UNIVERSITY OF CALIFORNIA
Los Angeles

Core Training:
Learning Deep Neuromuscular Control of the Torso for Anthropomimetic Animation

A dissertation submitted in partial satisfaction
of the requirements for the degree
Doctor of Philosophy in Computer Science

by

Tao Zhou

2019

ABSTRACT OF THE DISSERTATION

Core Training:

Learning Deep Neuromuscular Control of the Torso for Anthropomimetic Animation

by

Tao Zhou

Doctor of Philosophy in Computer Science

University of California, Los Angeles, 2019

Professor Demetri Terzopoulos, Chair

Despite its importance, the core of the human body has to date received inadequate attention in the computer graphics literature. We tackle the challenge of biomechanically simulating and controlling the torso, including of course the spine, in its full musculoskeletal complexity, thus providing a whole-body biomechanical human model with a full set of articular degrees of freedom actuated by many hundreds of muscles embedded in a finite-element soft tissue simulation. Performing skillful (non-locomotive) motor tasks while bipedally balancing upright in gravity has never before been attempted with a musculoskeletal model of such realism and complexity. Our approach to tackling the challenge is machine learning, specifically deep learning. The neuromuscular motor control system of our virtual human comprises 12 trained deep neural networks (DNNs), including a core voluntary/reflex DNN pair devoted to innervating the 443 muscles of the torso. By synthesizing its own training data offline, our virtual human automatically learns efficient, online, active control of the core musculoskeletal complex as well as its proper coordination with the five extremities—the cervicocephalic, arm, and leg musculoskeletal complexes—in order to perform nontrivial motor tasks such as sitting and standing, doing calisthenics, stepping, and golf putting. Moreover, we equip our virtual human with a full sensorimotor control system, thus making it autonomous. Afforded suitable NN-based machine perception, our model can also visually analyze drawings and manually sketch similar drawings as it balances in an upright stance before a large touchscreen display.

The dissertation of Tao Zhou is approved.

Guy Van den Broeck

Song-Chun Zhu

Joseph M. Teran

Demetri Terzopoulos, Committee Chair

University of California, Los Angeles

2019

To my mother and father

TABLE OF CONTENTS

LIST OF FIGURES

ACKNOWLEDGMENTS

I have always loved writing acknowledgements because I am really thankful for many things in my life. Five years ago, I chanced upon an video on biomimetic human simulation from Professor Terzopoulos' group. I was deeply attracted to the fascinating work on neural network based neuromuscular and sensorimotor control. Later that day, I wrote to Demetri asking him if he would be willing to let me pursue a 2nd PhD degree in the UCLA Computer Science Department. It was one of the best things I have done in my life.

Pushing, apprenticing, nurturing and guiding ones mentees is a challenging and sometimes daunting job that Demetri makes look very easy. During the past years, I owe him so much for his help, and it might also have been an unique experience for him to mentor a student who already has a PhD degree. He provided his unconditional support whenever I needed it. Even with his tight schedule, he often spent hours discussing the progress of my research. This thesis would never have materialized without his guidance and support. I have learned a lot from Demetri and for that I will always be indebted.

I thank Professors Song-Chun Zhu, Joseph Teran, and Guy Van Den Broeck for serving on my thesis committee and offering me their advice on how to improve my dissertation.

I would also like to express my appreciation to my collaborators on this project. Dr. Masaki Nakada acted as my associate mentor on the biomechanical model project and always provided technical support and valuable advice thorough my PhD study. My project was follow-up work to his PhD research on neuromuscular and sensorimotor control with biomimetic perception, and his research set a high standard for my thesis. I express my appreciation to Professor Sung-Hee Lee and Dr. Weiguang Justin Si. The biomechanical human model and the simulation of soft tissues (which was further enhanced by Professor Eftychios Sifakis) that I have used in my research is also largely based on the work that they developed for their UCLA PhD theses. I must thank Alan Litteneker for his help converting from OpenSceneGraph to POV-Ray. Without him, the rendering in this thesis would have been mission impossible in such a short time period. Arjun Lakshmipathy also helped my paper submissions and I was inspired by his work on biomechanical eye modeling and simulation.

It has been ten years since I began my research on control, from robot control, to human motor control, and finally to virtual human control. I would like to express my deep appreciation to my previous PhD advisor at Penn State University, Professor Mark Latash, and to my MS advisor at Tongji University, Professor Zhuping Wang. I thank them immensely for all their valuable inputs to my career.

I am very fortunate to have been surrounded by wonderful labmates in the UCLA Computer Graphics and Vision Laboratory who helped me work through the PhD program. I would like to express my gratitude to Dr. Tomer Weiss, Dr. Chenfanfu Jiang, Dr. Sharath Gopal, Dr. Garett Ridge, Dr. Xiaowei Ding, Dr. Gergely Klar, Dr. Andre Pradhana, Abdullah-Al-Zubaer Imran, Ziran Lin, Yajun Shi, Yingyue Qiu, and Hao Ding.

Lastly, my parents spared no sacrifice to nurture me and avail me of the best possible education. I thank them for their support and unconditional love. I am deeply grateful to my family for making all of this possible.

VITA

2008	B.S. Electrical Engineering Tongji University Shanghai, China
2011	M.S. Control Theory and Control Engineering Tongji University Shanghai, China
2015	Ph.D. Kinesiology The Pennsylvania State University University Park, PA
2015–2016	Research Assistant Computer Graphics & Vision Laboratory University of California, Los Angeles Los Angeles, California
2016–2019	Teaching Assistant Computer Science Department University of California, Los Angeles Los Angeles, California
2017	Research Intern Adobe Research San Jose, CA
2018	Apply Scientist Intern A9, Amazon Palo Alto, CA

PUBLICATIONS

Nakada M, Lakshmipathy A, Ling X, Chen HL, Zhou T, Terzopoulos D. "Biomimetic eye modeling and deep neuromuscular oculomotor control." *ACM Transactions on Graphics*, **38**(6), November 2019, 221:1–14. *Proc. ACM SIGGRAPH ASIA 19 Conference*, Brisbane Australia, November 2019.

Zhou T, Fang C, Wang ZW, Yang JM, Kim B, Chen ZL, Brandt J, Terzopoulos D. "Learning to doodle with deep Q-networks and demonstrated strokes." In *Proc. British Machine Vision Conference (BMVC)*, Newcastle, England, September 2018, 13:1–13.

Nakada, M, Zhou T, Chen, H, Weiss, T, Terzopoulos, D. "Deep learning of biomimetic sensorimotor control for biomechanical human animation." *ACM Transactions on Graphics*, **37**(4), August 2018, 56:1–15. *Proc. ACM SIGGRAPH 18 Conference*, Vancouver, Canada, August 2018.

Chen MH, Zhou T, Zaniolo C. "Multi-graph affinity embeddings for multilingual knowledge graphs." In *Proc. 6th NIPS Workshop on Automated Knowledge Base Construction (AKBC)*, Long Beach, CA, December 2017.

Zhou T, Chen MH, Yu J, Terzopoulos D. "Attention-based natural language person retrieval." In *Proc. 3rd IEEE Workshop on Vision Meets Cognition: Functionality, Physics, Intentionality, and Causality (FPIC)*, Honolulu, HI, July 2017, 1–8.

Zhou T, Yu J. "Natural language person retrieval." (abstract) In *31st AAAI Conference on Artificial Intelligence (AAAI)*, San Fancisco, CA, February 2017.

Hu RH, Peng XC, Zhou T, Yu J, Skaff S, Darrell T, Saenko K. "Object detection and retrieval using natural language." (abstract and demo) In *14th European Conference on Computer Vision (ECCV)*, Amsterdam, The Netherlands, October 2016.

Zhou T, Falaki A, Latash ML. "Unintentional movements induced by sequential transient perturbations in a multi-joint positional task." *Human Movement Science*, **46**:1–9, 2016.

Qiao M, Zhou T, Latash ML. "Positional errors introduced by transient perturbations applied to a multi-joint limb." *Neuroscience Letters*, **595**:104–107, 2015.

Zhou T, Latash ML. "Unintentional changes in the apparent stiffness of the endpoint of a multi-joint limb." *Experimental Brain Research*, **233**(10):2989–3004, 2015.

Ambike S, Zhou T, Latash ML. "Moving a hand-held object: Reconstruction of referent coordinate and apparent stiffness trajectories." *Neuroscience*, **298**:336–356, 2015.

Zhou T, Zhang L, and Latash ML. "Intentional and unintentional multi-joint movements: Their nature and structure of variance." *Neuroscience*, **289**:181–193, 2015.

Zhou T, Zhang L, Latash ML. "Characteristics of unintentional movements by a multijoint effector." *Journal of Motor Behavior*, **47**(4):1–10, 2015.

Wang ZP, Zhou T, Mao Y, and Chen QJ. "Adaptive recurrent neural network control of uncertain constrained nonholonomic mobile manipulators." *International Journal of Systems Science*, **45**(2):133–144, 2014.

Zhou T, Solnik S, Wu YH, and Latash ML. "Unintentional movements produced by back-coupling between the actual and referent body configurations: Violations of equifinality in multi-joint positional tasks." *Experimental Brain Research*, **232**(12):3847–3859, 2014.

Falaki A, Zhou T, Towhidkhah F, and Latash ML. "Task-specific stability in muscle activation space during unintentional movements." *Experimental Brain Research*, **232**(11):3645–3658, 2014.

Zhou T, Solnik S, Wu YH, Latash ML. "Equifinality and its violations in a redundant system: control with referent configurations in a multi-joint positional task." *Motor Control*, **18**(4):405–424, 2014.

Zhou T, Wu YH, Bartsch A, Cuadra C, Zatsiorsky VM, and Latash ML. "Anticipatory synergy adjustments: preparing a quick action in an unknown direction." *Experimental Brain Research*, **226**(4):565–573, 2013.

Wang ZP, and Zhou T. Control of an uncertain nonholonomic mobile manipulator based on the Diagonal Recurrent Neural Network. *In: 2011 Chinese Control and Decision Conference (CCDC)*, p. 4044-4047.

Wang ZP, Zhou T, and Chen QJ. "Control of uncertain constrained nonholonomic mobile manipulator based on recurrent neural network." In *Proc. 8th World Congress on Intelligent Control and Automation (WCICA)*, 532–536, 2010.

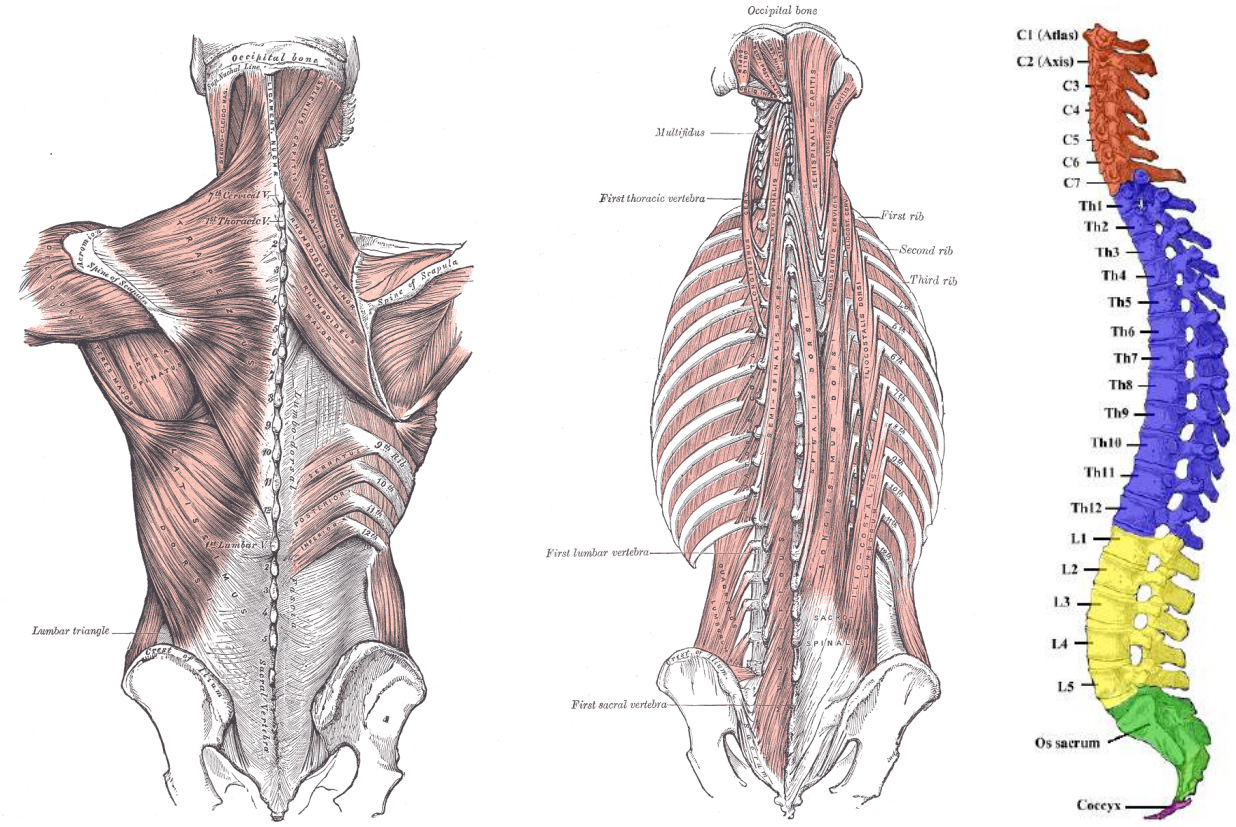
CHAPTER 1

Introduction

Anthropomimetic animation differs from conventional human animation in that it aims to achieve realism by taking advantage of increasingly accurate simulation of the anatomical structures of the human body, not only the bones, joints, and muscles, but also the human sensory organs and, of course, the brain. The torso, or trunk, is the anatomical term for the central part—in common speech, the core—of many animal bodies, excluding the extremities; i.e., the head and neck and limbs. Although it is indeed of core importance to the human body, the torso has to date received insufficient attention in the computer graphics literature. The goal of this thesis is to rectify this deficiency by taking an anthropomimetic approach.

Comprising the thorax or chest, abdomen, and pelvis, the torso complex houses most of the critical organs of the body as well as its major muscle groups (Figure 1.1a,b), including the pectoral muscles, abdominal muscles, lateral muscles, etc. The core muscles of the torso work together to help stabilize the body, transfer energy from the legs to the upper body, and transfer energy from the upper body to the legs. The vertebral column or spine, with its striking segmented structure (Figure 1.1c), is a very important skeletal component of the torso complex. Its thoracic and lumbar portions within the torso itself provide the main support for the human body, enabling standing, bending, twisting, and a variety of other whole-body motor actions. Furthermore, the majority of the spinal cord, the main pathway for sensorimotor information flow between the brain and the peripheral nervous system, branches out within the torso.

This disseration reports on an unprecedented attempt to tackle the challenge of biomechanically simulating and motor controlling the torso in its full musculoskeletal complexity, which includes more than 100 articular Degrees of Freedom (DoFs) and well over 400 muscle



(a) (b) (c)

Figure 1.1: (a) Superficial and intermediate muscles of the back. (b) Deep muscles of the back. (c) The spine, including the 7 cervical (C1–C7), 12 thoracic (T1–T12), and 5 lumbar (L1–L5) vertebra. (From Gray's Anatomy, 1918.)

actuators. Motor control is an area of biological science that explores how the nervous system exploits interactions between body parts and the environment to produce purposeful, coordinated actions that accomplish specific tasks. A central problem of motor control is that of dealing with abundant actuation and redundant DoFs (Latash, 2012), a challenge that is perhaps most acute in the context of the torso musculoskeletal complex. Numerous publications have provided support for the view that synergies as neural organizations ensure task-specific co-variation of elemental variables to provide the desired stability properties of motor actions while dealing with secondary tasks and unexpected perturbations.[1]

[1]Indeed, since the end of 19th century, researchers have agreed that the brain does not control muscles individually but unifies them into groups that are controlled in a synergistic manner (Jackson, 1889). The postural synergies for these simple daily movements may be considered as building blocks that the central nervous system uses to mitigate its computational burden (Scholz and Schöner, 1999).

1.1 Contributions

With biology serving as our guide, we take the natural, neuromuscular approach to addressing the tough challenge of controlling the torso as the human body's core synergistic subsystem. In particular, our work leverages modern artificial neural network and deep learning techniques. The specific contributions of our work are as follows:

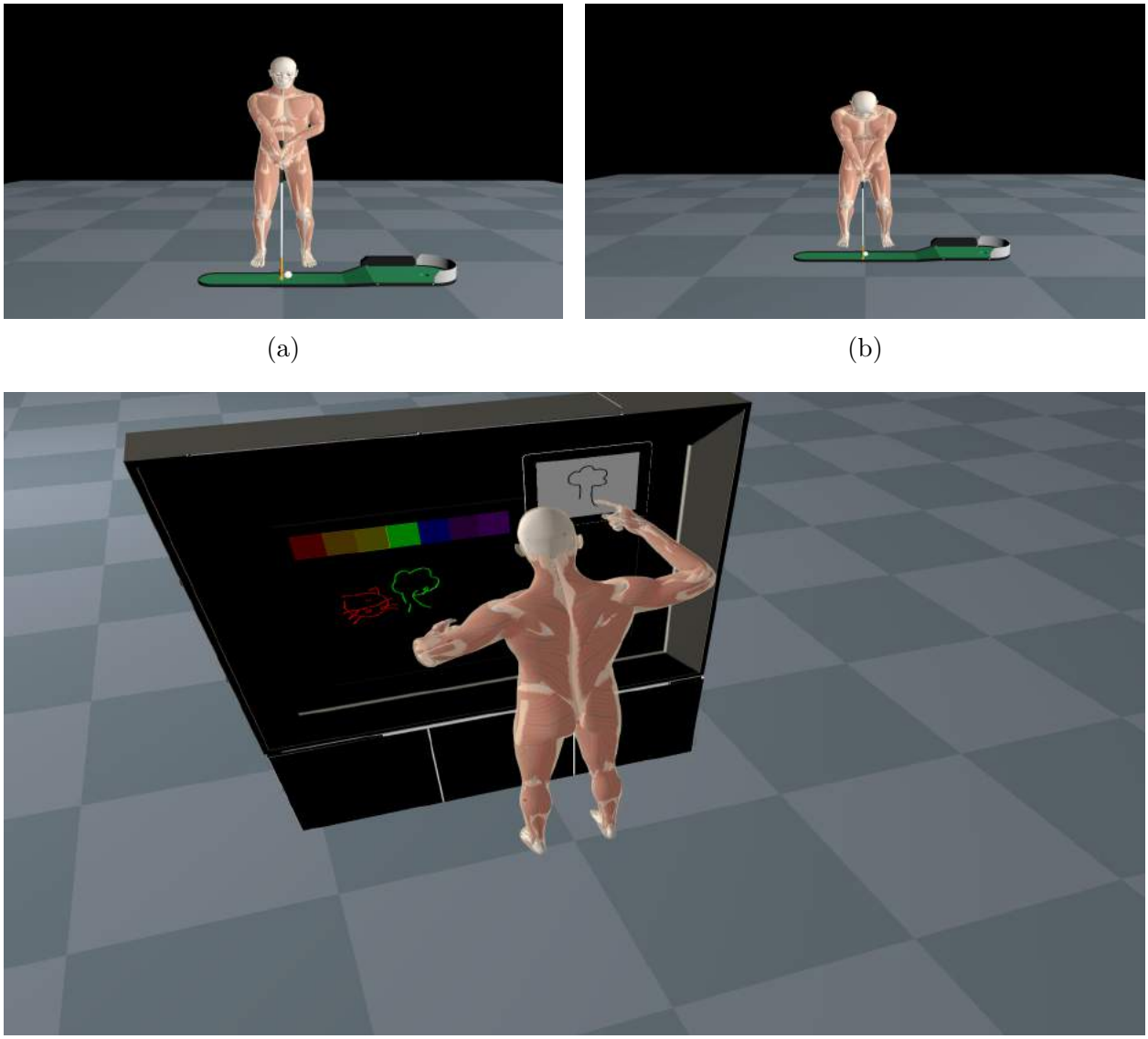
1. We develop the first neuromuscular motor control system for the spine and torso in the field of computer graphics and the most comprehensive and sophisticated one to date in any field.

2. We demonstrate that our control framework for the core musculoskeletal complex can work in concert with compatible neuromuscular controllers specialized to the five extremities—the cervicocephalic, two arm, and two leg musculoskeletal complexes.

3. We show how the six neuromuscular motor controllers, which include twelve Deep Neural Networks (DNNs), can form the motor subsystem of a whole-body sensorimotor control system.

4. We demonstrate and validate the robust online operation of our biomechanical virtual human's sensorimotor system in carrying out several skillful (non-locomotive) motor tasks, such as shown in Figure 1.2.

Of particular importance is the fact that our core neuromuscular motor controller is automatically trained offline on data synthesized by the biomechanical torso musculoskeletal model itself. Once trained, our controller works efficiently online to control the forward dynamics of our biomechanical virtual human. This is done without the online computation of inverse kinematics and inverse dynamics with muscle optimization as is typical for motor control schemes used in robotics.



(a) (b)

(c)

Figure 1.2: Our biomechanical human musculoskeletal model can perform various skillful motor control tasks while balancing its body in gravity, such as preparing to putt a golf ball (a) in a relatively upright stance with a "right hand low" grip on the golf club, and (b) in a crouched stance with a "prayer" grip. (c) Our virtual human performs an elaborate and prolonged sensorimotor control scenario, sketching on a large touchscreen display a set of reference drawings that it observes on a tablet. The body is rendered with translucent skin to reveal its internal simulated soft tissues.

1.2 Overview

The remainder of the dissertation is organized as follows:

Chapter 2 reviews relevant prior work.

Chapter 3 presents the multiphysics simulation framework of our biomechanical human musculoskeletal model.

Chapter 4 develops the neuromuscular motor control system for the musculoskeletal model.

Chapter 5 presents our experiments and results.

Chapter 6 applies our virtual human model to the challenging problem of sensorimotor control.

Chapter 7 concludes the dissertation with a summary of our work and suggestions for future work.

CHAPTER 2

Related Work

Our work builds upon techniques in the fields of computer graphics, biomechanics, robotics, machine learning, and neuroscience to model the anatomy and biomechanical functionality of the relevant musculoskeletal tissues of the human body and emulate the neuromuscular control abilities of the motor center of the brain.

2.1 Biomechanical Human Models

Anthropomorphic characters have been of central interest in computer animation. There exists a large body of work in human anatomical modeling, such as for the hand (Sueda et al., 2008; van Nierop et al., 2007; Tsang et al., 2005), torso (DiLorenzo et al., 2008; Zordan et al., 2004), face (Sifakis et al., 2005; Kähler et al., 2002; Lee et al., 1995), and neck (Lee and Terzopoulos, 2006). Also, various modeling schemes have been proposed regarding the modeling of muscle (Ng-Thow-Hing, 2001; Irving et al., 2004). Such earlier efforts applied various methods to the modeling of certain parts of the human body, but not to full-body modeling and control.

With regard to full-body models, Hodgins et al. (1995) and Faloutsos et al. (2001a,b) developed robotic joint-torque-driven articulated body control, which did not incorporate muscles into the control mechanism, but was nonetheless nontrivial to demonstrate at that time. Subsequently, Lee et al. (2009) developed a comprehensive upper body, muscle-actuated biomechanical model, and an extended, full-body version of this model was used by Si et al. (2014) to animate human swimming in simulated water using a neural central pattern generator (CPG) control system that synthesizes sustained rhythmic muscle activations suitable

for locomotion.

2.1.1 Modeling and Animating the Spine and Torso

Several computer graphics researchers have addressed the difficult problem of modeling and animating the human spine and torso in some or all of its complexity.

In their early 1990s pioneering work, Monheit and Badler (1991) introduced a purely kinematic model of the spine and torso based on the anatomy of the vertebra and the passive effects of the discs and surrounding soft tissues. They animated their spine model using the inverse kinematics technique. Shao and Ng-Thow-Hing (2003) proposed the use of more sophisticated joint models in the spine and shoulder, including non-orthogonal, non-intersecting axes of rotation and changing joint centers that are often found in the kinematics of biological joints (see also (Lee and Terzopoulos, 2008)).

By the mid to late 1990s the community had progressed to physics-based animation of articulated anthropomorphic models using joint-torque actuation control (Hodgins et al., 1995; Faloutsos et al., 2001a). However, the difficulties of this method at the time made it necessary to reduce the complexity of spine submodels to just a few rotational joints, which in terms of articulatory realism may be regarded as a step backward relative to the earlier kinematic model.

Roughly contemporaneously, researchers were experimenting with more realistic, muscle-based models. Wilhelms and Van Gelder (1997) and Scheepers et al. (1997) developed procedural muscle models that did not drive the anthropomorphic model, but instead deformed kinematically in response to kinematic skeletal articulation, which added realism to human character animation through keyframing or motion capture. Porcher Nedel and Thalmann (2000) took the additional step of using physically-based deformable muscle models, but again with a simplified kinematic spine.

Anatomically and biomechanically accurate musculoskeletal simulation is now superseding these earlier models. In the context of the torso, Zordan et al. (2004) introduced the first biomechanical model of the torso, which was designed to simulate breathing, including

a simplified spine model and a relevant subset of the torso muscles. Lee and Terzopoulos (2006) introduced the first biomechanical cervicocephalic musculoskeletal model with full anatomical accuracy in the cervical column. Extending this work, Lee et al. (2009) introduced a comprehensive biomechanical model of the upper body, including the arms, torso with full spine, plus a finite element flesh model, and Si et al. (2014) employed a whole-body version of this model. We use an improved version of the same whole-body model in our work.

Also loosely relevant to the scope of our work is that of Lee et al. (2018) and Saito et al. (2015).

2.2 Neuromuscular Motor Control of Biomechanical Models

Cruz Ruiz et al. (2017) present a comprehensive survey on muscle-based control for character animation. Our work falls into the category of neuromuscular motor control of realistic musculoskeletal models.

The full body model of Lee et al. (2009) comprises 103 bones with 163 DoFs and 823 muscles embedded in a finite-element flesh simulation. Hence the challenge originates from the complexity of the modeling of anatomically accurate human skeleton, joints, muscles, flesh, and skin models. The authors animated their model using biologically-implausible inverse dynamics techniques. By contrast, our objective is to emulate lifelike forward-dynamics-based motor control. That is, our motor controllers must provide appropriate muscle innervations at every simulation time step so as to actuate the complex anthropomorphic anatomical system to carry out nontrivial motor tasks in a fully autonomous manner.

Following the pioneering work of Grzeszczuk et al. (1998) on exploiting artificial neural networks and backpropagation learning in computer graphics, Lee and Terzopoulos (2006) were the first to apply them as neuromuscular motor controllers, specifically in their biomechanical model of the human cervicocephalic system. Unlike traditional inverse dynamics control, which is both unnatural and computationally expensive, learning neuromuscular motor controllers is a promising, biomimetic approach to tackling high-dimensional, mus-

culoskeletal motor control problems. To control the neck-head system in gravity, the work employed a two-layer, fully-connected artificial neural network trained offline through back-propagation learning. The training data comprised tens of thousands of random target head orientations as network inputs and, as associated outputs, the corresponding muscle activation levels that achieve these target orientations. Their neck-head biomechanical model was actuated by 72 muscles, so the trained regression network with adequate generalization continuously mapped 3 inputs (a target skull orientation) to 72 outputs (the muscle activations required to achieve this orientation). The trained network served as a neuromuscular controller, which was capable of efficiently controlling the biomechanical model online, thereby achieving a nearly real-time neuromuscular neck-head motor control system. Thus, Lee and Terzopoulos (2006) demonstrated the potential of biomimetic neuromuscular learning approaches to addressing the high-dimensional problems of muscle-actuated biomechanical motor control.

Shallow neural networks with only one or two hidden layers are not capable of dealing with the higher dimensionality and much greater quantities of training data required to train a biomechanical model controller of an order of magnitude greater complexity, such as the one presented by Lee et al. (2009). Indeed, Lee and Terzopoulos (2006) did not attempt to generalize their neck-head controller to deal with non-horizontally-oriented shoulders, which requires significantly larger quantities of training data. Thus, their trained neuromuscular controller fails to maintain satisfactory control when the shoulders tilt more than a certain amount. Lee et al. (2009) accomplished upper body control by solving inverse kinematics, then inverse dynamics and finally static muscle optimization problems, which requires a biologically infeasible, complete specification of desired movements. They could not achieve biomimetic, forward-dynamic neuromuscular control with neural networks due to the greater DoF and actuator complexity of the upper-body musculoskeletal system.

To tackle a more challenging generalization of the cervicocephalic control problem, Nakada and Terzopoulos (2015) were the first to apply deep learning techniques (Goodfellow et al., 2016). Their system showed that it is possible to train a deeper architecture on greater quantities of training data by applying a unsupervised pre-training phase to initialize the

parameters of a stacked denoising autoencoder prior to a regular back-propagation based fine-tuning process. The work demonstrated the viability of overcoming the limitations of shallow networks by successfully applying deeper network architectures, and it motivated the pursuit of full-body musculoskeletal motor control through the use of deep learning.

The work reported in this dissertation was inspired by that of Nakada et al. (2018), who developed a biomimetic sensorimotor system for a comprehensive, anatomically-accurate, muscle-actuated biomechanical human model. However, by immobilizing the pelvis as well as the lumbar and thoracic spinal column vertebra and other bones of the torso, leaving free to articulate only the extremities—the cervical column and four limbs—they skirted the challenge that this thesis confronts. A key contribution of our work is a previously intractable neuromuscular motor controller that not only can handle the full musculoskeletal complexity of the spine and torso, but also couple the cervicocephalic and four limb neuromuscular complexes from the preliminary sensorimotor model presented in (Nakada et al., 2018) with a common unrestricted core, thus enabling our simulated human to stand erect, balance, step, and perform skillful upper-body motor tasks.

CHAPTER 3

Simulation Framework

In this chapter, we present our simulation framework including the details of our comprehensive, whole-body biomechanical virtual human model.

3.1 Overview

Our simulation framework for human neuromuscular and sensorimotor control comprises two coupled component simulators—an articulated multibody dynamics simulator for the muscle-actuated skeleton and a deformable body dynamics simulator for the flesh. Figure 3.1 shows frontal and dorsal musculoskeletal, flesh, and opaque-skinned views of our biomechanical human model.

The musculoskeletal system of our anatomically accurate biomechanical virtual human (Figure 3.1a,b) includes all of the relevant articular bones and muscles—103 bones (hand and foot bones included) plus a total of 823 muscle actuators—comprising 163 articular DoFs. Each skeletal muscle is modeled as one or more Hill-type uniaxial contractile actuator that applies forces to the bones at its points of insertion and attachment.[1]

For added realism, our virtual human includes a finite element flesh model (Figure 3.1c,d), with the contractile actuators embedded in the flesh. The passive flesh simulation is accomplished by deforming a lattice-based discretization of quasi-incompressible elastic material augmented with active muscle terms from the embedded actuators. This approach avoids

[1] Our musculoskeletal model is a highly enhanced version of the preliminary model of Nakada et al. (2018). The arms include 29 muscles each, the legs include 39 muscles each, and the cervicocephalic complex includes 244 muscles (compared to the 216 in *ibid.*). Thus our model has 823 active muscles compared to the 352 active muscles in the earlier model.

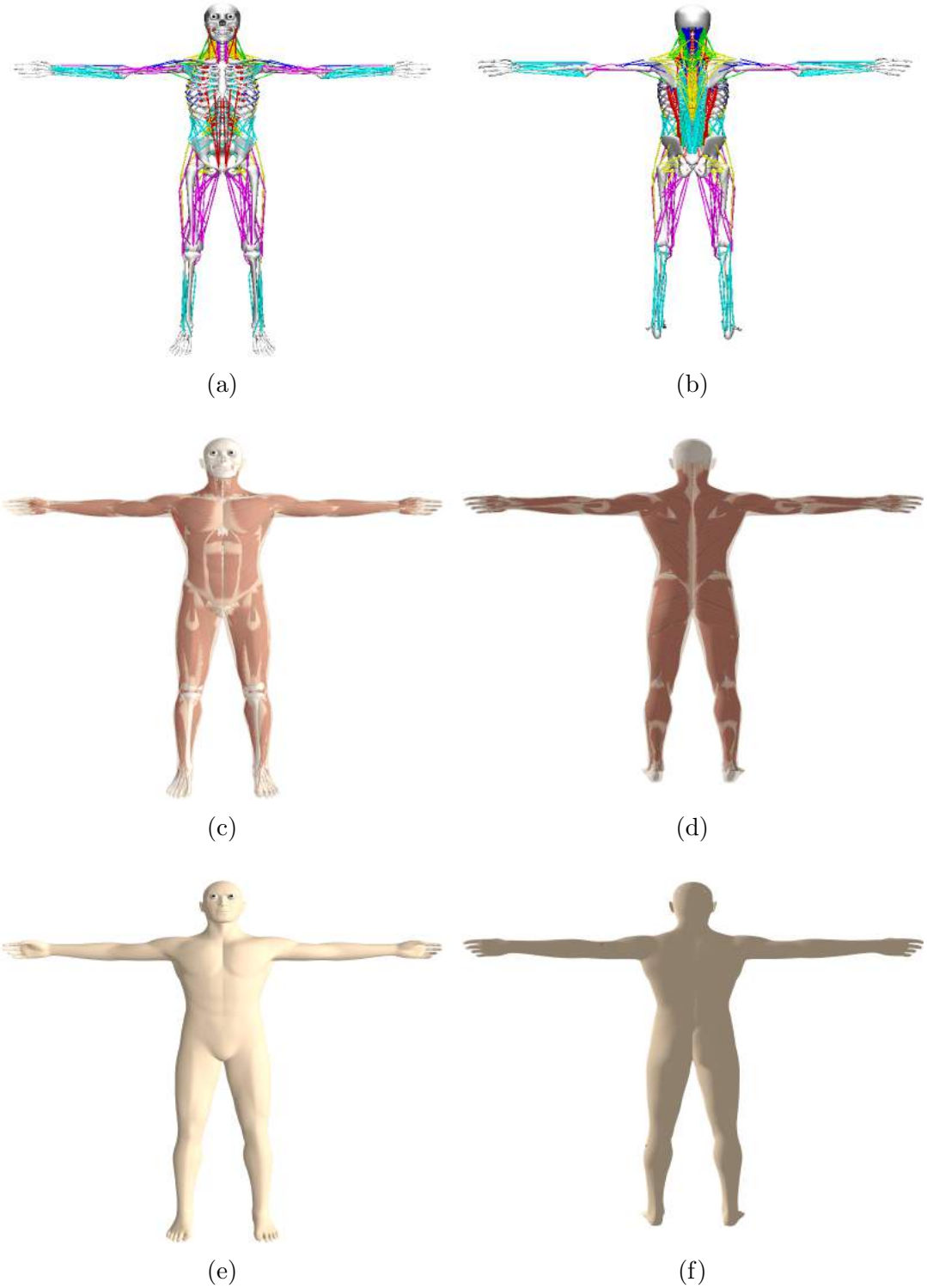


Figure 3.1: (a)–(b) The biomechanical human musculoskeletal model, showing the skeletal system with its 193 bones and 823 Hill-type muscle actuators surrounded by the 3D muscle geometries. (c)–(d) The biomechanical model showing its deformable finite element flesh submodel. (e)–(f) The biomechanical model with opaque skin.

the need for multiple meshes conforming to individual muscles and its regular structure offers significant opportunities for performance optimization. The inertial properties of the musculoskeletal system are approximated from the dense volumetric physical parameters of the soft-tissue elements—each bone's inertial tensor is augmented by the inertial parameters of its associated soft tissues.

The natural dynamics of the simulated human are induced by muscle forces generated by the contractile actuators. The low-level motor control inputs comprise the activation level of each muscle. The activated muscles generate forces that drive the skeletal simulation. Given the contractile muscle forces and the external forces, the skeleton is simulated using Featherstone's method (Featherstone, 2014) to compute the forward dynamics in conjunction with a backward Euler time-integration scheme as in (Lee et al., 2009).

The following two sections present the details of the musculoskeletal and flesh simulations, borrowing heavily from (Lee et al., 2009).

3.2 Musculoskeletal Simulation

3.2.1 Skeletal System

The equations of motion of the skeletal system are written as

$$\boldsymbol{M}(\boldsymbol{q}) \begin{bmatrix} \ddot{\boldsymbol{q}}_m \\ \ddot{\boldsymbol{q}}_p \end{bmatrix} + \boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}}) = \begin{bmatrix} \boldsymbol{P}(\boldsymbol{q})\boldsymbol{f}_c \\ \boldsymbol{0} \end{bmatrix} + \boldsymbol{J}^T \boldsymbol{f}_e, \tag{3.1}$$

where $\dot{\boldsymbol{q}}$ and $\ddot{\boldsymbol{q}}$ are the joint velocities and accelerations, $\boldsymbol{M}$ is the mass matrix, $\boldsymbol{C}$ accounts for forces such as the force from connecting tissues and muscle parallel elements $\boldsymbol{f}_p$ as well as Coriolis forces and centrifugal forces, $\boldsymbol{J}$ is the Jacobian matrix that transforms applied external forces to joint torques $\boldsymbol{\tau}$. The moment arm matrix $\boldsymbol{P}$ maps the contractile muscle forces $\boldsymbol{f}_c$ to the space of joint torques. The computation technique for the moment arm matrix is introduced in (Gonzalez et al., 1997).

Equation (3.1) can be compactly written as

$$\ddot{\boldsymbol{q}} = \phi(\boldsymbol{q}, \dot{\boldsymbol{q}}, \boldsymbol{\tau}). \tag{3.2}$$

We use forward dynamics to compute ϕ by computing the $\ddot{\boldsymbol{q}}$ from the generated torque produced by the muscle forces. Then, we use the implicit Euler time integration method to solve the linearized equations of motion. We can compute velocity at the next time step, where Δt is the step size, by solving

$$\dot{\boldsymbol{q}}(t + \Delta t) - \dot{\boldsymbol{q}}(t) = \Delta t \phi(\boldsymbol{q}(t + \Delta t), \dot{\boldsymbol{q}}(t + \Delta t), \tau). \tag{3.3}$$

The problem here is that this equation has the variable at the next time step on the right hand side of the equation. We use first-order approximation and rewrite the equation as follows:

$$\begin{aligned} \delta\dot{\boldsymbol{q}} &= \Delta t \left[\phi(\boldsymbol{q}(t), \dot{\boldsymbol{q}}(t), \tau) + \frac{\partial \phi}{\partial \boldsymbol{q}} \delta \boldsymbol{q} + \frac{\partial \phi}{\partial \dot{\boldsymbol{q}}} \delta \dot{\boldsymbol{q}} \right] \\ &= \Delta t \left[\phi(\boldsymbol{q}(t), \dot{\boldsymbol{q}}(t), \tau) + \frac{\partial \phi}{\partial \boldsymbol{q}} \Delta t (\dot{\boldsymbol{q}}(t) + \delta \dot{\boldsymbol{q}}) + \frac{\partial \phi}{\partial \dot{\boldsymbol{q}}} \delta \dot{\boldsymbol{q}} \right]. \end{aligned} \tag{3.4}$$

Thus, we can compute the joint velocities at the next time step. Then the joint angles at the next time step may be computed with an explicit Euler time integration.

3.2.2 Muscle System

We use the Hill-type muscle model (Lee and Terzopoulos, 2006; Lee et al., 2009). The muscle force $f_m = f_P + f_C$ is the combination of two components (Figure 3.2). The passive element f_P, which produces a restoring force due to the material elasticity to the deformation, is represented as a uniaxial exponential spring, as follows:

$$f_P = \max(0, k_s(\exp(k_c e) - 1) + k_d \dot{e}), \tag{3.5}$$

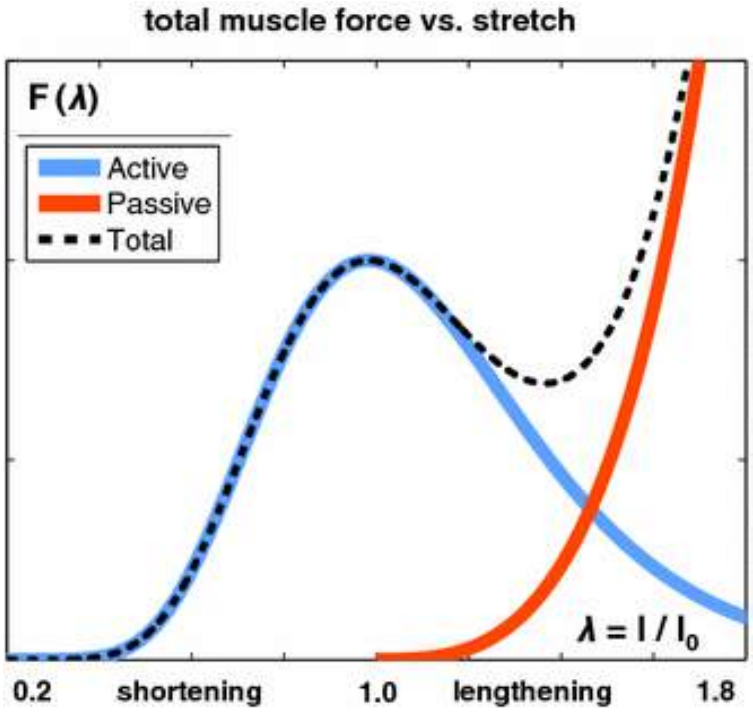


Figure 3.2: Total muscle force versus stretch ratio of the Hill-type muscle model. As the sum of the active and passive forces, the total muscle force F peaks then drops in accordance with the active force and then increases exponentially in accordance with the passive force.

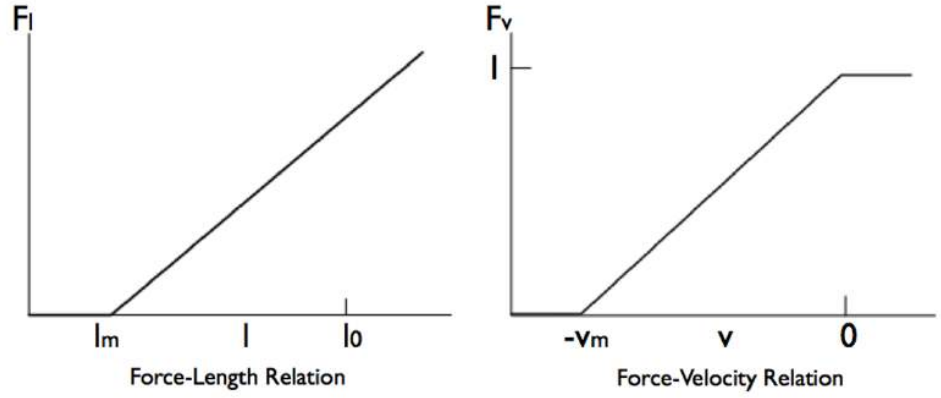


Figure 3.3: The force-length and force-velocity relations of the Hill-type muscle model.

where k_s and k_d are the stiffness and damping coefficient, respectively, e is the strain of the muscle and $\dot{e}$ is its strain rate. The contractile element f_C, which actively generates the contractile force of the muscle, is computed as

$$f_C = aF_l(l)F_v(\dot{l}), \tag{3.6}$$

where F_l is the force-length relation, F_v is the force-velocity relation, and a is the muscle activation level ($0 \leq a \leq 1$), which serves as the control input to the actuator.

Figure 3.3 plots the force-length and force-velocity relations. The former is represented as

$$F_l(l) = \max(0, k_{max}(l - l_m)), \tag{3.7}$$

where l_m is the minimum length for muscle to generate the force, and k_{max} is the maximum stiffness of activated muscle. The latter is represented as

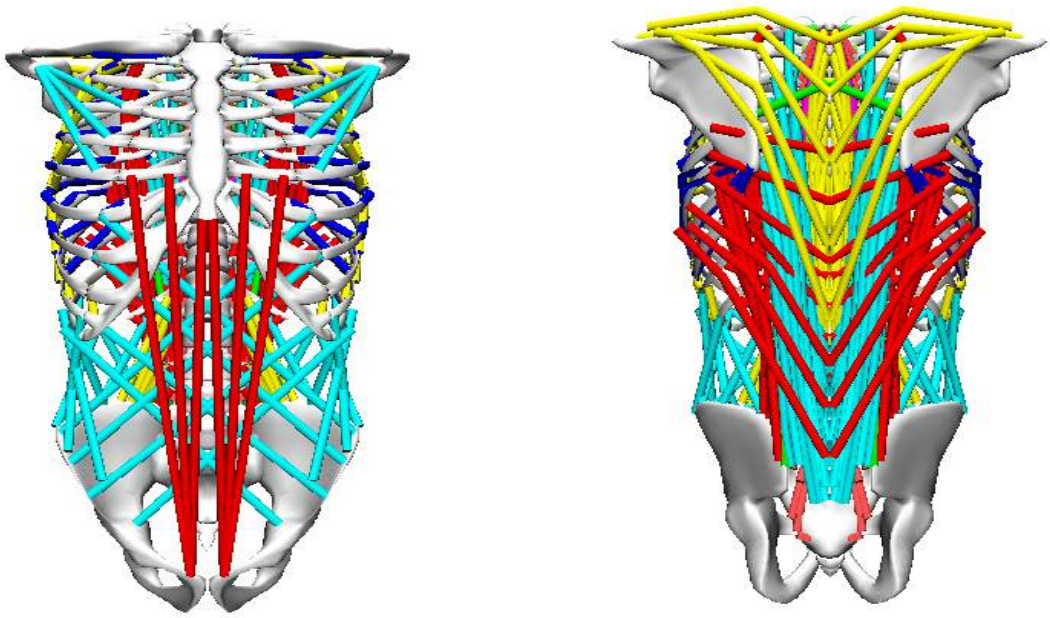
$$F_v(\dot{l}) = \max(0, 1 + \min(\dot{l}, 0)/v_m), \tag{3.8}$$

where v_m is the maximum contraction velocity with no load. The coefficient k_c is set to 7 for all the muscles, and I_m is set to $0.5l_0$ and v_m is $l_0 sec^{-1}$. The other coefficients k_s, k_d, and k_{max} are scaled to be proportional to the strength of weight factor of each muscle, which is calculated as roughly proportional to the cross-sectional area of the muscle. The list of weights can be found in (Lee et al., 2009).

In our modified Hill-type model, $F_l(l)$ increases monotonically. This works for our biomechanical human model because it stretches only a limited amount due to the constraints of the bones. Thus, we avoid negative stiffness, which could potentially cause instability in the numerical simulation of the musculoskeletal system.

3.2.3 Torso Musculoskeletal Complex

Unlike the musculoskeletal model of Nakada et al. (2018), our virtual human includes a fully functional torso musculoskeletal complex whose 50 bones (112 articular DoFs) are actuated



(a) (b)

Figure 3.4: (a)–(b) Close-up views of the torso with its 50 bones and 443 hill-type muscle actuators.

by no fewer than 443 muscles. Figure 3.4a,b shows in greater detail the torso musculoskeletal complex, which is rooted at the pelvis, with its five lumbar vertebrae, L5 through L1, and twelve thoracic vertebrae, T12 through T1, progressing up the spinal column to the T1 vertebra, which can be considered an "end-effector" (Figure 1.1c). Short, intermediate, and long Hill-type uniaxial muscle actuators situated with anatomical precision in deep, intermediate, and superficial layers, respectively, actuate the twelve 3-DoF joints of the spine and other joints of the torso musculoskeletal complex, such as the rib and sternum. Each spinal joint incorporates damped rotational springs to approximate the passive elasticity of the intervertebral discs and ligaments.

3.3 Flesh Simulation

The shape and deformation of the flesh volume is determined by the dynamics of the articulated skeleton and the gravitational force acting on the flesh. Naturally, the exact tissue behavior is also dependent on the geometric layout and material properties of the heterogeneous array of tissue components that constitute the flesh. Some of these material traits are encoded as static distributions of scalar (e.g., elastic moduli) or vector (e.g., muscle fiber orientation) quantities. In addition, the time-varying muscle activation signals, are input to the flesh simulation along with the skeletal dynamics.

The physical behavior of the virtual human's soft tissue and musculature is computed via numerical simulation of a discrete volumetric model. In designing the discrete representation, we simplify the model by striking a reasonable balance between computational complexity, geometric resolution, biomechanical accuracy, and robustness of the simulation. First, the skin is not modeled as a distinct simulation component; we model the entirety of the space between the skin and bones as an elastic continuum; no air-filled cavities or fluid volumes are explicitly simulated as such, although we are free to modulate the elastic properties (e.g., stiffness or compressibility) of such areas to reflect their macroscopic behavior. In addition, the entire flesh volume is assumed to deform as a connected continuum; that is, we do not allow slip or separation in the interior of the flesh volume. Note that connective tissue typically limits the extent of such motions, but there are parts of the real human body anatomy where true sliding or separation is possible.

For the purpose of simulating the dynamic deformation of flesh and muscle, we employ a lattice-based discretization of quasi-incompressible elasticity augmented with active (contractile) muscle terms. The lattice-based representation captures the shape of the deforming flesh volume. This discrete model is simply created by superimposing a cubic lattice (we use a lattice size of 10 mm) on a three-dimensional model of the human body, and we discard all cells that do not intersect the flesh volume (i.e., cells that are outside the body, or wholly within solid bones). Of course, the lattice representation thus created does not accurately capture the geometry of the flesh volume, but provides only a "cubed" approxi-

mation. Despite this, the discrete governing equations are constructed so as to compensate for the geometric discrepancy. The equations of elasticity are discretized using the method in (Patterson et al., 2012), which captures the fact that elastic material partially fills lattice elements on the boundary of the flesh volume. The actual skin surface differs from the jagged boundary of the lattice-derived simulation volume; we compensate by embedding a high-resolution skin surface mesh within the cubic lattice and distributing the forces acting on the skin surface into the volumetric lattice by scaling with the appropriate embedding weights, as discussed in (Zhu et al., 2010). Finally, since the contact surface between the flesh and bones is not resolved in the lattice-derived mesh, stiff zero-rest-length springs are applied to elastically attach points sampled on bone surfaces to embedded locations in the flesh simulation lattice, as detailed by Lee et al. (2009); McAdams et al. (2011).

3.3.1 Flesh Constitutive Model

The deformable collection of flesh, skin, and muscle are modeled as a hyperelastic solid. The elastic energy associated with it is partitioned as follows:

$$E_{\text{total}} = E_{\text{iso}} + E_{\text{muscle}} + E_{\text{vol}} + E_{\text{att}}. \tag{3.9}$$

In this expression,

$$E_{\text{iso}} = \int_{\Omega} \Psi_{\text{iso}}(\mathbf{F}) d\boldsymbol{X} \tag{3.10}$$

is a strain energy of an isotropic "foundation" material which corresponds to passive flesh (predominantly fatty tissue), where $\mathbf{F} = \partial\phi/\partial\boldsymbol{X}$ denotes the deformation gradient of the 3D deformation function $\phi : \Omega \to \mathbf{R}^3$, which maps material coordinates $\boldsymbol{X}$ to world-space deformed locations $\boldsymbol{x} = \phi(\boldsymbol{X})$. For the subset Ω_m of the body, which is covered by muscle, an additional anisotropic energy term

$$E_{\text{muscle}} = \int_{\Omega_m} \Psi_{\text{muscle}}(\mathbf{F}) d\boldsymbol{X} \tag{3.11}$$

is added, to account for the directional passive/active response of fibrous muscle tissue. The energy term E_{vol} enforces volume preservation in the flesh. Finally, the energy term E_{att} is associated with the elastic attachment constraints that couple the flesh and bone.

The isotropic component of the strain energy density is formulated as a Mooney-Rivlin material

$$\Psi_{\text{iso}}(\mathbf{F}) = \mu_{10}(\|\mathbf{F}\|_F^2 - 3) + \frac{\mu_{01}}{2}(\|\mathbf{F}\|_F^4 - \|\mathbf{F}^T\mathbf{F}\|_F^2 - 6) \tag{3.12}$$

with parameter values $\mu_{10} = 20\,\text{KPa}$, $\mu_{01} = 60\,\text{KPa}$. Here a simple, non-deviatoric formulation of Mooney-Rivlin hyperelasticity is used. The formulation supports strong incompressibility; thus, factoring out the hydrostatic stress component is not essential ($\mathbf{F}$ will be forced to have unit determinant by means of the strong incompressibility penalty). The anisotropic component of the strain energy is expressed as

$$\Psi_{\text{muscle}} = \sum_k \Psi_m^{(k)}(\mathbf{F}; \boldsymbol{w}_k; a_k)\,[\boldsymbol{X} \text{ covered by muscle } k], \tag{3.13}$$

where $\boldsymbol{w}_k$ and a_k are the fiber direction and activation level of muscle k, respectively. The energy density term associated with muscle k is a function $\Psi_m^{(k)}(\lambda_k, a_k)$ of the along-fiber stretch ratio $\lambda_k = \|\mathbf{F}\boldsymbol{w}_k\|_2$ and the respective activation value a_k. Quantity $\Psi_m^{(k)}$ is defined indirectly through its derivative with respect to λ_k; in fact, $\partial\Psi_m^{(k)}/\partial\lambda_k = T(\lambda_k, a_k)$ is the directional tension function resulting from the sum of the passive elasticity and force-length terms used in the muscle-actuated skeleton simulation. The strain rate and force-velocity terms have been omitted for simplicity, a decision further motivated by the fact that we use a non-validated generic Rayleigh damping model for the isotropic flesh, which would dilute the accuracy of an elaborate force-velocity formulation.

3.3.2 Incompressibility

Volume preservation in the elastic material is enforced via a penalty term $E_{\text{vol}} = \int_\Omega \Psi_{\text{vol}}(J)d\boldsymbol{X}$, where $\Psi_{\text{vol}}(J) = \kappa \log^2(J)/2$, with $J = \det\mathbf{F}$, is the volume change ratio, and the bulk modulus κ is set to 20 MPa. We transition to a mixed displacement/pressure energy formulation

in order to improve the numerical conditioning of this quasi-incompressible material:

$$\hat{E}(\mathbf{x}, p) = E_{\text{att}} + \int_{\Omega} \left[\Psi_{\text{iso}}(\mathbf{F}) + \Psi_{\text{muscle}}(\mathbf{F}) + \alpha p \log(J) - \frac{\alpha^2 p^2}{2\kappa} \right] d\boldsymbol{X}. \tag{3.14}$$

From the theory of mixed discretizations, the spatial gradients $\mathbf{f} = -\partial E/\partial \mathbf{x}$ and $\hat{\mathbf{f}} = -\partial \hat{E}/\partial \mathbf{x}$ of the two energy formulas (corresponding to elastic forces) are *equal* when the mixed energy $\hat{E}$ is stationary with respect to a variation in the pressure field. Thus, a time-integration scheme is constructed by computing forces according to $\hat{\mathbf{f}}(\mathbf{x}, p) = -\partial \hat{E}(\mathbf{x}, p)/\partial \mathbf{x}$, and the stationarity condition $\partial \hat{E}/\partial p = 0$ is appended to the time integration equations. The pure-displacement formulation is the same as the result, but the discrete equations remain well conditioned in spite of the degree of incompressibility. The cost for this conditioning is that the discrete time integration equations become symmetric indefinite, thus a Krylov solver such as MINRES, SYMMLQ, or symmetric QMR must be used in place of conjugate gradients.

3.3.3 Skeletal Attachments

Since the surfaces of attachment between flesh and bone do not coincide with nodes of the simulation lattice, the attachments are enforced by soft, spring-based constraints. The attachment energy is formulated as:

$$E_{\text{att}}(\mathbf{x}) = \sum_i \frac{k_i}{2} \|\mathbf{W}_i \mathbf{x} - \boldsymbol{t}_i\|_2^2, \tag{3.15}$$

where $\boldsymbol{t}_i$ is the target location (on the moving skeleton) of a flesh attachment, $\mathbf{W}_i$ is a trilinear interpolation operator that computes the interpolated location of an interior flesh point from the vector of nodal positions $\mathbf{x}$, and k_i is the stiffness parameter of attachment i. We sampled discrete attachment points on the bone surfaces as a preprocess. The attachment points are sampled uniformly, with a target density that yields an average of 3 to 6 attachment points per cell of the simulation lattice. We adapt the stiffness parameters such that a constant stiffness per bone surface area can be achieved.

3.3.4 Discretization Given Musculature and Skeletal Structure

We use standard trilinear hexahedral elements on a cubic lattice for the discretization of the deformation map ϕ, while the pressure field is only approximated as cell-wise constant. The geometries of the skin, muscles, and bones are embedded in the simulated hexahedra. The geometry of the muscles are used to modulate the material properties assigned to each simulation element. The nonlinear energy integrals are computed by defining a 4-point quadrature rule on an individual lattice cell basis, which is designed in the integration of polynomials of degree up to 2, yielding a (locally) second-order accurate approximation to the energy.

To achieve second-order accuracy on arbitrarily integration domains, a numerical quadrature scheme is used. The scheme integrates exactly all monomials $X^pY^qZ^r$ with $0 \leq p+q+r \leq 2$; i.e., Take the following quadrature as an example (Patterson et al., 2012):

$$\int_{\Omega_k} \begin{bmatrix} 1 & X & Y & Z \\ X & X^2 & XY & XZ \\ Y & XY & Y^2 & YZ \\ Z & XZ & YZ & Z^2 \end{bmatrix} d\boldsymbol{X}. \tag{3.16}$$

Here $\boldsymbol{X} = (X, Y, Z)$ is the material point in the undeformed configuration and $\Omega_k = \Omega \cap C_k$ is the elastic sub-domain within each lattice cell C_k. Monte-Carlo integration is used to compute the relevant moments of fractional cells C_k. A number of randomly generated points are uniformly distributed in each simulation hexahedron. We use 1.6×10^6 points for each simulated hexahedron of size $20\,\text{mm} \times 20\,\text{mm} \times 20\,\text{mm}$. We also examine whether each of these sample points is located inside any muscle volume, in which case the direction of the muscle fiber at the given location has to be associated with the sample point. Points not inside any muscle volume are considered as locations of passive flesh or fatty tissue. We then compute the quadrature (3.16) for the fraction of the simulation hexahedron covered by each muscle. We also compute the quadrature for boundary cells that are partially covered by muscles or passive flesh. The fiber directions of the sample points inside muscle k are

averaged and normalized the result to unit length to obtain a representative fiber direction $\boldsymbol{w}_k$ in (3.13).

3.4 Summary

This chapter presented the details of our anthropomimetic simulation framework, including both its musculoskeletal simulation and flesh simulation components. Given this biomechanical substrate, the inputs needed by our virtual human model to produce various motor behaviors are the time-varying activation levels to innervate its many Hill-type uniaxial muscles to actuate its skeleton and flesh. In the next chapter, we will develop the neuromuscular motor control system that achieves this.

CHAPTER 4

Neuromuscular Motor Control Framework

Our biomechanical human model is actuated by groups of skeletal muscles driven by neuromuscular motor controllers that provide efferent activation signals to the muscles. The motor subsystem includes a total of 6 neuromuscular motor controllers (Figure 4.1), a core controller for the torso musculoskeletal complex, one for the cervicocephalic complex, two for the arm complexes, and two for the leg complexes. Each controller generates the muscle activations $\boldsymbol{a}$ to actuate its associated musculoskeletal complex in a purposeful manner.

4.1 Neuromuscular Motor Controllers

Figure 4.2 shows the architecture of a neuromuscular motor controller. As in (Nakada et al., 2018), it is comprised of a voluntary controller and a reflex controller, both of which are trained DNNs that produce muscle activation adjustment (Δ) signals. The voluntary controller produces signals $\Delta\boldsymbol{a}_v$, which induce the desired actuation of the associated musculoskeletal complex, while the reflex controller produces muscle-stabilizing signals $\Delta\boldsymbol{a}_r$. Thus, the output of the neuromuscular motor controller is given by

$$\boldsymbol{a}(t + \Delta t) = \boldsymbol{a}(t) + (\Delta\boldsymbol{a}_v(t) + \Delta\boldsymbol{a}_r(t)) \,. \tag{4.1}$$

Note in Figure 4.2 that the neuromuscular controllers are recurrent neural networks with the muscle activation signal $\boldsymbol{a}$ forming a feedback loop.

We adopted a DNN architecture that works well for motor DNNs; specifically, rectangularly-shaped, fully-connected networks with six hidden layers. The DNN for the torso complex has 600-unit-wide hidden layers, while those for the simpler musculoskeletal complexes of

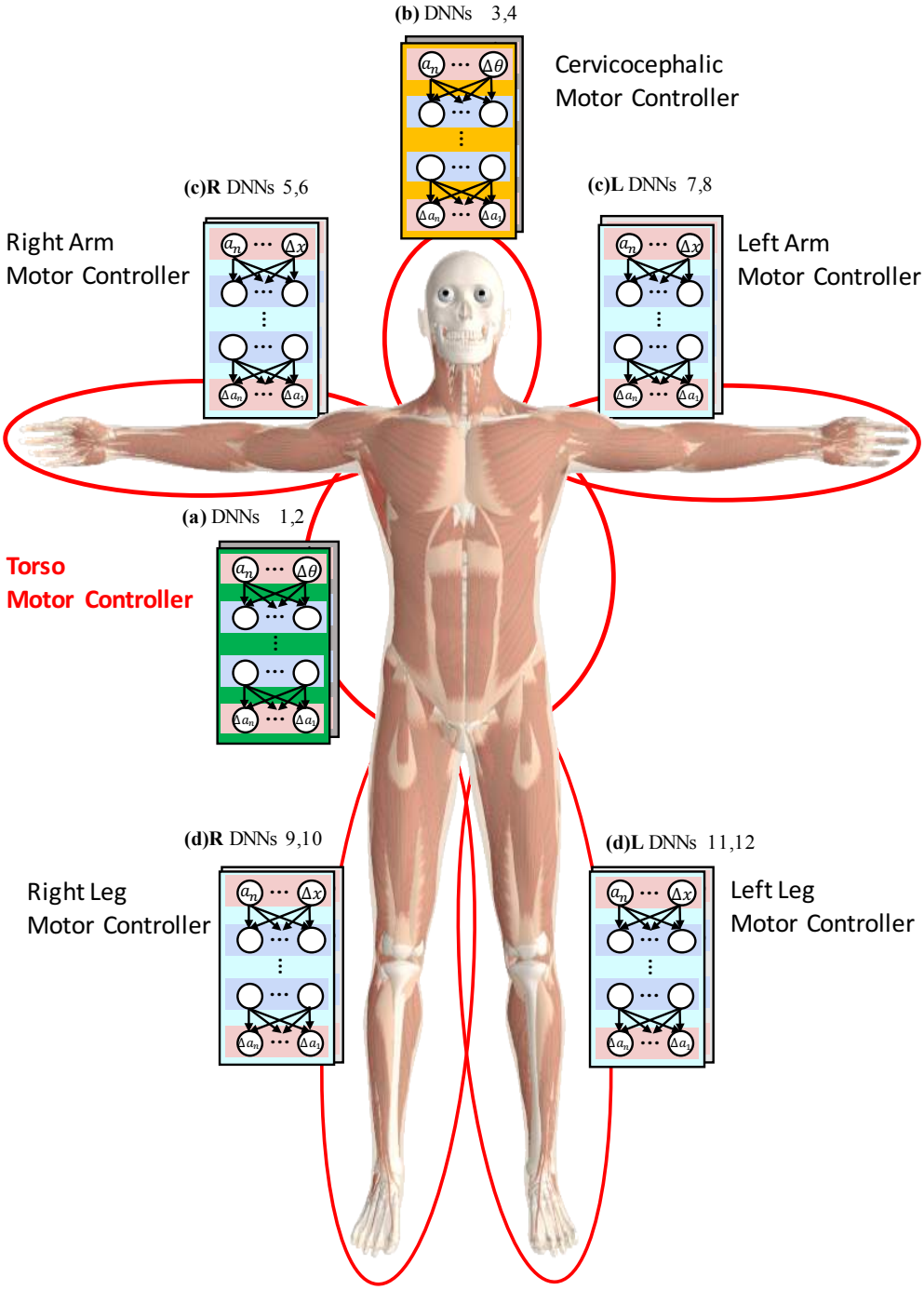


Figure 4.1: The motor subsystem architecture of our biomechanical human musculoskeletal model, illustrating its 6 neuromuscular motor controllers, each of which employs a pair of trained DNNs. They are (a) the torso controller (DNNs 1,2), (b) the cervicocephalic controller (DNNs 3,4), (c) the two arm controllers (DNNs 5,6 and DNNs 7,8), and (d) the two leg controllers (DNNs 9,10 and DNNs 11,12).

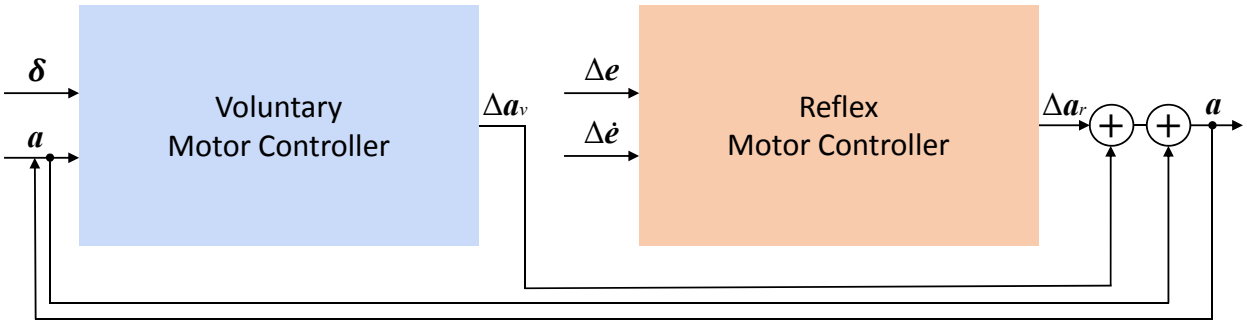


Figure 4.2: Neuromuscular motor controller architecture. The voluntary controller inputs a target discrepancy $\boldsymbol{\delta}$ and, recurrently, the muscle activations $\boldsymbol{a}$. The reflex controller inputs the changes in muscle strains $\boldsymbol{e}$ and strain rates $\dot{e}$. Both controllers output muscle activation adjustments.

the extremities have 300-unit-wide hidden layers. All the DNNs employ rectified linear units (i.e., the ReLU activation function). The DNNs were implemented and trained using the Keras library running on an Nvidia Titan X GPU in a Ubuntu 16.04 system with a 3.2 GHz Core 17 CPU.

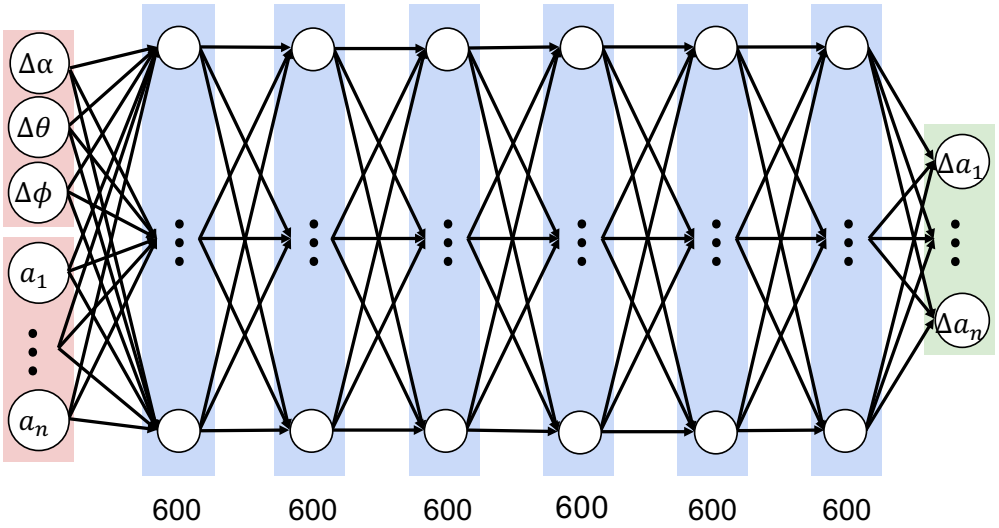
In the subsequent sections, we will present the details of the voluntary and reflex motor DNNs that control the torso and the extremities.

4.2 Torso Voluntary Motor DNN

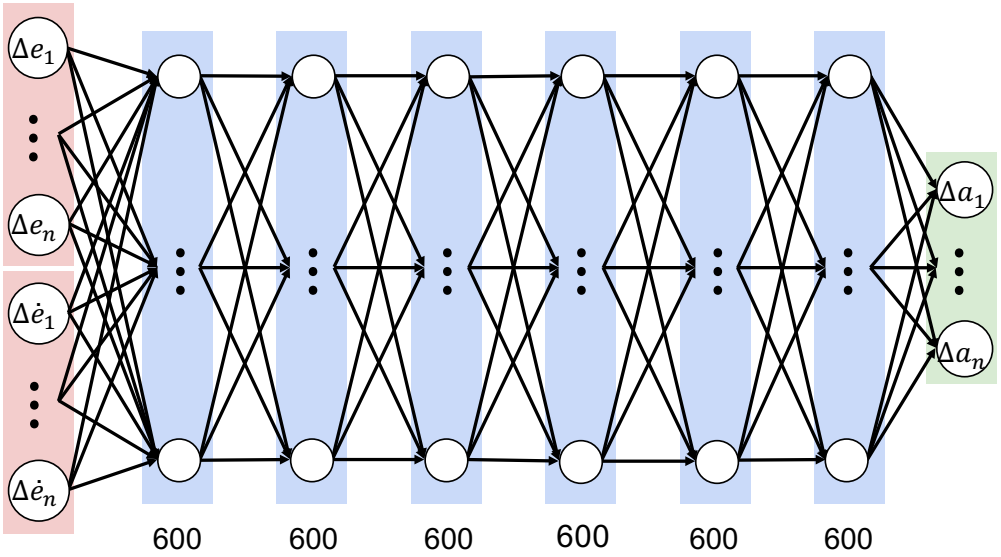
The function of the torso voluntary motor DNN (Figure 4.3a) is to generate efferent activation signals to the torso muscles in order to balance the mass of the trunk in gravity atop the flexible spinal column while actuating realistic torso movements to achieve target trunk poses that serve the purposes of the arms and head.

4.2.1 DNN Architecture

As shown in Figure 4.3a, the input layer of the DNN includes units that represent the (angular or linear) components of the discrepancy ($\boldsymbol{\delta}$ in Figure 4.2) between the value of some relevant feature of the torso's state, such as the orientation of the T1 vertebra, and the target value of that feature, as well as units that represent the current activations, $a_i(t)$,



(a) Voluntary Motor Controller DNN



(b) Reflex Motor Controller DNN

Figure 4.3: Architecture of the neuromuscular motor DNNs of the torso complex.

for $1 \leq i \leq n$, of each of the $n = 443$ muscles in the torso musculoskeletal complex. The 6 hidden layers comprise 600 units each. The output layer consists of units that encode the adjustments $\Delta a_i(t)$, for $1 \leq i \leq n$, to the muscle activations, which then contribute additively as $\boldsymbol{a}_v$ to updating the muscle activations according to (4.1).

4.2.2 Offline Training Data Synthesis and Network Training

To train the DNN, we use our biomechanical human musculoskeletal model to synthesize training data, as follows: Specifying a target orientation for the torso yields angular discrepancies, $\Delta\alpha$, $\Delta\theta$ and $\Delta\phi$, between the current torso orientation and the target torso orientation. With these angular discrepancies serving as the target control input, we compute inverse kinematics followed by inverse dynamics with minimal muscle effort optimization applied to the biomechanical torso model. This determines muscle activation adjustments, which serve as the desired output of the torso voluntary motor DNN. Hence, the input/output training pair consists of an input comprising the concatenation of the desired angle discrepancies, $\Delta\alpha$, $\Delta\theta$ and $\Delta\phi$, and the current muscle activations a_i, for $1 \leq i \leq 443$, along with an associated output comprising the desired muscle activation adjustments Δa_i, for $1 \leq i \leq 443$.

Appendix A presents additional details about the offline training data synthesis process. The process takes 10 sec of computation time to solve for 0.03 simulation seconds on a 3.2 GHz Intel Core i7 CPU with 16 GB RAM. Thus, we generated a training set of approximately 1M input-output pairs.

To train the DNNs, we apply backpropagation with the mean-squared-error loss function and the Adaptive Moment Estimation (Adam) stochastic optimizer (Kingma and Ba, 2014), with learning rate $\eta = 10^{-6}$. Figure 4.4 plots the progress of the training process. Figure 4.5 and Figure 4.6 show the same progress of the DNN with 300 units in each of the hidden layers. Comparing to Figure 4.4 and Figure 4.7, although the difference is not obvious between the two reflex motor DNNs, we observe a better result in terms of mean squared error from the voluntary motor DNN with 600 units; hence our choice of 600-unit hidden layers.

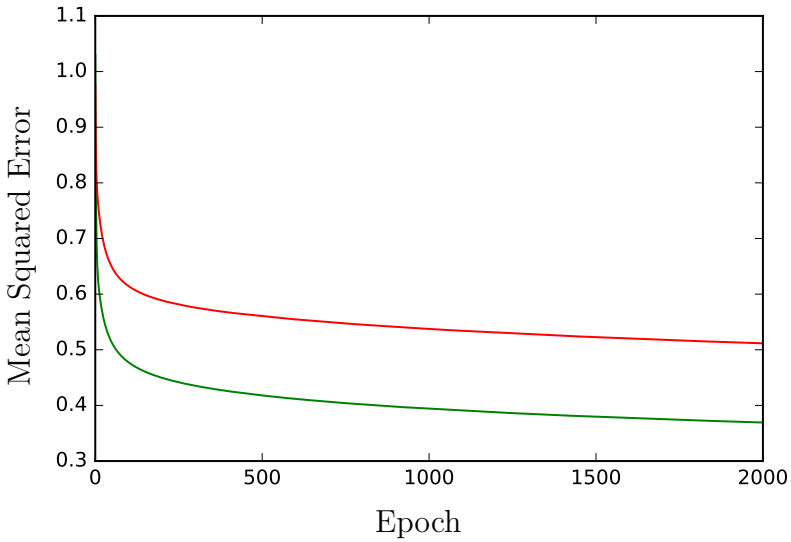


Figure 4.4: Progress of the backpropagation training of the torso voluntary motor DNN on the training (green) and validation (red) datasets.

After the DNN is trained, it serves as the *online* torso voluntary motor controller.

4.3 Torso Reflex Motor DNN

4.3.1 DNN Architecture

Figure 4.3b shows the architecture of the torso reflex motor DNN. The input layer consists of $2n$ units, where $n = 443$, the number of muscles in the torso musculoskeletal complex. It comprises units that represent the change in muscle strain Δe_i and in strain rate $\Delta \dot{e}_i$, for $1 \leq i \leq n$. Like the voluntary motor DNN, the network has 6 hidden layers with 600 units each. The output layer consists of n units providing muscle activation adjustments Δa_i, for $1 \leq i \leq n$, which then contribute additively as $\boldsymbol{a}_r$ to updating the muscle activations according to (4.1).

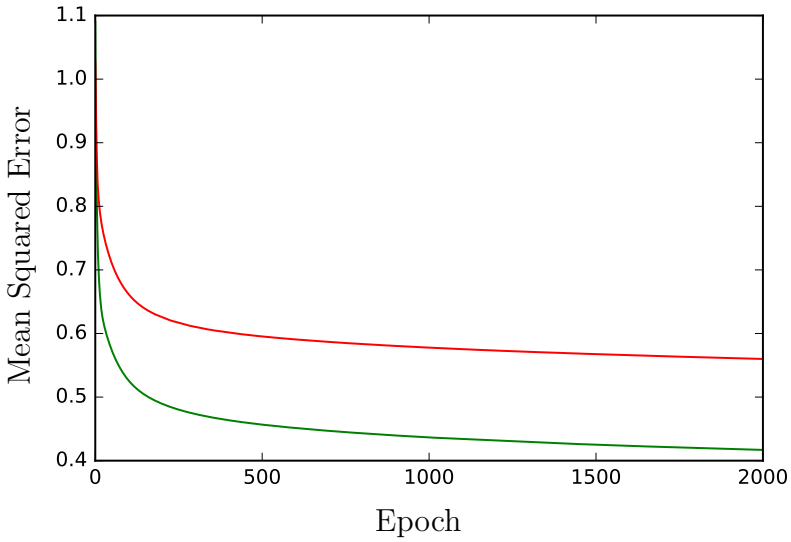


Figure 4.5: Progress of the backpropagation training of the torso voluntary motor DNN on the training (green) and validation (red) datasets with hidden layers of 300 units.

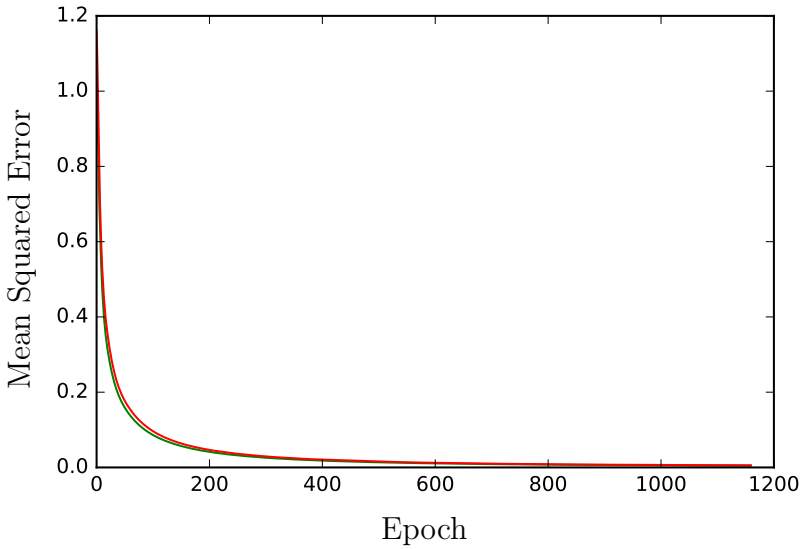


Figure 4.6: Progress of the backpropagation training of the torso reflex motor DNN on the training (green) and validation (red) datasets with hidden layers of 300 units.

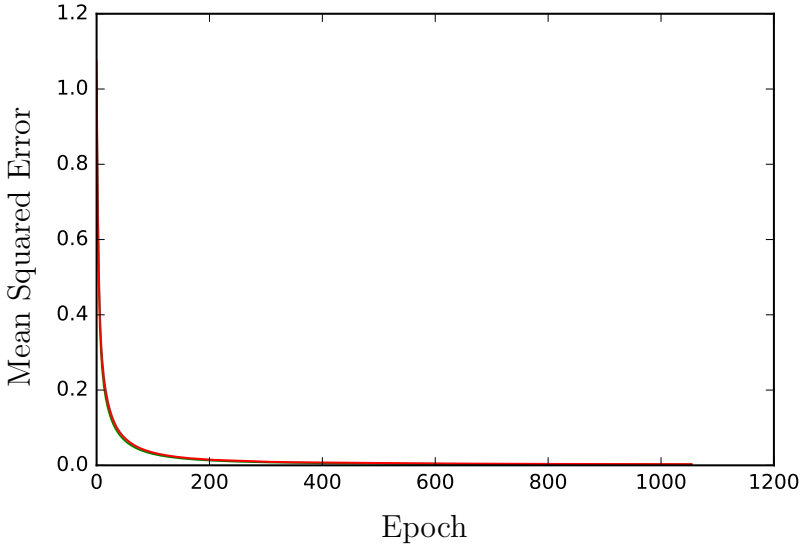


Figure 4.7: Progress of the backpropagation training of the torso reflex motor DNN on the training (green) and validation (red) datasets.

4.3.2 Offline Training Data Synthesis and Network Training

To train the torso reflex motor DNN, we employ the same training methods as for the voluntary motor DNN. We again use our biomechanical human musculoskeletal model to synthesize training data. The process for the torso complex reflex data generation is similar to the voluntary motor control data generation process and we employed the same computing hardware.

4.4 Motor DNNs for the Extremities

The architecture of the DNNs in the 5 neuromuscular controllers for the five extremities—the cervicocephalic complex, arm complexes, and leg complexes—is identical to that of the torso (Figure 4.2); however, the sizes of the input and output layers are determined by the number n of muscles in each complex, $n = 219$ for the cervicocephalic complex, $n = 29$ for the arm complexes, and $n = 39$ for the leg complexes. We refer the reader to (Nakada et al., 2018) for additional details.

For the purposes of our work in this thesis, we focus on balancing in gravity while sitting

or while in an upright stance and performing several motor tasks with the upper body. Confined to such scenarios, the task for legs is mainly to support an upright posture for the body and cooperate with the torso to maintain proper balance. This task may be satisfied through reflex-based pose control.

4.4.1 Offline Training Data Synthesis and Network Training for the Legs

The reflex motor controller DNNs of the legs are architected like the network in Figure 4.2b, but with 300 hidden units in each of the 6 hidden layers, and with $n = 39$ in the input and output layers in accordance with the 39 muscles in each leg musculoskeletal complex.

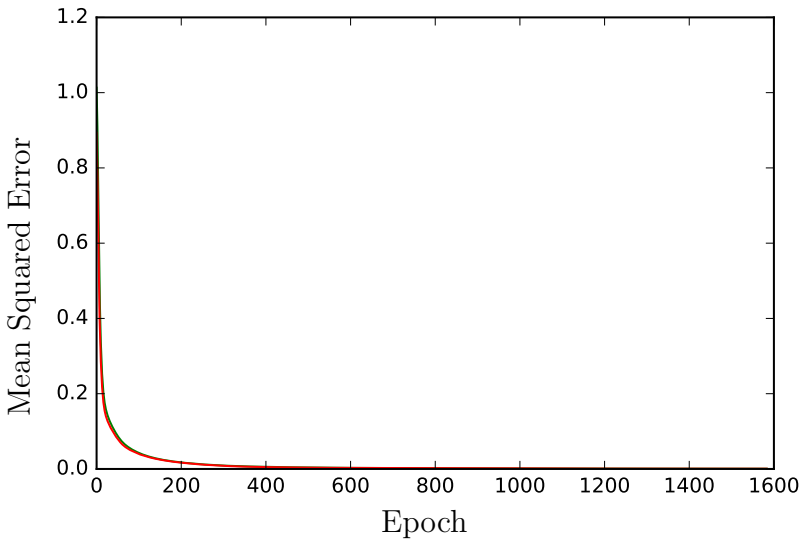
To synthesize training data for the network, pose control requires that the networks achieve and maintain a specified desired pose for the legs. Each desired pose determines a unique set of strains e_i and strain rates $\dot{e}_i$ for the $1 \leq i \leq 39$ leg muscles. The input to the leg reflex motor controller DNN is the discrepancy Δe_i between the actual and desired strains concatenated with the discrepancy $\Delta \dot{e}_i$ between the actual and desired strain rates, while the desired output is the muscle activation adjustments Δa_i. In this way, we generated approximately 300K input-output training data pairs.

Figure 4.8 shows the progress of the *offline* training process for the leg reflex DNN. After the DNNs are trained, they serve as *online* reflex motor controllers.

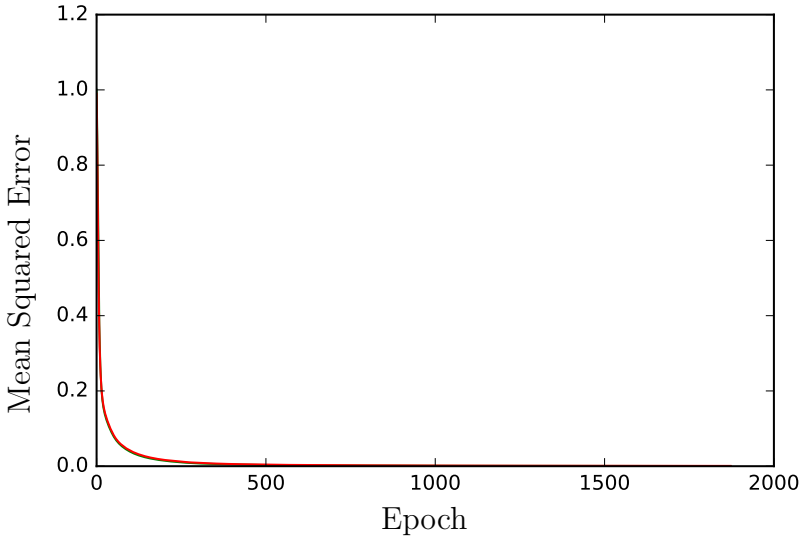
4.5 Coupling the Torso and Extremities

Despite the fact that the musculoskeletal complexes of the core and each of the 5 extremities include different sets of muscles under the control of different neuromuscular controllers, the force couplings between the extremities and the core are accomplished in a completely natural manner, through joints and muscles spanning the different musculoskeletal complexes.

First, the articulated biomechanical skeletal structure remains connected while moving, by its joints. In particular, the spinal joint between the C7 vertebra of the cervical spine and the T1 vertebra of the thoracic spine applies forces that constrain the cervicocephalic



(a) Left leg



(b) Right leg

Figure 4.8: Progress of the backpropagation training of the reflex motor DNNs for the legs on the training (green) and validation (red) datasets.

musculoskeletal complex to the torso. At the shoulders, the head of each humerus articulates with the glenoid fossa of its respective scapula in the torso, forming shoulder joints that apply forces to constrain the arm musculoskeletal complexes to the torso. At the hips, the head of each femur articulates with the acetabulum in the pelvic bone in the torso, forming hip joints that apply forces to constrain the leg musculoskeletal complexes to the torso.

Second, each of the musculoskeletal complexes of the extremities include multiple significant muscles that attach to major bones in the torso. These include the long, superficial muscles of the neck, such as the trapezius; the muscles of the shoulder, such as the biceps, triceps, and the muscles composing the rotator cuff; and the muscles of the hip, such as the gluteal group, adductor group, etc. The virtual counterparts of these muscles exist in our musculoskeletal model and they apply common forces between major bones of the torso and the proximal bones of the musculoskeletal complexes of the extremities.

Unlike Nakada et al. (2018) where we treat each musculoskeletal complex in isolation and train their motor controllers independently of one another, for viable core training, we must regard the whole-body model as a unified system. Thus, during torso training data synthesis, it is important to introduce random forces from the extremities onto the torso, such that the torso neuromuscular controller learns the consequences of forces derived from the extremities. Therefore, by sending random activation signals to their muscles, we randomly actuate the arm and cervicocephalic complexes by modest amounts as we synthesize the training data for the torso neuromuscular motor controller.

If the legs are to support the torso in a balanced stance, it is furthermore important to do the same for them. We therefore send random activation signals to the torso muscles as we synthesize training data for the leg neuromuscular motor controllers. To learn balance, if the center of pressure comes too close to the margin of the support polygon, the biomechanical model is reset to an upright posture and the data synthesis procedure is restarted.

4.6 Summary

Our approach in this chapter to tackling the challenge of neuromuscular motor control of our biomechanical human musculoskeletal model was machine learning, specifically deep learning. The neuromuscular motor control system of our virtual human comprises 12 trained DNNs, including a core voluntary/reflex DNN pair devoted to innervating the 443 muscles of the torso. By synthesizing its own training data offline, our virtual human automatically learns efficient, online, active control of the core musculoskeletal complex as well as its proper coordination with the five extremities—the cervicocephalic, arm, and leg musculoskeletal complexes. In the following two chapters, we will apply our novel, full-body neuromuscular motor control system in several animation scenarios.

CHAPTER 5

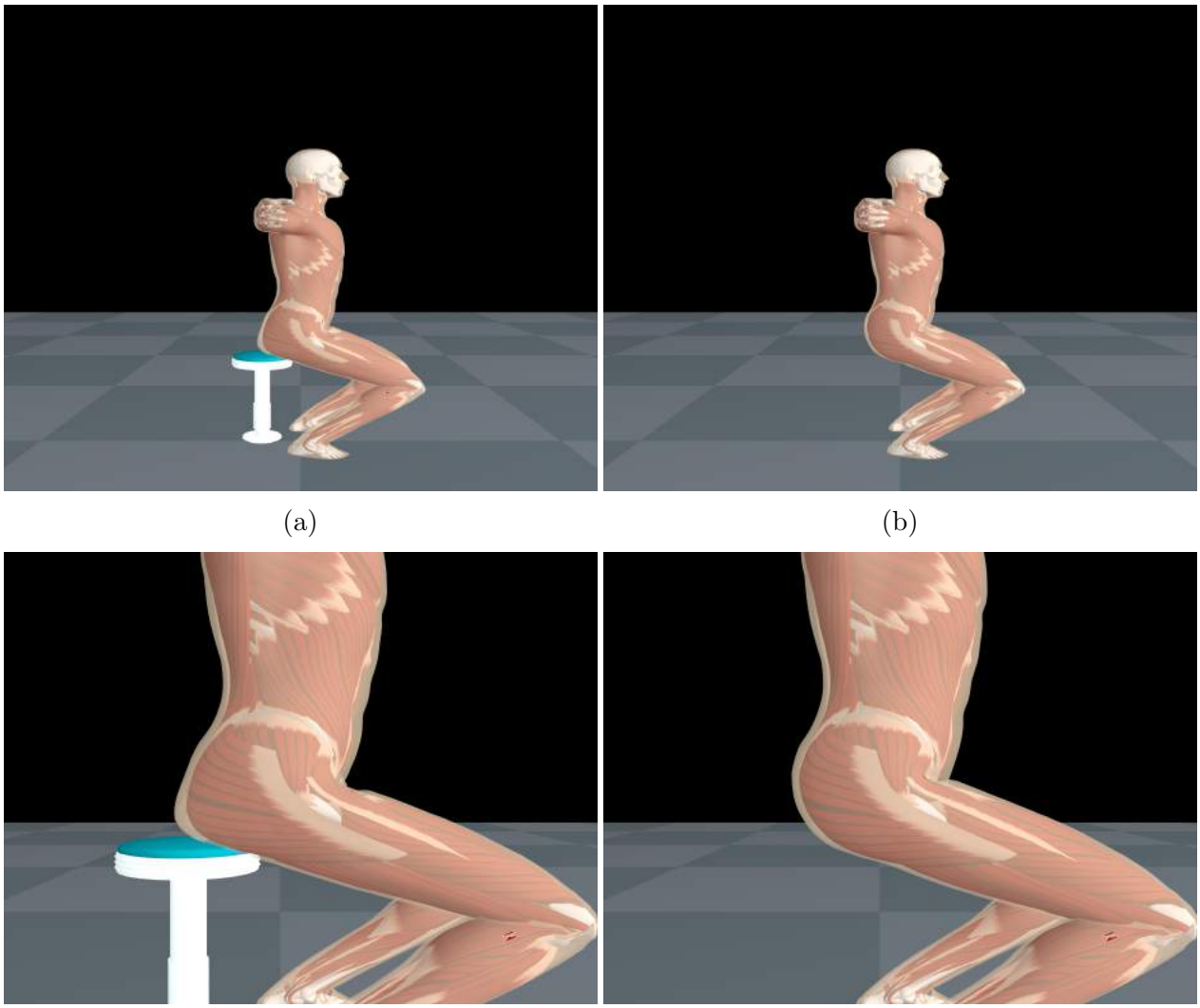
Experiments and Results

In this chapter, we report our experiments with the whole-body biomechanical human musculoskeletal model and show our results.

To visualize the soft-tissue components in our simulation framework, we demonstrate the detailed anatomical animation of the human body by rendering the skin translucently. Appendix B presents additional details about the body rendering process. For example, Figure 5.1 shows a sitting posture with close-up views demonstrating the deformation of the hip muscles when sitting on a stool.

5.1 Sit-to-Stand

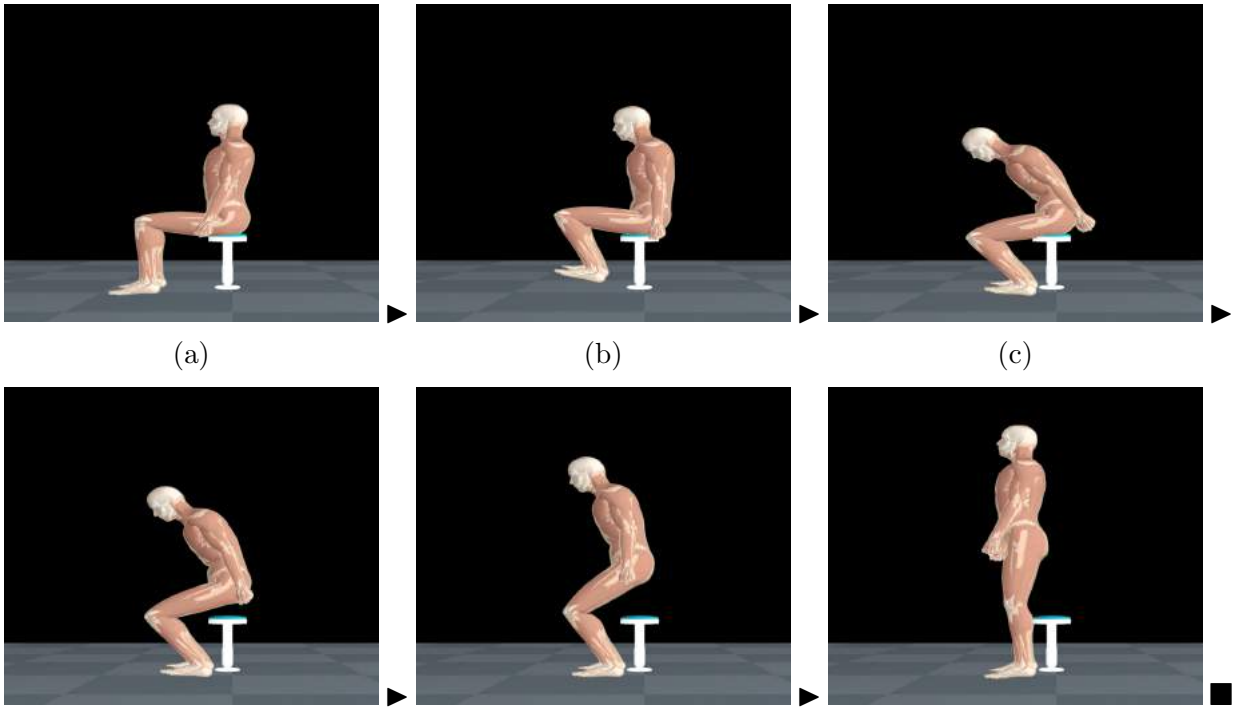
Figure 5.2 shows frames from a demonstration of our virtual human model sitting on a stool and standing upright. The musculoskeletal model must continually hold its upper body erect and maintain balance in gravity as it executes this action, as well as when it is standing in place and seated on the stool. Our demonstration video shows this animation, along with a visualization of the progression of the neuromuscular controller training process, in which our virtual human loses balance with inadequately trained torso and leg controllers and eventually learns to be proficient in performing the sit-to-stand action.



(a) (b)

(c) (d)

Figure 5.1: Anatomically detailed simulation and visualization of a sitting posture. (a) and (b) show a sitting posture of the virtual human with and without contact with a stool, respectively. Close-up views of the flesh near the pelvis are shown in (c) and (d).



(a) (b) (c)

(d) (e) (f)

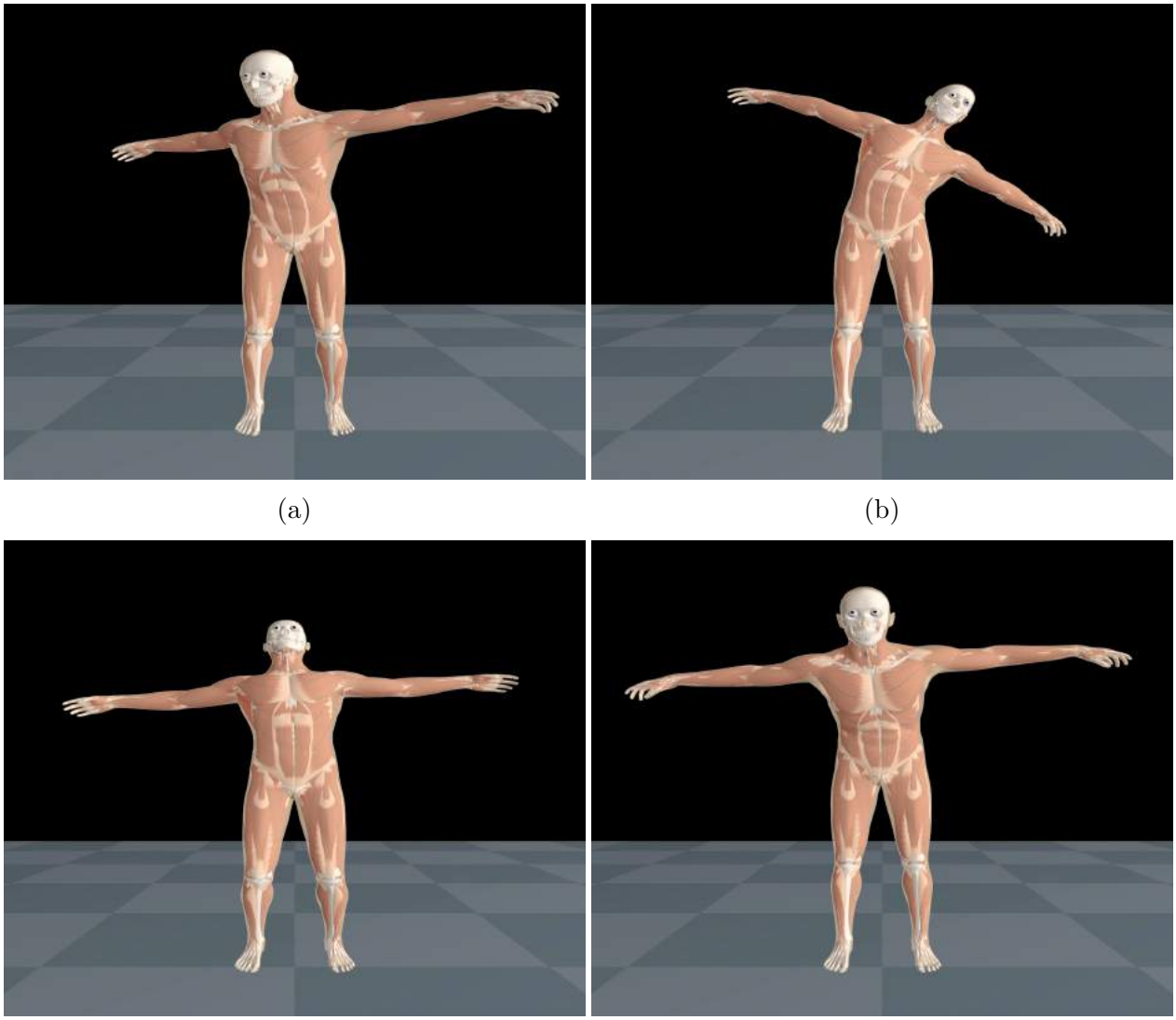
Figure 5.2: Sequence of frames from a sit-to-stand simulation.

5.2 Calisthenic Exercises

Figure 5.3 shows frames from an animation of our virtual human performing calisthenic exercises that involve significant torso articulation, including substantial bending and twisting of the lumbar and thoracic spinal articulatory DoFs. Not only is our exerciser able to articulate its upper body in a realistic manner, but by virtue of its reactive motor control ability it also maintains bipedal balance in gravity despite the large center-of-gravity shifts that typically result from strongly bending the upper body.

5.3 Stepping

A basic motor functionality for a free-standing autonomous virtual human is to be able to shift its position by taking steps. Purposeful stepping is a transient task-based motor function that differs from locomotion, which requires long-term, repetitive, rhythmic motor actions.



(a) (b)

(c) (d)

Figure 5.3: Calisthenic exercising of the torso. (a) and (b) demonstrate twist movements while (c) and (d) demonstrate backward and forward leaning movements.

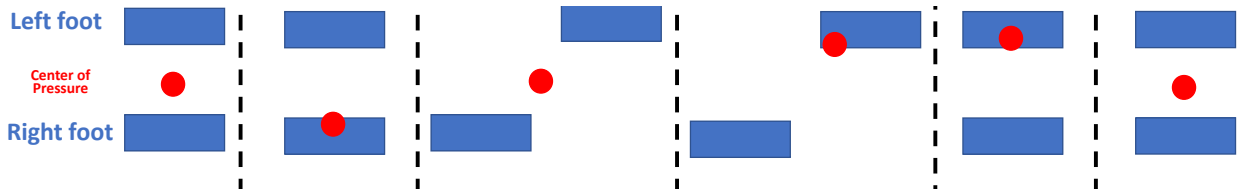
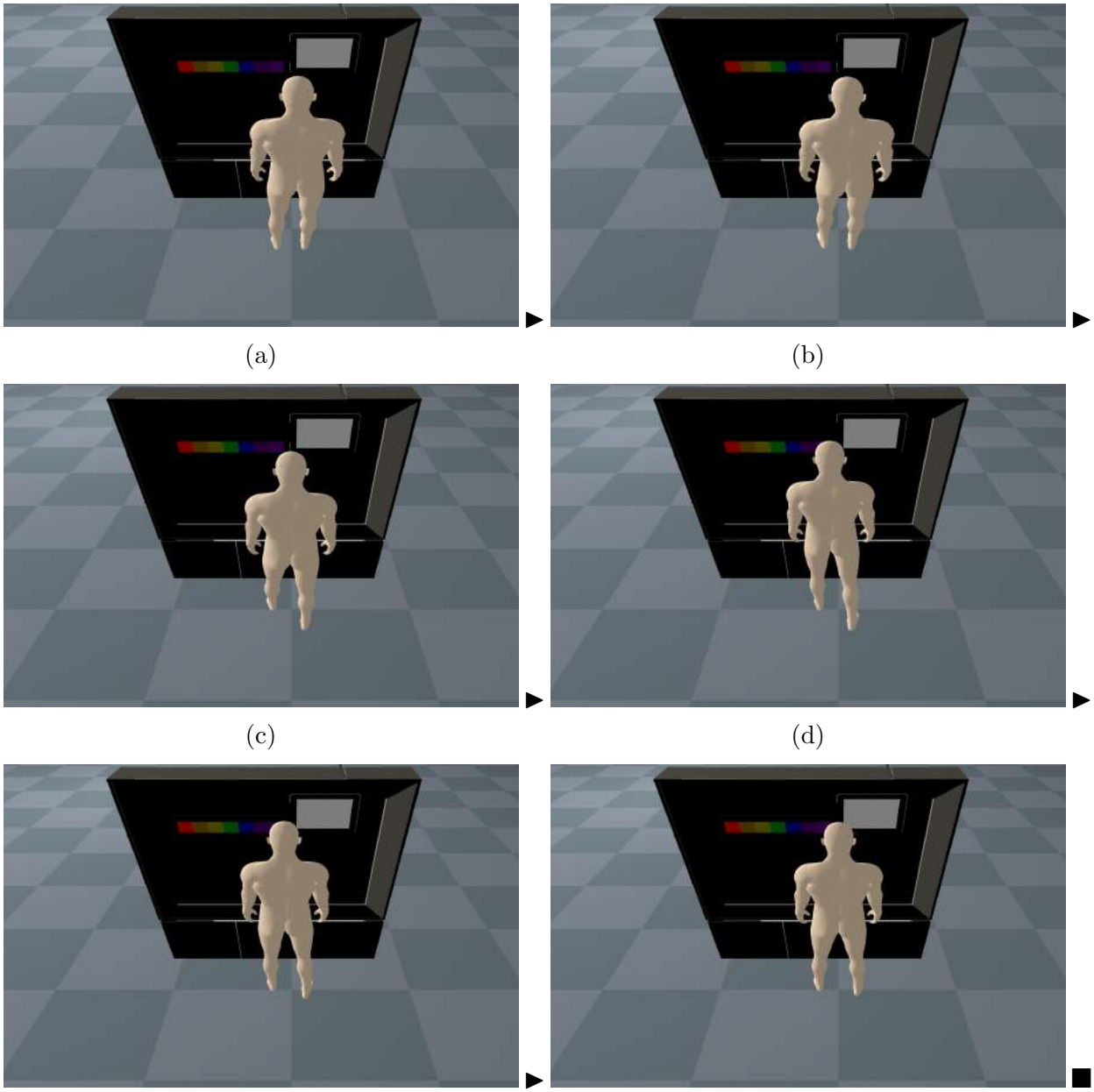


Figure 5.4: The stepping strategy.

Figure 5.4 illustrates our stepping control strategy. Our virtual human starts from a centered bipedal stance posture where the center of pressure (COP), represented by the red circle, falls near the center of the support polygon, the convex hull surrounding the boundaries of the two feet represented by the blue rectangles. Through the appropriate leg muscle activations, the virtual human shifts its COP towards one foot thereby reducing the pressure on the other foot until that foot's friction against the floor decreases enough that it can easily by shifted anteriorly, all the while maintaining its balanced upright stance. If necessary, the process is repeated by shifting the COP through the center of the support polygon towards the opposite foot. When stepping is completed, the bipedal stance is again centered by shifting the COP back near the center of the support polygon. The strategy for posterior or lateral shifts of a free foot is similar.

Figure 5.5 and Figure 5.6 show frames from simulations of our virtual human autonomously stepping forward toward a touchscreen display in order to move close enough to operate it. Proper balance is maintained throughout the stepping action.



(a) (b) (c) (d) (e) (f)

Figure 5.5: Sequence of frames from a stepping simulation with wide stance.

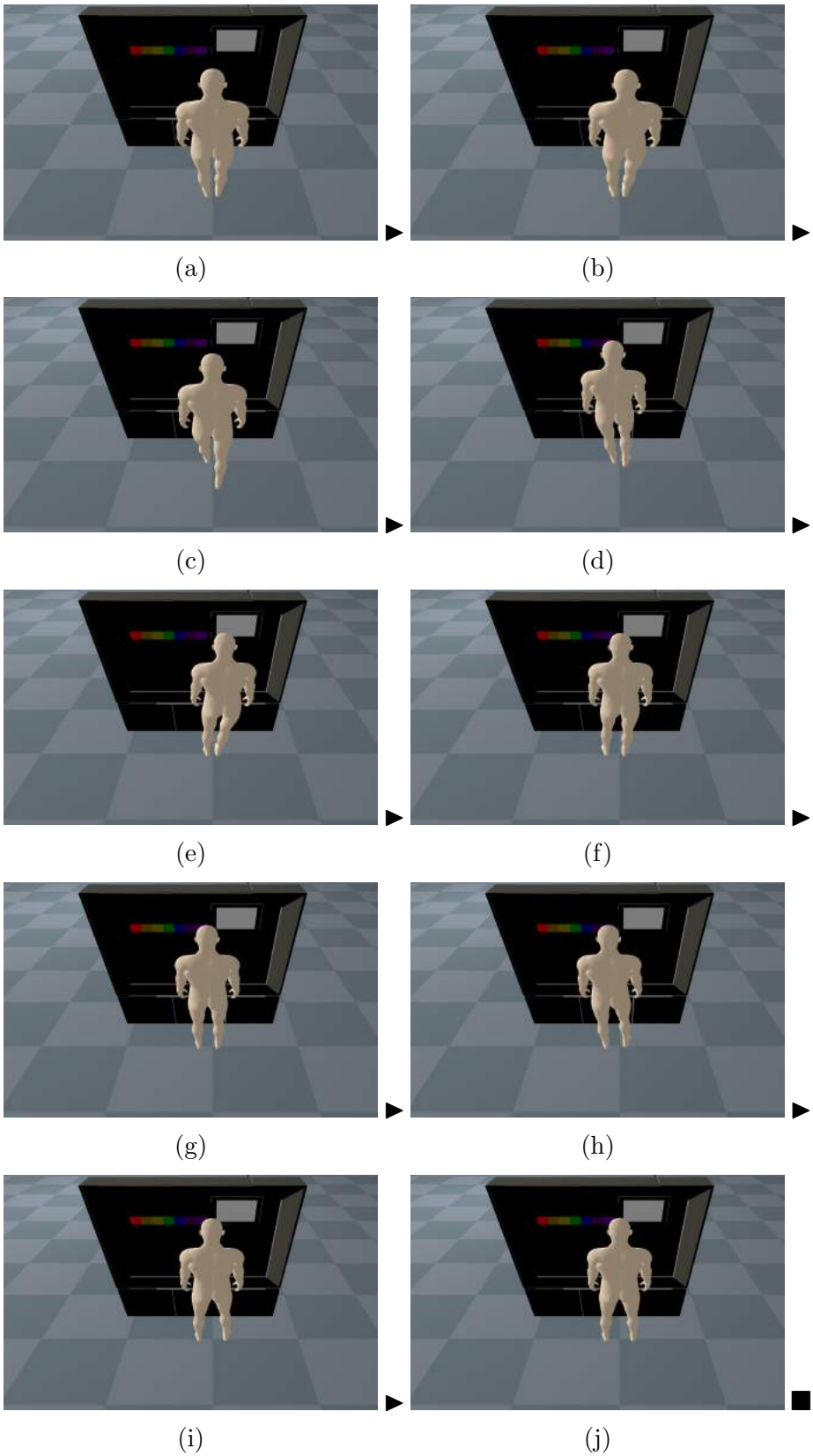
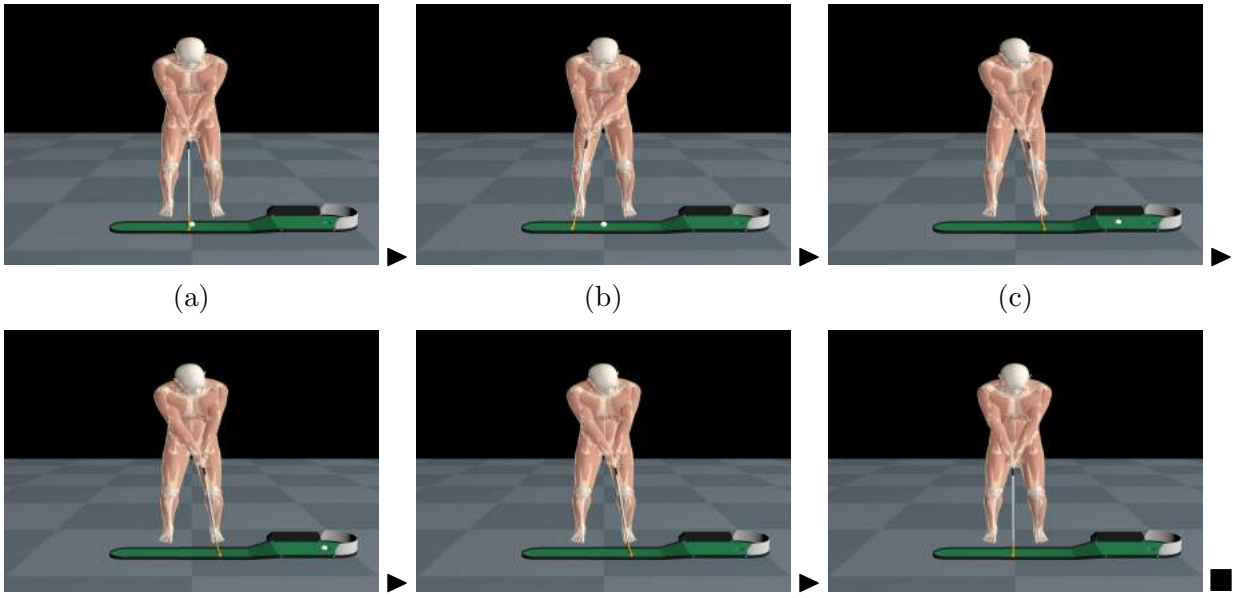


Figure 5.6: Sequence of frames from a stepping simulation with narrow stance.



(a) (b) (c)

(d) (e) (f)

Figure 5.7: Sequence of frames from a golf putting simulation (front view).

5.4 Golf Putting

Putting is the most individual of the golf strokes. The most important consideration is what feels natural, right, and good to each individual. Therefore multiple putt grips, stances, and actions are used. Figure 1.2a shows our biomechanical human musculoskeletal model gripping the golf club in a "right hand low" grip, while Figure 1.2b shows an alternative "prayer" grip. Our virtual golfer is balancing in gravity in an appropriately crouched stance in preparation for the putt.

Figure 5.7 and Figure 5.8 show frames from a putting simulation in which our golfer positions itself by stepping up to the ball, and after a number of attempts, putts the ball into the hole, throughout which our golfer's body stays bipedally well balanced and performs realistic strokes.

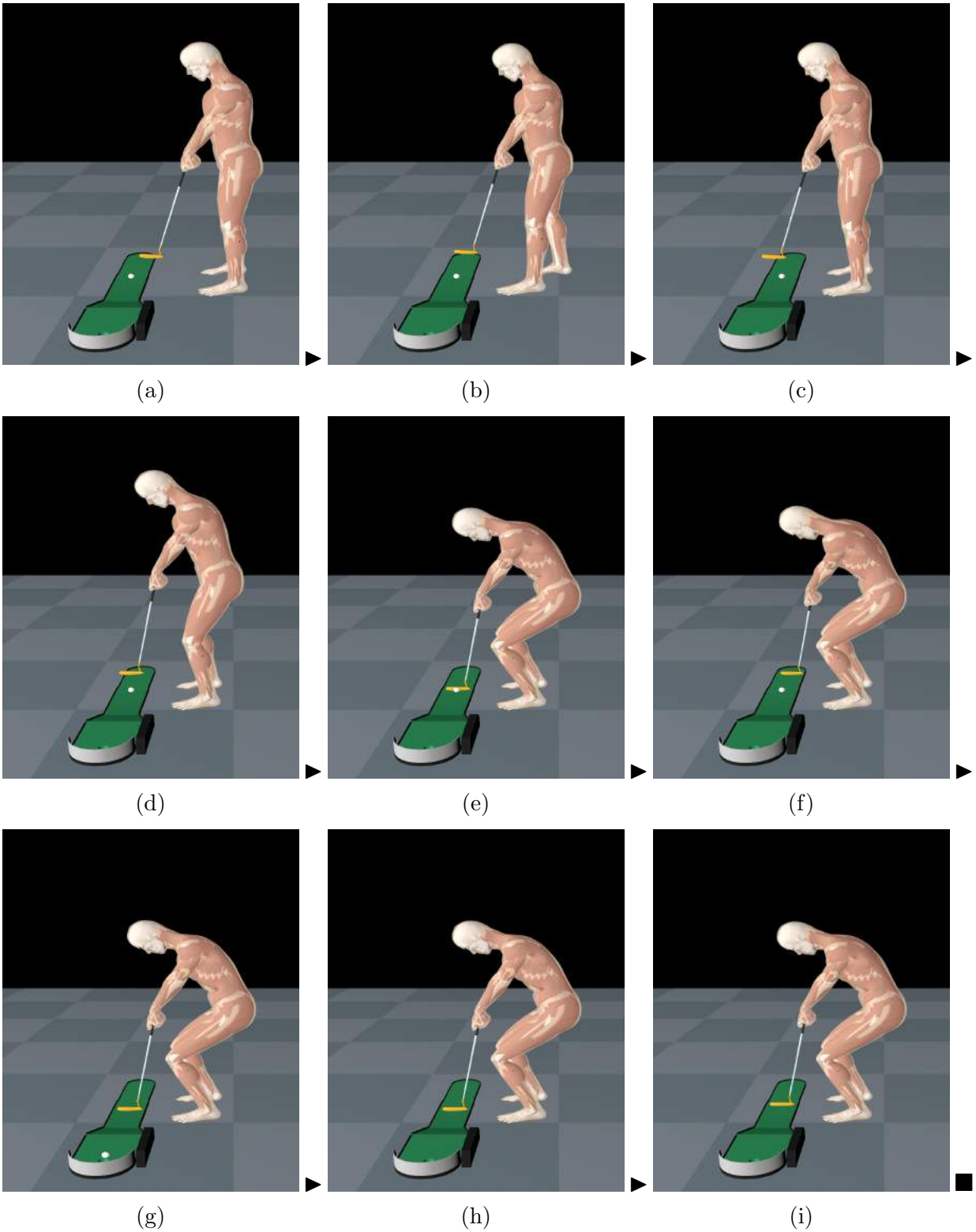


Figure 5.8: Sequence of frames from a golf stepping and putting simulation (side view). (a) The virtual human initially hold the putter and look at the ball; (b)–(c) the virtual human bent the knee and lean forward, prepare to put; (f)–(i) the virtual human swing the putter, hit and drive the ball to the hole.

CHAPTER 6

Applications to Sensorimotor Control

In this chapter, we demonstrate our whole-body biomechanical human musculoskeletal model and its neuromuscular control system in two substantially more complex applications that involve sophisticated sensorimotor control. The first demonstration, which involves observing and reproducing sketches, combines computer graphics with nontrivial computer vision. The second demonstration equips our virtual human with biomechanical eyes with the objective of validating the performance of these advanced eye models.

6.1 Autonomous Sketching

6.1.1 Background

There exist several studies related to sketching in the fields of robotics and AI. Traditionally, a robot arm is programmed to sketch lines on a canvas so as to mimic a given digitized portrait (Tresset and Leymarie, 2013). Calligraphy skills can be acquired by "Learning from Demonstration" (Sun et al., 2014). DNN-based approaches to art generation have recently been developed (Gatys et al., 2016; Elgammal et al., 2017). Furthermore, deep reinforcement learning based algorithms have been proposed to mimic the drawing task. However, these systems lack natural human drawing behaviors (Ganin et al., 2018; Zhou et al., 2018a).

In a departure from traditional pixel image modeling approaches, Simhon and Dudek (2004) and Zhang et al. (2017) proposed generative models of vector graphics. Graves (2013) focused on handwriting generation with Recurrent Neural Networks that generates a sequence of points. Subsequently, a sketch-RNN model was proposed to synthesize sketches (Jongejan et al., 2016; Ha and Eck, 2017), which was trained in a fully supervised manner,

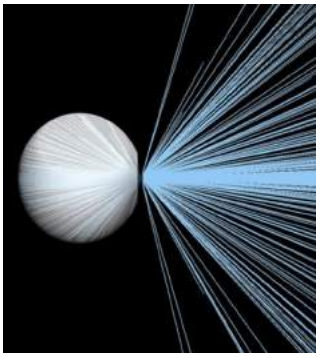


Figure 6.1: Rays (blue lines) cast from the positions of photoreceptors on the retina through the pinhole aperture and out into the scene by the raytracing procedure that computes the irradiance responses of the photodectors.

and the features learned by the model were represented as a sequence of pen stroke positions. However, humans do not naturally sense their visual environments as uniform grids of pixels and, although sketches are naturally generated by pen strokes, strokes do not capture the way humans mentally represent sketches.

In this chapter, we modify the sketch-RNN architecture to adapt the sketching technique to our human model. We employ an enhanced version of the biomimetic vision model proposed by Nakada et al. (2018), which includes a pair of virtual eyes, capable of eye movements. The eyes have retinas populated by photoreceptors (cones) that are nonuniformly distributed in a biologically consistent, foveated pattern.

6.1.2 Eye and Retina Model

The eye is modeled as a sphere that can be rotated with respect to its center around its vertical y axis by a horizontal angle of θ and around its horizontal x axis by a vertical angle of ϕ. The eyes are in their neutral positions looking straight ahead when $\theta = \phi = 0$.

We model the eye as an idealized pinhole camera with an infinitesimal aperture at the center of the pupil. To simulate biomimetic foveated perception, we place the photoreceptors on the hemispherical retinal surface at the interior rear of the eyeball according to a noisy log-polar distribution. In particular, we include 9,936 photoreceptors situated at $d_k = (d_{k,x}, d_{k,y})$,

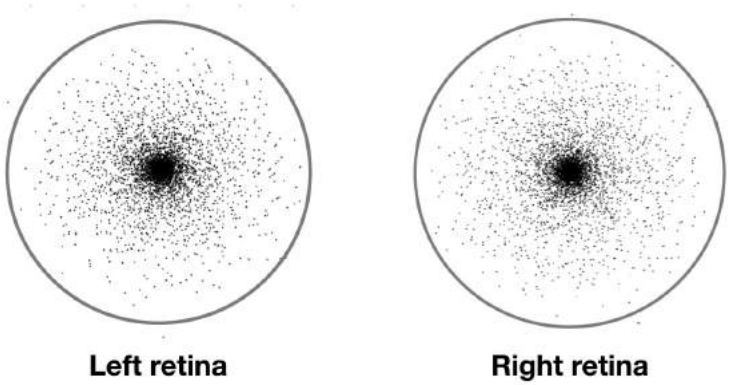


Figure 6.2: Locations of the photo-receptors (black dots) on the left retina and right retina according to the noisy log-polar model.

for $1 \leq k \leq 9{,}936$, such that

$$\begin{aligned} d_{k,x} &= e^{\rho_j} \cos\theta_i + N(\mu, \sigma^2); \\ d_{k,y} &= e^{\rho_j} \sin\theta_i + N(\mu, \sigma^2), \end{aligned} \tag{6.1}$$

where $0 < \rho_j \leq 138$, incremented in steps of 1, and $0 \leq \theta_i < 360°$, incremented in 5° steps, and where N is additive IID Gaussian noise of mean $\mu = 0$ and variance $\sigma^2 = 0.0225$, which places the photoreceptors in slightly different positions on the two retinas. Figure 6.2 illustrates the photoreceptor distributions on the left and right retinas.

Retinal imaging is performed using ray tracing. In each eye, sample rays from the positions of each of the 9,936 photoreceptors on the hemispherical retinal surface are cast through the aperture and out into the 3D virtual world where they recursively intersect with the visible surfaces of virtual objects and sample the light sources. As opposed to a conventional 2D image, the irradiance thus computed at each retinal photoreceptor comprises elements of a 1D Optic Nerve Vector (ONV) of length 9,936. As the virtual human observes its world, the ONV outputs from its eyes drive the processing in its visual perception system. For the purposes of our current application the retinal photoreceptors are binary; i.e., their response is either black or white.

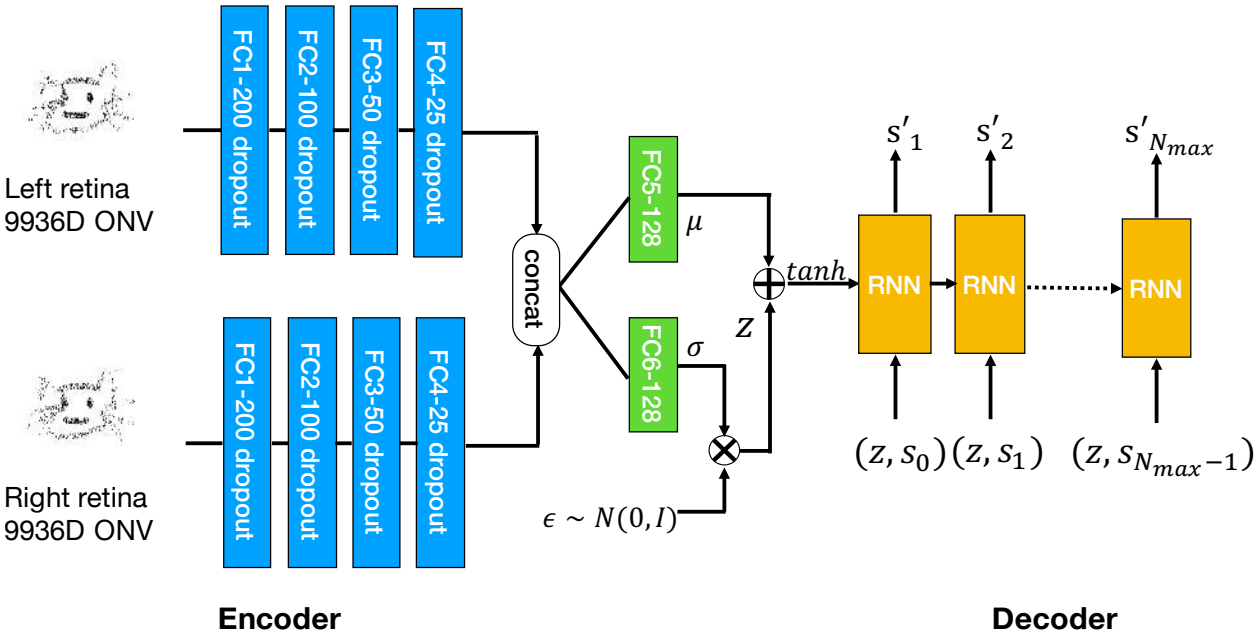


Figure 6.3: Architecture of our sketch-ONV2seq model. S_i is a 5-dimensional feature vector at time t, which includes the sketch offset and the drawing command, and (z, S_i) denotes the concatenation of latent vector and feature vector.

6.1.3 Sketch Visual Perception System

Figure C.2 illustrates the architecture of our virtual human's visual perception system for sketching, which is an ONV-to-sequence (ONV2seq) Variational Autoencoder (VAE). A trained DNN encoder compresses the sensed irradiance information in the ONV into a much smaller hidden state vector, from which the trained decoder reconstructs the viewed sketches as a sequence of finger strokes. The main differences between our model and the sketch-RNN model of Ha and Eck (2017) are in the DNN encoder and the loss function.

Our DNN encoder comprises two streams of fully connected layers, each of which processes the ONV from one eye, yielding two hidden state streams. Each stream is a 4-layer, fully-connected, tapered network, with 200, 100, 50, and 25 units in each hidden layer. The hidden states from both eyes are concatenated and projected into two vectors μ and σ, which are combined to produce a 128-dimensional latent random vector z that is conditioned on the input sketch.

The decoder is an autoregressive RNN that, from sketches represented by z, outputs the

finger strokes as a sequence of linear motor actions to successive screen coordinate offsets $(\Delta x_i, \Delta y_i)$ along with commands to touch the finger to the screen, to lift it from the screen, and to stop sketching.

Following Ha and Eck (2017), we model $(\Delta x_i, \Delta y_i)$ as a Gaussian mixture model

$$p(\Delta x, \Delta y) = \sum_{j=1}^{m} w_j \mathcal{N}(\Delta x, \Delta y | \mu_{x_j}, \mu_{y_j}, \sigma_{x_j}, \sigma_{y_j}, \rho_{xy_j}); \tag{6.2}$$

i.e., a weighted sum, with weights w_j, of m bivariate normal distributions $\mathcal{N}$ conditioned on the means μ_x and μ_y, standard deviations σ_x and σ_y, and correlation coefficient ρ_{xy}. The objective function for training the DNNs in the model is a reconstruction loss

$$L = L_s + L_p. \tag{6.3}$$

Here, the loss of the sketch movements is

$$L_s = -\frac{1}{n_{\max}} \sum_{i=1}^{n_s} \log\big(p(\Delta x_i, \Delta y_i)\big), \tag{6.4}$$

where n_s is the length of the sketch in movements and $n_{\max}$ is the total sequence length, and the loss of the sketch state is

$$L_p = -\frac{1}{N_{\max}} \sum_{i=1}^{N_{\max}} \sum_{k=1}^{3} p_{ki} \log(q_{ki}), \tag{6.5}$$

where p is a state variable such that $p = 1$ when the finger is in contact with the touchscreen, $p = 2$ when it is not in contact, and $p = 3$ when the drawing is finished.

We train our sketch ONV2seq model to minimize only the reconstruction loss L; i.e., to maximize the log-likelihood of the generated probability distribution of the training data. The loss is minimized using Adam stochastic optimization with minibatches of size 500 with the initial step size $\alpha = 0.00001$, gradually decaying with the training step at a decay rate of 0.995. The dropout rate for the fully connected layers in the encoder is 0.5.

Additional details are presented in Appendix C.

6.1.4 Sketching Demonstration

Figure 6.4 shows frames from a simulation in which the biomechanical human musculoskeletal model stands before a large touchscreen display. An auxiliary tablet displays images of drawings from the QuickDraw dataset (Jongejan et al., 2016). While the virtual human balances its body in gravity by virtue of its trained musculoskeletal controllers, it must observe its environment with its two eyes, supported by natural, muscle-actuated cervicocephalic movements. The virtual human controls itself autonomously, foveating the two displays as necessary, lifting its right arm to control the tablet display by swiping with its fingertip, and lifting its left arm and using its finger to select colors from a palette on the large display and sketch its interpretation (via its visual perception system described in the previous section) of the drawing that it sees displayed on the tablet. The scenario in the simulation evolves as follows:

Stepping up to the display to reach a comfortable working position (a), our virtual human (b) swipes the blank tablet screen with his right finger and (c) observes a cat reference image that appears on the tablet. (d) Continuing the observation, he prepares to sketch. Using his left arm and hand, he (e) selects the red color from the palette and (f) sketches the cat in red, (g) occasionally looking back to the screen to review the cat drawing. (h) He continues sketching the cat, (i) comparing the drawing on the tablet with his cat sketch. Next, he (j) swipes the tablet to display a tree, (k) prepares to draw the tree by (l) selecting the green color, then (m) sketches the tree in green, (n) occasionally looking back to the tablet to review the tree image and then (o) continuing to sketch the tree, and finally (p) compares the tree image on the tablet with the sketched tree. He then (q) swipes the tablet to display a bus, (r) prepares to draw the bus, (s) selects the blue color, (t) starts sketching the bus in blue, (u) occasionally looking back to the tablet to review the bus image. He (v) continues sketching the bus, (w) looks back to the tablet screen, (x) adds the final details to complete the bus sketch. Finally, he steps away from the touchscreen display.

In autonomously performing this scenario, our virtual human must deal with the varying external contact forces between its feet and the floor as it actuates its legs and arms as well

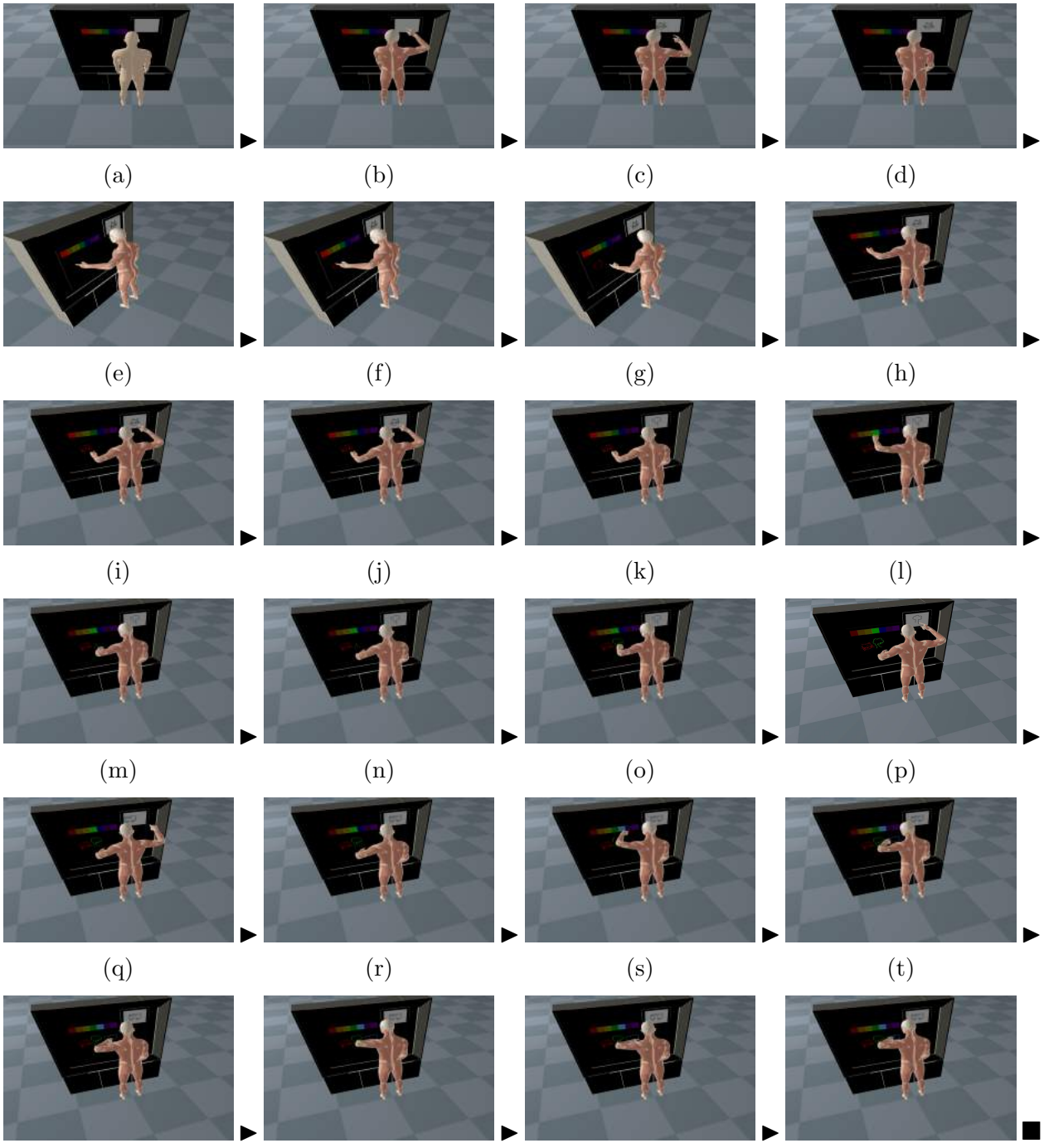


Figure 6.4: A sequence of frames from the sketching simulation. The virtual human controls itself autonomously.

as the external forces from its manual interaction with the two touchscreens.[1]

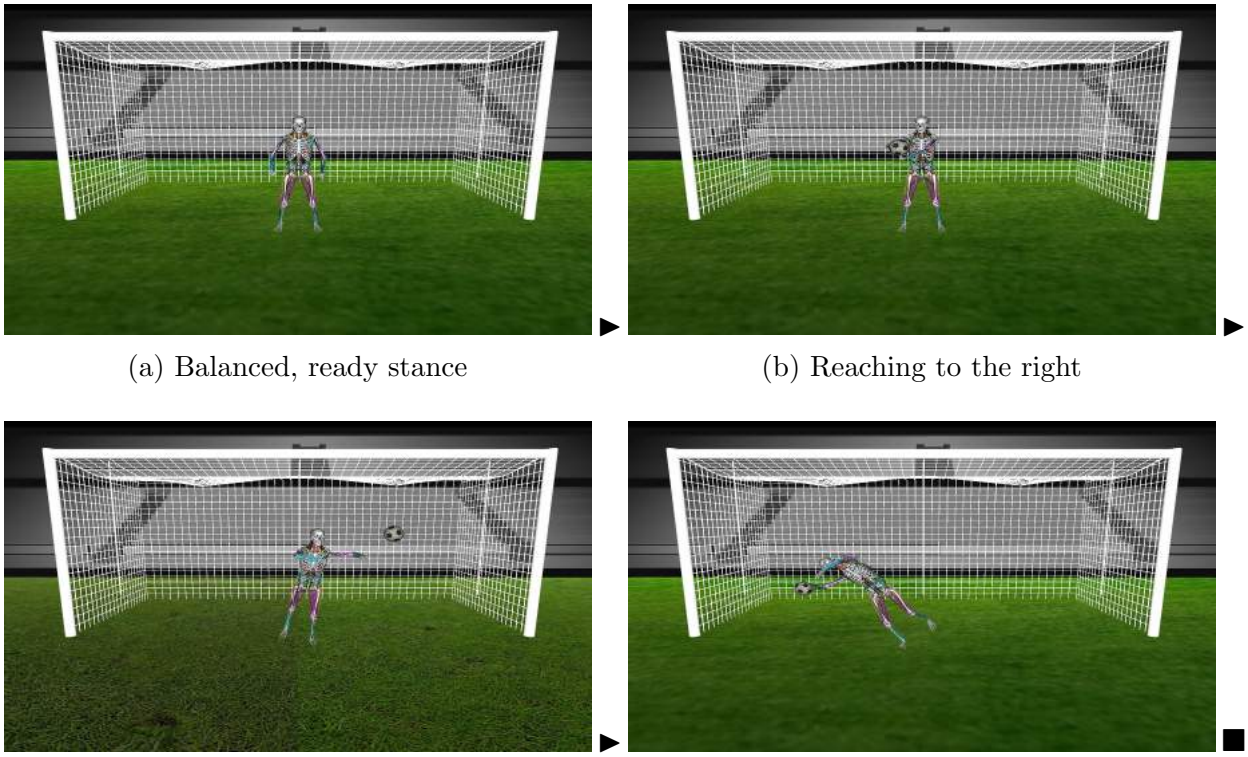
6.2 Autonomous Soccer Goaltending

Our biomechanical musculoskeletal model has served in validating the performance of an advanced biomechanical model of the eye in a novel sensorimotor control scenario, where the virtual human assumes the role of an autonomous soccer goaltender (Figure 6.5).

The biomechanical eye model, which was developed by Nakada et al. (2019), represents a significant improvement in fidelity over the simple, kinematic eye model described in the previous section, inasmuch as it includes functional submodels of the relevant optical organs (cornea, iris, deformable lens), a much higher-acuity retina, and its movements are actuated by six extraocular muscles driven by deep neuromuscular oculomotor controllers.

Equipped with two of these biomechanical eyes, our virtual human puts its binocular vision to use in a simulated soccer goaltending scenario, as shown in (Figure 6.5). While balancing its body in an upright ready stance, its eyes successfully foveate and visually pursue a moving target—the soccer ball. Our virtual human autonomously reacts to the approaching ball's trajectory, reaching out with its arms and possibly leaping at the ball in order to deflect it away from the goal, and its two eyes continue to visually track the moving ball while its head and body are in motion.

[1] In terms of the (transient spring-damper) interaction forces between the fingertips and the planes of the touchscreens, the maximum permitted penetration of the fingertip is set to 5 mm and the stiffness of the touchscreen is set to 10 N/mm. If the current fingertip position is beyond the allowed threshold depth, the virtual human adjusts its fingertip position via neuromuscular arm control.



(a) Balanced, ready stance

(b) Reaching to the right

(c) Springing to the left

(d) Leaping to the right

Figure 6.5: Sequence of frames from a simulated soccer goaltending scenario involving our autonomous biomechanical human musculoskeletal model incorporating biomimetic virtual eye models. Under deep neuromuscular oculomotor control, our goalie's eyes observe and persistently track incoming soccer balls by making saccadic foveation and smooth pursuit eye movements. The eye movements furthermore drive cervicocephalic head movements also under deep neuromuscular motor control. With its additional deep neuromuscular motor controllers, our goalie controls its torso, arms, and legs to reach out and even leap at approaching balls to deflect them away from the goal.

CHAPTER 7

Conclusion

The torso plays a major role in human motor control by transmitting voluntary control signals from the brain to the extremities to perform desired motor control tasks, while collecting proprioceptive sensory information as feedback. Indeed, controlling torso movement itself is nontrivial considering the complexities of the torso musculoskeletal complex, which includes more than half the muscles of the body and two thirds of its articular degrees of freedom. This thesis has addressed the challenge of biomechanically simulating and controlling the torso in its full musculoskeletal complexity, thus enabling a fully autonomous, whole-body biomechanical human model with an extensive set of articular degrees of freedom actuated by many hundreds of muscles. It would be a virtually insurmountable challenge to control such a complex musculoskeletal system using traditional optimization-based methods, considering the interaction joint forces between body parts and the contractile forces from muscles spanning multiple joints and connecting multiple musculoskeletal complexes. The only viable approach is the biomimetic one of neuromuscular control.

Our demonstration of the successful application of fully-connected deep neural networks to controlling the torso musculoskeletal complex is a primary contribution of this thesis. Our torso neuromuscular motor controller trains itself using data synthesized by the biomechanical torso musculoskeletal model, and it takes into consideration the interactions caused by the co-variation between the torso and the extremities resulting in natural, realistic whole-body human animation. Including our core neuromuscular motor controller with a voluntary/reflex DNN pair devoted to innervating the 443 muscles of the torso, the neuromuscular motor control system of our virtual human comprises a total of 12 trained deep neural networks (DNNs).

We have successfully validated our approach in several ways: For example, the upper-body calisthenic exercises require a stable bipedal stance while bending and rotating the torso, as does the golf putting task, and the sit-to-stand task requires even more challenging balance maintenance, as does the stepping task. We have also provided an unprecedentedly sophisticated demonstration of a fully autonomous, free-standing, biomechanical human musculoskeletal model with full sensorimotor control observing drawings with its eyes, analyzing its retinal percepts via its internal vision system, and sketching the drawings with its finger on a touchscreen.

We regard it a significant achievement spanning the fields of computer graphics and biomechanics that the six musculoskeletal complexes—torso, neck, arms, and legs—are now controlled by a dozen artificial neural networks trained on data synthesized by the virtual human model itself, analogous to real life.

7.1 Limitations and Future Work

7.1.1 Biomechanical, Muscle-Actuated Hands and Feet

For the time being, the hands and feet of our virtual human are animated in a purely kinematic matter. Their anatomical structures include complete bone and muscle geometries, but they presently lack the functional muscle actuators necessary to produce muscle-driven biomechanical hand and foot movements. It is worthwhile to explore the suitability of Hill-type muscle actuators for the hands and feet. If Hill-type actuators prove to be inadequate, one can turn to the strand musculotendon actuators of Sueda et al. (2008). In either case, in accordance with our paradigm, deep neuromuscular motor controllers will need to be developed and trained for the hands and feet.

7.1.2 Task-Specific Variance Structure

The inverse kinematics needed to calculate desired joint angles and desired muscle strains adopt an optimization method, which always provides a unique solution. From the per-

spective of motor control, this solution does not allow variance across trials (Latash et al., 2002).

In general, the Uncontrolled Manifold (UCM) hypothesis (Scholz and Schöner, 1999) assumes that the space of elemental variables may be divided into two subspaces, with one subspace (UCM) corresponding to no effect on the task-specific performance variables, whereas the other subspace is orthogonal to the UCM and does affect the performance variables. The variance in the UCM is V_{UCM}, whereas the variance in the space orthogonal to the UCM is V_{ORT}. Obviously, reducing V_{ORT} is desirable for tasks requiring high accuracy. Nevertheless, there is no obvious criterion for the V_{UCM}.

In fact, two factors make a larger V_{UCM} desirable. First, the selected elemental variables for one task are also involved in other tasks, thus contributing to the stability of other performance variables. A large V_{UCM} indicates a larger solution space with regard to such performance variables. If the solution space is small, any secondary task would require new solutions for the first task. Thus, given a large solution space, the secondary task can also be performed without affecting the original task (Zhang et al., 2008). Second, unpredictable changes both within the body and in the environment always influence the performance of any task. A large V_{UCM} is able to mediate the effects of such perturbations (Mattos et al., 2011).

To mimic real human movement, a concept of back coupling should be introduced to solve the motor redundancy problem and allow UCM variance (Martin et al., 2009, 2019).

7.1.3 Active Balance and Locomotion

Our neuromuscular controllers are capable of controlling our musculoskeletal model to sit, stand up, and balance upright; however, beyond stepping, our virtual human cannot yet perform continuous bipedal locomotion. Our next goal is to enable it to locomote and navigate its environment. Balancing strategies from animation and robotics may be incorporated into our biomechanical human model. Although a challenging proposition given its musculoskeletal complexity, this is an exciting research direction.

In terms of active balance control, certain balance controllers define objectives in terms of Center of Pressure (CoP) (Abdallah and Goswami, 2005), while others use momenta (Kajita et al., 2003). Although Ground Reaction Forces (GRF) and Center of Pressure (CoP) have a one-to-one relationship with the rate of change of spatial momentum, their physical meanings for balance differ. Whereas the CoP relates to robot motion, the GRF characterizes the constraints of the ground contact.

As for locomotion task, two categories of control schemes have been proposed. The first use per-joint PD servos, coordinated by a high-level state machine (Yin et al., 2007; Faloutsos et al., 2001a; Hodgins et al., 1995). However, the combination of high-gain tracking and discrete state machines frequently leads to rigid motions that can be very difficult to tune to produce natural-looking full-body movements. The second category of methods take biologically inspired approaches, such as Central Pattern Generators (CPGs), which are neural circuits capable of producing coordinated patterns of rhymic activity without any rhythmic inputs from sensory feedback or from higher control centers. The method has been successfully applied to multi-legged locomotion of Myriapoda (Fang et al., 2013) as well as to the control of swimming locomotion with a sophisticated biomechanical human model, the precursor to ours (Si et al., 2014). Low-level CPGs should be integrated into our free-standing biomechanical human model to drive bipedal locomotion.

7.1.4 Reinforcement Learning

Our neuromuscular motor controllers are trained offline using fully-connected deep neural networks. This gives the baseline capability of performing fundamental tasks such as reaching and standing. In order to achieve more natural motions with a greater variety of tasks, we plan to implement online learning methods with Deep Reinforcement Learning (DRL), which promises to give our model the ability to learn continuously from its experience. Such learning processes are especially important for motions that come with clear objectives such as playing sports, which humans continuously practice to improve their skills.

Previous studies have successfully applied DRL to articulated skeletal control (Peng et al.,

2017, 2018). However, these methods adopt deep reinforcement learning only at joint level while they apply supervised learning at the muscle level. While it seems infeasible to apply DRL to control on the order of 1,000 muscle actuators directly, the brain does not control muscles individually but unifies them into groups that are controlled in a synergistic manner (Jackson, 1889; Bizzi and Cheung, 2013; Ting and Macpherson, 2005). Synergies are considered building blocks that the central nervous system uses to mitigate its computational burden. It seems promising to decrease the dimensionality of the task by applying DRL directly to synergistic muscle groups. Furthermore, DRL can be applied to visual perception tasks, such as for doodling (see Appendix D).

APPENDIX A

Synthesizing Training Data

The DNNs that implement the voluntary neuromuscular controllers in our sensorimotor control system are trained offline in a supervised manner. Once trained, the DNNs can quickly produce the required muscle activation signals online.

We synthesize training data using our biomechanical human musculoskeletal system simulator. Given the current state of the musculoskeletal system, the current muscle activations, and a desired target state, we compute an appropriate correction to each muscle activation in order to drive the musculoskeletal system closer to the target state, subject to the downward pull of gravity.

For example, if the end effector is in some given pose and we want it to achieve a new pose, we first compute the desired changes of the joint angles $\boldsymbol{q}$ by solving an inverse kinematics problem. Second, we compute a desired acceleration for each joint to achieve the desired motion in a given time step. Third, we compute the required joint torques by solving the inverse dynamics problem for each joint, subject to external forces, such as gravity. Finally, a muscle optimization technique is applied to compute the minimal muscle activations that can generate the desired torque for each joint.

For the purposes of training the neural network, the input is the concatenation of the desired movement vector $\boldsymbol{e} = \boldsymbol{p}_d - \boldsymbol{p}_c$ between the current pose $\boldsymbol{p}_c$ and desired pose $\boldsymbol{p}_d$, and current muscle activations a_c, while the desired output of the network in response to this input is the change of activation $\Delta\boldsymbol{a} = \boldsymbol{a}_d - \boldsymbol{a}_c$, where $\boldsymbol{a}_d$ is a desired muscle activation; i.e.,

$$\text{Training input-output pair} \begin{cases} \text{input:} & [\boldsymbol{e}, \boldsymbol{a}_c]; \\ \text{output:} & \Delta\boldsymbol{a}. \end{cases} \tag{A.1}$$

This constitutes a single training pair for the network.

We iteratively update the joint angles in a gradient descent manner such that the difference between the current pose $\boldsymbol{q}$ and target pose $\boldsymbol{q}^*$ is minimized. The controller determines required accelerations to reach the target at each time step h using the following PD function:

$$\ddot{\boldsymbol{q}}^* = k_p(\boldsymbol{q}^* - \boldsymbol{q}) + k_d(\dot{\boldsymbol{q}}^* - \dot{\boldsymbol{q}}), \tag{A.2}$$

with proportional $k_p = 2(1-\gamma)/h^2$ and derivative $k_d = 2/h$ gains and error reduction rate γ. We set $h = \gamma = 0.1$.

We use the hybrid recursive dynamics algorithm due to Featherstone (2014), which makes it possible to compute the desired accelerations for acceleration-specified joints and the desired torques for torque-specified joints, as follows: First, we set the muscle-driven joints as acceleration-specified joints, while the passive joints remain torque-specified joints. We compute the desired accelerations for the muscle-driven joints and run the hybrid dynamics algorithm to compute the resulting accelerations for the passive joints. Second, we advance the system to the next time step in accordance with the first-order implicit Euler method, as was explained in the previous section, and use the hybrid dynamics algorithm to compute the required torques for the muscle-driven joints, thus obtaining the desired torques for the muscle-driven joints and accelerations for the passive joints.

After the desired torques are obtained, an optimization problem for agonist and antagonist muscles is solved to compute the desired minimal muscle activation levels.

Further details about the above numerical techniques can be found in (Lee et al., 2009) and Chapter 3.

APPENDIX B

Rendering

We use the open-source POV-Ray software to render our simulations. We first run the simulation in OpenSceneGraph and store the whole scene except for the musculoskeletal model into a .osg file. The conversion from the internal OpenSceneGraph visualization data to external rendering systems required custom code to output Wavefront OBJ and POV-Ray POV files. The implementation of this was mainly comprised of several interacting C++ software modules following the visitor design pattern. Special care was taken to ensure the geometry and material properties matched before and after the conversion to all desired files types. However, this was not always generically possible. For example, the POV filetype does not support line shapes. Instead, we converted to cylinders with a predefined radius, empirically chosen to be in the range 1.0–5.0, which was satisfactory.

We also store the kinematic state (joint angles and velocities) and import it to the flesh simulator. The simulation system we use is based on PhysBAM (Dubey et al., 2011), which is an object-oriented C++ library capable of solving a variety of problems in computational dynamics, computational mechanics, computer graphics, and computer vision. The geometries of the skin, muscles, and skeleton are embedded into the hexahedra of the embedding volume (Patterson et al., 2012). This embedding framework preserves high resolution geometry for rendering. The scene script of POV-Ray contains definitions of the camera, illumination, background, and material properties of all the geometric entities. We add the flesh model and other objects, such as the putter or touchscreen, into the scene. It takes around 10 seconds to render one frame of 1,920 $\times$ 1,080 pixels on a single core of our 3.2 GHz Intel Core i7-3930k computer. POV-Ray enables adjustment of the transparency of the skin to show an opaque or translucent skin surface that reveals the underlying flesh/muscle model.

APPENDIX C

ONV2seq: Biomimetic Perception Learning for Sketch Generation

Our raw datasets came from QuickDraw, a public sketch database built by Google. All the sketch sequence data were collected by The Quick, Draw!, an online game that requires participants to draw a sketch within 20 seconds. We use a data format that represents a sketch as a set of finger stroke actions. In this data, the initial absolute coordinate of the drawing is located at the origin of the finger. A sketch is a list of points, and each point is a vector consisting of 5 elements: (Δx, Δy, p_1, p_2, p_3). The first two elements are the offset distance in the x and y directions of the finger from the previous point. The last 3 elements represents a binary one-hot vector of 3 possible states. The first finger state, p_1, indicates that the finger is currently touching the screen, and that a line will be drawn connecting the next point with the current point. The second finger state, p_2, indicates that the finger will be lifted from the touchscreen after the current point, and that no line will be drawn next. The final finger state, p_3, indicates that the drawing has ended, and subsequent points, including the current point, will not be rendered. The number of training samples for each category is 70,000, while the validation and test samples are both 2,500 samples.

The hidden state extracted from the images is project to two vectors μ and σ. We use μ and σ, along with $N(0, I)$, a vector of IID Gaussian variables of size N_z to construct a random vector, $z \in R^{N_z}$ in the classical VAE method (Kingma and Welling, 2013): $z = \mu + \sigma \odot N(0, I)$.

At each step i of the decoder RNN, the previous point x_{i-1} and the latent vector z are presented as a concatenated input, where x_0 is defined as (0,0,1,0,0). The output at each time step comprises parameters of the probability distribution of the next data point x_i.

We model $(\Delta x, \Delta y)$ as a Gaussian Mixture Model (GMM) with M normal distributions and $(q_1,\ q_2,\ q_3)$ as a categorical distribution to model the ground truth data $(p_1,\ p_2,\ p_3)$, where $q_1 + q_2 + q_3 = 1$. The generated sequence is conditioned from a latent code z sampled from our encoder, which is trained end-to-end alongside the decoder:

$$p(\Delta x, \Delta y) = \sum_{j=1}^{M} \Pi_j N(\Delta x, \Delta y | \mu_{x,j}, \mu_{y,j}, \sigma_{x,j}, \sigma_{y,j}, \rho_{xy,j}), \tag{C.1}$$

where

$$\sum_{n=1}^{M} \Pi_j = 1. \tag{C.2}$$

Here, $N(\Delta x, \Delta y | \mu_{x,j}, \mu_{y,j}, \sigma_{x,j}, \sigma_{y,j}, \rho_{xy,j})$ is the probability distribution function for a bivariate normal distribution; each of the M bivariate normal distributions consist of five parameters: $(\mu_x,\ \mu_y,\ \sigma_x,\ \sigma_y,\ \rho_{xy})$, where μ_x and μ_y are the means, σ_x and σ_y are the standard deviations, and ρ_{xy} is the correlation parameter of each bivariate normal distribution. The vector Π contains the mixture weights of the Gaussian mixture model.

During data preparation, all sequences are generated to a length of N_{max} where N_{max} is a preset length of the sketch in the training dataset. Since the length of x is always shorter than N_{max}, we set x_i to be (0, 0, 0, 0, 1) for $i > N_s$, where N_s is the sketch length. During the sampling process, we generate the parameters for both GMM and categorical distributions at each time step, and sample an outcome x_i for each time step. The sampled outcome x_i is fed as the input for the next time step.

The training loss is to maximize the log-likehood of the generated probability distribution to explain the training data x. The reconstruction loss

$$L_R = L_s + L_p \tag{C.3}$$

has two components—the offset term $(\Delta x, \Delta y)$

$$L_s = -\frac{1}{N_{max}} \sum_{i=1}^{N_s} \log \left(\sum_{j=1}^{M} \Pi_{j,i} N(\Delta x_i, \Delta y_i | \mu_{x,j,i}, \mu_{y,j,i}, \sigma_{x,j,i}, \sigma_{y,j,i}, \rho_{xy,j,i}) \right) \tag{C.4}$$

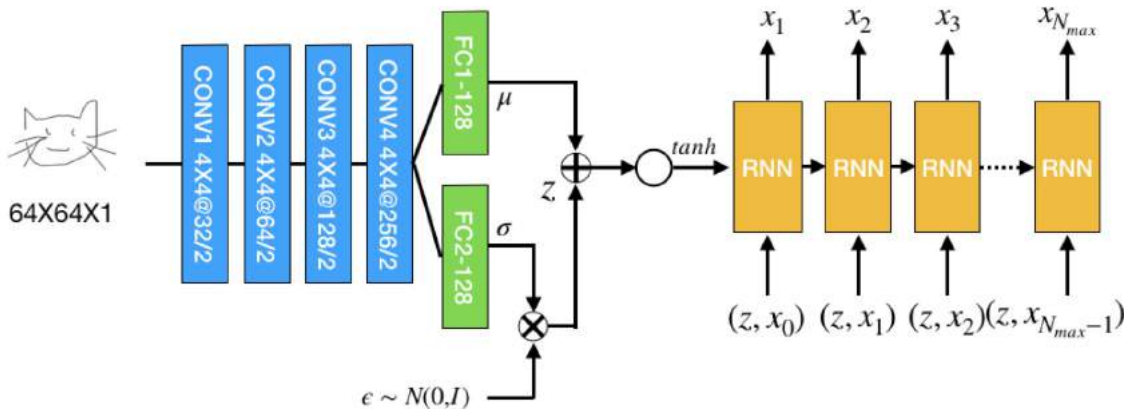


Figure C.1: Model Architecture of sketch-pix2seq

and the finger state terms (p_1, p_2, p_3)

$$L_p = -\frac{1}{N_{max}} \sum_{i=1}^{N_{max}} \sum_{k=1}^{3} p_{k,i} \log(q_{k,i}). \tag{C.5}$$

Note that points beyond N_s is not used for modeling the offset term $(\Delta x, \Delta y)$ while the finger state terms require all of the PDF parameters until N_{max}.

The images fed into the encoder of the sketch-pix2seq model (Figure C.1) were produced by first converting the raw sequences to vector image files (.svg) and then converting to gray-scale pixel image files (.png, .jpg, etc.) of size 64×64. The ONV data used by the encoder of the sketch-ONV2seq model (Figure C.2) were converted from the pixel image files of 600×600 after augmenting the stroke width and adding sufficient padding space such that more image detail can be captured by the retinal photoreceptors.

Figure C.1 shows the model architecture for sketch-pix2seq, which used a gray-scale pixel image as the neural network input. Our model architecture is shown in Figure C.2. It is a ONV2seq Variational Autoencoder (VAE), which has similar structure to sketch-RNN and sketch-pix2seq. The main difference in the network architecture lies in the DNN encoder, which consists of two parallel channels of four-layer fully-connected networks, rather than a CNN-based network.

We conducted experiments for three models—sketch-ONV2seq, sketch-ONV2seq (pre-trained decoder of sketch-pix2seq), and sketch-ONV2seq (pretrained decoder of sketch-

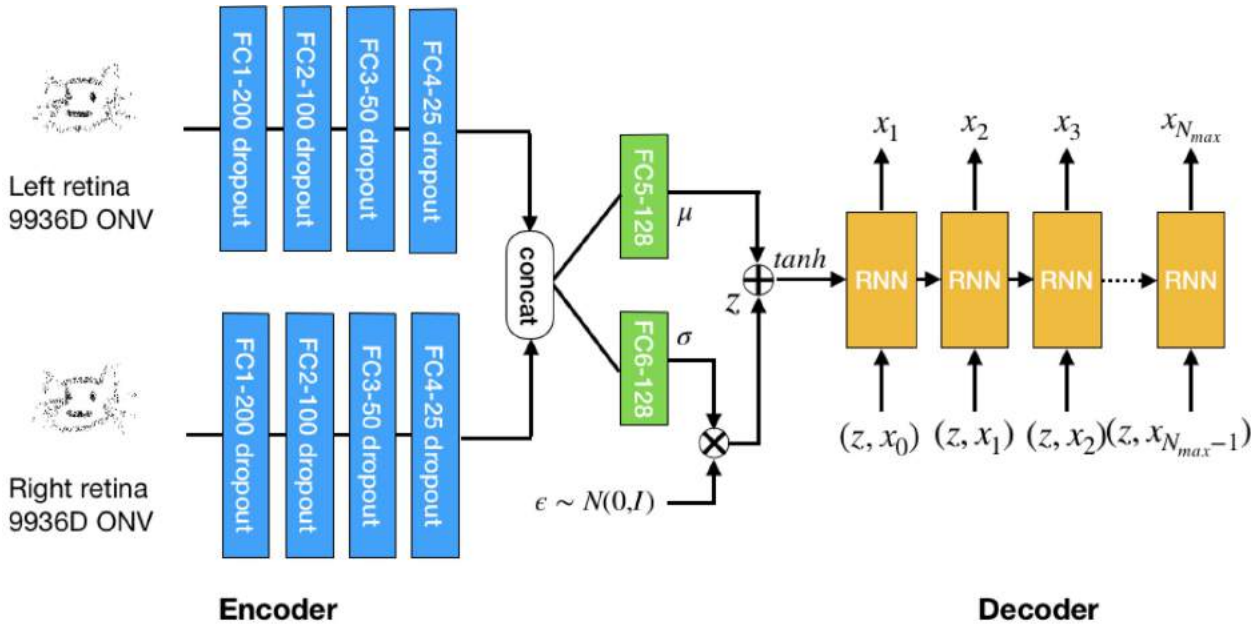


Figure C.2: Model Architecture of sketch-ONV2seq

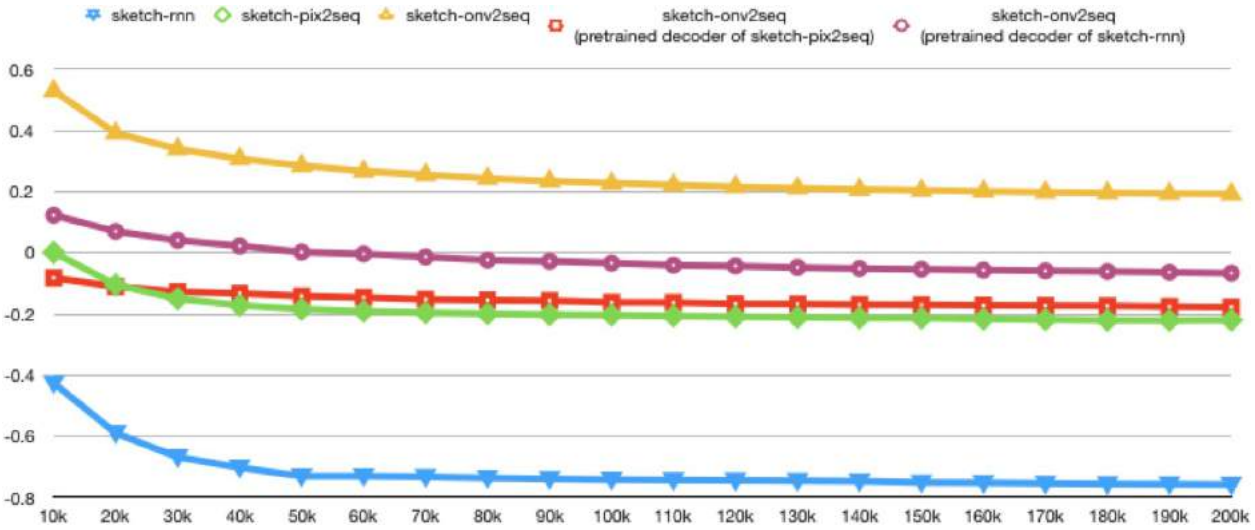


Figure C.3: Evaluation cost of models over training epochs

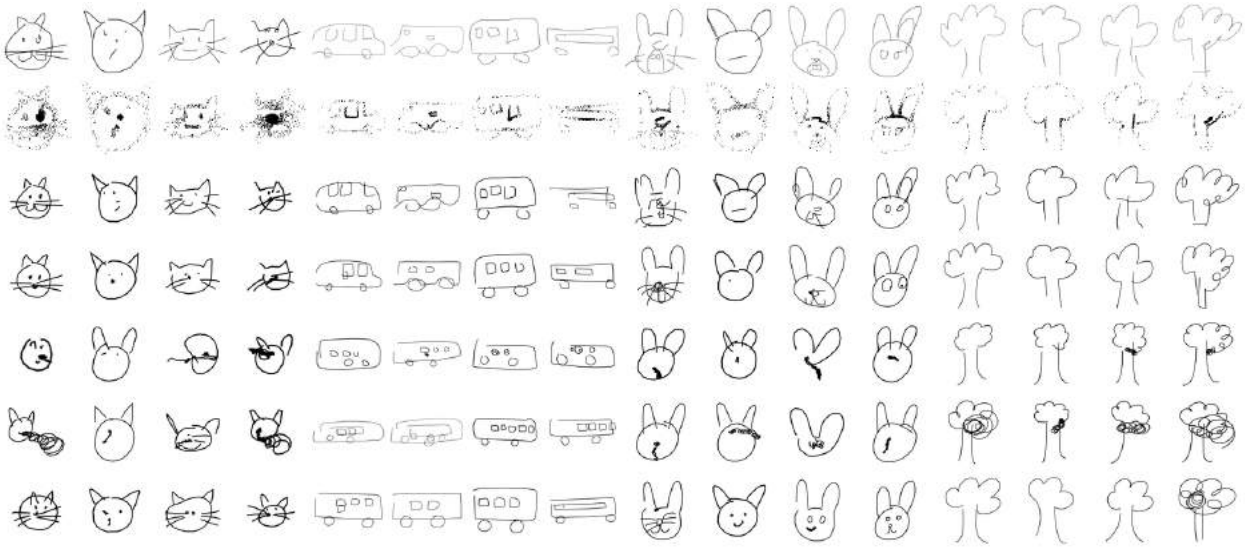


Figure C.4: The first row contains hand-drawn sketches (original test samples) and the second row is the ONV (of left retina) generated based on the original test samples. Rows 3–7 are the reconstructed sketches based on the models: (3) sketch-RNN, (4) sketch-pix2seq, (5) sketch-ONV2seq, (6) sketch-ONV2seq (pretrained decoder of sketch-RNN), (7) sketch-ONV2seq (pretrained decoder of sketch-pix2seq).

RNN)—and compared them with the sketch-RNN and sketch-pix2seq models. The sketch-ONV2seq is trained from the scratch with all the weights initialized with a uniform random distribution. The decoder weights of sketch-ONV2seq (pretrained decoder of sketch-pix2seq) and sketch-ONV2seq (pretrained decoder of sketch-RNN) are initialized with the pretrained weights of the sketch-pix2seq and sketch-RNN models, respectively, while the rest of the weights are randomly initialized.

For simplicity, we selected four categories—cat, bus, rabbit and tree—to carry out the experiments. Figure C.3 plots the evaluation loss for each training epoch. As is shown in the figure, sketch-ONV2seq performs the worst and it has the highest cost. Furthermore, sketch-ONV2seq also converges most slowly when training from scratch. This is expected as the fully-connected layers are not as good as convolutional layers at capturing image features. However, if we initialize the decoder weights for the sketch-ONV2seq models using the one in sketch-pix2seq, convergence is significantly faster.

Figure C.4 is the generated sketch result based on the input of selected test samples. The generated sketches of the sketch-pix2seq model looks the best and those of the sketch-

ONV2seq (pretrained decoder of sketch-pix2seq) look similar. Furthermore, although the sketch-ONV2seq model can achieve lower loss using the pretrained sketch-RNN decoder, its visual result is still disappointing. This might be because the RNN encoder in the sketch-RNN model fails to capture local image features as well as the DNN encoder and the CNN encoder does, but it captures the sequence order information instead. Thus their encoded latent vectors may have quite different distributions and the trained decoder weights based on the latent vectors are unlikely to be suitable for transfer to the other models. Another thing to note is that the sketch-RNN model has the lowest loss, but its reconstructed sketches are not the best. According to Nakada et al. (2018), the sketches reconstructed by the sketch-pix2seq model look more 'human-like' (likely to be drawn by a human) than the sketch-RNN model, but its construction loss is larger, which indicates that the loss is not correlated to how 'human-like' the sketch is.

APPENDIX D

Learning to Doodle with Deep Q-Networks and Demonstrated Strokes

This appendix reproduces publication (Zhou et al., 2018b).

Doodling is a useful and common intelligent skill that people can learn and master. In this work, we propose a two-stage learning framework to teach a machine to doodle in a simulated painting environment via Stroke Demonstration and deep Q-learning (SDQ). The developed system, *Doodle-SDQ*, generates a sequence of pen actions to reproduce a reference drawing and mimics the behavior of human painters. In the first stage, it learns to draw simple strokes by imitating in supervised fashion from a set of stroke-action pairs collected from artist paintings. In the second stage, it is challenged to draw real and more complex doodles without ground truth actions; thus, it is trained with Q-learning. Our experiments confirm that (1) doodling can be learned without direct step-by-step action supervision and (2) pretraining with stroke demonstration via supervised learning is important to improve performance. We further show that *Doodle-SDQ* is effective at producing plausible drawings in different media types, including sketch and watercolor. A short video can be found at https://www.youtube.com/watch?v=-5FVUQFQTaE.

D.1 Introduction

Doodling is a common, simple, and useful activity for communication, education, and reasoning. It is sometimes very effective at capturing complex concepts and conveying complicated ideas Brown (2014). Doodling is also quite popular as a simple form of creative art, compared to other types of fine art. We all learn, practice, and master the skill of doodling in

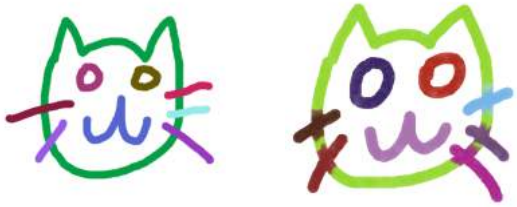


Figure D.1: Cat doodles rendered using color sketch (left) and water color (right) media types.

one way or another. Therefore, for the purposes of building a computer-based doodling tool or enabling computers to create art, it is interesting and meaningful to study the problem of teaching a machine to doodle.

Recent progress in visual generative models—e.g., Generative Adversarial Networks Goodfellow et al. (2014) and Variational Autoencoders Kingma and Welling (2013)—have enabled computer programs to synthesize complex visual patterns, such as natural images Denton et al. (2015), videos Vondrick et al. (2016), and visual arts Elgammal et al. (2017). By contrast to these efforts, which model pixel values, we model the relationship between pen actions and visual outcomes, and use that to generate doodles by acting in a painting environment. More concretely, given a reference doodle drawing, our task is to doodle in a painting environment so as to generate a drawing that resembles the reference. In order to facilitate the experiment setup and be more focused and expedient on algorithm design, we employ an internal Simulated Painting Environment (SPE) that supports major media types; for example, sketch and watercolor (Figure D.1).

Our seemingly simple task faces at least *three challenges*:

First, our goal is to enable machines to doodle like humans. This means that rather than mechanically printing pixel by pixel like a printer, our system should be able to decompose a given drawing into strokes, assign them a drawing order, and reproduce the strokes with pen action sequences. These abilities require the system to visually parse the given drawing, understand the current status of the canvas, make and adjust drawing plans, and implement the plans by invoking correct actions in a painting environment. Rather than designing a

rule-based or heuristic system that is likely to fail in corner cases, we propose a machine learning framework for teaching computers to accomplish these tasks.

The second challenge is the lack of data to train such a system. The success of modern machine learning heavily relies on the availability of large-scale labeled datasets. However, in our domain, it is expensive, if not impossible, to collect paintings and their corresponding action data (i.e., recordings of artists' actions). This is compounded by the fact that the artistic paintings space features rich variations, including media types, brush settings, personal styles, etc., that are difficult to cover. Hence, the traditional paradigm of collecting ground truth data for model learning does not work in our case.

Consequently, we propose a hybrid learning framework that consists of two stages of training, which are driven by different learning mechanisms. In Stage 1, we collect stroke demonstration data, which comprises a picture of randomly placed strokes and its corresponding pen actions recorded from a painting device, and train a model to draw simple strokes in a supervised manner. Essentially, the model is trained to imitate human drawing behaviour at the stroke level with step-by-step supervision. Note that it is significantly easier to collect human action data at the stroke level than for the entire painting. In Stage 2, we challenge the model learned in Stage 1 with real and more complex doodles, for which there are no associated pen action data. To train the model, we adopt a Reinforcement Learning (RL) paradigm, more specifically Q-learning with reward for reproducing a given reference drawing. We name our proposed system *Doodle-SDQ*, which stands for Doodle with Stroke Demonstration and deep Q-Networks. We experimentally show that both stages are required to achieve good performance.

Third, it is challenging to induce good painting behaviour with RL due to large state/action space. At each step, the agent faces at least 200 different action choices, including the pen state, pen location, and color. The action space is larger than in other settings where RL has been applied successfully Mnih et al. (2015); Peng et al. (2016); Levine et al. (2016). We empirically observe that Q-learning with a high probability of random exploration is not effective in our large action space, and reducing the chance of random exploration significantly helps stabilize the training process, thus improving the accumulated reward.

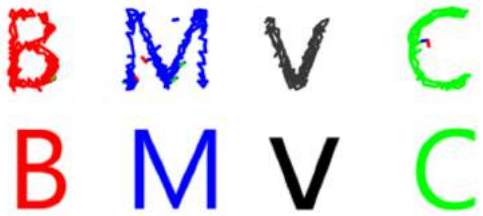


Figure D.2: Sketch drawing examples: **BMVC**. (top) The images produced by unrolling the Doodle-SDQ model for 100 steps. (bottom) The corresponding reference images.

To summarize, Doodle-SDQ leverages demonstration data at the stroke level and generates a sequence of pen actions given only reference images. Our algorithm models the relationship between pen actions and visual outcomes and works in a relatively large action space. We apply our trained model to draw various concepts (e.g., characters and objects) in different media types (e.g., black and white sketch, color sketch, and watercolor). In Figure D.2, our system has automatically sketched a colored "BMVC".

D.2 Related Work

D.2.1 Imitation Learning and Deep Reinforcement Learning

Imitation learning techniques aim to mimic human behavior in a given task. An agent (a learning machine) is trained to perform a task from demonstrations by learning a mapping between observations and actions Hussein et al. (2017). Naive imitation learning, however, is unable to help the agent recover from its mistakes, and the demonstrations usually cannot cover all the scenarios the agent will experience in the real world. To tackle this problem, DAGGER Ross et al. (2011) iteratively produces new policies based on polling the expert policy outside its original state space. Therefore, DAGGER requires an expert to be available during training to provide additional feedback to the agent. When the demonstration data or the expert are unavailable, RL is a natural choice for an agent to learn from experience by exploring the world. Nevertheless, reward functions have to be designed based on a large number of hand-crafted features or rules Xie et al. (2012).

The breakthrough of Deep RL (DRL) Mnih et al. (2015) came from the introduction of a target network to stabilize the training process and experience replay to learn from past experiences. Hasselt et al. (2016) proposed *Double DQN* (DDQN) to solve an over-estimation issue in deep Q-learning due to the use of the maximum action value as an approximation to the maximum expected action value. Schaul et al. (2016) developed the concept of *prioritized experience replay*, which replaced DQN's uniform sampling strategy from the replay memory with a sampling strategy weighted by TD errors. Our algorithm starts with Double DQN with prioritized experience replay (DDQN + PER) Schaul et al. (2016).

Recently, there has also been interest in combining imitation learning with the RL problem Cruz Jr et al. (2017); Subramanian et al. (2016). Silver et al. (2016) trained human demonstrations in supervised learning and used the supervised learner's network to initialize RL's policy network while Hester et al. (2018) proposed Deep Q-learning from Demonstrations (DQfD), which leverages even very small amounts of demonstration data to accelerate learning dramatically.

D.2.2 Sketch and Art Generation

There are outstanding studies related to drawing in the fields of robotics and AI. Traditionally, a robot arm is programmed to sketch lines on a canvas to mimic a given digitized portrait Tresset and Leymarie (2013). Calligraphy skills can be acquired via Learning from Demonstration Sun et al. (2014). Recently, Deep Neural Network-based approaches for art generation have been developed Gatys et al. (2016); Elgammal et al. (2017). An earlier work by Gregor et al. (2015) introduced a network combining an attention mechanism with a sequential auto-encoding framework that enables the iterative construction of complex images. The high-level idea is similar to ours; that is, updating only part of the canvas at each step. Their method, however, operates on the canvas matrix while ours generates pen actions that make changes to the canvas. More recently, a SPIRAL model Ganin et al. (2018) used Reinforced Adversarial Learning to produce impressive drawings without supervision; however, the model generates control points for quadratic Bezier curves, rather than directly

controlling the pen's drawing actions.

Rather than focusing on traditional pixel image modeling approaches, Zhang et al. (2017) and Simhon and Dudek (2004) proposed generative models for vector images. Graves (2013) focused on handwriting generation with Recurrent Neural Networks to generate continuous data points. Following the handwriting generation work, a sketch-RNN model was proposed to generate sketches Ha and Eck (2017); Jongejan et al. (2016), which was learned in a fully supervised manner. The features learned by the model were represented as a sequence of pen stroke positions. In our work, we process the sketch sequence data and, using an internal simulated painting environment, render onto the canvas as in the reference images.

D.3 Methodology

Given a reference image and a blank canvas for the first iteration, our Doodle-SDQ model predicts the pen's action. When the pen moves to the next location, a new canvas state is produced. The model takes the new canvas state as the input, predicts the action based on the difference between the current canvas and the reference image, and repeats the process for a fixed number of steps. (Figure D.3a).

D.3.1 Our Model

The network has two input streams (Figure D.3b-A). The global stream has 4 channels, which comprise the current canvas, the reference image, the distance map and the color map. The distance map and the color map encode the pen's position and state. The local stream has 2 channels—the cropped patch of the current canvas centered at the pen's current location with size equal to the pen's movement range, and the corresponding patch on the reference image. Unlike the classical DQN structure Mnih et al. (2015), which stacks four frames, the input in this model includes only the current frame and no history information.

The convnet for global feature extraction consists of three convolutional layers Mnih et al. (2015). The first hidden layer convolves 32 8×8 filters with stride 4. The second hidden

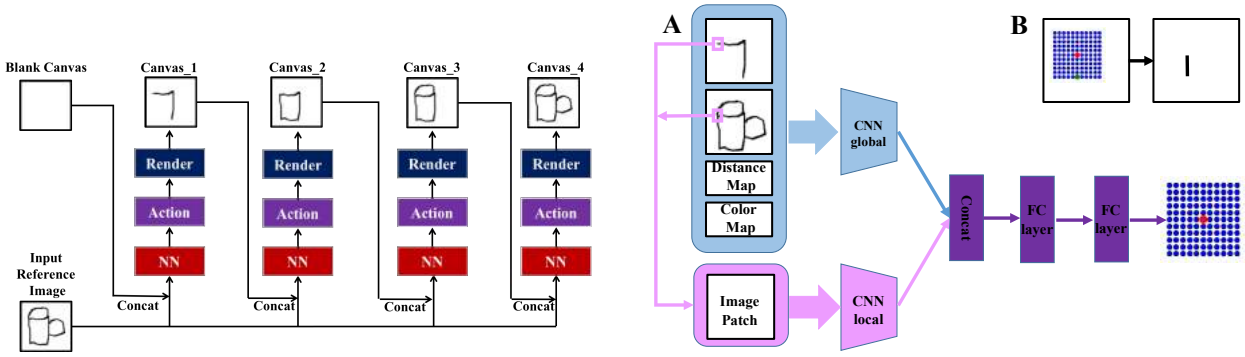


Figure D.3: Doodle-SDQ structure. (a) The algorithm starts with a blank canvas and an input reference image. The neural network predicts the action of the pen and sends rendering commands to a painting engine. The new canvas and the reference image are then concatenated and the process is repeated for a fixed number of steps. (b) A: Two CNNs extract global scene-level contextual features and local image patch descriptors. The local and global features are concatenated for action prediction. B: Given the current position (red dot) and the predicted action (green dot), the painting engine renders a segment to connect them. The rectangle of blue dots represents the movement range, which is the same size as the local image patch.

layer convolves 64 4×4 filters with stride 2. The third hidden layer convolves 64 3×3 filters with stride 1. The only convnet layer of the local CNN stream convolves 128 11×11 filters with stride 1. The two streams are then concatenated and input to a fully-connected linear layer, and the output layer is another fully-connected linear layer with a single output for each valid action.

At each time step, the pen must decide where to move (Figure D.3b-B). The pen is designed to have maximum 5 pixels offset movement horizontally and vertically from its current position.[1] Therefore, the movement range is 11×11 and there are in total 121 positional choices.

The pen's state is determined by the type of reference image. Specifically, the pen's state is either up or down (i.e., draw) for a grayscale image. For a color image, the pen's state can be up or down with a color selected from the three options (i.e., red, green, and blue).[2]

[1]The maximal offset movement of the pen is set arbitrarily; it could also be 4 or 6.

[2]The painting engine allows more colors; however, to simplify our experiments, we limit it to three colors.

Image	Input global stream	Input local stream	Output action space
Grayscale	$84 \times 84 \times 4$	$11 \times 11 \times 2$	$11 \times 11 \times 2 = 242$
RGB	$84 \times 84 \times 8$	$11 \times 11 \times 6$	$11 \times 11 \times 4 = 484$

Table D.1: Input and output dimensionalities.

Therefore, the dimension of the action space is 242 for grayscale images and 484 for color images. Figure D.3b-B shows a segment rendered given the pen's current position and the predicted action.

Rather than memorizing absolute coordinates of a pen on a canvas, humans tend to encode the relative positions between points. To represent the current location of the pen, an L2 distance map is constructed by computing

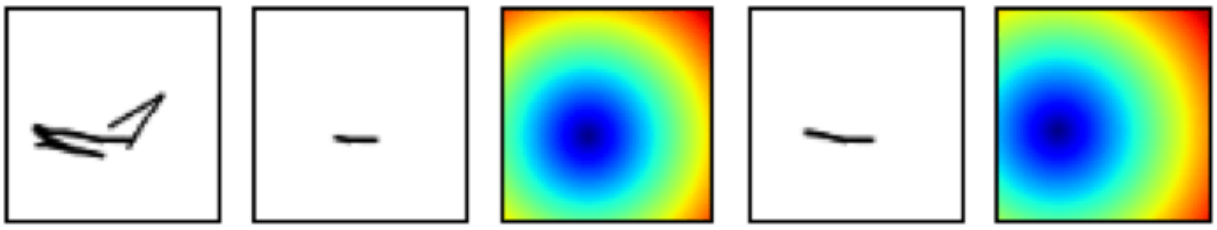
$$D(x, y) = \frac{\sqrt{(x - x_o)^2 + (y - y_o)^2}}{L}, \quad \forall (x, y) \in \Omega, \tag{D.1}$$

where Ω denotes the canvas which is an $L \times L$ discrete grid, L being the length of the canvas' side, and (x_o, y_o) is the current pen location. In terms of a color map, all elements are 1 when the pen is put down and 0 when the pen is lifted up for grayscale images. For an image with red, green, and blue color, all elements are 0 when the pen is lifted up, 1 for red color drawing, 2 for green color and 3 for blue color. The size of distance map and the color map is the same as the canvas size, which is 84×84 (Figure D.3b-A). Table D.1 summarizes the dimensionalities of the input and output for grayscale or color reference images.

D.3.2 Pre-Training Networks Using Demonstration Strokes

DRL can be difficult to train from scratch. Therefore, we pre-train the network in a supervised manner using synthesized data with ground truth actions. The synthetic data are generated by randomly placing real strokes on canvas (Figure D.4a). The real strokes are collected from recordings of a few artist paintings.

In the learning from demonstration phase, each training sample consists of the reference image (Figure D.4a), the current canvas (Figure D.4b), the color map, the distance map



(a) (b) (c) (d) (e)

Figure D.4: Data preparation for pre-training the network. (a) A reference image comprising two strokes randomly placed on the canvas; (b) the current canvas as part of the reference image; (c) the distance map of the current canvas, whose center is the pen's location on the current canvas; (d) the next step canvas after a one step action of the pen; (e) The distance map of the next step canvas, which represents the pen's location on the next step canvas.

(Figure D.4c), the small patch of the reference image, and the current canvas. The ground truth output will be the drawing action producing Figure D.4d from Figure D.4b. After training, the learned weights and biases are used to initialize the Doodle-SDQ network in the RL stage.

D.3.3 Doodle-SDQ

To encourage the agent to draw a picture similar to the reference image, the similarity between the k^{th} step canvas and the reference image is measured as

$$s_k = \frac{\sum_{i=1}^{L}\sum_{j=1}^{L}(P_{ij}^{k} - P_{ij}^{\text{ref}})^2}{L^2}, \tag{D.2}$$

where P_{ij}^{k} is the pixel value at position (i, j) in the k^{th} step canvas and P_{ij}^{ref} is the pixel value at that position in the reference image.

The pixel reward of executing action at the k^{th} step is defined as

$$r_{\text{pixel}} = s_k - s_{k+1}. \tag{D.3}$$

An intuitive interpretation is that r_{pixel} is 0 when the pen is up and increases with the similarity between the canvas and reference image.

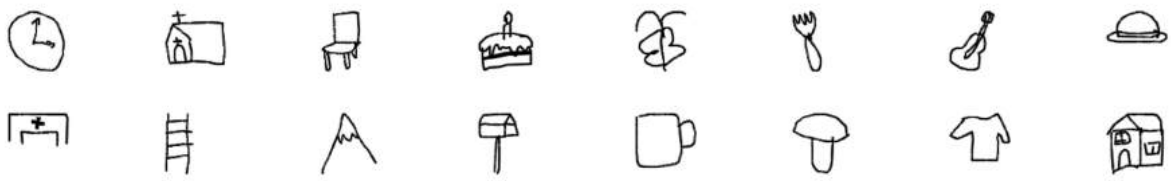


Figure D.5: Reference images for training and testing. 16 classes are randomly chosen from the QuickDraw dataset Jongejan et al. (2016): clock, church, chair, cake, butterfly, fork, guitar, hat, hospital, ladder, mountain, mailbox, mug, mushroom, T-shirt, house.

To avoid slow movement or pixel by pixel printing, we penalize small steps. Specifically, if the pen moves less than 5 pixels/step when the pen is drawing or if it moves while being up, the agent will be penalized with P_{step}. If the input is an RGB image, we additionally penalize the incorrectness of the chosen color P_{color}.

Thus, the final reward is

$$r_k = r_{\text{pixel}} + P_{\text{step}} + \beta P_{\text{color}}, \tag{D.4}$$

where P_{step} and P_{color} are constants set based on the observation, and β depends on the input image type: 0 for a grayscale image and 1 for a color image.

In the RL phase, we use QuickDraw Jongejan et al. (2016), a dataset of vector drawings, as the input reference image. Since the scale of the drawings in QuickDraw varies across samples, the drawing sequence data is processed such that all the drawings can be squeezed onto an 84×84 pixel canvas. We randomly selected sixteen classes, and each class includes 200 reference images (Figure D.5). For RL training, the images except for the 'house' class are applied. Therefore, 3,000 reference images are adopted for training.

D.4 Experiments

During the pretraining phase, we use a softmax cross entropy loss for the classification task. The loss is minimized using Adam Kingma and Ba (2014) with minibatches of size 128 for optimization with the initial step size $\alpha = 0.001$, and gradually decays with the training step.

		Naive SDQ	SDQ + Rare exp	Pretrain on random	Pretrain on QuickDraw	SDQ + Rare exp + weight init	Max reward
House Class	Sketch	93	1,404	1,726	1,738	1,927	2,966
	Color Sketch	−13	1,651	1,765	1,747	1,808	3,484
	Water Color	−162	407	596	620	670	1,492
Training Classes	Sketch	67	1024	1,539	1,521	1,805	2,645
	Color Sketch	−15	1,464	1,669	1,683	1,731	3,533
	Water Color	−182	363	446	473	509	1,527

Table D.2: Average accumulated rewards for the models tested.

Instead of using random initialization, the learned weights from the pretrained classification model are used to initialize Doodle-SDQ's network. Due to the large action space, the pen is likely to draw a wrong stroke following a random action in the RL phase. Thus, exploration in action space is rarely applied unless the pen is stuck at some point.[3] For the RL stage, we train for a total of 0.6M frames and use a replay memory of 20 thousand frames. The weights are updated based on the difference between the Q value and the output of the target Q network Schaul et al. (2016). The loss is minimized using Adam with $\alpha = 0.001$. Our model is implemented in Tensorflow Abadi et al. (2016).

To visualize the effect of the algorithm, the model is unrolled for 100 steps starting from an empty canvas. We chose 100 steps because more steps do not lead to further improvement. Figure D.6 shows the drawing given the reference images from different categories in the test set using different media types. Additional sketch drawing examples are presented (Figure D.7) and the algorithm was tested on reference images not in the QuickDraw dataset, where we found that, although it was trained on QuickDraw, the agent has the ability to draw quite diverse doodles. For a reference image, the reward from each step is summed up and the accumulated reward is a quantitative measure of the performance of the algorithm. The maximum reward is achieved when the agent perfectly reproduces the reference image. In the test phase, we used 100 house reference images and 100 reference images randomly selected from the test sets belonging to the training classes.

[3]From our observations, the pen is likely to stop moving at some location or move back and forth between two spots. Only in these scenarios, the pen will be given a random action to avoid local minima.

(a) Sketch: butterfly, guitar, church, cake, mailbox, hospital

(b) Color sketch: mailbox, chair, hat, house, mug, T-shirt

(c) Watercolor: T-shirt, butterfly, cake, mug, house, mailbox

Figure D.6: Comparisons between drawings and reference images in different media types: (a) sketch, (b) color sketch, (c) watercolor. The left image in each pair is the drawing after 100 steps of the model and the right is the reference image. The drawings in watercolor mode are enlarged to visualize the stroke distortion and color mixing

Table D.2 presents the average accumulated rewards and the average maximum rewards across reference images. In the table, the 'Naive SDQ' model is the Doodle-SDQ model trained from scratch following a ϵ-greedy strategy with ϵ annealed linearly from 1.0 to 0.1 over the first fifty thousand frames and fixed at 0.1 thereafter. The 'SDQ + Rare exp' is the Doodle-SDQ model trained from scratch with rare exploration. The 'Pretrain on random' model is the model with supervised pretraining on the synthesized random stroke sequence data (Figure D.4). The 'Pretrain on QuickDraw' model is the model with supervised pretraining on the QuickDraw sequence data. The 'SDQ + Rare exp + weight init' model is the Doodle-SDQ model with rare exploration and weight initialization from the 'Pretrain on random' model. Based on the average accumulated reward, Doodle-SDQ with weight initialization is significantly better than all the other methods. Furthermore, pretraining on the QuickDraw sequence data directly does not lead to superior performance over the RL method. This indicates the advantage of using DRL in the drawing task.

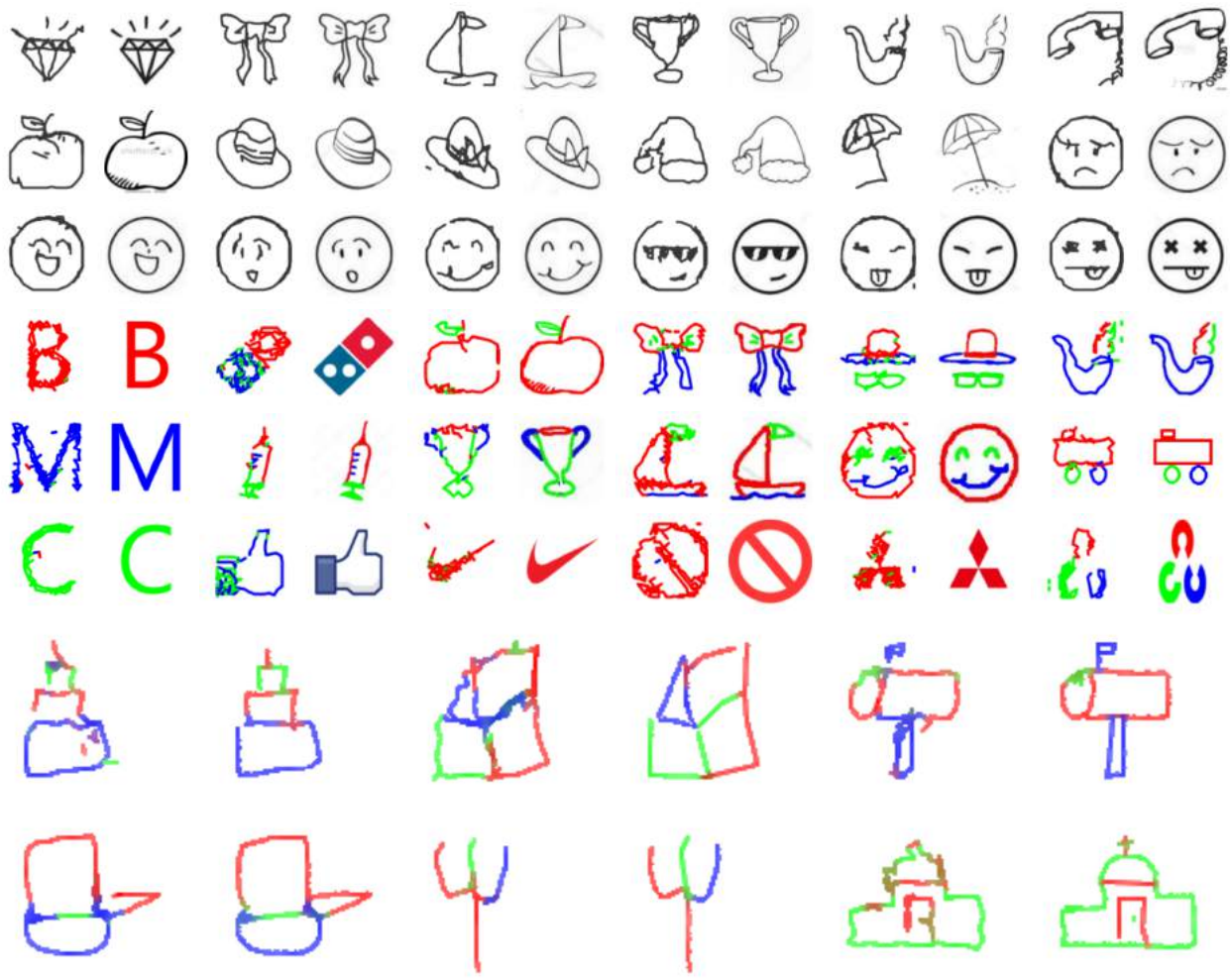


Figure D.7: Additional sketch drawing examples.

D.5 Discussion

We now list several key factors that contribute to the success of our Doodle-SDQ algorithm and compare it to the DDQN + PER model of Schaul et al. (2016) (Table D.3).

Since Naive SDQ cannot be directly used for the drawing task, we first pretrain the network to initialize the weights. Referring to Table D.2, pretraining with stroke demonstration via supervised learning leads to an improvement in performance (Columns 4 and 7). Based on our observations, the 4-frame history used in Schaul et al. (2016) introduces a movement momentum that compels the agent to move in a straight line and rarely turn. Therefore, history information is excluded in our current model. In Schaul et al. (2016), the probability

	Doodle-SDQ	DDQN + PER Schaul et al. (2016)
History	No	Yes
Exploration	Rare	Yes
Pretrain	Yes	No
Input stream	2	1

Table D.3: Differences between the proposed method and Schaul et al. (2016).

for the exploration of random action decays from 0.9 to 0.1 with increasing epochs. Since we pretrained the network, the agent does not need to explore the environment with a large rate Cruz Jr et al. (2017). Thus, we initially set the exploration rate to 0.1. However, Doodle-SDQ cannot outperform the pretrained model until we remove exploration.[4] The countereffect of the exploration may be caused by the large action space. The small patch in the two streams structure (Figure D.3) makes the agent attend to the region where the pen is located. More specifically, when the lifted pen is within one step action distance to the target drawing, the local stream is able to move the pen to a correct position and start drawing. Without this stream, the RL training cannot be successful even after removing the exploration or pretraining the network. The average accumulated rewards for the global stream only network varies around 100 depending on the media types.

Despite the success of our SDQ model in simple sketch drawing, there are several limitations to be addressed in the future work. On the one hand, our motivation is to design an algorithm to enable machines to doodle like humans, rather than competing with GAN Goodfellow et al. (2014) to generate complex image types, at least not at the current stage. However, it has been demonstrated that an adversarial framework Ganin et al. (2018) interprets and generates images in the space of visual programs. Therefore, it will be a promising direction to mimic human drawing by combining adversarial training technique and reinforcement learning. On the other hand, although the SDQ model works in a relatively large action space due to rare exploration, the average accumulated rewards introduced by the component of reinforcement learning still suffers from the increase of the dimension

[4] A random movement will be generated only when the agent gets stuck at some position, such as moving back and forth or remaining at the same spot.

of action space by allowing colorful drawing as shown by a comparison between sketch and color sketch (Column 6 and 7 in Table D.2). Since our future work will incorporate more action variables (e.g., the pen's pressure and additional colors) and explore doodling on large canvases, the actions might be embed in a continuous space Dulac-Arnold et al. (2015).

D.6 Conclusion

We addressed the challenging problem of emulating human doodling. To solve this problem, we proposed a deep-reinforcement-learning-based method, Doodle-SDQ. Due to the large action space, Naive SDQ fails to draw appropriately. Thus, we designed a hybrid approach that combines supervised imitation learning and reinforcement learning. We train the agent in a supervised manner by providing demonstration strokes with ground truth actions. We then further trained the pre-trained agent with Q-learning using a reward based on the similarity between the current drawing and the reference image. Drawing step-by-step, our model reproduces reference images by comparing the similarity between the current drawing and the reference image. Our experimental results demonstrate that our model is robust and generalizes to classes not presented during training, and that it can be easily extended to other media types, such as watercolor.

Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., et al. (2016). Tensorflow: A system for large-scale machine learning. In *OSDI*, volume 16, pages 265–283. 78

Abdallah, M. and Goswami, A. (2005). A biomechanically motivated two-phase strategy for biped upright balance control. In *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, pages 1996–2001. IEEE. 57

Bizzi, E. and Cheung, V. C. (2013). The neural origin of muscle synergies. *Frontiers in Computational Neuroscience*, 7:51. 58

Brown, S. (2014). *The Doodle revolution: Unlock the power to think differently*. Penguin. 68

Cruz Jr, G. V., Du, Y., and Taylor, M. E. (2017). Pre-training neural networks with human demonstrations for deep reinforcement learning. *arXiv preprint arXiv:1709.04083*. 72, 81

Cruz Ruiz, A., Pontonnier, C., Pronost, N., and Dumont, G. (2017). Muscle-based control for character animation. *Computer Graphics Forum*, 36(6):122–147. 8

Denton, E. L., Chintala, S., Fergus, R., et al. (2015). Deep generative image models using a Laplacian pyramid of adversarial networks. In *Advances in Neural Information Processing Systems*, pages 1486–1494. 69

DiLorenzo, P., Zordan, V., and Sanders, B. (2008). Laughing out loud: Control for modeling anatomically inspired laughter using audio. *ACM Transactions on Graphics (TOG) (Proc. ACM SIGGRAPH Asia 2008)*, 27(5):125. 6

Dubey, P., Hanrahan, P., Fedkiw, R., Lentine, M., and Schroeder, C. (2011). PhysBAM: Physically based simulation. In *ACM SIGGRAPH 2011 Courses*, SIGGRAPH '11, pages 10:1–10:22, New York, NY, USA. ACM. 61

Dulac-Arnold, G., Evans, R., van Hasselt, H., Sunehag, P., Lillicrap, T., Hunt, J., Mann, T., Weber, T., Degris, T., and Coppin, B. (2015). Deep reinforcement learning in large discrete action spaces. *arXiv preprint arXiv:1512.07679*. 82

Elgammal, A., Liu, B., Elhoseiny, M., and Mazzone, M. (2017). CAN: Creative adversarial networks, generating "art" by learning about styles and deviating from style norms. *arXiv preprint arXiv:1706.07068*. 45, 69, 72

Faloutsos, P., van de Panne, M., and Terzopoulos, D. (2001a). Composable controllers for physics-based character animation. In *Proc. 28th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '01)*, pages 251–260, Los Angeles, CA. 6, 7, 57

Faloutsos, P., van de Panne, M., and Terzopoulos, D. (2001b). The virtual stuntman: Dynamic characters with a repertoire of autonomous motor skills. *Computers & Graphics*, 25(6):933–953. 6

Fang, J., Jiang, C., and Terzopoulos, D. (2013). Modeling and animating myriapoda: A real-time kinematic/dynamic approach. In *Proceedings of the 12th ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 203–212. ACM. 57

Featherstone, R. (2014). *Rigid Body Dynamics Algorithms*. Springer, New York, NY. 13, 60

Ganin, Y., Kulkarni, T., Babuschkin, I., Eslami, S., and Vinyals, O. (2018). Synthesizing programs for images using reinforced adversarial learning. *arXiv preprint arXiv:1804.01118*. 45, 72, 81

Gatys, L., Ecker, A., and Bethge, M. (2016). Image style transfer using convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2414–2423. 45, 72

Gonzalez, R., Buchanan, T., and Delp, S. (1997). How muscle architecture and moment arms affect wrist flexion-extension moments. *Journal of Biomechanics*, 30(7):705–712. 13

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press, Cambridge, MA. 9

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative adversarial nets. In *Advances in Neural Information Processing Systems*, pages 2672–2680. 69, 81

Graves, A. (2013). Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*. 45, 73

Gregor, K., Danihelka, I., Graves, A., Rezende, D. J., and Wierstra, D. (2015). DRAW: A recurrent neural network for image generation. *arXiv preprint arXiv:1502.04623*. 72

Grzeszczuk, R., Terzopoulos, D., and Hinton, G. (1998). Neuroanimator: Fast neural network emulation and control of physics-based models. In *Computer Graphics Proceedings, Annual Conference Series*, pages 9–20, Orlando, FL. Proc. ACM SIGGRAPH 98. 8

Ha, D. and Eck, D. (2017). A neural representation of sketch drawings. *arXiv preprint arXiv:1704.03477*. 45, 48, 49, 73

Hasselt, H. v., Guez, A., and Silver, D. (2016). Deep reinforcement learning with double Q-learning. In *AAAI Conference on Artificial Intelligence*, pages 2094–2100. AAAI Press. 72

Hester, T., Vecerik, M., Pietquin, O., Lanctot, M., Schaul, T., Piot, B., Sendonaris, A., Dulac-Arnold, G., Osband, I., Agapiou, J., et al. (2018). Deep Q-learning from demonstrations. *Association for the Advancement of Artificial Intelligence (AAAI)*. 72

Hodgins, J., Wooten, W., Brogan, D., and O'Brien, J. (1995). Animating human athletics. In *Proc. 22nd Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '95)*, pages 71–78, Los Angeles, CA. 6, 7, 57

Hussein, A., Gaber, M. M., Elyan, E., and Jayne, C. (2017). Imitation learning: A survey of learning methods. *ACM Computing Surveys (CSUR)*, 50(2):21. 71

Irving, G., Teran, J., and Fedkiw, R. (2004). Invertible finite elements for robust simulation of large deformation. In *Proc. ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, page 131. 6

Jackson, J. (1889). On the comparative study of diseases of the nervous system. *British Medical Journal*, 2:355–362. 2, 58

Jongejan, J., Rowley, H., Kawashima, T., Kim, J., and Fox-Gieg, N. (2016). The Quick, Draw! AI experiment. *https://quickdraw.withgoogle.com*. 45, 50, 73, 77

Kähler, K., Haber, J., Yamauchi, H., and Seidel, H.-P. (2002). Head shop: Generating animated head models with anatomical structure. In *Proc. 2002 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 55–63, San Antonio, TX. 6

Kajita, S., Kanehiro, F., Kaneko, K., Fujiwara, K., Harada, K., Yokoi, K., and Hirukawa, H. (2003). Resolved momentum control: Humanoid motion planning based on the linear and angular momentum. In *Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003)(Cat. No. 03CH37453)*, volume 2, pages 1644–1650. IEEE. 57

Kingma, D. and Ba, J. (2014). Adam: A method for stochastic optimization. Technical report, arXiv preprint arXiv:1412.6980. 28, 77

Kingma, D. P. and Welling, M. (2013). Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*. 62, 69

Latash, M. (2012). The bliss (not the problem) of motor abundance (not redundancy). *Experimental Brain Research*, 217(1):1–5. 2

Latash, M. L., Scholz, J. P., and Schöner, G. (2002). Motor control strategies revealed in the structure of motor variability. *Exercise and Sport Sciences Reviews*, 30(1):26–31. 56

Lee, S., Yu, R., Park, J., Aanjaneya, M., Sifakis, E., and Lee, J. (2018). Dexterous manipulation and control with volumetric muscles. *ACM Transactions on Graphics (TOG)*, 37(4):57. 8

Lee, S.-H., Sifakis, E., and Terzopoulos, D. (2009). Comprehensive biomechanical modeling and simulation of the upper body. *ACM Transactions on Graphics (TOG)*, 28(4):99:1–17. 6, 8, 9, 13, 14, 16, 19, 60

Lee, S.-H. and Terzopoulos, D. (2006). Heads up! Biomechanical modeling and neuromuscular control of the neck. *ACM Transactions on Graphics*, 23(212):1188–1198. Proc. ACM SIGGRAPH 2006. 6, 8, 9, 14

Lee, S.-H. and Terzopoulos, D. (2008). Spline joints for multibody dynamics. *ACM Transactions on Graphics (TOG)*, 27(3):22. 7

Lee, Y., Terzopoulos, D., and Waters, K. (1995). Realistic modeling for facial animation. In *Proc. 22nd Annual Conference on Computer Graphics and Interactive Techniques*, pages 55–62. ACM. 6

Levine, S., Finn, C., Darrell, T., and Abbeel, P. (2016). End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research*, 17(1):1334–1373. 70

Martin, V., Reimann, H., and Schöner, G. (2019). A process account of the uncontrolled manifold structure of joint space variance in pointing movements. *Biological Cybernetics*, pages 1–15. 56

Martin, V., Scholz, J. P., and Schöner, G. (2009). Redundancy, self-motion, and motor control. *Neural Computation*, 21(5):1371–1414. 56

Mattos, D., Latash, M. L., Park, E., Kuhl, J., and Scholz, J. P. (2011). Unpredictable elbow joint perturbation during reaching results in multijoint motor equivalence. *Journal of Neurophysiology*, 106(3):1424–1436. 56

McAdams, A., Zhu, Y., Selle, A., Empey, M., Tamstorf, R., Teran, J., and Sifakis, E. (2011). Efficient elasticity for character skinning with contact and collisions. *ACM Transactions on Graphics (TOG)*, 30(4):37. 19

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wiestra, D., Legg, S., and Hassabis, D. (2015). Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–33. 70, 72, 73

Monheit, G. and Badler, N. (1991). A kinematic model of the human spine and torso. *IEEE Computer Graphics and Applications*, 11(2):29–38. 7

Nakada, M., Lakshmipathy, A., Chen, H., Ling, N., Zhou, T., and Terzopoulos, D. (2019). Biomimetic eye modeling and deep neuromuscular oculomotor control. *ACM Transactions on Graphics (TOG)*, 38(6):221:1–14. Proc. ACM SIGGRAPH Asia 2019 Conference, Brisbane, Australia, November 2019. 52

Nakada, M. and Terzopoulos, D. (2015). Deep learning of neuromuscular control for biomechanical human animation. In *Advances in Visual Computing*, Lecture Notes in Computer Science, Vol. 9474, pages 339–348, Berlin. Springer. Proc. *International Symposium on Visual Computing*, Las Vegas, NV, December 2015. 9

Nakada, M., Zhou, T., Chen, H., Weiss, T., and Terzopoulos, D. (2018). Deep learning of biomimetic sensorimotor control for biomechanical human animation. *ACM Transactions on Graphics (TOG)*, 37(4):56:1–15. Proc. ACM SIGGRAPH 2018 Conference, Los Angeles, CA, August 2018. 10, 11, 16, 24, 31, 34, 46, 67

Ng-Thow-Hing, V. (2001). *Anatomically-Based Models for Physical and Geometric Reconstruction of Humans and Other Animals*. PhD thesis, University of Toronto, Computer Science Department. 6

Patterson, T., Mitchell, N., and Sifakis, E. (2012). Simulation of complex nonlinear elastic bodies using lattice deformers. *ACM Transactions on Graphics (TOG)*, 31(6):197. 19, 22, 61

Peng, X., Berseth, G., Yin, K., and Van De Panne, M. (2017). Deeploco: Dynamic locomotion skills using hierarchical deep reinforcement learning. *ACM Transactions on Graphics*, 36(4):41. 57

Peng, X. B., Abbeel, P., Levine, S., and van de Panne, M. (2018). Deepmimic: Example-guided deep reinforcement learning of physics-based character skills. *ACM Transactions on Graphics (TOG)*, 37(4):143. 58

Peng, X. B., Berseth, G., and Van de Panne, M. (2016). Terrain-adaptive locomotion skills using deep reinforcement learning. *ACM Transactions on Graphics (TOG)*, 35(4):81. 70

Porcher Nedel, L. and Thalmann, D. (2000). Anatomic modeling of deformable human bodies. *The Visual Computer*, 16(6):306–321. 7

Ross, S., Gordon, G., and Bagnell, D. (2011). A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the 14th International Conference on Artificial Intelligence and Statistics*, pages 627–635. 71

Saito, S., Zhou, Z.-Y., and Kavan, L. (2015). Computational bodybuilding: Anatomically-based modeling of human bodies. *ACM Transactions on Graphics (TOG)*, 34(4):41. 8

Schaul, T., Quan, J., Antonoglou, I., and Silver, D. (2016). Prioritized experience replay. In *International Conference on Learning Representations (ICLR)*. 72, 78, 80, 81

Scheepers, F., Parent, R., Carlson, W., and May, S. (1997). Anatomy-based modeling of the human musculature. In *Proceedings of the ACM SIGGRAPH Conference*, pages 163–172. ACM Press/Addison-Wesley Publishing Co. 7

Scholz, J. and Schöner, G. (1999). The uncontrolled manifold concept: Identifying control variables for a functional task. *Experimental Brain Research*, 126(3):289–306. 2, 56

Shao, W. and Ng-Thow-Hing, V. (2003). A general joint component framework for realistic articulation in human characters. In *Proc. Symposium on Interactive 3D Graphics*, pages 11–18. ACM. 7

Si, W., Lee, S.-H., Sifakis, E., and Terzopoulos, D. (2014). Realistic biomechanical simulation and control of human swimming. *ACM Transactions on Graphics (TOG)*, 34(1):10:1–15. 6, 8, 57

Sifakis, E., Neverov, I., and Fedkiw, R. (2005). Automatic determination of facial muscle activations from sparse motion capture marker data. *ACM Transactions on Graphics*, 1(212):417–425. 6

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., and Hassabis, D. (2016). Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489. 72

Simhon, S. and Dudek, G. (2004). Sketch interpretation and refinement using statistical models. In *Rendering Techniques*, pages 23–32. 45, 73

Subramanian, K., Isbell Jr, C. L., and Thomaz, A. L. (2016). Exploration from demonstration for interactive reinforcement learning. In *Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems*, pages 447–456. 72

Sueda, S., Kaufman, A., and Pai, D. (2008). Musculotendon simulation for hand animation. *ACM Transactions on Graphics*, 27(3):83. 6, 55

Sun, Y., Qian, H., and Xu, Y. (2014). Robot learns Chinese calligraphy from demonstrations. In *Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on*, pages 4408–4413. IEEE. 45, 72

Ting, L. H. and Macpherson, J. M. (2005). A limited set of muscle synergies for force control during a postural task. *Journal of Neurophysiology*, 93(1):609–613. 58

Tresset, P. and Leymarie, F. (2013). Portrait drawing by Paul the robot. *Computers & Graphics*, 37(5):348–363. 45, 72

Tsang, W., Singh, K., and Fiume, E. (2005). Helping hand: An anatomically accurate inverse dynamics solution for unconstrained hand motion. In *Proc. 2005 ACM SIGGRAPH/ Symposium on Computer Animation*, pages 319–328. 6

van Nierop, O. A., van der Helm, A., Overbeeke, K. J., and Djajadiningrat, T. J. (2007). A natural human hand model. *The Visual Computer*, 24(1):31–44. 6

Vondrick, C., Pirsiavash, H., and Torralba, A. (2016). Generating videos with scene dynamics. In *Advances In Neural Information Processing Systems*, pages 613–621. 69

Wilhelms, J. and Van Gelder, A. (1997). Anatomically based modeling. In *Proceedings of the ACM SIGGRAPH Conference*, pages 173–180. ACM Press/Addison-Wesley Publishing Co. 7

Xie, N., Hachiya, H., and Sugiyama, M. (2012). Artist agent: a reinforcement learning approach to automatic stroke generation in oriental ink painting. In *Proceedings of the 29th International Conference on Machine Learning*, pages 1059–1066. Omnipress. 71

Yin, K., Loken, K., and van de Panne, M. (2007). Simbicon: Simple biped locomotion control. *ACM Transactions on Graphics (TOG)*, 26(3):105. 57

Zhang, W., Scholz, J. P., Zatsiorsky, V. M., and Latash, M. L. (2008). What do synergies do? Effects of secondary constraints on multidigit synergies in accurate force-production tasks. *Journal of Neurophysiology*, 99(2):500–513. 56

Zhang, X.-Y., Yin, F., Zhang, Y.-M., Liu, C.-L., and Bengio, Y. (2017). Drawing and recognizing Chinese characters with recurrent neural network. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. 45, 73

Zhou, T., Fang, C., Wang, Z., Yang, J., Kim, B., Chen, Z., Brandt, J., and Terzopoulos, D. (2018a). Learning to doodle with stroke demonstrations and deep Q-networks. In *British Machine Vision Conference (BMVC)*, pages 13:1–13, Newcastle, UK. 45

Zhou, T., Fang, C., Wang, Z., Yang, J., Kim, B., Chen, Z., Brandt, J., and Terzopoulos, D. (2018b). Learning to sketch with deep Q-networks and demonstrated strokes. *arXiv preprint arXiv:1810.05977*. 68

Zhu, Y., Sifakis, E., Teran, J., and Brandt, A. (2010). An efficient multigrid method for the simulation of high-resolution elastic solids. *ACM Transactions on Graphics (TOG)*, 29(2):16. 19

Zordan, V., Celly, B., Chiu, B., and DiLorenzo, P. (2004). Breathe easy: Model and control of simulated respiration for animation. In *Proc. ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 29–37. 6, 7