more stable than walking, and is therefore easier to learn, but is nonetheless sub-optimal. Therefore, over the course of the first 2000 training iterations, the upper and lower boundaries of the randomized dynamics parameters are linearly annealed from fixed default values to the maximum ranges specified in Tab. 9.2.

9.3 Experiments

The walking policy is trained with a MuJoCo simulation of the Cassie robot [270]. The performance of the learned policy is evaluated in three domains: MuJoCo, MATLAB SimMechanics, and on Cassie. Performance is first evaluated in the MuJoCo simulator, which is the domain used for training. Later, SimMechanics provides a safe and high-fidelity simulated environment that closely replicates the physical system to extensively test the learned policy. However, the high-fidelity simulation is slower than real-time by an order of magnitude, so it is primarily used for testing. Finally, the policy is deployed and validated on the real Cassie robot.

Learning Performance

To evaluate the effects of the randomization curriculum, we compare the performance of policies trained with and without the curriculum. Fig. 9.3(a) compares learning curves for the different policies. The policy trained with the randomization curriculum (CR) starts training with a small amount of randomization, which is then gradually increased over the

course of training. The policy trained without the curriculum (NCR) starts training with the full range of randomization. The large amount of randomization at the start of training leads the policy to adopt an excessively conservative and sub-optimal behavior. The policy trained with the curriculum exhibits substantially faster learning progress, while also achieving a higher return.

The effects of residual control are evaluated by comparing the performance of our policy that uses non-residual control (NRC), with a policy that uses residual control (RC) [295, 296]. Learning curves comparing the different policies are available in Fig. 9.3(b). In the absence of external perturbations, the performance of the two policies is similar, with the non-residual policy performing marginally better than the residual policy. However, as we show in the following experiments, our non-residual policy with dynamics randomization is more robust than the residual policy, which may be due to the non-residual policy’s greater flexibility to deviate from the behaviors prescribed by the reference motion in order to recover from perturbations.

Robustness Analysis in High-Fidelity Simulation

A *Feasible command set* is a set of input gait parameters p^d that will not cause a controller to fail. A *safe set* is defined as a set of gait parameters that a controller actually achieves on the robot while maintaining a stable walking gait. During each iteration, a gait parameter p^d is provided to the controller as a command in MATLAB SimMechanics, if the controller succeeds in maintaining a stable gait for 15 seconds, then p^d will be added to the feasible command set and the actual achieved gait parameter \hat{p} will be added to the safe set. Through extensive tests of a controller, the resulting feasible command set and safe set provide informative metrics to evaluate the control performance and robustness of the controller when deployed on the robot. Typically, a walking controller with a larger feasible command set can handle more scenarios, and a controller with a larger safe set can achieve more dynamic motions. Moreover, a controller with better tracking performance can result in a similar shape between the feasible command set and safe set, as the difference between these two sets indicates tracking errors between p^d and \hat{p} .

We compare the feasible command sets and safe sets between the learned policy and prior HZD-based variable walking height controller developed in [153] and based on [79]. The procedures for generating the command sets and safe sets of these two controllers are identical, and the testing range for p^d is set to be between $[-1.1, -0.6, 0.65]^T$ and $[2, 0.6, 1.0]^T$, with a resolution of $[0.1, 0.1, 0.05]^T$. The resulting feasible command sets and safe sets are shown in Fig. 9.4. As shown in Fig. 9.4(a), the proposed RL-based controller is able to cover almost the entire testing range, while the HZD-based controller can only handle a smaller bowl-shape region. Quantitatively, the feasible command set of the RL-based walking controller is more than 4 times larger than HZD-based controller. Moreover, as illustrated in Fig. 9.4(b), the RL-based walking controller covers a broader safe set than the HZD-based controller. In practice, this means that the RL-based controller can achieve faster forward and backward walking (from -1.2 m/s to 1.2 m/s) than HZD-based one (below 1 m/s).

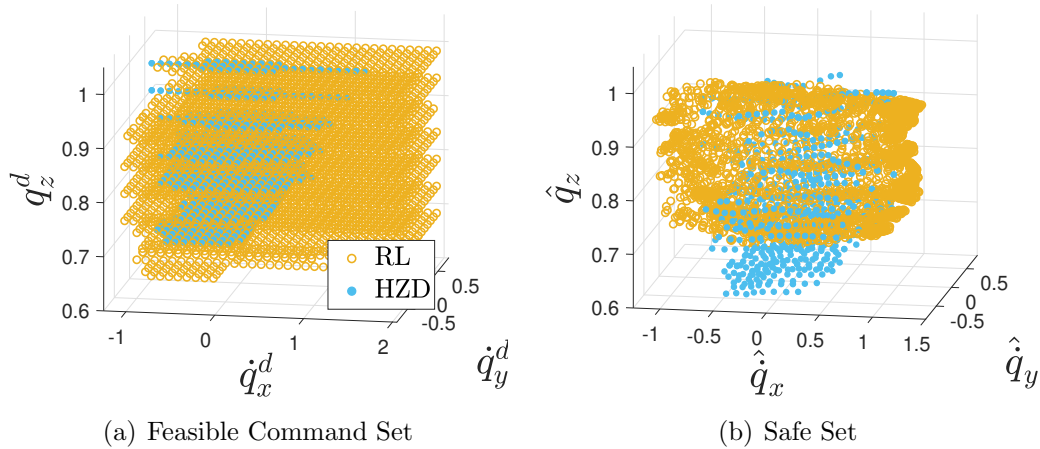


Figure 9.4: Comparison of proposed commands (Feasible Command Set) and achieved commands (Safe Set) between HZD-based controller [153] and proposed RL-based controller. Our RL-based controller can handle more tracking commands than the HZD-based baseline and thus results in a larger feasible command set. The safe set of RL-based policy is also larger in the sagittal walking velocity \hat{q}_x and walking height \hat{q}_z direction. The tracking performance of the RL-based policy also shows advantages as the shapes of feasible command set and safe set are closer.

Although Fig. 9.4(b) shows that the HZD-based controller can achieve walking gaits with lower heights (0.6 m) than the RL-based one (0.65 m), the tracking error of the HZD-based controller is not negligible as its minimum feasible walking height command can only reach 0.7 m while RL-based one can go to 0.65 m. Therefore, the RL-based controller exhibits better performance on tracking commands. Moreover, by inspecting the relationship between Fig. 9.4(a) and Fig. 9.4(b), we find that the RL-based controller can handle the commands that are outside of the given gait library in Tab. 9.1, *e.g.*, 2 m/s in the sagittal direction. The resulting actual velocity \hat{q}_x is around 1.2 m/s, which is also outside of the range seen in the gait library. With the standard HZD-based controller, the robot only approaches 1 m/s when it is being given a 2 m/s command, due to a large tracking error. This shows that the RL-based controller is not strictly tracking the commands, and instead finds a more optimal gait that is close to the commands while maintaining stability.

Robustness in the Real World

The deployed policy on Cassie in the real world can reliably control the robot to perform various behaviors, such as changing walking heights in Fig. 9.1(a),9.1(b), fast walking in Fig. 9.5(a), walking sideways in Fig. 9.5(b), turning around in Fig. 9.5(c). Moreover, the policy also shows robustness to the changes of the robot itself and the environment.

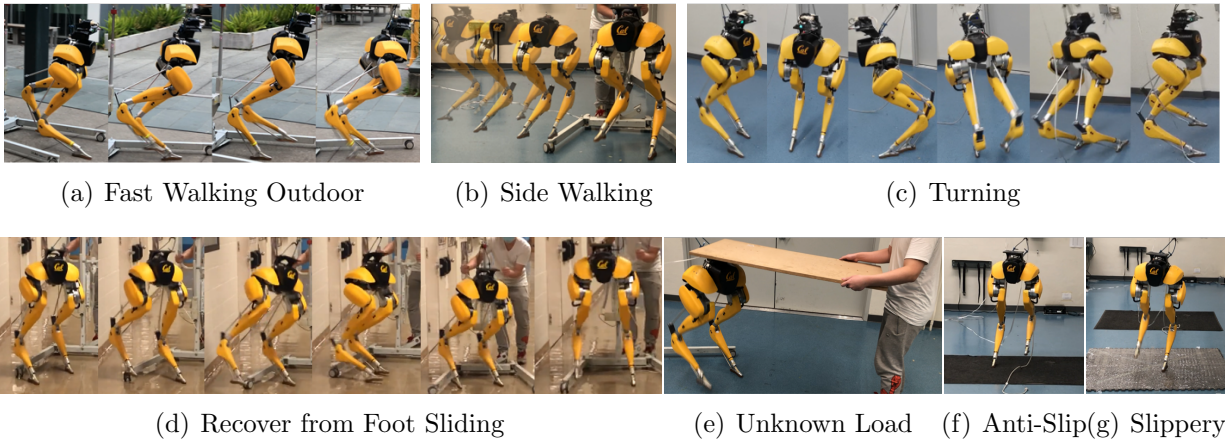


Figure 9.5: Experiment Results. The proposed learned walking policy extensively on Cassie in real world in different scenarios. In the experiments, the policy enables the Cassie to perform various agile behaviors such as fast forward and backward walking, sideways walking, changing walking height, and turning around. Moreover, empowered by the proposed policy, the robot is able to recover from random load perturbation and also able to adapt to change different ground frictions and unknown load.

Modeling Error

During the experiments in this paper, a malfunction caused two motors on the Cassie to behave abnormally. Specifically, the right rotation and right knee motors were partially damaged, rendering them unable to produce as much torque as the corresponding motors on the left side. Following this malfunction, model-based walking controllers, such as the factory default controller and the HZD-based variable walking controller [153], were no longer able to reliably produce a walking gait. The baseline HZD-based controller was no longer able to recover to a normal height after a reduction in walking height, since the right leg was weaker than the left leg. However, by training with dynamics randomization (Sec. 9.2), especially the damping ratio of each joint, the proposed learned walking controller could control the robot even with partially damaged motors. Indeed, this policy was able to successfully control the robot the very first time it was deployed, without additional tuning.

Perturbation

To show that our approach is more robust, three quantitative experiments are done in the MuJoCo simulator: 1) a non-residual policy trained with the gait library, 2) a non-residual policy trained using a single reference motion from the gait library, and 3) a residual policy trained with the same single gait as the previous work [296]. All policies are trained without domain randomization. During the evaluation, the pelvis is perturbed randomly by a 6 DoF force with a probability of $0.15\% \beta$ at each time step lasting for a random time span sampling

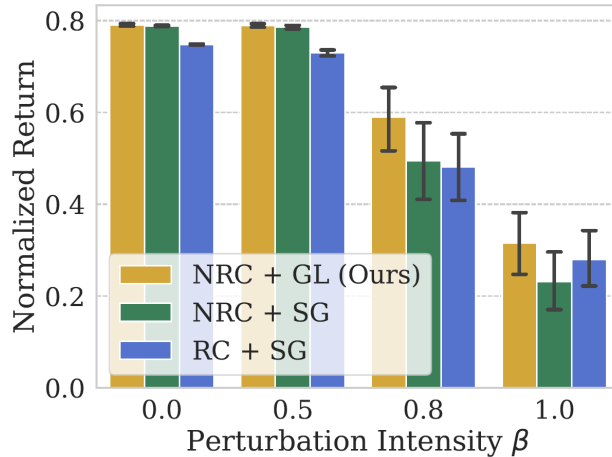


Figure 9.6: Comparison of robustness to perturbation among 3 different methods in MuJoCo simulation. Non-Residual Controlled Gait Library (NRC+GL) trained with the gait library; Non-Residual Controlled Single Gait (NRC+SG) and Residual Controlled Single Gait (RC+SG) are trained with only one single gait. A 6 DoF force is randomly applied on the pelvis with probability $0.15\beta\%$. The *Normalized Return* is computed with the mean reward of 32 roll-outs for each model and β .

from $[0, 0.8\beta]$ s, where $\beta \in [0, 1]$ stands for perturbation intensity. The achieved return of each model is illustrated in Fig. 9.6. When larger perturbations like $\beta \in \{0.8, 1\}$ are applied, the model trained by the proposed method shows significant advantages over other models.

To further demonstrate the robustness in the real world qualitatively, we randomly push Cassie with a rod in different directions Fig. 9.1(c). The feet of Cassie are also perturbed during walking, including stepping on the gantry in Fig. 9.5(d). In addition an unknown load is applied in Fig. 9.5(e) and changes in ground friction in Fig. 9.5(f),9.5(g). The proposed learned policy shows improved robustness over previous work across all scenarios.

9.4 Discussion

In this chapter, we presented a reinforcement learning system that combines motion imitation and domain randomization to create robust parameterized bipedal locomotion policies that can walk, turn and squat. The proposed learning method shows benefits over a baseline model-based walking controller, produces a larger feasible command set, a larger safe set, and better tracking performance. In real world experiments, the policies also demonstrate considerable robustness, allowing the Cassie to walk over floors with different friction, recover from perturbations, and even walk with malfunctioning motors. An exciting future direction is to explore more dynamic and agile behaviors with the Cassie by building on the approach presented in this work.

Chapter 10

Domain Adaptation

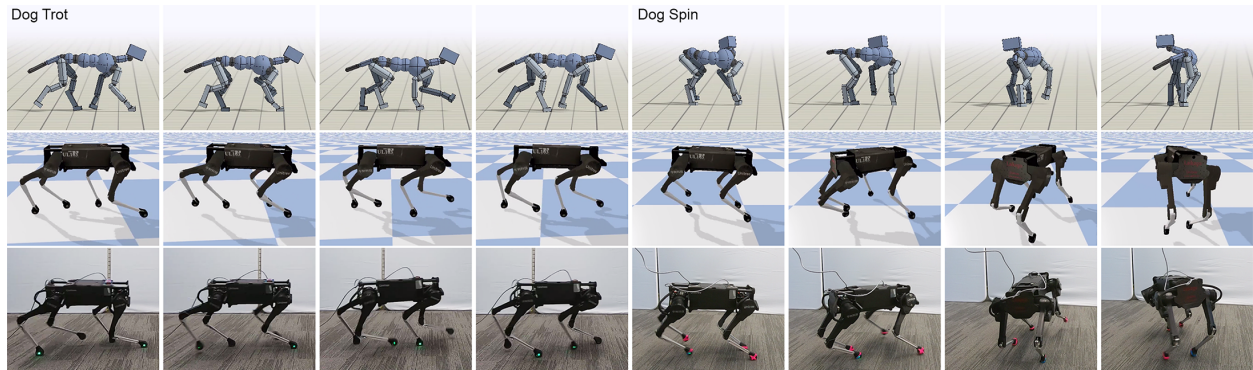


Figure 10.1: Laikago robot performing locomotion skills learned by imitating motion data recorded from a real dog. **Top:** Motion capture data recorded from a dog. **Middle:** Simulated Laikago robot imitating reference motions. **Bottom:** Real Laikago robot imitating reference motions. (Video¹)

Animals can traverse complex environments with remarkable agility, bringing to bear broad repertoires of agile and acrobatic skills. Reproducing such agile behaviors has been a long-standing challenge in robotics, with a large body of work devoted to designing control strategies for various locomotion skills [179, 217, 240, 76, 23]. However, designing control strategies often involves a lengthy development process, and requires substantial expertise of both the underlying system and the desired skills. Despite the many success in this domain, the capabilities achieved by these systems are still far from the fluid and graceful motions seen in the animal kingdom.

In this chapter, we propose an imitation learning framework that enables legged robots to learn agile locomotion skills from real-world animals. To address the high sample requirements of current RL algorithms, the initial training phase is performed in simulation.

¹ Supplementary video: https://xbpeng.github.io/projects/Robotic_Imitation/

However, the directly sim-to-real transfer approach used in Chapter 9 may not always be successful when transferring more agile behaviors, which can be more sensitive to variations in the dynamics of a system. In order to transfer these agile skills from simulation to the real world, we propose a sample efficient adaptation technique, which fine-tunes the behavior of a policy using a learned dynamics representation. The central contribution of this chapter is a system that enables legged robots to learn agile locomotion skills by imitating animals. We demonstrate the effectiveness of our framework on a variety of dynamic locomotion skills with the Laikago quadruped robot [286], including different locomotion gaits, as well as dynamic hops and turns. In our ablation studies, we explore the impact of different design decisions made for the various components of our system.

10.1 Related Work

The development of controllers for legged locomotion has been an enduring subject of interest in robotics, with a large body of work proposing a variety of control strategies for legged systems [179, 217, 240, 81, 76, 302, 44, 23]. However, many of these methods require in-depth knowledge and manual engineering for each behavior, and as such, the resulting capabilities are ultimately limited by the designer’s understanding of how to model and represent agile and dynamic behaviors. Trajectory optimization and model predictive control can mitigate some of the manual effort involved in the design process, but due to the high-dimensional and complex dynamics of legged systems, reduced-order models are often needed to formulate tractable optimization problems [138, 74, 54, 12]. These simplified abstractions tend to be task-specific, and again require significant insight of the properties of each skill.

Motion imitation: Imitating reference motions provides a general approach for robots to perform a rich variety of behaviors that would otherwise be difficult to manually encode into controllers [213, 83, 252, 299]. But applications of motion imitation to legged robots have predominantly been limited to behaviors that emphasize upper-body motions, with fairly static lower-body movements, where balance control can be delegated to separate control strategies [188, 123, 128]. In contrast to physical robots, substantially more dynamic skills can be reproduced by agents in simulation [184, 145, 46, 160]. Recently, motion imitation with reinforcement learning has been effective for learning a large repertoire of highly acrobatic skills in simulation [207, 158, 210, 143]. But due to the high sample complexity of RL algorithms and other physical limitations, many of the capabilities demonstrated in simulation have yet to be replicated in the real world.

Sim-to-real transfer: The challenges of applying RL in the real world have driven the use of domain transfer approaches, where policies are first trained in simulation (source domain), and then transferred to the real world (target domain). Sim-to-real transfer can be facilitated by constructing more accurate simulations [259, 296], or adapting the simulator with real-world data [256, 92, 111, 165, 37]. However, building high-fidelity simulators remains a

challenging endeavour, and even state-of-the-art simulators provide only a coarse approximation of the rich dynamics of the real world. Domain randomization can be incorporated into the training process to encourage policies to be robust to variations in the dynamics [232, 269, 212, 200, 195]. Sample efficient adaptation techniques, such as finetuning [231] and meta-learning [58, 65, 41] can also be applied to further improve the performance of pre-trained policies in new domains. In this work, we leverage a class of adaptation techniques, which we broadly referred to as *latent space* methods [98, 311, 310], to transfer locomotion policies from simulation to the real world. During pre-training, these methods learn a latent representation of different behaviors that are effective under various scenarios. When transferring to a new domain, a search can be conducted in the latent space to find behaviors that successfully execute a desired task in the new domain. We show that by combining motion imitation and latent space adaptation, our system is able to learn a diverse corpus of dynamic locomotion skills that can be transferred to legged robots in the real world.

RL for legged locomotion: Reinforcement learning has been effective for automatically acquiring locomotion skills in simulation [207, 158, 143] and in the real world [129, 264, 59, 259, 90, 111]. Kohl and Stone [129] applied a policy gradient method to tune manually-crafted walking controllers for the Sony Aibo robot. By carefully modeling the motor dynamics of the Minitaur quadruped robot, Tan et al. [259] was able to train walking policies in simulation that can be directly deployed on a real robot. Hwangbo et al. [111] proposed learning a motor dynamics model using real-world data, which enabled direct transfer of a variety of locomotion skills to the ANYmal robot. Their system trained policies using manually-designed reward functions for each skill, which can be difficult to specify for more complex behaviors. Imitating reference motions can be a general approach for learning diverse repertoires of skills without the need to design skill-specific reward functions [160, 207, 210]. Xie et al. [296] trained bipedal walking policies for the Cassie robot by imitating reference motions recorded from existing controllers and keyframe animations. The policies are again transferred from simulation to the real world with the aid of careful system identification. Yu et al. [311] transferred bipedal locomotion policies from simulation to a physical Darwin OP2 robot using a latent space adaptation method, which mitigates the dependency on accurate simulators. In this work, we leverage a similar latent space method, but by combining it with motion imitation, our system enables real robots to perform more diverse and agile behaviors than have been demonstrated by these previous methods.

10.2 Overview

The objective of our framework is to enable robots to learn skills from real animals. Our framework receives as input a reference motion that demonstrates a desired skill for the robot, which may be recorded using motion capture (mocap) of real animals (e.g. a dog). Given a reference motion, it then uses reinforcement learning to synthesize a policy that enables a robot to reproduce that skill in the real world. A schematic illustration of our framework

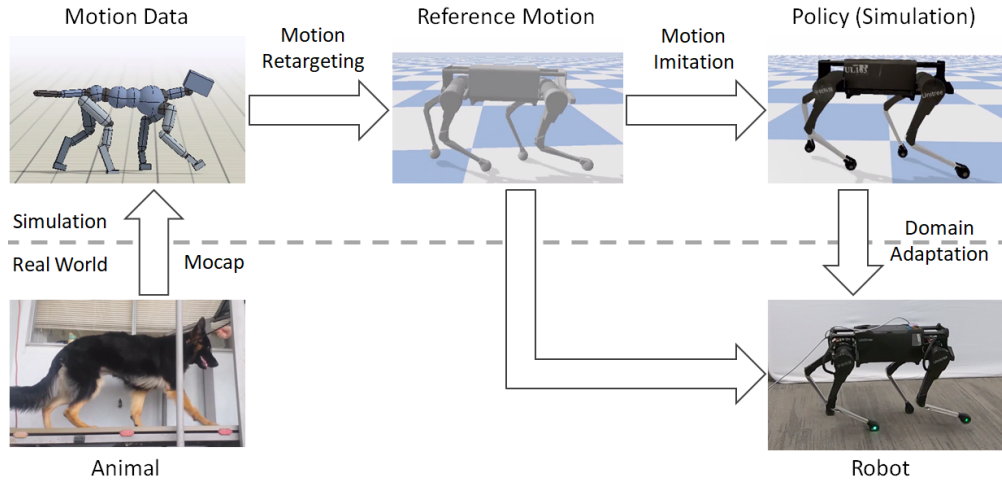


Figure 10.2: The framework consists of three stages: motion retargeting, motion imitation, and domain adaptation. It receives as input motion data recorded from an animal, and outputs a control policy that enables a real robot to reproduce the motion.

is available in Figure 10.2. The process is organized into three stages: motion retargeting, motion imitation, and domain adaptation. 1) The reference motion is first processed by the motion retargeting stage, where the motion clip is mapped from the original subject’s morphology to the robot’s morphology via inverse-kinematics. 2) Next, the retargeted reference motion is used in the motion imitation stage to train a policy to reproduce the motion with a simulated model of the robot. To facilitate transfer to the real world, domain randomization is applied in simulation to train policies that can adapt to different dynamics. 3) Finally, the policy is transferred to a real robot via a sample efficient domain adaptation process, which adapts the policy’s behavior using a learned latent dynamics representation.

10.3 Motion Retargeting

When using motion data recorded from animals, the subject’s morphology tends to differ from that of the robot’s. To address this discrepancy, the source motions are retargeted to the robot’s morphology using inverse-kinematics [78]. First, a set of source keypoints are specified on the subject’s body, which are paired with corresponding target keypoints on the robot’s body. An illustration of the keypoints is available in Figure 10.3. The keypoints include the positions of the feet and hips. At each timestep, the source motion specifies the 3D location $\hat{\mathbf{x}}_i(t)$ of each keypoint i . The corresponding target keypoint $\mathbf{x}_i(\mathbf{q}_t)$ is determined by the robot’s pose \mathbf{q}_t , represented in generalized coordinates [63]. IK is then applied to construct a sequence of poses $\mathbf{q}_{0:T}$ that track the keypoints at each frame,

$$\arg \min_{\mathbf{q}_{0:T}} \sum_t \sum_i \|\hat{\mathbf{x}}_i(t) - \mathbf{x}_i(\mathbf{q}_t)\|^2 + (\bar{\mathbf{q}} - \mathbf{q}_t)^T \mathbf{W}(\bar{\mathbf{q}} - \mathbf{q}_t). \quad (10.1)$$

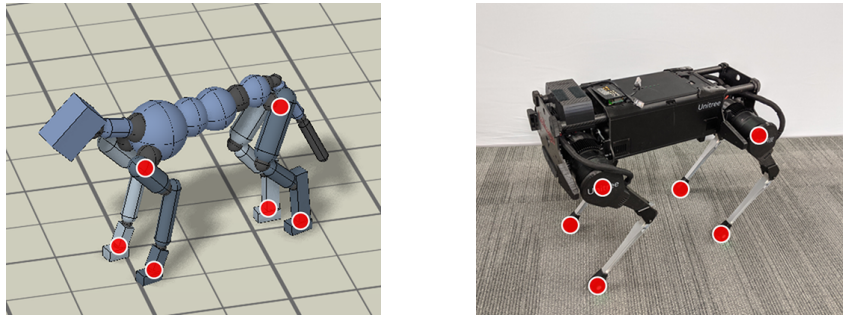


Figure 10.3: Inverse-kinematics (IK) is used to retarget mocap clips recorded from a real dog (**left**) to the Laikago robot (**right**). Corresponding pairs of keypoints (red) are specified on the dog and robot’s bodies, and then IK is used to compute a pose for the robot that tracks the keypoints.

An additional regularization term is included to encourage the poses to remain similar to a default pose $\bar{\mathbf{q}}$, and $\mathbf{W} = \text{diag}(w_1, w_2, \dots)$ is a diagonal matrix specifying regularization coefficients for each joint.

10.4 Motion Imitation

To imitate a given reference motion, we follow a similar motion imitation approach as Chapter 3. The inputs to the policy is augmented with an additional goal \mathbf{g}_t , which specifies the motion that the robot should imitate. The policy is modeled as a feedforward network that maps a given state \mathbf{s}_t and goal \mathbf{g}_t to a distribution over actions $\pi(\mathbf{a}_t | \mathbf{s}_t, \mathbf{g}_t)$. The policy is queried at 30Hz for a new action at each timestep. The state $\mathbf{s}_t = (\mathbf{q}_{t-2:t}, \mathbf{a}_{t-3:t-1})$ is represented by the poses $\mathbf{q}_{t-2:t}$ of the robot in the three previous timesteps, and the three previous actions $\mathbf{a}_{t-3:t-1}$. The pose features \mathbf{q}_t consist of IMU readings of the root orientation (roll, pitch, yaw) and the local rotations of every joint. The root position is not included among the pose features to avoid the need to estimate the root position during real-world deployment. The goal $\mathbf{g}_t = (\hat{\mathbf{q}}_{t+1}, \hat{\mathbf{q}}_{t+2}, \hat{\mathbf{q}}_{t+10}, \hat{\mathbf{q}}_{t+30})$ specifies target poses from the reference motion at four future timesteps, spanning approximately 1 second. The action \mathbf{a}_t specifies target rotations for PD controllers at each joint. To ensure smoother motions, the PD targets are first processed by a low-pass filter before being applied on the robot [32].

Reward Function: The reward function encourages the policy to track the sequence of target poses $(\hat{\mathbf{q}}_0, \hat{\mathbf{q}}_1, \dots, \hat{\mathbf{q}}_T)$ from the reference motion at every timestep. The reward r_t at each timestep is given by:

$$r_t = w^p r_t^p + w^v r_t^v + w^e r_t^e + w^{\text{rp}} r_t^{\text{rp}} + w^{\text{rv}} r_t^{\text{rv}} \quad (10.2)$$

$$w^p = 0.5, w^v = 0.05, w^e = 0.2, w^{\text{rp}} = 0.15, w^{\text{rv}} = 0.1$$

The pose reward r_t^p encourages the robot to minimize the difference between the joint rotations specified by the reference motion and those of the robot. In the equation below, $\hat{\mathbf{q}}_t^j$ represents the 1D local rotation of joint j from the reference motion at time t , and \mathbf{q}_t^j represents the robot's joint,

$$r_t^p = \exp \left[-5 \sum_j \|\hat{\mathbf{q}}_t^j - \mathbf{q}_t^j\|^2 \right]. \quad (10.3)$$

Similarly, the velocity reward r_t^v is calculated according to the joint velocities, with $\hat{\dot{\mathbf{q}}}_t^j$ and $\dot{\mathbf{q}}_t^j$ being the angular velocity of joint j from the reference motion and robot respectively,

$$r_t^v = \exp \left[-0.1 \sum_j \|\hat{\dot{\mathbf{q}}}_t^j - \dot{\mathbf{q}}_t^j\|^2 \right]. \quad (10.4)$$

Next, the end-effector reward r_t^e , encourages the robot to track the positions of the end-effectors, where \mathbf{x}_t^e denotes the relative 3D position of end-effector e with respect to the root,

$$r_t^e = \exp \left[-40 \sum_e \|\hat{\mathbf{x}}_t^e - \mathbf{x}_t^e\|^2 \right]. \quad (10.5)$$

Finally, the root pose reward r_t^{rp} and root velocity reward r_t^{rv} encourage the robot to track the reference root motion. $\mathbf{x}_t^{\text{root}}$ and $\dot{\mathbf{x}}_t^{\text{root}}$ denotes the root's global position and linear velocity, while $\mathbf{q}_t^{\text{root}}$ and $\dot{\mathbf{q}}_t^{\text{root}}$ are the rotation and angular velocity,

$$r_t^{\text{rp}} = \exp \left[-20 \|\hat{\mathbf{x}}_t^{\text{root}} - \mathbf{x}_t^{\text{root}}\|^2 - 10 \|\hat{\mathbf{q}}_t^{\text{root}} - \mathbf{q}_t^{\text{root}}\|^2 \right] \quad (10.6)$$

$$r_t^{\text{rv}} = \exp \left[-2 \|\hat{\dot{\mathbf{x}}}_t^{\text{root}} - \dot{\mathbf{x}}_t^{\text{root}}\|^2 - 0.2 \|\hat{\dot{\mathbf{q}}}_t^{\text{root}} - \dot{\mathbf{q}}_t^{\text{root}}\|^2 \right]. \quad (10.7)$$

10.5 Domain Adaptation

Due to discrepancies between the dynamics of the simulation and the real world, policies trained in simulation tend to perform poorly when deployed on a physical system. Therefore, we propose a sample efficient adaptation technique for transferring policies from simulation to the real world.

Domain Randomization

Domain randomization is a simple strategy for improving a policy's robustness to dynamics variations [232, 269, 200]. Instead of training a policy in a single environment with fixed dynamics, domain randomization varies the dynamics during training, thereby encouraging the policy to learn strategies that are functional across different dynamics. However, there may be no single strategy that is effective across all environments, and due to unmodeled effects in the real world, strategies that are robust to different simulated dynamics may nonetheless fail when deployed in a physical system.

Domain Adaptation

In this work, we aim to learn strategies that are robust to variations in the dynamics of the environment, while also being able to adapt its behaviors as necessary for new environments. Let μ represent the values of the dynamics parameters that are randomized during training in simulation (Table 10.1). At the start of each episode, a random set of parameters are sampled according to $\mu \sim p(\mu)$. The dynamics parameters are then encoded into a latent embedding $\mathbf{z} \sim E(\mathbf{z}|\mu)$ by a stochastic encoder E , and \mathbf{z} is provided as an additional input to the policy $\pi(\mathbf{a}|\mathbf{s}, \mathbf{z})$. For brevity, we have excluded the goal input \mathbf{g} for the policy. When transferring a policy to the real world, we follow a similar approach as Yu, Liu, and Turk [306], where a search is performed to find a latent encoding \mathbf{z}^* that enables the policy to successfully execute the desired behaviors on the physical system. Next, we propose an extension that addresses potential issues due to over-fitting with the previously proposed method.

A potential degeneracies of the previously described approach is that the policy may learn strategies that depend on \mathbf{z} being an accurate representation of the true dynamics of the system. This can result in brittle behaviors where the strategies utilized by the policy for a given \mathbf{z} can overfit to the precise dynamics from the corresponding parameters μ . Furthermore, due to unmodeled effects in the real world, there might be no μ that accurately models real-world dynamics. Therefore, to encourage the policy to be robust to uncertainty in the dynamics, we incorporate an information bottleneck into the encoder. The information bottleneck enforces an upper bound I_c on the mutual information $I(\cdot, \mathbf{Z})$ between the dynamics parameters and the encoding \mathbf{Z} . This results in the following constrained policy optimization objective,

$$\arg \max_{\pi, E} \quad \mathbb{E}_{\mu \sim p(\mu)} \mathbb{E}_{\mathbf{z} \sim E(\mathbf{z}|\mu)} \mathbb{E}_{\tau \sim p(\tau|\pi, \mu, \mathbf{z})} \left[\sum_{t=0}^{T-1} \gamma^t r_t \right] \quad (10.8)$$

$$\text{s.t.} \quad I(\cdot, \mathbf{Z}) \leq I_c. \quad (10.9)$$

where the trajectory distribution is now given by,

$$p(\tau|\pi, \mu, \mathbf{z}) = p(\mathbf{s}_0) \prod_{t=0}^{T-1} p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t, \mu) \pi(\mathbf{a}_t|\mathbf{s}_t, \mathbf{z}). \quad (10.10)$$

Since computing the mutual information is intractable, the constraint in Equation 10.9 can be approximated with a variational upper bound using the KL divergence between E and a variational prior $\rho(\mathbf{z})$ [8],

$$I(\cdot, \mathbf{Z}) \leq \mathbb{E}_{\mu \sim p(\mu)} [\text{D}_{\text{KL}} [E(\cdot|\mu) || \rho(\cdot)]] . \quad (10.11)$$

We can further simplify the objective by converting Equation 10.9 into a soft constraint, to

Algorithm 5 Adaptation with Advantage-Weighted Regression

```

1:  $\pi \leftarrow$  trained policy
2:  $\omega_0 \leftarrow \mathcal{N}(0, I)$ 
3:  $\mathcal{D} \leftarrow \emptyset$ 
4: for iteration  $k = 0, \dots, k_{\max} - 1$  do
5:    $\mathbf{z}_k \leftarrow$  sampled encoding from  $\omega_k(\mathbf{z})$ 
6:   Rollout an episode with  $\pi$  conditioned  $\mathbf{z}_k$  and record the return  $\mathcal{R}_k$ 
7:   Store  $(\mathbf{z}_k, \mathcal{R}_k)$  in  $\mathcal{D}$ 
8:    $\bar{v} \leftarrow \frac{1}{k} \sum_{i=1}^k \mathcal{R}_i$ 
9:    $\omega_{k+1} \leftarrow \arg \max_{\omega} \sum_{i=1}^k [\log \omega(\mathbf{z}_i) \exp(\frac{1}{\alpha} (\mathcal{R}_i - \bar{v}))]$ 
10: end for

```

yield the following information-regularized objective,

$$\arg \max_{\pi, E} \mathbb{E}_{\mu \sim p(\mu)} \mathbb{E}_{\mathbf{z} \sim E(\mathbf{z}|\mu)} \mathbb{E}_{\tau \sim p(\tau|\pi, \mu, \mathbf{z})} \left[\sum_{t=0}^{T-1} \gamma^t r_t \right] - \beta \mathbb{E}_{\mu \sim p(\mu)} [\text{D}_{\text{KL}} [E(\cdot|\mu) || \rho(\cdot)]], \quad (10.12)$$

with $\beta \geq 0$ being a Lagrange multiplier. In our experiments, we model the encoder $E(\mathbf{z}|\mu) = \mathcal{N}(\mathbf{m}(\mu), \Sigma(\mu))$ as a Gaussian distribution with mean $\mathbf{m}(\mu)$ and standard deviation $\Sigma(\mu)$, and the prior $\rho(\mathbf{z}) = \mathcal{N}(0, \cdot)$ is given by the unit Gaussian. This objective can be interpreted as training a policy that maximizes the agent’s expected return across different dynamics, while also being able to adapt its behaviors when necessary by relying on only a minimal amount of information from the ground-truth dynamics parameters. In our formulation, the Lagrange multiplier β provides a trade-off between robustness and adaptability. Large values of β restrict the amount of information that the policy can access from μ . In the limit $\beta \rightarrow \infty$, the policy converges to a robust but non-adaptive policy that does not access the underlying dynamics parameters. Conversely, small values of $\beta \rightarrow 0$ provides the policy with unfettered access to the dynamics parameters, which can result in brittle strategies where the policy’s behaviors overfit to the nuances of each setting of the dynamics parameters, potentially leading to poor generalization to real-world dynamics.

Real World Transfer

To adapt a policy to the real world, we directly search for an encoding \mathbf{z} that maximizes the return on the physical system

$$\mathbf{z}^* = \arg \max_{\mathbf{z}} \mathbb{E}_{\tau \sim p^*(\tau|\pi, \mathbf{z})} \left[\sum_{t=0}^{T-1} \gamma^t r_t \right], \quad (10.13)$$

with $p^*(\tau|\pi, \mathbf{z})$ being the trajectory distribution under real-world dynamics. To identify \mathbf{z}^* , we use advantage-weighted regression (AWR) [190, 204], a simple off-policy RL algorithm.

Parameter	Training Range	Testing Range
Mass	$[0.8, 1.2] \times \text{default value}$	$[0.5, 2.0] \times \text{default value}$
Inertia	$[0.5, 1.5] \times \text{default value}$	$[0.4, 1.6] \times \text{default value}$
Motor Strength	$[0.8, 1.2] \times \text{default value}$	$[0.7, 1.3] \times \text{default value}$
Motor Friction	$[0, 0.05] \text{ Nms/rad}$	$[0, 0.075] \text{ Nms/rad}$
Latency	$[0, 0.04] \text{ s}$	$[0, 0.05] \text{ s}$
Lateral Friction	$[0.05, 1.25] \text{ Ns/m}$	$[0.04, 1.35] \text{ Ns/m}$

Table 10.1: Dynamic parameters and their respective range of values used during training and testing. A larger range of values are used during testing to evaluate the policies’ ability to generalize to unfamiliar dynamics.

Algorithm 5 summarizes the adaptation process. The search distribution is initialized with the prior $\omega_0(\mathbf{z}) = \mathcal{N}(0, I)$. At each iteration k , we sample an encoding from the current distribution $\mathbf{z}_k \sim \omega_k(\mathbf{z})$ and execute an episode with the policy conditioned on \mathbf{z}_k . The return \mathcal{R}_k for the episode is recorded and stored along with \mathbf{z}_k in a replay buffer \mathcal{D} containing all samples from previous iterations. $\omega_k(\mathbf{z})$ is then updated by fitting a new distribution that assigns higher likelihoods to samples with larger advantages. The likelihood of each sample \mathbf{z}_i is weighted by the exponentiated-advantage $\exp\left(\frac{1}{\alpha}(\mathcal{R}_i - \bar{v})\right)$, where the baseline \bar{v} is the average return of all samples in \mathcal{D} , and α is a manually specified temperature parameter. Note that, since $\omega_k(\mathbf{z})$ is Gaussian, the optimal distribution at each iteration (Line 9) can be determined analytically. However, we found that the analytic solution is prone to premature convergence to a suboptimal solution. Instead, we update $\omega_k(\mathbf{z})$ incrementally using a few steps of gradient descent. This process is repeated for k_{\max} iterations, and the mean of the final distribution $\omega_{k_{\max}}(\mathbf{z})$ is used as an approximation of the optimal encoding \mathbf{z}^* for deploying the policy in the real world.

10.6 Experimental Evaluation

We evaluate our robotic learning system by learning to imitating a variety of dynamic locomotion skills using the Laikago robot [286], an 18 degrees-of-freedom quadruped with 3 actuated degrees-of-freedom per leg, and 6 under-actuated degrees of freedom for the root (torso). Behaviors learned by the policies are best seen in the supplementary video¹, and snapshots of the behaviors are also available in Figure 10.4. In the following experiments, we aim to evaluate the effectiveness of our framework on learning a diverse set of quadruped skills, and study how well real-world adaptation can enable more agile behaviors. We show that our adaptation method can efficiently transfer policies trained in simulation to the real world with a small number of trials on the physical system. We further study the effects of regularizing the latent dynamics encoding with an information bottleneck, and show that this provides a mechanism to trade off between the robustness and adaptability of the learned



Figure 10.4: Laikago robot performing skills learned by imitating reference motions. **Top:** Reference motion. **Middle:** Simulated robot. **Bottom:** Real robot.

policies.

Experimental Setup

Retargeting via inverse-kinematics and simulated training is performed using PyBullet [47]. Table 10.1 summarizes the dynamics parameters and their respective range of values. The motion dataset contains a mixture of mocap clips recorded from a dog and clips from artist

Parameter	Value
Discount factor γ	0.95
Policy Adam learning rate	2×10^{-5}
Value function Adam learning rate	10^{-5}
PPO clip threshold	0.2
PPO batch size	10000
PPO epochs	10
Information penalty coefficient (β)	10^{-4}

Table 10.2: Hyper-parameters used during training in simulation with PPO.

Parameter	Value
Discount factor γ	1.0
Adam learning rate	5×10^{-3}
Gradient steps per iteration	10
AWR temperature α	0.01

Table 10.3: Hyper-parameters used for domain adaptation with AWR in the real world.

generated animations. The mocap clips are collected from a public dataset [314] and re-targeted to the Laikago following the procedure in Section 10.3. Figure 10.6 lists the skills learned by the robot and summarizes the performance of the policies when deployed in the real world. Motion clips recorded from a dog are designated with “Dog”, and the other clips correspond to artist animated motions. Performance is recorded as the average normalized return, with 0 corresponding to the minimum possible return per episode and 1 being the maximum return. Note that the maximum return may not be achievable, since the reference motions are generally not physically feasible for the robot. Performance is calculated using the average of 3 policies initialized with different random seeds. Each policy is trained with proximal policy optimization using about 200 million samples in simulation [238]. Both the encoder and policy are trained end-to-end using the reparameterization trick [125]. Domain adaptation is performed on the physical system with AWR in the latent dynamics space, using approximately 50 real-world trials to adapt each policy. Trials vary between 5s and 10s in length depending on the space requirements of each skill. Table 10.2 summarizes the hyper-parameter settings for training with proximal-policy optimization (PPO) in simulation, and Table 10.3 shows the hyper-parameters for domain adaptation with advantage-weighted regression (AWR). Gradient descent updates are performed using Adam kingma2014adam.

Model representation: All policies are modeled using the neural network architecture shown in Figure 10.5. The encoder $E(\mathbf{z}|\mu)$ is represented by a fully-connected network that maps the dynamics parameters μ to the mean $\mathbf{m}_E(\mu)$ and standard deviation $\Sigma_E(\mu)$ of the encoder distribution. The policy network $\pi(\mathbf{a}|\mathbf{s}, \mathbf{g}, \mathbf{z})$ receives as input the state \mathbf{s} , goal \mathbf{g} , and dynamics encoding \mathbf{z} , then outputs the mean $\mathbf{m}_\pi(\mathbf{s}, \mathbf{g}, \mathbf{z})$ of a Gaussian action distribution. The standard deviation $\Sigma_\pi = \text{diag}(\sigma_\pi^1, \sigma_\pi^2, \dots)$ of the action distribution is represented by a fixed matrix. The value function $V(\mathbf{s}, \mathbf{g}, \mu)$ receives as input the state, goal, and dynamics parameters.

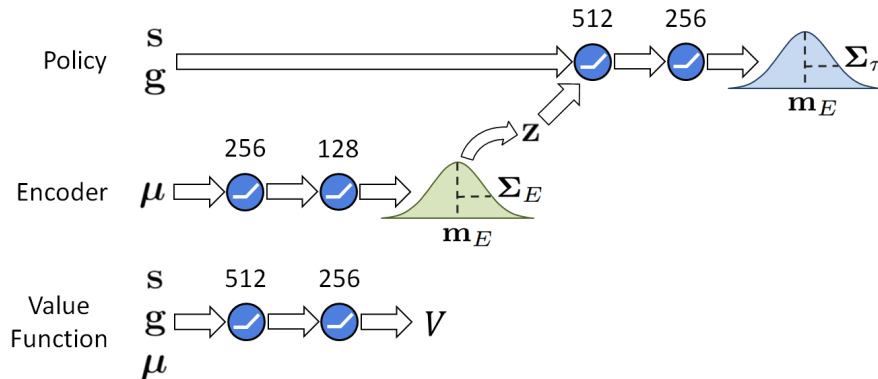


Figure 10.5: Schematic illustration of the network architecture used for the adaptive policy. The encoder $E(\mathbf{z}|\mu)$ receives the dynamics parameters μ as input, which are processed by two fully-connected layers with 256 and 128 ReLU units, and then mapped to a Gaussian distribution over the latent space \mathbf{Z} with mean $\mathbf{m}_E(\mu)$ and standard deviation $\Sigma_E(\mu)$. An encoding \mathbf{z} is sampled from the encoder distribution and provided to the policy $\pi(\mathbf{a}|\mathbf{s}, \mathbf{g}, \mu)$ as input, along with the state \mathbf{s} and goal \mathbf{g} . The policy is modeled with two layers of 512 and 256 units, followed by an output layer which specifies the mean $\mathbf{m}_\pi(\mathbf{s}, \mathbf{g}, \mathbf{z})$ of the action distribution. The standard deviation Σ_π of the action distribution is specified by a fixed diagonal matrix. The value function $V(\mathbf{s}, \mathbf{g}, \mu)$ is modeled by a separate network with 512 and 256 hidden units.

Learned Skills

Our framework is able to learn a diverse set of locomotion skills for the Laikago, including dynamic gaits, such as pacing and trotting, as well as agile turning and spinning motions (Figure 10.4). Pacing is typically used for walking at slower speeds, and is characterized by each pair of legs on the same side of the body moving in unison (Figure 10.4(a)) [218]. Trotting is a faster gait, where diagonal pairs of legs move together (Figure 10.1). We are able to train policies for these different gaits just by providing the system with different reference motions. Furthermore, by simply playing the mocap clips backwards, we are able to train policies for different backwards walking gaits (Figure 10.4(b)). The gaits learned by our policies are faster than those of the manually-designed controller from the manufacturer. The fastest manufacturer gait reaches a top speed of about 0.84m/s, while the Dog Trot policy reaches a speed of 1.08m/s. The backwards trotting gait reaches an even higher speed of 1.20m/s. In addition to imitating mocap data from animals, our system is also able to learn from artist animated motions. While these hand-animated motions are generally not physically correct, the policies are nonetheless able to closely imitate most motions with the real robot. This includes a highly dynamic Hop-Turn motion, in which the robot performs a 90 degrees turn midair (Figure 10.4(e)). While our system is able to imitate a variety of motions, some motions, such as Running Man (Figure 10.4(f)), prove challenging to reproduce. The motion requires the robot to travel backwards while moving in a forward-

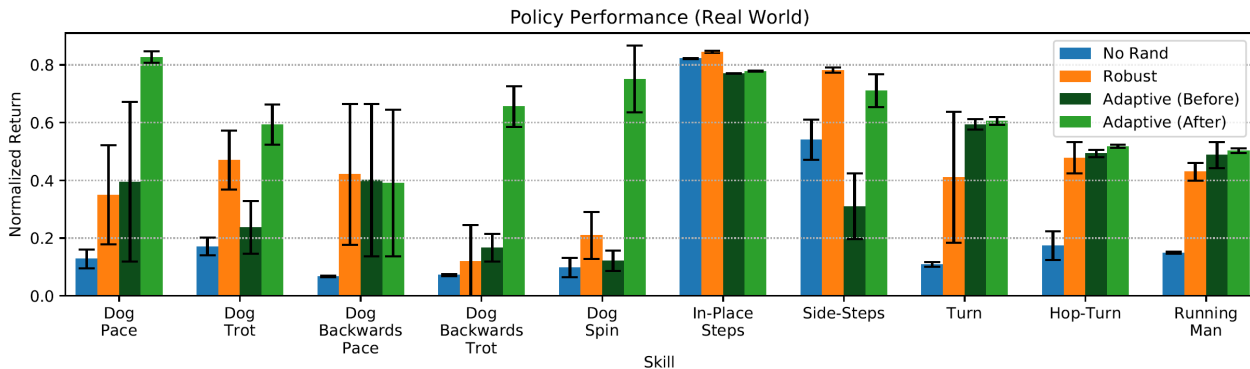


Figure 10.6: Performance statistics of imitating various skills in the real world. Performance is recorded as the average normalized return between $[0, 1]$. Three policies initialized with different random seeds are trained for each combination of skill and method. The performance of each policy is evaluated over 5 episodes, for a total of 15 trials per method. The adaptive policies outperform the non-adaptive policies on most skills.

Skill	No Rand	Robust	Adaptive (Before)	Adaptive (After)
Dog Pace	0.128 ± 0.033	0.350 ± 0.172	0.395 ± 0.277	0.827 ± 0.020
Dog Trot	0.171 ± 0.031	0.471 ± 0.102	0.237 ± 0.092	0.593 ± 0.070
Dog Backwards Pace	0.067 ± 0.003	0.421 ± 0.244	0.401 ± 0.264	0.390 ± 0.254
Dog Backwards Trot	0.072 ± 0.004	0.120 ± 0.126	0.167 ± 0.048	0.656 ± 0.071
Dog Spin	0.098 ± 0.033	0.209 ± 0.081	0.121 ± 0.035	0.751 ± 0.116
In-Place Steps	0.822 ± 0.002	0.845 ± 0.004	0.771 ± 0.001	0.778 ± 0.002
Side-Steps	0.541 ± 0.070	0.782 ± 0.009	0.310 ± 0.114	0.710 ± 0.057
Turn	0.108 ± 0.008	0.410 ± 0.227	0.594 ± 0.018	0.606 ± 0.014
Hop-Turn	0.174 ± 0.050	0.478 ± 0.054	0.493 ± 0.012	0.518 ± 0.005
Running Man	0.149 ± 0.004	0.430 ± 0.031	0.488 ± 0.045	0.503 ± 0.008

Table 10.4: Performance statistics of imitating various skills in the real world. The method that achieves the highest return for each skill on the real robot is highlighted. Our adaptive model achieves higher returns on most skills.

walking manner. Our policies learn to keep the robot’s feet on the ground and shuffle backwards, instead of lifting the feet during each step.

Domain Adaptation

To determine the effects of domain adaptation, we compare our method to non-adaptive policies trained in simulation without randomization (No Rand), and robust policies trained with randomization (Robust) but do not perform adaptation in new environments. Real-world performance comparisons of these methods are shown in Figure 10.6 and Table 10.4. When deployed on the real robot, the adaptive policies outperform their non-adaptive counterparts on most skills. For simpler skills, such as In-Place Steps and Side-Steps, the robust

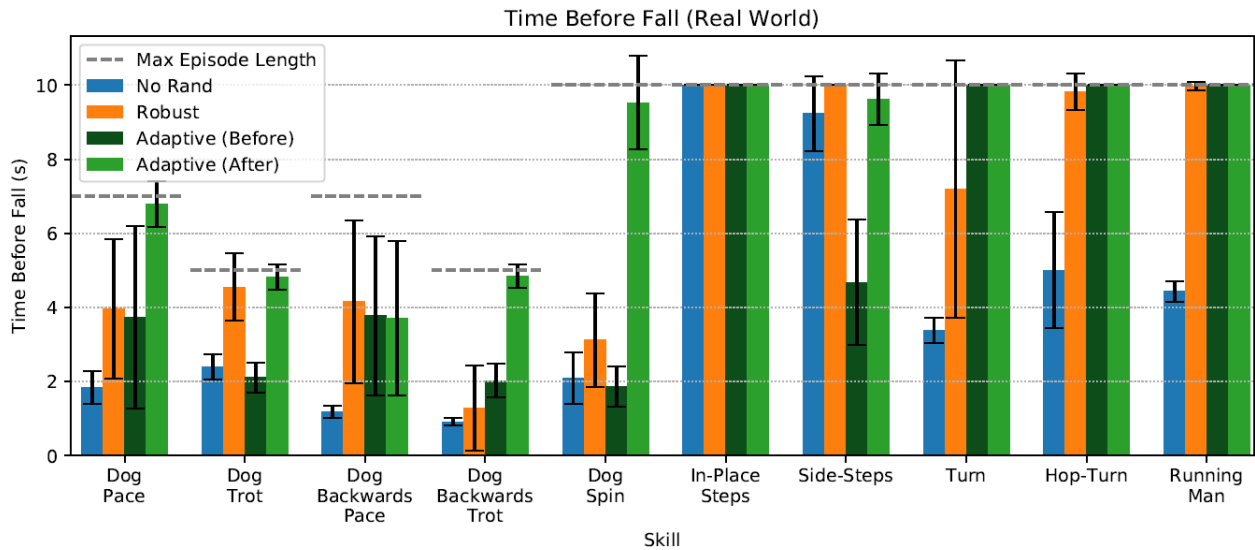


Figure 10.7: Comparison of the time elapsed before the robot falls when deploying various policies in the real world. The adaptive policies are often able to maintain balance longer than the other baselines policies, and tend to reach the max episode length without falling.

policies are sufficient for transfer to the real robot. But for more dynamic skills, such as Dog Pace and Dog Spin, the robust policies are prone to falling, while the adaptive policies can execute the skills more consistently. Policies trained without randomization fail to transfer to the real world for most skills. Figure 10.7 compares the time elapsed before the robot falls under the various policies. The adaptive policies are often able to maintain balance for a longer period of time than the other methods, with a significant performance improvement after adaptation.

To evaluate the policies' abilities to cope with unfamiliar dynamics, we test the policies in out-of-distribution simulated environments, where the dynamics parameters are sampled from a larger range of values than those used during training. The range of values used during training and testing are detailed in Table 10.1. Figure 10.8 visualizes the performance of the policies in 100 simulated environments with different dynamics. The vertical axis represents the normalized return, and the horizontal axis records the portion of environments in which a policy achieves a return higher than a particular value. For example, in the case of Dog Pace, the adaptive policies achieve a return higher than 0.6 in 50% of the environments, while the robust policy achieves a return higher than 0.6 in 38% of the environments. The experiments are repeated 3 times for each method using policies initialized with different random seeds. In these experiments, the adaptive policies tend to outperform their non-adaptive counterparts across the various skills. This suggests that the adaptation process is able to better generalize to environments that differ from those encountered during training. To analyze the performance of policies during the adaptation process, we record the performance of individual policies after each update iteration. Figure 10.9 illustrates the

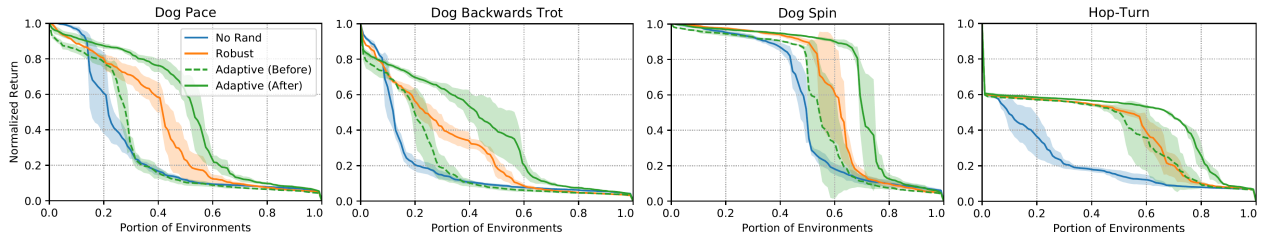


Figure 10.8: Performance of policies in 100 simulated environments with different dynamics. The y-axis represents the normalized return, and the x-axis records the portion of environments in which a policy achieves a return higher than a particular value. The adaptive policies achieve higher returns under more diverse dynamics than the non-adaptive policies.

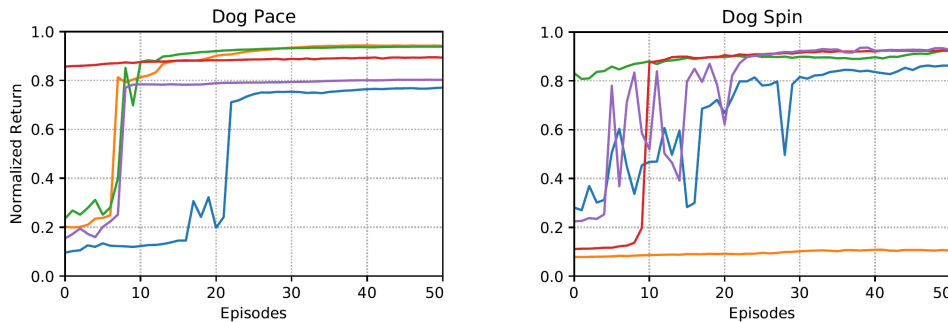


Figure 10.9: Learning curves of adapting policies to different simulated environments using the learned latent space. The policies are able to adapt to new environments in a relatively small number of episodes.

learning curves in 5 different environments for each skill. The policies are generally able to adapt to new environments in a relatively few number of episodes.

Information Bottleneck

Next we evaluate the effects of the information bottleneck on adaptation performance. Figure 10.10 summarizes the performance of policies trained with different values of β for the information penalty. Larger values of β produce policies that access fewer number of bits of information from the dynamics parameters during pre-training. This encourages a policy to be less reliant on precise knowledge of the underlying dynamics, which in turn results in more robust behaviors that attain higher performance before adaptation. However, since the policy’s behavior is less dependent on the latent variables, this can also result in less adaptable policies, which exhibit smaller performance improvements after adaptation. Similarly, smaller values of β tend to produce less robust but more adaptive policies, exhibiting lower performance before adaptation, but a larger improvement after adaptation. In our experiments, we find that $\beta = 10^{-4}$ provides a good trade-off between robustness and adaptability.

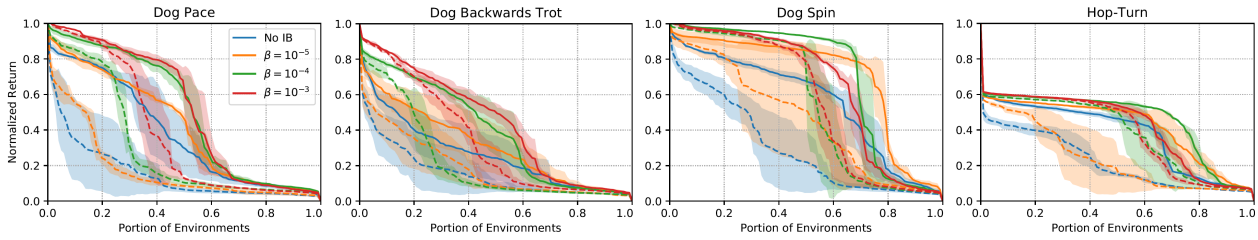


Figure 10.10: Performance of adaptive policies trained with different coefficients β for the information penalty. "No IB" corresponds to policies trained without an information bottleneck. The dotted lines represent performance before adaptation, and the solid lines represent after adaptation. Larger values of β results in more robust but less adaptable policies, which starting with better performance before adaptation, but exhibits smaller improvements after adaptation. Vice-versa for smaller values of β .

We also compare the information-constrained latent representations to the unconstrained counterparts (No IB). The information-constrained policies generally achieve better performance both before and after adaptation.

10.7 Discussion

In this chapter, we presented a framework for learning agile legged-locomotion skills by imitating reference motion data. By simply providing the system with different reference motions, we are able to learn policies for a diverse set of behaviors with a quadruped robot, which can then be efficiently transferred from simulation to the real world. However, due to hardware and algorithmic limitations, we have not been able to learn more dynamic behaviors such as large jumps and runs. Exploring techniques that are able to reproduce these behaviors in the real world could significantly increase the agility of legged robots. The behaviors learned by our policies are currently not as stable as the best manually-designed controllers. Improving the robustness of these learned controllers would be valuable for more complex real-world applications. We are also interested in learning from other sources of motion data, such video clips, which could substantially increase the volume of behavioral data that robots can learn from.

Chapter 11

Conclusion

Motor control models that can replicate the physical and athletic capabilities of humans and animals can enable a wide range of applications. In this thesis, we have taken steps towards this goal by developing learning-based frameworks that allow agents to reproduce a large and diverse repertoire of motor skills. A common theme throughout our work is the use of motion data to shape the behaviors of policies trained through reinforcement learning, which greatly reduces the reliance on carefully-crafted skill-specific control strategies and reward functions. In Chapter 3, we showed that a versatile motion-tracking approach can be applied to learn a large corpus of complex behaviors in simulation, ranging from common locomotion behaviors such as walking and running, to more athletic capabilities such as acrobatics and martial arts. The agents are able to produce robust and high-quality motions that are nearly indistinguishable in appearance from motion clips recorded from real-life actors. By incorporating vision-based pose estimation techniques, our approach can be extended to imitate challenging skills from video clips, such as those readily found on YouTube. In Chapter 6, we developed an adversarial imitation learning approach that enables agents to imitate behaviors from large unstructured motion datasets, resulting in policies that can automatically select, interpolate, and seamlessly transition between diverse behaviors.

In addition to developing motor skills for simulated agents, our methods can also be applied to develop motor skills for robots in the real world. To circumvent challenges of directly applying reinforcement learning on physical systems, we have primarily utilized sim-to-real transfer techniques to first train policies in simulation and then transfer the learned policies to robots operating in the real world. In order to bridge the reality gap, we showed that randomizing the dynamics of the simulated environment during training can lead to sufficiently robust policies that can then be deployed directly on real robots. However, for more dynamic locomotion skills, which can be more sensitive to variations in the dynamics, direct sim-to-real transfer may not always be successful. Therefore, we proposed a sample efficient adaptation technique that further fine-tunes the behavior of a policy using a small amount of real-world data. This approach then enables the transfer of more agile behaviors, such as fast trotting, hopping, and spinning motions.

The work we presented in this thesis have taken steps towards developing agents that can

replicate the rich physical capabilities of humans and animals. However, there remains a large gap between the capabilities demonstrated by humans and those that can be reproduced by our agents. Humans can not only perform a vast corpus of behaviors, but can also compose and sequence those skills in versatile ways to further overarching goals. While our techniques can train policies to imitate a large variety of skills, the breadth of behaviors that can be effectively modeled by an individual policy is still fairly limited. More expressive and versatile models will be needed before our agents can agilely interact and perform sophisticated tasks in complex environments. These challenges highlight a number of exciting directions for future exploration:

Reusable Motor Models: Large expressive models trained on massive datasets have been central to the recent successes in machine learning. Not only can these models solve complex tasks, but also provide reusable components for a wide range of downstream applications. This is in stark contrast to the current state-of-affairs in reinforcement learning and motor control, where models are most typically trained to specialize in a narrow set of skills. Motion imitation can provide a sufficiently rich set of tasks for training more expressive and general motor control models. Coupled with architectures that are amenable to reuse, these models may produce useful building blocks that enable novel downstream applications, which would otherwise not be possible with *tabula rasa* learning.

Learning in the Real World: While the work in this thesis has taken steps towards deploying policies trained in simulation to robots in the real world, there remains a large gap between the capabilities of simulated and real-world agents. Moving forward, simulation will likely continue to be a vital component of the skill acquisition pipeline, as a way of providing models with good initializations prior to real-world deployment. But as the focus shifts to more and more sophisticated behaviors, real-world training will likely be crucial for successful deployment of these skills. Developing algorithms that can learn safely, efficiently, and continuously learn in real world environments will therefore be of vital importance for developing robots that can replicate the capabilities of their real-life counterparts.

Directability: One of the impediments that have precluded physics-based character animation from being more widely adopted in computer graphics applications is the difficulty of directing the behaviors of physically simulated characters. Current learning-based frameworks are generally not well suited for the iterative process of animation workflows, where users incrementally refine a given motion according to feedback and production requirements. Developing intuitive interfaces for users to specify feedback, and algorithms that can quickly update a character's behaviors according to that feedback will be valuable for the wider adoption of these simulation-based animation systems. This direction can also provide useful tools for directing the behavior of real-world robots, providing accessible interfaces through which nontechnical users can control the behaviors of robotic agents.

Behavioral Priors: In addition to learning behaviors from motion data, it may also be possible to learn behavioral priors that quantify abstract characteristics, such as the naturalness of a motion. Our work on adversarial motion priors has taken steps towards this goal, but the priors from that system must be trained in tandem with a particular policy, and cannot be easily reused to train new policies without additional retraining. More portable priors can provide convenient objective functions for training agents to perform new tasks, while adopting behaviors that conform to characteristics specified by the priors. Such priors may also be useful tools for artists by providing a quantitative score of the *realism* of manually authored animations, and possibly suggesting instructive feedback for potential corrections. This line of work can provide relevant insights for inverse reinforcement learning and adversarial examples, which can have impact on a wide range of applications beyond computer graphics.

Cross-Domain Imitation Learning: While our work has made heavy use of motion data recorded from human actors, we have largely overlooked challenges associated with the domain shift between the demonstrator and the agent. Motion data recorded from real-life actors will rarely have the same embodiment and morphology as the agent. Our systems obviate this mismatch by using motion retargeting techniques to map a given motion clip to the target agent’s embodiment. However, this retargeting process tends to be manual and tedious, requiring users to directly specify correspondences between the demonstrator and the agent that captures the semantically salient characteristics of a desired skill. Developing imitation learning techniques that can automatically resolve such domain shifts can lead to more versatile and scalable systems that are able to imitate demonstrations with drastic domain shifts, such as directly imitating video clips of human actors without the need for explicit pose estimation. Effective cross-domain imitation learning may allow agents to continuously observe and learn from other agents in their environment, perpetually growing their repertoire of skills by mimicking others.

* * *

We hope the work presented in this thesis will provide useful building blocks on the path towards agents that can rival the athletic prowess of humans and animals. Just as the evolution of the brain has been spurred by the need for complex motor skills, the insights gained through building systems that can replicate these sophisticated behaviors may also help to shed light on the nature of intelligence.

Bibliography

- [1] Pieter Abbeel and Andrew Y. Ng. “Apprenticeship Learning via Inverse Reinforcement Learning”. In: *Proceedings of the Twenty-First International Conference on Machine Learning*. ICML '04. Banff, Alberta, Canada: Association for Computing Machinery, 2004, p. 1. ISBN: 1581138385. DOI: 10.1145/1015330.1015430. URL: <https://doi.org/10.1145/1015330.1015430>.
- [2] Farzad Adbolhosseini et al. “On Learning Symmetric Locomotion”. In: *Proc. ACM SIGGRAPH Motion, Interaction, and Games (MIG 2019)*. 2019.
- [3] Agility Robotics. *cassie-mujoco-sim*. (2018) [Online]. URL: <https://github.com/osudr1/cassie-mujoco-sim>.
- [4] Shailen Agrawal and Michiel van de Panne. “Task-based Locomotion”. In: *ACM Transactions on Graphics (Proc. SIGGRAPH 2016)* 35.4 (2016).
- [5] Shailen Agrawal, Shuo Shen, and Michiel van de Panne. “Diverse Motion Variations for Physics-based Character Animation”. In: *Symposium on Computer Animation* (2013).
- [6] Srinivas Akella and Matthew T. Mason. “Posing Polygonal Objects in the Plane by Pushing”. In: *The International Journal of Robotics Research* 17.1 (1998), pp. 70–88. DOI: 10.1177/027836499801700107.
- [7] M. Al Borno, M. de Lasa, and A. Hertzmann. “Trajectory Optimization for Full-Body Movements with Complex Contacts”. In: *IEEE Transactions on Visualization and Computer Graphics* 19.8 (2013), pp. 1405–1414. DOI: 10.1109/TVCG.2012.325.
- [8] Alexander A. Alemi et al. “Deep Variational Information Bottleneck”. In: *CoRR* abs/1612.00410 (2016). arXiv: 1612.00410. URL: <http://arxiv.org/abs/1612.00410>.
- [9] Marcin Andrychowicz et al. “Hindsight Experience Replay”. In: *Advances in Neural Information Processing Systems*. 2017.
- [10] Rika Antonova et al. “Reinforcement Learning for Pivoting Task”. In: *CoRR* abs/1703.00472 (2017). URL: <http://arxiv.org/abs/1703.00472>.
- [11] Dafni Antotsiou, Guillermo Garcia-Hernando, and Tae-Kyun Kim. “Task-Oriented Hand Motion Retargeting for Dexterous Manipulation Imitation”. In: *ECCV Workshops*. 2018.

- [12] Taylor Apgar et al. “Fast Online Trajectory Optimization for the Bipedal Robot Cassie”. In: June 2018. DOI: 10.15607/RSS.2018.XIV.054.
- [13] Andreas Argyriou, Theodoros Evgeniou, and Massimiliano Pontil. “Multi-Task Feature Learning”. In: *Advances in Neural Information Processing Systems 19*. Ed. by B. Schölkopf, J. C. Platt, and T. Hoffman. MIT Press, 2007, pp. 41–48. URL: <http://papers.nips.cc/paper/3143-multi-task-feature-learning.pdf>.
- [14] Martin Arjovsky, Soumith Chintala, and Léon Bottou. “Wasserstein Generative Adversarial Networks”. In: ed. by Doina Precup and Yee Whye Teh. Vol. 70. Proceedings of Machine Learning Research. International Convention Centre, Sydney, Australia: PMLR, June 2017, pp. 214–223. URL: <http://proceedings.mlr.press/v70/arjovsky17a.html>.
- [15] Kai Arulkumaran, Antoine Cully, and Julian Togelius. “AlphaStar: An Evolutionary Computation Perspective”. In: *Proceedings of the Genetic and Evolutionary Computation Conference Companion*. GECCO ’19. Prague, Czech Republic: Association for Computing Machinery, 2019, pp. 314–315. ISBN: 9781450367486. DOI: 10.1145/3319619.3321894. URL: <https://doi.org/10.1145/3319619.3321894>.
- [16] Salman Aslam. *YouTube by the Numbers*. <https://www.omnicoreagency.com/youtube-statistics/>. Accessed: 2018-05-15. 2018.
- [17] C.G. Atkeson and S. Schaal. “Learning tasks from a single demonstration”. In: *Proceedings of International Conference on Robotics and Automation*. Vol. 2. 1997, 1706–1712 vol.2. DOI: 10.1109/ROBOT.1997.614389.
- [18] Pierre-Luc Bacon, Jean Harb, and Doina Precup. “The Option-Critic Architecture”. In: *AAAI*. 2016.
- [19] Marc G. Bellemare et al. “Unifying Count-Based Exploration and Intrinsic Motivation”. In: *CoRR* abs/1606.01868 (2016). arXiv: 1606.01868.
- [20] Kevin Bergamin et al. “DReCon: Data-Driven Responsive Control of Physics-Based Characters”. In: *ACM Trans. Graph.* 38.6 (Nov. 2019). ISSN: 0730-0301. DOI: 10.1145/3355089.3356536. URL: <https://doi.org/10.1145/3355089.3356536>.
- [21] Christopher Berner et al. “Dota 2 with Large Scale Deep Reinforcement Learning”. In: *CoRR* abs/1912.06680 (2019). arXiv: 1912.06680. URL: <http://arxiv.org/abs/1912.06680>.
- [22] David Berthelot, Tom Schumm, and Luke Metz. “BEGAN: Boundary Equilibrium Generative Adversarial Networks”. In: *CoRR* abs/1703.10717 (2017). arXiv: 1703.10717. URL: <http://arxiv.org/abs/1703.10717>.
- [23] Gerardo Blede et al. “MIT Cheetah 3: Design and Control of a Robust, Dynamic Quadruped Robot”. In: *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (2018), pp. 2245–2252.

- [24] Federica Bogo et al. “Keep it SMPL: Automatic Estimation of 3D Human Pose and Shape from a Single Image”. In: *European Conference on Computer Vision, ECCV*. Lecture Notes in Computer Science. Springer International Publishing, Oct. 2016.
- [25] Mariusz Bojarski et al. “End to End Learning for Self-Driving Cars”. In: *CoRR* abs/1604.07316 (2016). arXiv: 1604.07316. URL: <http://arxiv.org/abs/1604.07316>.
- [26] Christoph Bregler and Jitendra Malik. “Tracking people with twists and exponential maps”. In: *IEEE Conference on Computer Vision and Pattern Recognition, CVPR*. IEEE. 1998, pp. 8–15.
- [27] Andrew Brock, Jeff Donahue, and Karen Simonyan. “Large Scale GAN Training for High Fidelity Natural Image Synthesis”. In: *International Conference on Learning Representations*. 2019. URL: <https://openreview.net/forum?id=B1xsqj09Fm>.
- [28] Greg Brockman et al. “OpenAI Gym”. In: *CoRR* abs/1606.01540 (2016). arXiv: 1606.01540.
- [29] Tom B. Brown et al. “Language Models are Few-Shot Learners”. In: *CoRR* abs/2005.14165 (2020). arXiv: 2005.14165. URL: <https://arxiv.org/abs/2005.14165>.
- [30] Bullet. *Bullet Physics Library*. <http://bulletphysics.org>. 2015.
- [31] W. Burgard, O. Brock, and C. Stachniss. “Learning Omnidirectional Path Following Using Dimensionality Reduction”. In: *Robotics: Science and Systems III*. 2008, pp. 257–264.
- [32] Stephen Butterworth et al. “On the theory of filter amplifiers”. In: *Wireless Engineer* 7.6 (1930), pp. 536–541.
- [33] Zhe Cao et al. “Realtime Multi-Person 2D Pose Estimation using Part Affinity Fields”. In: *CVPR*. 2017.
- [34] Rich Caruana. “Multitask Learning”. In: *Machine Learning* 28.1 (July 1997), pp. 41–75.
- [35] Guillermo A Castillo et al. “Hybrid zero dynamics inspired feedback control policy design for 3d bipedal locomotion using reinforcement learning”. In: *IEEE International Conference on Robotics and Automation*. 2020, pp. 8746–8752.
- [36] Guillermo A Castillo et al. “Velocity Regulation of 3D Bipedal Walking Robots with Uncertain Dynamics Through Adaptive Neural Network Controller”. In: *arXiv preprint arXiv:2008.00376* (2020).
- [37] Yevgen Chebotar et al. “Closing the Sim-to-Real Loop: Adapting Simulation Randomization with Real World Experience”. In: *CoRR* abs/1810.05687 (2018). arXiv: 1810.05687. URL: <http://arxiv.org/abs/1810.05687>.
- [38] Mark Chen et al. “Evaluating Large Language Models Trained on Code”. In: *CoRR* abs/2107.03374 (2021). arXiv: 2107.03374. URL: <https://arxiv.org/abs/2107.03374>.

- [39] Nuttapon Chentanez et al. “Physics-Based Motion Capture Imitation with Deep Reinforcement Learning”. In: *Proceedings of the 11th Annual International Conference on Motion, Interaction, and Games*. MIG '18. Limassol, Cyprus: Association for Computing Machinery, 2018. ISBN: 9781450360159. DOI: 10.1145/3274247.3274506. URL: <https://doi.org/10.1145/3274247.3274506>.
- [40] Paul Christiano et al. “Transfer from Simulation to Real World through Learning Deep Inverse Dynamics Model”. In: *CoRR* abs/1610.03518 (2016). URL: <http://arxiv.org/abs/1610.03518>.
- [41] Ignasi Clavera et al. “Learning to Adapt in Dynamic, Real-World Environments through Meta-Reinforcement Learning”. In: *International Conference on Learning Representations*. 2019. URL: <https://openreview.net/forum?id=HyztsoC5Y7>.
- [42] Alexander Clegg et al. “Learning to Dress: Synthesizing Human Dressing Motion via Deep Reinforcement Learning”. In: *ACM Trans. Graph.* 37.6 (Dec. 2018). ISSN: 0730-0301. DOI: 10.1145/3272127.3275048. URL: <https://doi.org/10.1145/3272127.3275048>.
- [43] CMU. *CMU Graphics Lab Motion Capture Database*. <http://mocap.cs.cmu.edu/>.
- [44] Stelian Coros, Philippe Beaudoin, and Michiel van de Panne. “Generalized Biped Walking Control”. In: *ACM Transactions on Graphics* 29.4 (2010), Article 130.
- [45] Stelian Coros, Philippe Beaudoin, and Michiel van de Panne. “Robust Task-based Control Policies for Physics-based Characters”. In: *ACM Trans. Graph. (Proc. SIGGRAPH Asia)* 28.5 (2009), Article 170.
- [46] Stelian Coros et al. “Locomotion Skills for Simulated Quadrupeds”. In: *ACM Transactions on Graphics* 30.4 (2011), Article TBD.
- [47] Erwin Coumans and Yunfei Bai. *PyBullet, a Python module for physics simulation for games, robotics and machine learning*. <http://pybullet.org>. 2016–2019.
- [48] Xingye Da et al. “From 2D design of underactuated bipedal gaits to 3D implementation: Walking with speed tracking”. In: *IEEE Access* 4 (2016), pp. 3469–3478.
- [49] M. Da Silva, Y. Abe, and J. Popovic. “Simulation of Human Motion Data using Short-Horizon Model-Predictive Control”. In: *Computer Graphics Forum* (2008). ISSN: 1467-8659.
- [50] Shreyansh Daftry, J. Andrew Bagnell, and Martial Hebert. “Learning Transferable Policies for Monocular Reactive MAV Control”. In: *CoRR* abs/1608.00627 (2016). URL: <http://arxiv.org/abs/1608.00627>.
- [51] Hongkai Dai, Andrés Valenzuela, and Russ Tedrake. “Whole-body motion planning with centroidal dynamics and full kinematics”. In: *IEEE-RAS International Conference on Humanoid Robots*. 2014, pp. 295–302.

- [52] Peter Dayan and Geoffrey E. Hinton. “Feudal Reinforcement Learning”. In: *Advances in Neural Information Processing Systems 5, [NIPS Conference]*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1993, pp. 271–278. ISBN: 1-55860-274-7. URL: <http://dl.acm.org/citation.cfm?id=645753.668239>.
- [53] J. Deng et al. “ImageNet: A Large-Scale Hierarchical Image Database”. In: *CVPR09*. 2009.
- [54] Jared Di Carlo et al. “Dynamic locomotion in the MIT cheetah 3 through convex model-predictive control”. In: *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2018, pp. 1–9.
- [55] Mehmet Dogar and Siddhartha Srinivasa. “A Framework for Push-grasping in Clutter”. In: *Robotics: Science and Systems VII*. Pittsburgh, PA: MIT Press, July 2011.
- [56] Jeff Donahue et al. “DeCAF: A Deep Convolutional Activation Feature for Generic Visual Recognition”. In: *Proceedings of the 31st International Conference on Machine Learning*. Ed. by Eric P. Xing and Tony Jebara. Vol. 32. Proceedings of Machine Learning Research 1. Beijing, China: PMLR, 22–24 Jun 2014, pp. 647–655. URL: <http://proceedings.mlr.press/v32/donahue14.html>.
- [57] Yan Duan et al. “Benchmarking Deep Reinforcement Learning for Continuous Control”. In: *CoRR* abs/1604.06778 (2016). arXiv: 1604.06778.
- [58] Yan Duan et al. “RL 2: Fast Reinforcement Learning via Slow Reinforcement Learning”. In: *CoRR* abs/1611.02779 (2016). arXiv: 1611.02779. URL: <http://arxiv.org/abs/1611.02779>.
- [59] Gen Endo et al. “Learning CPG Sensory Feedback with Policy Gradient for Biped Locomotion for a Full-Body Humanoid”. In: *Proceedings of the 20th National Conference on Artificial Intelligence - Volume 3. AAAI’05*. Pittsburgh, Pennsylvania: AAAI Press, 2005, pp. 1267–1273. ISBN: 157735236x.
- [60] Benjamin Eysenbach et al. “Diversity is All You Need: Learning Skills without a Reward Function”. In: *International Conference on Learning Representations*. 2019. URL: <https://openreview.net/forum?id=SJx63jRqFm>.
- [61] Petros Faloutsos, Michiel van de Panne, and Demetri Terzopoulos. “Composable Controllers for Physics-based Character Animation”. In: *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques. SIGGRAPH ’01*. New York, NY, USA: ACM, 2001, pp. 251–260. ISBN: 1-58113-374-X. DOI: 10.1145/383259.383287. URL: <http://doi.acm.org/10.1145/383259.383287>.
- [62] Nima Fazeli et al. “Parameter and contact force estimation of planar rigid-bodies undergoing frictional contact”. In: *The International Journal of Robotics Research* 0.0 (2016), p. 0278364917698749.
- [63] Roy Featherstone. *Rigid Body Dynamics Algorithms*. Berlin, Heidelberg: Springer-Verlag, 2007. ISBN: 0387743146.

- [64] Siyuan Feng et al. “3D walking based on online optimization”. In: *2013 13th IEEE-RAS International Conference on Humanoid Robots*. 2013, pp. 21–27.
- [65] Chelsea Finn, Pieter Abbeel, and Sergey Levine. “Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks”. In: *Proceedings of the 34th International Conference on Machine Learning*. Ed. by Doina Precup and Yee Whye Teh. Vol. 70. Proceedings of Machine Learning Research. International Convention Centre, Sydney, Australia: PMLR, June 2017, pp. 1126–1135. URL: <http://proceedings.mlr.press/v70/finn17a.html>.
- [66] Chelsea Finn, Ian J. Goodfellow, and Sergey Levine. “Unsupervised Learning for Physical Interaction through Video Prediction”. In: *CoRR* abs/1605.07157 (2016). URL: <http://arxiv.org/abs/1605.07157>.
- [67] Carlos Florensa, Yan Duan, and Pieter Abbeel. “Stochastic Neural Networks for Hierarchical Reinforcement Learning”. In: *Proceedings of the International Conference on Learning Representations (ICLR)*. 2017.
- [68] Jakob N. Foerster et al. “Learning to Communicate with Deep Multi-Agent Reinforcement Learning”. In: *CoRR* abs/1605.06676 (2016). URL: <http://arxiv.org/abs/1605.06676>.
- [69] Kevin Frans et al. “Meta Learning Shared Hierarchies”. In: *CoRR* abs/1710.09767 (2017). arXiv: 1710.09767. URL: <http://arxiv.org/abs/1710.09767>.
- [70] Justin Fu, Katie Luo, and Sergey Levine. “Learning Robust Rewards with Adversarial Inverse Reinforcement Learning”. In: *International Conference on Learning Representations*. 2018. URL: <https://openreview.net/forum?id=rkHyw1-A->.
- [71] Justin Fu, John Co-Reyes, and Sergey Levine. “EX2: Exploration with Exemplar Models for Deep Reinforcement Learning”. In: *Advances in Neural Information Processing Systems 30*. Curran Associates, Inc., 2017, pp. 2574–2584.
- [72] Levi Fussell, Kevin Bergamin, and Daniel Holden. “SuperTrack: Motion Tracking for Physically Simulated Characters Using Supervised Learning”. In: *ACM Trans. Graph.* 40.6 (Dec. 2021). ISSN: 0730-0301. DOI: 10.1145/3478513.3480527. URL: <https://doi.org/10.1145/3478513.3480527>.
- [73] Yaroslav Ganin et al. “Domain-adversarial Training of Neural Networks”. In: *J. Mach. Learn. Res.* 17.1 (Jan. 2016), pp. 2096–2030. ISSN: 1532-4435. URL: <http://dl.acm.org/citation.cfm?id=2946645.2946704>.
- [74] Christian Gehring et al. “Practice Makes Perfect: An Optimization-Based Approach to Controlling Agile Motions for a Quadruped Robot”. In: *IEEE Robotics & Automation Magazine* (Feb. 2016), pp. 1–1. DOI: 10.1109/MRA.2015.2505910.
- [75] Thomas Geijtenbeek, Michiel van de Panne, and A. Frank van der Stappen. “Flexible Muscle-Based Locomotion for Bipedal Creatures”. In: *ACM Transactions on Graphics* 32.6 (2013).

- [76] Hartmut Geyer, Andre Seyfarth, and Reinhard Blickhan. “Positive force feedback in bouncing gaits?” In: *Proceedings. Biological sciences / The Royal Society* 270 (Nov. 2003), pp. 2173–83. DOI: 10.1098/rspb.2003.2454.
- [77] Seyed Kamyar Seyed Ghasemipour, Richard Zemel, and Shixiang Gu. “A Divergence Minimization Perspective on Imitation Learning Methods”. In: *Proceedings of the Conference on Robot Learning*. Ed. by Leslie Pack Kaelbling, Danica Kragic, and Komei Sugiura. Vol. 100. Proceedings of Machine Learning Research. PMLR, 30 Oct–01 Nov 2020, pp. 1259–1277.
- [78] Michael Gleicher. “Retargetting Motion to New Characters”. In: *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '98. New York, NY, USA: ACM, 1998, pp. 33–42. ISBN: 0-89791-999-8. DOI: 10.1145/280814.280820. URL: <http://doi.acm.org/10.1145/280814.280820>.
- [79] Yukai Gong et al. “Feedback control of a cassie bipedal robot: Walking, standing, and riding a segway”. In: *American Control Conference*. 2019, pp. 4559–4566.
- [80] Ian Goodfellow et al. “Generative Adversarial Nets”. In: *Advances in Neural Information Processing Systems 27*. Ed. by Z. Ghahramani et al. Curran Associates, Inc., 2014, pp. 2672–2680. URL: <http://papers.nips.cc/paper/5423-generative-adversarial-nets.pdf>.
- [81] A. Goswami. “Foot rotation indicator (FRI) point: a new gait planning tool to evaluate postural stability of biped robots”. In: *Proceedings 1999 IEEE International Conference on Robotics and Automation (Cat. No.99CH36288C)*. Vol. 1. May 1999, 47–52 vol.1. DOI: 10.1109/ROBOT.1999.769929.
- [82] F. Sebastin Grassia. “Practical Parameterization of Rotations Using the Exponential Map”. In: *J. Graph. Tools* 3.3 (Mar. 1998), pp. 29–48. ISSN: 1086-7651. DOI: 10.1080/10867651.1998.10487493. URL: <https://doi.org/10.1080/10867651.1998.10487493>.
- [83] David B. Grimes, Rawichote Chalodhorn, and Rajesh P. N. Rao. “Dynamic Imitation in a Humanoid Robot through Nonparametric Probabilistic Inference.” In: *Robotics: Science and Systems*. Ed. by Gaurav S. Sukhatme et al. The MIT Press, 2006. ISBN: 0-262-69348-8. URL: <http://dblp.uni-trier.de/db/conf/rss/rss2006.html#GrimesCR06>.
- [84] J W. Grizzle et al. “3D Bipedal Robotic Walking: Models, Feedback Control, and Open Problems”. In: *IFAC Symposium on Nonlinear Control Systems*. 2010.
- [85] Shixiang Gu et al. “Deep Reinforcement Learning for Robotic Manipulation”. In: *CoRR* abs/1610.00633 (2016). arXiv: 1610.00633. URL: <http://arxiv.org/abs/1610.00633>.

- [86] Ishaan Gulrajani et al. “Improved Training of Wasserstein GANs”. In: *Advances in Neural Information Processing Systems 30*. Ed. by I. Guyon et al. Curran Associates, Inc., 2017, pp. 5767–5777. URL: <http://papers.nips.cc/paper/7159-improved-training-of-wasserstein-gans.pdf>.
- [87] Abhishek Gupta et al. “Learning Invariant Feature Spaces to Transfer Skills with Reinforcement Learning”. In: *CoRR* abs/1703.02949 (2017). URL: <http://arxiv.org/abs/1703.02949>.
- [88] Sehoon Ha and C Karen Liu. “Iterative training of dynamic skills inspired by human coaching techniques”. In: *ACM Transactions on Graphics* 34.1 (2014).
- [89] Tuomas Haarnoja et al. “Latent Space Policies for Hierarchical Reinforcement Learning”. In: *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*. 2018, pp. 1846–1855. URL: <http://proceedings.mlr.press/v80/haarnoja18a.html>.
- [90] Tuomas Haarnoja et al. “Learning to Walk via Deep Reinforcement Learning”. In: *CoRR* abs/1812.11103 (2018). arXiv: 1812.11103. URL: <http://arxiv.org/abs/1812.11103>.
- [91] Perttu Hämäläinen, Joose Rajamäki, and C Karen Liu. “Online control of simulated humanoids using particle belief propagation”. In: *ACM Transactions on Graphics (TOG)* 34.4 (2015), p. 81.
- [92] Josiah Hanna and Peter Stone. “Grounded Action Transformation for Robot Learning in Simulation”. In: *Proceedings of the 31st AAAI Conference on Artificial Intelligence (AAAI)*. San Francisco, CA, Feb. 2017.
- [93] T. Harada et al. “Quantitative evaluation method for pose and motion similarity based on human perception”. In: *4th IEEE/RAS International Conference on Humanoid Robots, 2004*. Vol. 1. 2004, 494–512 Vol. 1. DOI: 10.1109/ICHR.2004.1442140.
- [94] Matthew Hausknecht and Peter Stone. “Deep Reinforcement Learning in Parameterized Action Space”. In: *Proceedings of the International Conference on Learning Representations (ICLR)*. San Juan, Puerto Rico, May 2016. URL: <http://www.cs.utexas.edu/users/ai-lab/?hausknecht:iclr16>.
- [95] Karol Hausman et al. “Learning an Embedding Space for Transferable Robot Skills”. In: *International Conference on Learning Representations*. 2018. URL: <https://openreview.net/forum?id=rk07ZXZRb>.
- [96] G Clark Haynes and Alfred A Rizzi. “Gaits and gait transitions for legged robots”. In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2006, pp. 1117–1122.
- [97] Kaiming He et al. “Deep Residual Learning for Image Recognition”. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 770–778. DOI: 10.1109/CVPR.2016.90.

- [98] Zhanpeng He et al. “Zero-Shot Skill Composition and Simulation-to-Real Transfer by Learning Task Representations”. In: *CoRR* abs/1810.02422 (2018). arXiv: 1810.02422. URL: <http://arxiv.org/abs/1810.02422>.
- [99] Nicolas Heess et al. “Emergence of Locomotion Behaviours in Rich Environments”. In: *CoRR* abs/1707.02286 (2017). arXiv: 1707.02286.
- [100] Nicolas Heess et al. “Learning and Transfer of Modulated Locomotor Controllers”. In: *CoRR* abs/1610.05182 (2016). arXiv: 1610.05182.
- [101] Nicolas Heess et al. “Memory-based control with recurrent neural networks”. In: *CoRR* abs/1512.04455 (2015). URL: <http://arxiv.org/abs/1512.04455>.
- [102] A. Hereid et al. “Rapid Trajectory optimization Using C-FROST with Illustration on a Cassie-Series Dynamic Walking Biped”. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2019, pp. 4722–4729. DOI: 10.1109/IRoS40897.2019.8968056.
- [103] G. E. Hinton and R. R. Salakhutdinov. “Reducing the Dimensionality of Data with Neural Networks”. In: *Science* 313.5786 (2006), pp. 504–507. ISSN: 0036-8075. DOI: 10.1126/science.1127647. eprint: <http://science.sciencemag.org/content/313/5786/504.full.pdf>. URL: <http://science.sciencemag.org/content/313/5786/504>.
- [104] Jonathan Ho and Stefano Ermon. “Generative Adversarial Imitation Learning”. In: *Advances in Neural Information Processing Systems 29*. Curran Associates, Inc., 2016, pp. 4565–4573.
- [105] Jessica K. Hodgins et al. “Animating human athletics”. In: *Proceedings of the 22nd Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH 1995, Los Angeles, CA, USA, August 6-11, 1995*. Ed. by Susan G. Mair and Robert Cook. ACM, 1995, pp. 71–78. DOI: 10.1145/218380.218414. URL: <https://doi.org/10.1145/218380.218414>.
- [106] Daniel Holden, Taku Komura, and Jun Saito. “Phase-Functioned Neural Networks for Character Control”. In: *ACM Trans. Graph.* 36.4 (July 2017). ISSN: 0730-0301. DOI: 10.1145/3072959.3073663. URL: <https://doi.org/10.1145/3072959.3073663>.
- [107] Daniel Holden, Taku Komura, and Jun Saito. “Phase-functioned Neural Networks for Character Control”. In: *ACM Trans. Graph.* 36.4 (July 2017), 42:1–42:13. ISSN: 0730-0301.
- [108] Daniel Holden, Jun Saito, and Taku Komura. “A Deep Learning Framework for Character Motion Synthesis and Editing”. In: *ACM Trans. Graph.* 35.4 (July 2016), 138:1–138:11. ISSN: 0730-0301.
- [109] Rein Houthoofd et al. “Curiosity-driven Exploration in Deep Reinforcement Learning via Bayesian Neural Networks”. In: *CoRR* abs/1605.09674 (2016). arXiv: 1605.09674.

- [110] Marco Hutter et al. “ANYmal - a highly mobile and dynamic quadrupedal robot”. In: *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (2016), pp. 38–44.
- [111] Jemin Hwangbo et al. “Learning agile and dynamic motor skills for legged robots”. In: *Science Robotics* 4.26 (2019). DOI: 10.1126/scirobotics.aau5872. eprint: <https://robotics.sciencemag.org/content/4/26/eaau5872.full.pdf>. URL: <https://robotics.sciencemag.org/content/4/26/eaau5872>.
- [112] David Held Ignasi Clavera and Pieter Abbeel. “Policy Transfer via Modularity”. In: *IROS*. IEEE, 2017.
- [113] Catalin Ionescu et al. “Human3.6M: Large Scale Datasets and Predictive Methods for 3D Human Sensing in Natural Environments”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence, TPAMI* 36.7 (2014), pp. 1325–1339.
- [114] Robert A. Jacobs et al. “Adaptive Mixtures of Local Experts”. In: *Neural Comput.* 3.1 (Mar. 1991), pp. 79–87. ISSN: 0899-7667. DOI: 10.1162/neco.1991.3.1.79. URL: <http://dx.doi.org/10.1162/neco.1991.3.1.79>.
- [115] Stephen James and Edward Johns. “3D Simulation for Robot Arm Control with Deep Q-Learning”. In: *CoRR* abs/1609.03759 (2016). URL: <http://arxiv.org/abs/1609.03759>.
- [116] Yifeng Jiang et al. “Synthesis of Biologically Realistic Human Motion Using Joint Torque Actuation”. In: *ACM Trans. Graph.* 38.4 (July 2019). ISSN: 0730-0301. DOI: 10.1145/3306346.3322966. URL: <https://doi.org/10.1145/3306346.3322966>.
- [117] Tobias Johannink et al. “Residual reinforcement learning for robot control”. In: *International Conference on Robotics and Automation*. 2019, pp. 6023–6029.
- [118] Sam Johnson and Mark Everingham. “Clustered Pose and Nonlinear Appearance Models for Human Pose Estimation”. In: *Proceedings of the British Machine Vision Conference, BMVC*. 2010, pp. 12.1–12.11.
- [119] Angjoo Kanazawa et al. “End-to-end Recovery of Human Shape and Pose”. In: *Computer Vision and Pattern Recognition (CVPR)*. 2018.
- [120] Tero Karras, Samuli Laine, and Timo Aila. “A Style-Based Generator Architecture for Generative Adversarial Networks”. In: *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2019, pp. 4396–4405. DOI: 10.1109/CVPR.2019.00453.
- [121] Tero Karras et al. “Progressive Growing of GANs for Improved Quality, Stability, and Variation”. In: *CoRR* abs/1710.10196 (2017). arXiv: 1710.10196. URL: <http://arxiv.org/abs/1710.10196>.
- [122] Liyiming Ke et al. “Imitation Learning as f-Divergence Minimization”. In: *CoRR* abs/1905.12888 (2019). arXiv: 1905.12888. URL: <http://arxiv.org/abs/1905.12888>.

- [123] S. Kim et al. “Stable whole-body motion generation for humanoid robots to imitate human motions”. In: *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*. Oct. 2009, pp. 2518–2524. DOI: 10.1109/IR0S.2009.5354271.
- [124] Diederik P. Kingma and Jimmy Ba. “Adam: A Method for Stochastic Optimization”. In: *CoRR* abs/1412.6980 (2014). URL: <http://arxiv.org/abs/1412.6980>.
- [125] Diederik P. Kingma and Max Welling. “Auto-Encoding Variational Bayes.” In: *CoRR* abs/1312.6114 (2013). URL: <http://dblp.uni-trier.de/db/journals/corr/corr1312.html#KingmaW13>.
- [126] James Kirkpatrick et al. “Overcoming catastrophic forgetting in neural networks”. In: 114.13 (2017), pp. 3521–3526. DOI: 10.1073/pnas.1611835114.
- [127] Naveen Kodali et al. “How to Train Your DRAGAN”. In: *CoRR* abs/1705.07215 (2017). arXiv: 1705.07215. URL: <http://arxiv.org/abs/1705.07215>.
- [128] Jonas Koenemann, Felix Burget, and Maren Bennewitz. “Real-time imitation of human whole-body motions by humanoids”. In: *2014 IEEE International Conference on Robotics and Automation (ICRA)* (2014), pp. 2806–2812.
- [129] Nate Kohl and Peter Stone. “Policy Gradient Reinforcement Learning for Fast Quadrupedal Locomotion.” In: *ICRA*. IEEE, Nov. 2, 2007, pp. 2619–2624. URL: <http://dblp.uni-trier.de/db/conf/icra/icra2004-3.html#KohlS04>.
- [130] J. Zico Kolter and Andrew Y. Ng. “Learning omnidirectional path following using dimensionality reduction”. In: *in Proceedings of Robotics: Science and Systems*. 2007.
- [131] Twan Koolen et al. “Capturability-based analysis and control of legged locomotion, Part 1: Theory and application to three simple gait models”. In: *The international journal of robotics research* 31.9 (2012), pp. 1094–1113.
- [132] Twan Koolen et al. “Summary of team IHMC’s virtual robotics challenge entry”. In: *2013 13th IEEE-RAS International Conference on Humanoid Robots (Humanoids)*. IEEE. 2013, pp. 307–314.
- [133] Ilya Kostrikov, Ofir Nachum, and Jonathan Tompson. “Imitation Learning via Off-Policy Distribution Matching”. In: *International Conference on Learning Representations*. 2020. URL: <https://openreview.net/forum?id=Hyg-JC4FDr>.
- [134] Sanjay Krishnan et al. “DDCO: Discovery of Deep Continuous Options for Robot Learning from Demonstrations”. In: *CoRR* abs/1710.05421 (2017). arXiv: 1710.05421. URL: <http://arxiv.org/abs/1710.05421>.
- [135] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. “ImageNet Classification with Deep Convolutional Neural Networks”. In: *Advances in Neural Information Processing Systems*. Ed. by F. Pereira et al. Vol. 25. Curran Associates, Inc., 2012. URL: <https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf>.

- [136] Scott Kuindersma, Frank Permenter, and Russ Tedrake. “An efficiently solvable quadratic program for stabilizing dynamic locomotion”. In: *IEEE International Conference on Robotics and Automation*. 2014, pp. 2589–2594.
- [137] Taesoo Kwon and Jessica K. Hodgins. “Momentum-Mapped Inverted Pendulum Models for Controlling Dynamic Human Motions”. In: *ACM Trans. Graph.* 36.4 (Jan. 2017). ISSN: 0730-0301. DOI: 10.1145/3072959.2983616. URL: <https://doi.org/10.1145/3072959.2983616>.
- [138] Martin de Lasa, Igor Mordatch, and Aaron Hertzmann. “Feature-Based Locomotion Controllers”. In: *ACM Transactions on Graphics* 29.3 (2010).
- [139] H. Lee and Z. Chen. “Determination of 3D human body postures from a single view”. In: *Computer Vision Graphics and Image Processing* 30.2 (1985), pp. 148–168.
- [140] Jehee Lee et al. “Interactive Control of Avatars Animated with Human Motion Data”. In: *ACM Trans. Graph.* 21.3 (July 2002), pp. 491–500. ISSN: 0730-0301. DOI: 10.1145/566654.566607. URL: <https://doi.org/10.1145/566654.566607>.
- [141] Joonho Lee et al. “Learning quadrupedal locomotion over challenging terrain”. In: *Science Robotics* 5.47 (2020).
- [142] Kyungho Lee, Seyoung Lee, and Jehee Lee. “Interactive Character Animation by Learning Multi-Objective Control”. In: *ACM Trans. Graph.* 37.6 (Dec. 2018). ISSN: 0730-0301. DOI: 10.1145/3272127.3275071. URL: <https://doi.org/10.1145/3272127.3275071>.
- [143] Seunghwan Lee et al. “Scalable Muscle-Actuated Human Simulation and Control”. In: *ACM Trans. Graph.* 38.4 (July 2019). ISSN: 0730-0301. DOI: 10.1145/3306346.3322972. URL: <https://doi.org/10.1145/3306346.3322972>.
- [144] Yongjoon Lee et al. “Motion Fields for Interactive Character Locomotion”. In: *ACM SIGGRAPH Asia 2010 Papers*. SIGGRAPH ASIA ’10. Seoul, South Korea: ACM, 2010, 138:1–138:8. ISBN: 978-1-4503-0439-9.
- [145] Yoonsang Lee, Sungeun Kim, and Jehee Lee. “Data-Driven Biped Control”. In: *ACM Trans. Graph.* 29.4 (July 2010). ISSN: 0730-0301. DOI: 10.1145/1778765.1781155. URL: <https://doi.org/10.1145/1778765.1781155>.
- [146] Yoonsang Lee et al. “Locomotion Control for Many-muscle Humanoids”. In: *ACM Trans. Graph.* 33.6 (Nov. 2014), 218:1–218:11. ISSN: 0730-0301.
- [147] Sergey Levine, Nolan Wagnener, and Pieter Abbeel. “Learning Contact-Rich Manipulation Skills with Guided Policy Search”. In: *CoRR* abs/1501.05611 (2015). URL: <http://arxiv.org/abs/1501.05611>.
- [148] Sergey Levine et al. “Continuous Character Control with Low-Dimensional Embeddings”. In: *ACM Transactions on Graphics* 31.4 (2012), p. 28.
- [149] Sergey Levine et al. “End-to-End Training of Deep Visuomotor Policies”. In: *CoRR* abs/1504.00702 (2015). arXiv: 1504.00702.

- [150] Sergey Levine et al. “Learning Hand-Eye Coordination for Robotic Grasping with Deep Learning and Large-Scale Data Collection”. In: *CoRR* abs/1603.02199 (2016). URL: <http://arxiv.org/abs/1603.02199>.
- [151] Sergey Levine et al. “Space-Time Planning with Parameterized Locomotion Controllers”. In: *ACM Trans. Graph.* 30.3 (May 2011). ISSN: 0730-0301. DOI: 10.1145/1966394.1966402. URL: <https://doi.org/10.1145/1966394.1966402>.
- [152] Tianyu Li et al. “Learning generalizable locomotion skills with hierarchical reinforcement learning”. In: *IEEE International Conference on Robotics and Automation*. 2020, pp. 413–419.
- [153] Zhongyu Li, Christine Cummings, and Koushil Sreenath. “Animated Cassie: A Dynamic Relatable Robotic Character”. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2020.
- [154] Zhongyu Li et al. “Reinforcement Learning for Robust Parameterized Locomotion Control of Bipedal Robots”. In: *2021 IEEE International Conference on Robotics and Automation (ICRA)*. 2021, pp. 1–7.
- [155] Timothy P. Lillicrap et al. “Continuous control with deep reinforcement learning.” In: *ICLR*. Ed. by Yoshua Bengio and Yann LeCun. 2016. URL: <http://dblp.uni-trier.de/db/conf/iclr/iclr2016.html#LillicrapHPHETS15>.
- [156] Tsung-Yi Lin et al. “Microsoft COCO: Common Objects in Context”. In: *European Conference on Computer Vision (ECCV)*. Zürich, Jan. 1, 2014. URL: /se3/wp-content/uploads/2014/09/coco_eccv.pdf,%20http://mscoco.org.
- [157] Hung Yu Ling et al. “Character Controllers Using Motion VAEs”. In: *ACM Trans. Graph.* 39.4 (July 2020). ISSN: 0730-0301. DOI: 10.1145/3386569.3392422. URL: <https://doi.org/10.1145/3386569.3392422>.
- [158] Libin Liu and Jessica Hodgins. “Learning Basketball Dribbling Skills Using Trajectory Optimization and Deep Reinforcement Learning”. In: *ACM Trans. Graph.* 37.4 (July 2018). ISSN: 0730-0301. DOI: 10.1145/3197517.3201315. URL: <https://doi.org/10.1145/3197517.3201315>.
- [159] Libin Liu and Jessica Hodgins. “Learning to Schedule Control Fragments for Physics-Based Characters Using Deep Q-Learning”. In: *ACM Trans. Graph.* 36.3 (June 2017), 29:1–29:14. ISSN: 0730-0301.
- [160] Libin Liu, Michiel van de Panne, and KangKang Yin. “Guided Learning of Control Graphs for Physics-Based Characters”. In: *ACM Transactions on Graphics* 35.3 (2016).
- [161] Libin Liu et al. “Sampling-based Contact-rich Motion Control”. In: *ACM Transactions on Graphics* 29.4 (2010), Article 128.

- [162] Libin Liu et al. “Terrain runner: control, parameterization, composition, and planning for highly dynamic motions”. In: *ACM Transactions on Graphics (TOG)* 31.6 (2012), p. 154.
- [163] Matthew Loper et al. “SMPL: A Skinned Multi-Person Linear Model”. In: *ACM Trans. Graphics (Proc. SIGGRAPH Asia)* 34.6 (Oct. 2015), 248:1–248:16.
- [164] Ryan Lowe et al. “Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments”. In: *CoRR* abs/1706.02275 (2017). URL: <http://arxiv.org/abs/1706.02275>.
- [165] Kendall Lowrey et al. “Reinforcement learning for non-prehensile manipulation: Transfer from simulation to physical system”. In: *CoRR* abs/1803.10371 (2018). arXiv: 1803.10371. URL: <http://arxiv.org/abs/1803.10371>.
- [166] Ying-Sheng Luo et al. “CARL: Controllable Agent with Reinforcement Learning for Quadruped Locomotion”. In: *ACM Trans. Graph.* 39.4 (July 2020). ISSN: 0730-0301. DOI: 10.1145/3386569.3392433. URL: <https://doi.org/10.1145/3386569.3392433>.
- [167] Corey Lynch et al. “Learning Latent Plans from Play”. In: *Proceedings of the Conference on Robot Learning*. Ed. by Leslie Pack Kaelbling, Danica Kragic, and Komei Sugiura. Vol. 100. Proceedings of Machine Learning Research. PMLR, 30 Oct–01 Nov 2020, pp. 1113–1132. URL: <http://proceedings.mlr.press/v100/lynch20a.html>.
- [168] Kevin M. Lynch and Matthew T. Mason. “Stable Pushing: Mechanics, Controllability, and Planning”. In: *The International Journal of Robotics Research* 15.6 (1996), pp. 533–556. DOI: 10.1177/027836499601500602.
- [169] Viktor Makoviychuk et al. “Isaac Gym: High Performance GPU-Based Physics Simulation For Robot Learning”. In: *CoRR* abs/2108.10470 (2021). arXiv: 2108.10470. URL: <https://arxiv.org/abs/2108.10470>.
- [170] X. Mao et al. “Least Squares Generative Adversarial Networks”. In: *2017 IEEE International Conference on Computer Vision (ICCV)*. 2017, pp. 2813–2821. DOI: 10.1109/ICCV.2017.304.
- [171] Martin Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: <http://tensorflow.org/>.
- [172] Tad McGeer. “Passive bipedal running”. In: *Proceedings of the Royal Society of London. B. Biological Sciences* 240 (1990), pp. 107–134.
- [173] Dushyant Mehta et al. “VNect: Real-time 3D Human Pose Estimation with a Single RGB Camera”. In: *ACM Transactions on Graphics (TOG) - Proceedings of ACM SIGGRAPH* 36 (July 2017). URL: <http://gvv.mpi-inf.mpg.de/projects/VNect/>.

- [174] Josh Merel et al. “Catch and Carry: Reusable Neural Controllers for Vision-Guided Whole-Body Tasks”. In: *ACM Trans. Graph.* 39.4 (July 2020). ISSN: 0730-0301. DOI: 10.1145/3386569.3392474. URL: <https://doi.org/10.1145/3386569.3392474>.
- [175] Josh Merel et al. “Hierarchical Visuomotor Control of Humanoids”. In: *International Conference on Learning Representations*. 2019. URL: <https://openreview.net/forum?id=BJfYvo09Y7>.
- [176] Josh Merel et al. “Learning human behaviors from motion capture by adversarial imitation”. In: *CoRR* abs/1707.02201 (2017). arXiv: 1707.02201.
- [177] Josh Merel et al. “Neural Probabilistic Motor Primitives for Humanoid Control”. In: *International Conference on Learning Representations*. 2019. URL: <https://openreview.net/forum?id=BJl6TjRcY7>.
- [178] Lars Mescheder, Andreas Geiger, and Sebastian Nowozin. “Which Training Methods for GANs do actually Converge?” In: *Proceedings of the 35th International Conference on Machine Learning*. Ed. by Jennifer Dy and Andreas Krause. Vol. 80. Proceedings of Machine Learning Research. Stockholmsmässan, Stockholm Sweden: PMLR, Oct. 2018, pp. 3481–3490. URL: <http://proceedings.mlr.press/v80/mescheder18a.html>.
- [179] Hirofumi Miura and Isao Shimoyama. “Dynamic Walk of a Biped”. In: *The International Journal of Robotics Research* 3 (1984), pp. 60–74.
- [180] Volodymyr Mnih et al. “Human-level control through deep reinforcement learning”. In: *Nature* 518.7540 (Feb. 2015), pp. 529–533. ISSN: 00280836.
- [181] Igor Mordatch, Kendall Lowrey, and Emanuel Todorov. “Ensemble-CIO: Full-body dynamic motion planning that transfers to physical humanoids”. In: *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2015, Hamburg, Germany, September 28 - October 2, 2015*. 2015, pp. 5307–5314. DOI: 10.1109/IROS.2015.7354126. URL: <https://doi.org/10.1109/IROS.2015.7354126>.
- [182] Igor Mordatch, Emanuel Todorov, and Zoran Popović. “Discovery of Complex Behaviors through Contact-Invariant Optimization”. In: *ACM Trans. Graph.* 31.4 (July 2012). ISSN: 0730-0301. DOI: 10.1145/2185520.2185539. URL: <https://doi.org/10.1145/2185520.2185539>.
- [183] Jun Morimoto and Christopher G Atkeson. “Learning biped locomotion”. In: *IEEE Robotics & Automation Magazine* 14.2 (2007), pp. 41–51.
- [184] Uldarico Muico et al. “Contact-Aware Nonlinear Control of Dynamic Characters”. In: *ACM Trans. Graph.* 28.3 (July 2009). ISSN: 0730-0301. DOI: 10.1145/1531326.1531387. URL: <https://doi.org/10.1145/1531326.1531387>.
- [185] Uldarico Muico et al. “Contact-aware nonlinear control of dynamic characters”. In: *ACM Transactions on Graphics (TOG)*. Vol. 28. 3. ACM. 2009, p. 81.

- [186] Ashvin Nair et al. “Overcoming Exploration in Reinforcement Learning with Demonstrations”. In: *CoRR* abs/1709.10089 (2017). arXiv: 1709.10089.
- [187] Vinod Nair and Geoffrey E. Hinton. “Rectified Linear Units Improve Restricted Boltzmann Machines”. In: *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*. Ed. by Johannes Fürnkranz and Thorsten Joachims. Omnipress, 2010, pp. 807–814.
- [188] S. Nakaoka et al. “Generating whole body motions for a biped humanoid robot from captured human dances”. In: *2003 IEEE International Conference on Robotics and Automation (Cat. No.03CH37422)*. Vol. 3. Sept. 2003, 3905–3910 vol.3. DOI: 10.1109/ROBOT.2003.1242196.
- [189] Gerhard Neumann, Wolfgang Maass, and Jan Peters. “Learning Complex Motions by Sequencing Simpler Motion Templates”. In: *Proceedings of the 26th Annual International Conference on Machine Learning. ICML '09*. Montreal, Quebec, Canada: ACM, 2009, pp. 753–760. ISBN: 978-1-60558-516-1. DOI: 10.1145/1553374.1553471. URL: <http://doi.acm.org/10.1145/1553374.1553471>.
- [190] Gerhard Neumann and Jan R. Peters. “Fitted Q-iteration by Advantage Weighted Regression”. In: *Advances in Neural Information Processing Systems 21*. Ed. by D. Koller et al. Curran Associates, Inc., 2009, pp. 1177–1184. URL: <http://papers.nips.cc/paper/3501-fitted-q-iteration-by-advantage-weighted-regression.pdf>.
- [191] Alejandro Newell, Kaiyu Yang, and Jia Deng. “Stacked hourglass networks for human pose estimation”. In: *European Conference on Computer Vision, ECCV*. 2016, pp. 483–499.
- [192] Quan Nguyen et al. “Dynamic Walking on Randomly-Varying Discrete Terrain with One-step Preview”. In: *Robotics: Science and Systems*. 2017.
- [193] Sebastian Nowozin, Botond Cseke, and Ryota Tomioka. “f-GAN: Training Generative Neural Samplers using Variational Divergence Minimization”. In: *Advances in Neural Information Processing Systems*. Ed. by D. Lee et al. Vol. 29. Curran Associates, Inc., 2016, pp. 271–279. URL: <https://proceedings.neurips.cc/paper/2016/file/cedebb6e872f539bef8c3f919874e9d7-Paper.pdf>.
- [194] OpenAI et al. “Learning Dexterous In-Hand Manipulation”. In: *CoRR* abs/1808.00177 (2018). arXiv: 1808.00177. URL: <http://arxiv.org/abs/1808.00177>.
- [195] OpenAI et al. “Learning Dexterous In-Hand Manipulation”. In: *CoRR* abs/1808.00177 (2018). arXiv: 1808.00177. URL: <http://arxiv.org/abs/1808.00177>.
- [196] Michiel van de Panne, Ryan Kim, and Eugene Flume. “Virtual Wind-up Toys for Animation”. In: *Proceedings of Graphics Interface '94*. 1994, pp. 208–215.
- [197] Alexandros Paraschos et al. “Probabilistic Movement Primitives”. In: *Advances in Neural Information Processing Systems 26*. Ed. by C. J. C. Burges et al. Curran Associates, Inc., 2013, pp. 2616–2624. URL: <http://papers.nips.cc/paper/5177-probabilistic-movement-primitives.pdf>.

- [198] Soohwan Park et al. “Learning Predict-and-Simulate Policies from Unorganized Human Motion Data”. In: *ACM Trans. Graph.* 38.6 (Nov. 2019). ISSN: 0730-0301. DOI: 10.1145/3355089.3356501. URL: <https://doi.org/10.1145/3355089.3356501>.
- [199] Georgios Pavlakos et al. “Coarse-to-Fine Volumetric Prediction for Single-Image 3D Human Pose”. In: *IEEE Conference on Computer Vision and Pattern Recognition, CVPR*. 2017.
- [200] X. B. Peng et al. “Sim-to-Real Transfer of Robotic Control with Dynamics Randomization”. In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. May 2018, pp. 1–8. DOI: 10.1109/ICRA.2018.8460528.
- [201] Xue Bin Peng, Glen Berseth, and Michiel van de Panne. “Dynamic Terrain Traversal Skills Using Reinforcement Learning”. In: *ACM Trans. Graph.* 34.4 (July 2015), 80:1–80:11. ISSN: 0730-0301.
- [202] Xue Bin Peng, Glen Berseth, and Michiel van de Panne. “Terrain-Adaptive Locomotion Skills Using Deep Reinforcement Learning”. In: *ACM Transactions on Graphics (Proc. SIGGRAPH 2016)* 35.4 (2016).
- [203] Xue Bin Peng, Michiel van de Panne, and KangKang Yin. “Learning Locomotion Skills Using DeepRL: Does the Choice of Action Space Matter?” In: *Proc. ACM SIGGRAPH / Eurographics Symposium on Computer Animation*. 2017.
- [204] Xue Bin Peng et al. “Advantage-Weighted Regression: Simple and Scalable Off-Policy Reinforcement Learning”. In: *CoRR* abs/1910.00177 (2019). arXiv: 1910.00177. URL: <https://arxiv.org/abs/1910.00177>.
- [205] Xue Bin Peng et al. “AMP: Adversarial Motion Priors for Stylized Physics-Based Character Control”. In: *ACM Trans. Graph.* 40.4 (July 2021). DOI: 10.1145/3450626.3459670. URL: <http://doi.acm.org/10.1145/3450626.3459670>.
- [206] Xue Bin Peng et al. “DeepLoco: Dynamic Locomotion Skills Using Hierarchical Deep Reinforcement Learning”. In: *ACM Transactions on Graphics (Proc. SIGGRAPH 2017)* 36.4 (2017).
- [207] Xue Bin Peng et al. “DeepMimic: Example-Guided Deep Reinforcement Learning of Physics-Based Character Skills”. In: *ACM Transactions on Graphics (Proc. SIGGRAPH 2018 - to appear)* 37.4 (2018).
- [208] Xue Bin Peng et al. “Learning Agile Robotic Locomotion Skills by Imitating Animals”. In: *Robotics: Science and Systems*. July 2020. DOI: 10.15607/RSS.2020.XVI.064.
- [209] Xue Bin Peng et al. “MCP: Learning Composable Hierarchical Control with Multiplicative Compositional Policies”. In: *Advances in Neural Information Processing Systems 32*. Ed. by H. Wallach et al. Curran Associates, Inc., 2019, pp. 3681–3692. URL: <http://papers.nips.cc/paper/8626-mcp-learning-composable-hierarchical-control-with-multiplicative-compositional-policies.pdf>.

- [210] Xue Bin Peng et al. “SFV: Reinforcement Learning of Physical Skills from Videos”. In: *ACM Trans. Graph.* 37.6 (Nov. 2018).
- [211] Xue Bin Peng et al. “Variational Discriminator Bottleneck: Improving Imitation Learning, Inverse RL, and GANs by Constraining Information Flow”. In: *International Conference on Learning Representations*. 2019. URL: <https://openreview.net/forum?id=HyxPx3R9tm>.
- [212] Lerrel Pinto et al. “Robust Adversarial Reinforcement Learning”. In: *Proceedings of the 34th International Conference on Machine Learning*. Ed. by Doina Precup and Yee Whye Teh. Vol. 70. Proceedings of Machine Learning Research. International Convention Centre, Sydney, Australia: PMLR, June 2017, pp. 2817–2826. URL: <http://proceedings.mlr.press/v70/pinto17a.html>.
- [213] Nancy Pollard et al. “Adapting Human Motion for the Control of a Humanoid Robot”. In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA '02)*. May 2002.
- [214] Dean A. Pomerleau. “ALVINN: An Autonomous Land Vehicle in a Neural Network”. In: *Proceedings of the 1st International Conference on Neural Information Processing Systems*. NIPS'88. Cambridge, MA, USA: MIT Press, 1988, pp. 305–313.
- [215] Jerry Pratt et al. “Capturability-based analysis and control of legged locomotion, Part 2: Application to M2V2, a lower-body humanoid”. In: *The international journal of robotics research* 31.10 (2012), pp. 1117–1133.
- [216] Alec Radford, Luke Metz, and Soumith Chintala. “Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks”. In: *CoRR* abs/1511.06434 (2015). arXiv: 1511.06434. URL: <http://arxiv.org/abs/1511.06434>.
- [217] M. H. Raibert. “Hopping in legged systems — Modeling and simulation for the two-dimensional one-legged case”. In: *IEEE Transactions on Systems, Man, and Cybernetics* SMC-14.3 (May 1984), pp. 451–463. ISSN: 2168-2909. DOI: 10.1109/TSMC.1984.6313238.
- [218] Marc H Raibert. “Trotting, pacing and bounding by a quadruped robot”. In: *Journal of biomechanics* 23 (1990), pp. 79–98.
- [219] Marc H. Raibert, Jr H. Benjamin Brown, and Michael Chepponis. “Experiments in Balance with a 3D One-Legged Hopping Machine”. In: *The International Journal of Robotics Research* 3.2 (1984), pp. 75–92. DOI: 10.1177/027836498400300207. eprint: <https://doi.org/10.1177/027836498400300207>. URL: <https://doi.org/10.1177/027836498400300207>.
- [220] Marc H. Raibert and Jessica K. Hodgins. “Animation of Dynamic Legged Locomotion”. In: *Proceedings of the 18th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '91. New York, NY, USA: Association for Computing Machinery, 1991, pp. 349–358. ISBN: 0897914368. DOI: 10.1145/122718.122755. URL: <https://doi.org/10.1145/122718.122755>.

- [221] Marc H. Raibert and Jessica K. Hodgins. “Animation of Dynamic Legged Locomotion”. In: *SIGGRAPH Comput. Graph.* 25.4 (July 1991), pp. 349–358. ISSN: 0097-8930. DOI: 10.1145/127719.122755. URL: <https://doi.org/10.1145/127719.122755>.
- [222] Aravind Rajeswaran et al. “EPOpt: Learning Robust Neural Network Policies Using Model Ensembles”. In: *CoRR* abs/1610.01283 (2016). arXiv: 1610.01283.
- [223] Aravind Rajeswaran et al. “Learning Complex Dexterous Manipulation with Deep Reinforcement Learning and Demonstrations”. In: *CoRR* abs/1709.10087 (2017). arXiv: 1709.10087.
- [224] Nathan Ratliff, J Andrew Bagnell, and Siddhartha S Srinivasa. “Imitation learning for locomotion and manipulation”. In: *IEEE-RAS International Conference on Humanoid Robots*. 2007, pp. 392–397.
- [225] Nathan D. Ratliff, Jan Issac, and Daniel Kappler. “Riemannian Motion Policies”. In: *CoRR* abs/1801.02854 (2018). arXiv: 1801.02854. URL: <http://arxiv.org/abs/1801.02854>.
- [226] Gregory Rogez and Cordelia Schmid. “MoCap-guided Data Augmentation for 3D Pose Estimation in the Wild”. In: *Advances in Neural Information Processing Systems, (NIPS)*. 2016.
- [227] Stephane Ross, Geoffrey Gordon, and Drew Bagnell. “A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning”. In: ed. by Geoffrey Gordon, David Dunson, and Miroslav Dudík. Vol. 15. Proceedings of Machine Learning Research. Fort Lauderdale, FL, USA: JMLR Workshop and Conference Proceedings, Nov. 2011, pp. 627–635. URL: <http://proceedings.mlr.press/v15/ross11a.html>.
- [228] Olga Russakovsky et al. “ImageNet Large Scale Visual Recognition Challenge”. In: *International Journal of Computer Vision (IJCV)* 115.3 (2015), pp. 211–252. DOI: 10.1007/s11263-015-0816-y.
- [229] Andrei A. Rusu et al. “Progressive Neural Networks”. In: *CoRR* abs/1606.04671 (2016). arXiv: 1606.04671. URL: <http://arxiv.org/abs/1606.04671>.
- [230] Andrei A. Rusu et al. “Sim-to-Real Robot Learning from Pixels with Progressive Nets”. In: *Proceedings of the 1st Annual Conference on Robot Learning*. Ed. by Sergey Levine, Vincent Vanhoucke, and Ken Goldberg. Vol. 78. Proceedings of Machine Learning Research. PMLR, 13–15 Nov 2017, pp. 262–270. URL: <http://proceedings.mlr.press/v78/rusu17a.html>.
- [231] Andrei A. Rusu et al. “Sim-to-Real Robot Learning from Pixels with Progressive Nets”. In: *Proceedings of the 1st Annual Conference on Robot Learning*. Ed. by Sergey Levine, Vincent Vanhoucke, and Ken Goldberg. Vol. 78. Proceedings of Machine Learning Research. PMLR, 13–15 Nov 2017, pp. 262–270. URL: <http://proceedings.mlr.press/v78/rusu17a.html>.

- [232] Fereshteh Sadeghi and Sergey Levine. “(CAD) 2RL : RealSingle–ImageFlightwithoutaSingleRealImage”. In: *CoRR* abs/1611.04201 (2016). arXiv: 1611.04201.
- [233] Alla Safonova and Jessica K Hodgins. “Construction and optimal search of interpolated motion graphs”. In: *ACM Transactions on Graphics (TOG)*. Vol. 26. 3. ACM. 2007, p. 106.
- [234] H. Sakoe and S. Chiba. “Dynamic programming algorithm optimization for spoken word recognition”. In: *IEEE Transactions on Acoustics, Speech, and Signal Processing* 26.1 (1978), pp. 43–49. DOI: 10.1109/TASSP.1978.1163055.
- [235] Tim Salimans et al. “Improved Techniques for Training GANs”. In: *CoRR* abs/1606.03498 (2016). arXiv: 1606.03498. URL: <http://arxiv.org/abs/1606.03498>.
- [236] Claude Sammut et al. “Learning to Fly”. In: *Proceedings of the Ninth International Workshop on Machine Learning (ML 1992), Aberdeen, Scotland, UK, July 1-3, 1992*. Ed. by Derek H. Sleeman and Peter Edwards. Morgan Kaufmann, 1992, pp. 385–393. DOI: 10.1016/b978-1-55860-247-2.50055-3. URL: <https://doi.org/10.1016/b978-1-55860-247-2.50055-3>.
- [237] John Schulman et al. “High-Dimensional Continuous Control Using Generalized Advantage Estimation”. In: *CoRR* abs/1506.02438 (2016).
- [238] John Schulman et al. “Proximal Policy Optimization Algorithms”. In: *CoRR* abs/1707.06347 (2017). arXiv: 1707.06347.
- [239] John Schulman et al. “Trust Region Policy Optimization”. In: *CoRR* abs/1502.05477 (2015). arXiv: 1502.05477.
- [240] William J. Schwind and Daniel E. Koditschek. “Spring loaded inverted pendulum running: a plant model”. In: 1998.
- [241] Pierre Sermanet et al. “Time-Contrastive Networks: Self-Supervised Learning from Multi-View Observation”. In: *CoRR* abs/1704.06888 (2017). arXiv: 1704.06888. URL: <http://arxiv.org/abs/1704.06888>.
- [242] SFU. *SFU Motion Capture Database*. <http://mocap.cs.sfu.ca/>.
- [243] Dana Sharon and Michiel van de Panne. “Synthesis of Controllers for Stylized Planar Bipedal Walking”. In: *Proceedings of the 2005 IEEE International Conference on Robotics and Automation* (2005), pp. 2387–2392.
- [244] Dana Sharon and Michiel van de Panne. “Synthesis of Controllers for Stylized Planar Bipedal Walking”. In: *Proc. of IEEE International Conference on Robotics and Animation*. 2005.
- [245] Jonah Siekmann et al. “Learning Memory-Based Control for Human-Scale Bipedal Locomotion”. In: *arXiv preprint arXiv:2006.02402* (2020).

- [246] Marco da Silva, Yeuhi Abe, and Jovan Popović. “Interactive Simulation of Stylized Human Locomotion”. In: *ACM SIGGRAPH 2008 Papers*. SIGGRAPH ’08. Los Angeles, California: ACM, 2008, 82:1–82:10. ISBN: 978-1-4503-0112-1. DOI: 10.1145/1399504.1360681. URL: <http://doi.acm.org/10.1145/1399504.1360681>.
- [247] David Silver et al. “Mastering the Game of Go with Deep Neural Networks and Tree Search”. In: *Nature* 529.7587 (Jan. 2016), pp. 484–489. DOI: 10.1038/nature16961.
- [248] David Silver et al. “Mastering the game of Go without human knowledge”. In: *Nature* 550 (Oct. 2017), pp. 354–. URL: <http://dx.doi.org/10.1038/nature24270>.
- [249] Kwang Won Sok, Manmyung Kim, and Jeehe Lee. “Simulating biped behaviors from human motion data”. In: *ACM Transactions on Graphics (TOG)*. Vol. 26. 3. ACM. 2007, p. 107.
- [250] Koushil Sreenath et al. “A Compliant Hybrid Zero Dynamics Controller for Stable, Efficient and Fast Bipedal Walking on MABEL”. In: *Int. J. Rob. Res.* 30.9 (Aug. 2011), pp. 1170–1193. ISSN: 0278-3649. DOI: 10.1177/0278364910379882. URL: <https://doi.org/10.1177/0278364910379882>.
- [251] Sebastian Starke et al. “Neural State Machine for Character-Scene Interactions”. In: *ACM Trans. Graph.* 38.6 (Nov. 2019). ISSN: 0730-0301. DOI: 10.1145/3355089.3356505. URL: <https://doi.org/10.1145/3355089.3356505>.
- [252] W. Suleiman et al. “On human motion imitation by humanoid robot”. In: *2008 IEEE International Conference on Robotics and Automation*. May 2008, pp. 2697–2704. DOI: 10.1109/ROBOT.2008.4543619.
- [253] R. Sutton et al. *Policy Gradient Methods for Reinforcement Learning with Function Approximation*. 2001.
- [254] Richard S. Sutton and Andrew G. Barto. *Introduction to Reinforcement Learning*. 1st. Cambridge, MA, USA: MIT Press, 1998. ISBN: 0262193981.
- [255] Richard S. Sutton, Doina Precup, and Satinder Singh. “Between MDPs and semi-MDPs: A Framework for Temporal Abstraction in Reinforcement Learning”. In: *Artif. Intell.* 112.1-2 (Aug. 1999), pp. 181–211. ISSN: 0004-3702. DOI: 10.1016/S0004-3702(99)00052-1. URL: [http://dx.doi.org/10.1016/S0004-3702\(99\)00052-1](http://dx.doi.org/10.1016/S0004-3702(99)00052-1).
- [256] J. Tan et al. “Simulation-based design of dynamic controllers for humanoid balancing”. In: *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Oct. 2016, pp. 2729–2736. DOI: 10.1109/IROS.2016.7759424.
- [257] Jie Tan, Karen Liu, and Greg Turk. “Stable Proportional-Derivative Controllers”. In: *IEEE Comput. Graph. Appl.* 31.4 (2011), pp. 34–44. ISSN: 0272-1716.
- [258] Jie Tan et al. “Learning Bicycle Stunts”. In: *ACM Trans. Graph.* 33.4 (July 2014). ISSN: 0730-0301. DOI: 10.1145/2601097.2601121. URL: <https://doi.org/10.1145/2601097.2601121>.

- [259] Jie Tan et al. “Sim-to-Real: Learning Agile Locomotion For Quadruped Robots”. In: *Proceedings of Robotics: Science and Systems*. Pittsburgh, Pennsylvania, June 2018. DOI: 10.15607/RSS.2018.XIV.010.
- [260] Jeff Tang et al. “Emulating human perception of motion similarity”. In: *Computer Animation and Virtual Worlds* 19 (Aug. 2008), pp. 211–221. DOI: 10.1002/cav.260.
- [261] Yuval Tassa, Tom Erez, and Emanuel Todorov. “Synthesis and stabilization of complex behaviors through online trajectory optimization”. In: *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*. IEEE. 2012, pp. 4906–4913.
- [262] Yuval Tassa et al. “DeepMind Control Suite”. In: *CoRR* abs/1801.00690 (2018). arXiv: 1801.00690. URL: <http://arxiv.org/abs/1801.00690>.
- [263] C. Taylor. “Reconstruction of articulated objects from point correspondences in single uncalibrated image”. In: *Computer Vision and Image Understanding, CVIU* 80.10 (2000), pp. 349–363.
- [264] Russ Tedrake, Teresa Weirui Zhang, and H. Sebastian Seung. “Stochastic policy gradient reinforcement learning on a simple 3D biped”. In: *Proceedings of the 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2004)*. Vol. 3. Piscataway, NJ, USA: IEEE, 2004, pp. 2849–2854. ISBN: 0-7803-8463-6. URL: <http://www.cs.cmu.edu/~cga/legs/01389841.pdf>.
- [265] Yee Whye Teh et al. “Distal: Robust Multitask Reinforcement Learning”. In: *CoRR* abs/1707.04175 (2017). arXiv: 1707.04175.
- [266] Bugra Tekin et al. “Direct Prediction of 3D Body Poses from Motion Compensated Sequences”. In: *IEEE Conference on Computer Vision and Pattern Recognition, CVPR*. 2016, pp. 991–1000.
- [267] Chen Tessler et al. “A Deep Hierarchical Approach to Lifelong Learning in Minecraft”. In: *AAAI*. 2016.
- [268] Sebastian Thrun. “Is Learning The n-th Thing Any Easier Than Learning The First?”. In: *Advances in Neural Information Processing Systems*. The MIT Press, 1996, pp. 640–646.
- [269] Joshua Tobin et al. “Domain Randomization for Transferring Deep Neural Networks from Simulation to the Real World”. In: *CoRR* abs/1703.06907 (2017). URL: <http://arxiv.org/abs/1703.06907>.
- [270] Emanuel Todorov, Tom Erez, and Yuval Tassa. “MuJoCo: A physics engine for model-based control.” In: *IROS*. IEEE, 2012, pp. 5026–5033. ISBN: 978-1-4673-1737-5.
- [271] Jonathan J Tompson et al. “Joint training of a convolutional network and a graphical model for human pose estimation”. In: *Advances in neural information processing systems*. 2014, pp. 1799–1807.

- [272] Faraz Torabi, Garrett Warnell, and Peter Stone. “Generative Adversarial Imitation from Observation”. In: *CoRR* abs/1807.06158 (2018). arXiv: 1807.06158. URL: <http://arxiv.org/abs/1807.06158>.
- [273] Alexander Toshev and Christian Szegedy. “DeepPose: Human Pose Estimation via Deep Neural Networks”. In: *IEEE Conference on Computer Vision and Pattern Recognition, CVPR*. 2014, pp. 1653–1660.
- [274] Adrien Treuille, Yongjoon Lee, and Zoran Popović. “Near-Optimal Character Animation with Continuous Control”. In: *ACM SIGGRAPH 2007 Papers*. SIGGRAPH ’07. San Diego, California: Association for Computing Machinery, 2007, 7–es. ISBN: 9781450378369. DOI: 10.1145/1275808.1276386. URL: <https://doi.org/10.1145/1275808.1276386>.
- [275] Hsiao-Yu Tung et al. “Self-supervised Learning of Motion Capture”. In: *Advances in Neural Information Processing Systems*. 2017, pp. 5242–5252.
- [276] Eric Tzeng et al. “Adapting Deep Visuomotor Representations with Weak Pairwise Constraints”. In: *CoRR* abs/1511.07111 (2015). URL: <http://arxiv.org/abs/1511.07111>.
- [277] Alexander Sasha Vezhnevets et al. “FeUdal Networks for Hierarchical Reinforcement Learning”. In: *Proceedings of the 34th International Conference on Machine Learning - Volume 70*. ICML’17. Sydney, NSW, Australia: JMLR.org, 2017, pp. 3540–3549. URL: <http://dl.acm.org/citation.cfm?id=3305890.3306047>.
- [278] Marek Vondrak et al. “Video-based 3D Motion Capture Through Biped Control”. In: *ACM Trans. Graph.* 31.4 (July 2012), 27:1–27:12. ISSN: 0730-0301. DOI: 10.1145/2185520.2185523. URL: <http://doi.acm.org/10.1145/2185520.2185523>.
- [279] Miomir Vukobratovic and Branislav Borovac. “Zero-Moment Point - Thirty Five Years of its Life.” In: *Int. J. Humanoid Robotics* 1.1 (2004), pp. 157–173. URL: <http://dblp.uni-trier.de/db/journals/ijhr/ijhr1.html#VukobratovicB04>.
- [280] Miomir Vukobratovic and Branislav Borovac. “Zero-moment point—thirty five years of its life”. In: *International journal of humanoid robotics* 1.01 (2004), pp. 157–173.
- [281] Kevin Wampler, Zoran Popović, and Jovan Popović. “Generalizing Locomotion Style to New Animals with Inverse Optimal Regression”. In: *ACM Trans. Graph.* 33.4 (July 2014), 49:1–49:11. ISSN: 0730-0301.
- [282] Jack M. Wang, David J. Fleet, and Aaron Hertzmann. “Optimizing Walking Controllers”. In: *ACM SIGGRAPH Asia 2009 Papers*. SIGGRAPH Asia ’09. Yokohama, Japan: Association for Computing Machinery, 2009. ISBN: 9781605588582. DOI: 10.1145/1661412.1618514. URL: <https://doi.org/10.1145/1661412.1618514>.

- [283] Jack M. Wang, David J. Fleet, and Aaron Hertzmann. “Optimizing Walking Controllers for Uncertain Inputs and Environments”. In: *ACM SIGGRAPH 2010 Papers*. SIGGRAPH '10. Los Angeles, California: ACM, 2010, 73:1–73:8. ISBN: 978-1-4503-0210-4. DOI: 10.1145/1833349.1778810. URL: <http://doi.acm.org/10.1145/1833349.1778810>.
- [284] Jack M. Wang et al. “Optimizing locomotion controllers using biologically-based actuators and objectives”. In: *ACM Trans. Graph* (2012).
- [285] Tingwu Wang et al. *UniCon: Universal Neural Controller For Physics-based Character Motion*. 2020. arXiv: 2011.15119 [cs.GR].
- [286] Xingxing Wang. *Laikago Pro, Unitree Robotics*. 2018. URL: <http://www.unitree.cc/e/action/ShowInfo.php?classid=6&id=355>.
- [287] Ziyu Wang et al. “Robust Imitation of Diverse Behaviors”. In: *Advances in Neural Information Processing Systems*. Ed. by I. Guyon et al. Vol. 30. Curran Associates, Inc., 2017, pp. 5320–5329. URL: <https://proceedings.neurips.cc/paper/2017/file/044a23cadb567653eb51d4eb40acaa88-Paper.pdf>.
- [288] Shih-En Wei et al. “Convolutional Pose Machines”. In: *IEEE Conference on Computer Vision and Pattern Recognition, CVPR*. 2016, pp. 4724–4732.
- [289] Ronald J. Williams. “Simple statistical gradient-following algorithms for connectionist reinforcement learning”. In: *Machine Learning* 8.3 (May 1992), pp. 229–256. ISSN: 1573-0565.
- [290] Jungdam Won, Deepak Gopinath, and Jessica Hodgins. “A Scalable Approach to Control Diverse Behaviors for Physically Simulated Characters”. In: *ACM Trans. Graph*. 39.4 (July 2020). ISSN: 0730-0301. DOI: 10.1145/3386569.3392381. URL: <https://doi.org/10.1145/3386569.3392381>.
- [291] Jungdam Won, Deepak Gopinath, and Jessica Hodgins. “Control Strategies for Physically Simulated Characters Performing Two-Player Competitive Sports”. In: *ACM Trans. Graph*. 40.4 (July 2021). ISSN: 0730-0301. DOI: 10.1145/3450626.3459761. URL: <https://doi.org/10.1145/3450626.3459761>.
- [292] Jungdam Won and Jehee Lee. “Learning Body Shape Variation in Physics-Based Characters”. In: *ACM Trans. Graph*. 38.6 (Nov. 2019). ISSN: 0730-0301. DOI: 10.1145/3355089.3356499. URL: <https://doi.org/10.1145/3355089.3356499>.
- [293] Jungdam Won et al. “How to Train Your Dragon: Example-guided Control of Flapping Flight”. In: *ACM Trans. Graph*. 36.6 (Nov. 2017), 198:1–198:13. ISSN: 0730-0301.
- [294] Markus Wulfmeier, Ingmar Posner, and Pieter Abbeel. “Mutual Alignment Transfer Learning”. In: *CoRR* abs/1707.07907 (2017). URL: <http://arxiv.org/abs/1707.07907>.

- [295] Zhaoming Xie et al. “Feedback control for cassie with deep reinforcement learning”. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2018, pp. 1241–1246.
- [296] Zhaoming Xie et al. “Learning Locomotion Skills for Cassie: Iterative Design and Sim-to-Real”. In: *Proc. Conference on Robot Learning (CORL 2019)*. 2019.
- [297] Xiaobin Xiong and Aaron D Ames. “Coupling reduced order models via feedback control for 3d underactuated bipedal robotic walking”. In: *IEEE-RAS International Conference on Humanoid Robots*. 2018.
- [298] Weipeng Xu et al. “MonoPerfCap: Human Performance Capture from Monocular Video”. In: *ACM Transactions on Graphics (2018)*. URL: <http://gvv.mpi-inf.mpg.de/projects/wxu/MonoPerfCap>.
- [299] K. Yamane, S. O. Anderson, and J. K. Hodgins. “Controlling humanoid robots with human motion data: Experimental validation”. In: *2010 10th IEEE-RAS International Conference on Humanoid Robots*. Dec. 2010, pp. 504–510. DOI: 10.1109/ICHR.2010.5686312.
- [300] Yuting Ye and C Karen Liu. “Optimal feedback control for character animation using an abstract model”. In: *ACM Transactions on Graphics (TOG)*. Vol. 29. 4. ACM. 2010, p. 74.
- [301] Yuting Ye and C. Karen Liu. “Synthesis of Responsive Motion Using a Dynamic Model”. In: *Computer Graphics Forum 29.2 (2010)*, pp. 555–562. ISSN: 1467-8659.
- [302] KangKang Yin, Kevin Loken, and Michiel van de Panne. “SIMBICON: Simple Biped Locomotion Control”. In: *ACM Trans. Graph.* 26.3 (2007), Article 105.
- [303] Kuan-Ting Yu et al. “More than a Million Ways to Be Pushed: A High-Fidelity Experimental Data Set of Planar Pushing”. In: *CoRR abs/1604.04038 (2016)*. URL: <http://arxiv.org/abs/1604.04038>.
- [304] Ri Yu, Hwangpil Park, and Jehee Lee. “Figure Skating Simulation from Video”. In: *Computer Graphics Forum*. Vol. 38. 7. Wiley Online Library. 2019, pp. 225–234.
- [305] Tianhe Yu et al. “One-Shot Imitation from Observing Humans via Domain-Adaptive Meta-Learning”. In: *CoRR abs/1802.01557 (2018)*. arXiv: 1802.01557. URL: <http://arxiv.org/abs/1802.01557>.
- [306] Wenhao Yu, C. Karen Liu, and Greg Turk. “Policy Transfer with Strategy Optimization”. In: *International Conference on Learning Representations*. 2019. URL: <https://openreview.net/forum?id=H1g6osRcFQ>.
- [307] Wenhao Yu, C. Karen Liu, and Greg Turk. “Preparing for the Unknown: Learning a Universal Policy with Online System Identification”. In: *CoRR abs/1702.02453 (2017)*. URL: <http://arxiv.org/abs/1702.02453>.

- [308] Wenhao Yu, Greg Turk, and C. Karen Liu. “Learning Symmetric and Low-Energy Locomotion”. In: *ACM Trans. Graph.* 37.4 (July 2018). ISSN: 0730-0301. DOI: 10.1145/3197517.3201397. URL: <https://doi.org/10.1145/3197517.3201397>.
- [309] Wenhao Yu, Greg Turk, and C. Karen Liu. “Learning Symmetry and Low-energy Locomotion”. In: *CoRR* abs/1801.08093 (2018). arXiv: 1801.08093. URL: <http://arxiv.org/abs/1801.08093>.
- [310] Wenhao Yu et al. “Learning fast adaptation with meta strategy optimization”. In: *IEEE Robotics and Automation Letters* 5.2 (2020), pp. 2950–2957.
- [311] Wenhao Yu et al. “Sim-to-Real Transfer for Biped Locomotion”. In: *CoRR* abs/1903.01390 (2019). arXiv: 1903.01390. URL: <http://arxiv.org/abs/1903.01390>.
- [312] Y. Yuan et al. “SimPoE: Simulated Character Control for 3D Human Pose Estimation”. In: *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. Los Alamitos, CA, USA: IEEE Computer Society, June 2021, pp. 7155–7165. DOI: 10.1109/CVPR46437.2021.00708. URL: <https://doi.ieeecomputersociety.org/10.1109/CVPR46437.2021.00708>.
- [313] He Zhang et al. “Mode-Adaptive Neural Networks for Quadruped Motion Control”. In: *ACM Trans. Graph.* 37.4 (July 2018). ISSN: 0730-0301. DOI: 10.1145/3197517.3201366. URL: <https://doi.org/10.1145/3197517.3201366>.
- [314] He Zhang et al. “Mode-adaptive Neural Networks for Quadruped Motion Control”. In: *ACM Trans. Graph.* 37.4 (July 2018), 145:1–145:11. ISSN: 0730-0301. DOI: 10.1145/3197517.3201366. URL: <http://doi.acm.org/10.1145/3197517.3201366>.
- [315] X. Zhou et al. “Sparse Representation for 3D Shape Estimation: A Convex Relaxation Approach”. In: *IEEE Conference on Computer Vision and Pattern Recognition, CVPR*. 2015, pp. 4447–4455.
- [316] X. Zhou et al. “Sparseness Meets Deepness: 3D Human Pose Estimation from Monocular Video”. In: *IEEE Conference on Computer Vision and Pattern Recognition, CVPR*. 2016, pp. 4966–4975.
- [317] Xingyi Zhou et al. “Deep Kinematic Pose Regression”. In: *ECCV Workshop on Geometry Meets Deep Learning*. 2016, pp. 186–201.
- [318] Xingyi Zhou et al. “Weakly-supervised Transfer for 3D Human Pose Estimation in the Wild”. In: *IEEE International Conference on Computer Vision, ICCV*. 2017.
- [319] Brian D. Ziebart et al. “Maximum Entropy Inverse Reinforcement Learning”. In: *Proceedings of the 23rd National Conference on Artificial Intelligence - Volume 3*. AAAI’08. Chicago, Illinois: AAAI Press, 2008, pp. 1433–1438. ISBN: 9781577353683.

- [320] Victor Brian Zordan and Jessica K. Hodgins. “Motion Capture-Driven Simulations That Hit and React”. In: *Proceedings of the 2002 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. SCA '02. San Antonio, Texas: Association for Computing Machinery, 2002, pp. 89–96. ISBN: 1581135734. DOI: 10.1145/545261.545276. URL: <https://doi.org/10.1145/545261.545276>.