UNIVERSITY OF CALIFORNIA
Los Angeles

Deep Learning of Neuromuscular and Sensorimotor Control with Biomimetic Perception for Realistic Biomechanical Human Animation

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

by

Masaki Nakada

2017

ABSTRACT OF THE DISSERTATION

Deep Learning of Neuromuscular and Sensorimotor Control with Biomimetic Perception for Realistic Biomechanical Human Animation

by

Masaki Nakada
Doctor of Philosophy in Computer Science
University of California, Los Angeles, 2017
Professor Demetri Terzopoulos, Chair

We introduce a biomimetic simulation framework for investigating human perception and sensorimotor control. Our framework is unique in that it features a biomechanically simulated musculoskeletal human model actuated by 823 muscles. The anthropomorphic model has two human-like eyes whose retinas contain spatially nonuniform arrangements of photoreceptors. The sensorimotor control system of our human model comprises a set of 15 automatically-trained, fully-connected deep neural networks. Two networks control the saccadic eye movement functionality of its binocular, foveated perception system. The remaining networks achieve neuromuscular control of the skeletal muscles. One network controls the 216 neck muscles that actuate the neck-head biomechanical complex, producing controlled head movements. In our prototype model, 3 networks control each limb; in particular, the 29 muscles in each of the two arms and the 39 muscles in each of the two legs. Thus, the virtual human demonstrates effective sensorimotor control of its eyes, head, and four limbs driven exclusively by visual perception to achieve a nontrivial motor task. We also demonstrate that its foveated perceptual system is capable of appearance-based recognition.

The dissertation of Masaki Nakada is approved.

Michael G. Dyer

Song-Chun Zhu

Joseph M. Teran

Demetri Terzopoulos, Committee Chair

University of California, Los Angeles

2017

To my family, friends and mentors ...

who, among so many other things,

supported me from my childhood

to my PhD degree

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

First, I cannot thank Professor Demetri Terzopoulos enough for supporting me as my advisor throughout my PhD program. While I was pursuing a master's degree in Japan, I had a chance to work with him as a Rotary Foundation visiting scholar at UCLA. That experience made me want to pursue a PhD degree and immerse myself in research. Without him, I probably would not even have entered a PhD program. During my PhD, he provided his unconditional support whenever I needed it. He gave me a clear research direction and often spent hours discussing the progress of my research. Without his guidance, I would not have accomplished the results reported in this thesis. He is my role model as a researcher, and also I respect him as a person who is always kind, cheerful, and hard working. I am truly grateful that I got to know him.

Secondly, I would like to express my appreciation to the other members of my PhD thesis committee. The advice I got from those committee members were helpful to advance my research with various insights in their course lectures and their feedback during my oral qualifying exam and other interactions. Professor Song-Chun Zhu taught me computer vision and pattern recognition, and helped strengthen my knowledge of machine learning. Professor Michael Dyer taught me animats-based modeling, where I learned interesting learning algorithms, which I applied to develop a virtual creature. Professor Joseph Teran taught me scientific computation and strengthened my mathematical background. Thanks to my committee, I was able to build my knowledge and skills to make progress in my research, and write this dissertation.

I would also like to express my appreciation to Professor Sung-Hee Lee of KAIST. He gave me much valuable advice on the multibody dynamics simulation of the biomechanical human musculoskeletal model that he developed during his PhD program in our UCLA lab. His model served as the starting point of my research. I was greatly inspired by and learned a lot from his work.

Furthermore, I would like to express my deep appreciation to Professor Shigeo Morishima.

He supervised me through my undergraduate and master degrees in Japan and taught me much about doing research and being a better person. Even after my master's program, he has continued to support me. We have spent many days together going on trips and discussing research and life in general. My life in the past 10 years would have been very different without him. I cannot thank him enough.

I am very fortunate to have been surrounded by wonderful labmates in the UCLA Computer Graphics & Vision Laboratory who helped me work through the PhD program. I would like to express my gratitude to Tomer Weiss, Tao Zhao, Honglin Chen, Chenfanfu Jiang, Sharath Gopal, Jingyi Fang, Wenjia Huang, Weiguang Si, Lap-Fai Yu, Kresimir Petrinec, Garett Ridge, Matthew Wang, Xiaowei Ding, Noah Duncan, Gergely Klar, Eduardo Poyart, Andre Pradhana, Gabriele Nataneli, Konstantinos Sideris, Shawn Singh, Mubbasir Kapadia, Brian Allen, Billy Hewlett, and Alan Litteneker.

My special thanks goes to the Japan Student Organization (JASSO), which financially supported me for 5 years during my PhD program through a Scholarship for Long-Term Study Abroad. Without their support, I would not have been able to start a PhD program, let alone graduate. I am deeply grateful to all those involved with my fellowship, especially at the Study-Abroad Center at Waseda University who helped me to process paperwork among other things during my PhD program.

Lastly, I would like to thank my parents, Mayumi and Tatsuya Nakada, who gave me life in this world and raised me with unconditional love. I could not have come this far without them. I also thank my grandparents, Yukiko and Masami Chakihara, Kimiko and Ichiro Nakada, my sister, Emi Nakada, and other family members who have supported me a great deal from my childhood till now.

VITA

2009	B.S. Applied Physics Waseda University Tokyo, Japan
2010	Visiting Researcher Computer Science Department University of California, Los Angeles Los Angeles, CA
2011	M.S. Pure and Applied Physics Waseda University Tokyo, Japan
2011	Intel Corp. Software Engineer Tsukuba, Japan
2017	Teaching Assistant Computer Science Department University of California, Los Angeles Los Angeles, CA

PUBLICATIONS

M. Nakada, H. Chen, T. Zhou, T. Weiss, P. Sodani and D. Terzopoulos. "Biomimetic Perception Learning for Human Sensorimotor Control." In review by the *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Salt Lake City, UT, July 2018.

M. Nakada, T. Weiss and D. Terzopoulos. "Automated Layout Synthesis and Visualization From Images of Interior or Exterior Spaces." In *Third IEEE Workshop on Vision Meets Cognition: Functionality, Physics, Intentionality, and Causality (FPIC)*, *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Honolulu, HI, July 2017, pp. 41–47.

M. Nakada, H. Wang and D. Terzopoulos. "AcFR: Active Face Recognition Using Convolutional Neural Networks." In *Third IEEE Workshop on Vision Meets Cognition: Functionality, Physics, Intentionality, and Causality (FPIC)*, *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Honolulu, HI, July 2017, pp. 35–40.

M. Nakada and D. Terzopoulos. "Deep Learning of Neuromuscular Control for Biomechanical Human Animation." In *11th International Symposium on Visual Computing (ISVC 2015)*, Las Vegas, NV, December 2015, *Lecture Notes in Computer Science, Vol. 9474*, G. Bebis et al. (eds.), Springer-Verlag, Berlin, 2015, pp. 339–348.

M. Nakada, B. Allen, S. Morishima and D. Terzopoulos. "Learning Arm Motion Strategies for Balance Recovery of Humanoid Robots." In *2010 International Symposium on Learning and Adaptive Behavior in Robotic Systems*, *2010 International Symposium on Learning and Adaptive Behavior in Robotic Systems*, Canterbury, UK, September 2010, pp. 165–170.

CHAPTER 1

Introduction

The modeling of graphical characters based on human anatomy is attracting increasing attention in the field of computer animation. Such models are applicable in various domains, including self-animating characters for motion pictures and games, interactive characters for virtual reality and augmented reality, and virtual human subjects for biomedical research.

Increasingly accurate biomechanical modeling of the human body should, in principle, result in more realistic human animation. However, effectively controlling complex biomechanical human musculoskeletal models remains a daunting research challenge. Given the unavoidable complexity of anatomically realistic biomechanical models of humans and other animals, we must confront a variety of difficult motor control tasks.

For example, a biomechanical neck-head musculoskeletal model developed by Lee and Terzopoulos (2006) has 72 muscle actuators, while the full-body biomechanical model developed by Lee et al. (2009) includes 823 muscle actuators. Therefore, controlling the latter model requires specifying the 823 muscle activation levels at every simulation time step of the biomechanical simulation. A biomechanical control problem of this magnitude has not yet been been tackled, but in this thesis we take a major stride towards this goal by pursuing a neuromuscular control approach based machine learning, specifically using deep neural networks (DNNs).

Beyond pure motor control, we must eventually also confront the problem of sensorimotor control. Sensorimotor functionality in biological organisms refers to the process of receiving sensory input and producing motor output responses. As animals, we humans receive sensory information from the environment and our bodies through our sensory systems (visual, auditory, tactile, olfactory, gustatory, vestibular, and proprioceptive). This sensory informa-

tion must be organized and processed in order to produce appropriate motor responses that achieve behavioral goals. DNNs (at least of the convolutional variety) have been demonstrably effective in computer vision, especially for image segmentation and recognition; however, their application to the online control of strongly biomimetic sensorimotor systems has received virtually no attention in the fields of computer vision, computer graphics, and even robotics.

1.1 Scope of the Research

The main research problems that we tackle in this dissertation are as follows:

- **Biomimetic DNN-based motor control:** The human brain generates the required muscle activations seemingly instantaneously, apparently without solving the Newtonian equations of motion underlying biomechanics, and its motor control mechanisms appear to be at odds with any online control approach that relies on solving inverse kinematics, inverse dynamics, and/or muscle optimization problems to compute the required muscle activations needed to produce desired motions. Rather, the brain is capable of learning muscle controls from experience. Therefore, as in one or two recent precedents, we investigate a neuromuscular control approach based on machine learning, particularly based on DNNs.

- **Biomimetic perception:** Since our research goal is to create a realistic virtual human, we revisit the fundamentals of human vision taking a strongly biomimetic approach to computer vision for virtual humans. We investigate the possibility of creating an anatomically consistent 3D eye model, with a retina populated by photoreceptors in an irregular, foveated pattern, which is capable of undergoing directed eye movements and of controlling these eye movements using a perceptual DNN approach.

- **Biomimetic sensorimotor control:** We investigate the possibility of tying the above motor control and perception models together with additional perceptual DNN controllers to achieve an integrated biomimetic sensorimotor control system capable of

performing certain nontrivial tasks. These include well-controlled coordinated eye-head movements supporting active vision as well as limb motions to reach static and kinetic visual targets.

1.2 Contributions

In this thesis, we introduce a biomimetic simulation framework for investigating human perception and sensorimotor control. Our framework is unique in that it features a biomechanically simulated, 823 muscle actuated, anatomically accurate human musculoskeletal model, shown in Figure 1.1. As is evident in the figure, our anthropomorphic model has two human-like eyes whose retinas contain cone-like photoreceptors in spatially nonuniform arrangements. The sensorimotor control system of our human model consists of a set of 15 automatically-trained, *fully-connected* deep neural networks (DNNs), as illustrated in Figure 4.1.

Two networks control the saccadic eye movement functionality of our virtual human's binocular, foveated perception system. The remaining networks achieve neuromuscular control of the skeletal muscles. One network controls the 216 neck muscles that balance the head atop the cervical column against the downward pull of gravity and actuate the neck-head biomechanical complex, thereby producing controlled head movements. In our prototype sensorimotor system, 3 networks control each limb; in particular, the 29 muscles of each of the two arms and the 39 muscles of each of the two legs. Thus, the virtual human demonstrates effective sensorimotor control of its eyes, head, and four limbs driven exclusively by its visual perception.

To our knowledge, our sensorimotor control system is unprecedented, both in its use of a sophisticated biomechanical human model as well as in its use of modern machine learning methodologies to control realistic musculoskeletal systems and online visual processing for active, foveated perception through a modular set of DNNs. It is important to note that all the DNNs are automatically trained offline with training data synthesized using the biomechanical model itself, which we regard a contribution of interest to the machine learning

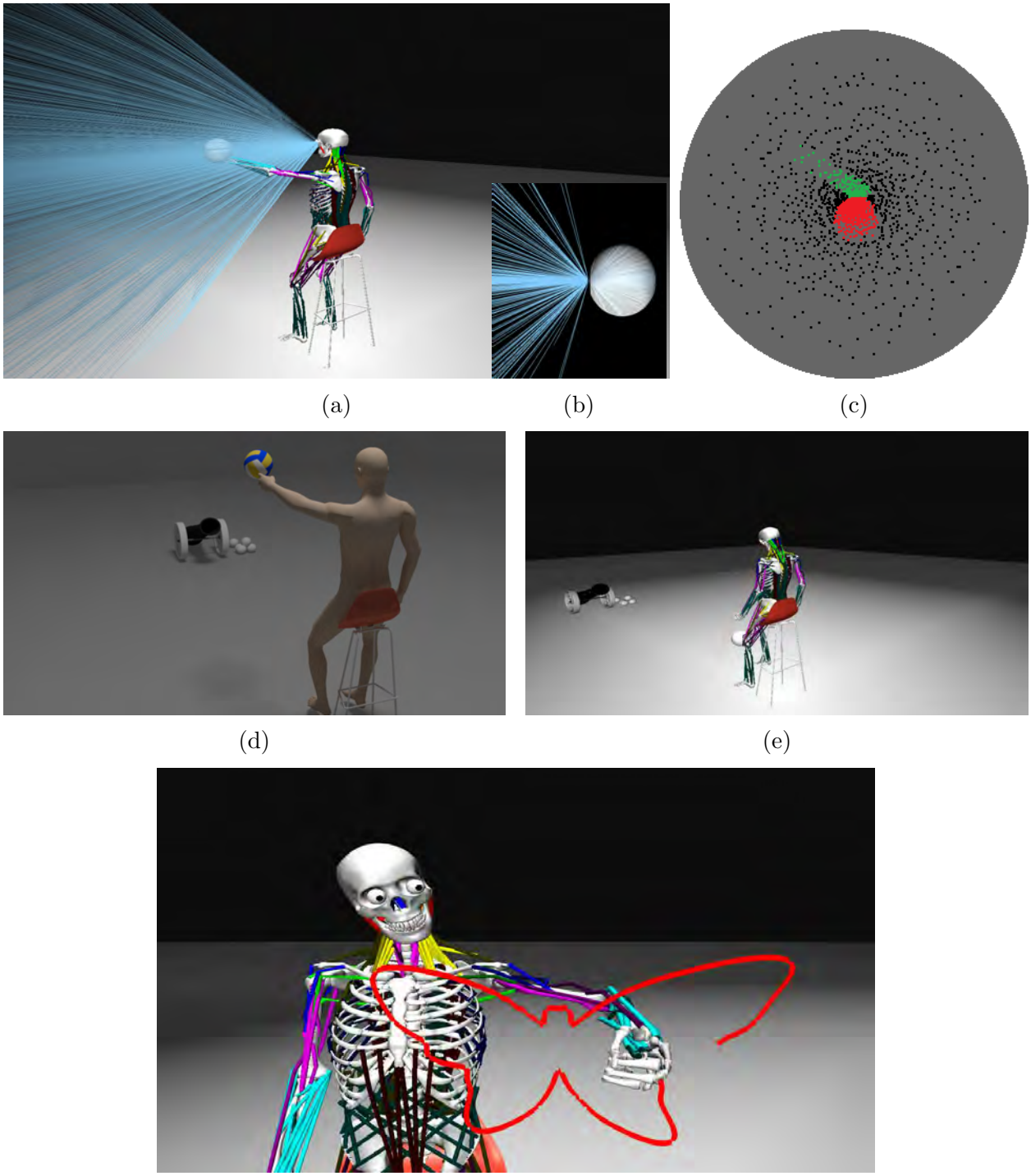


Figure 1.1: Our biomechanical human musculoskeletal model with its biomimetic sensorimotor control system, which includes virtual eyes (b) that visually sense the virtual environment via ray tracing (blue rays) (a) to compute the irradiance at cone-like RGB photoreceptors distributed on the foveated retina (c). The virtual human visually senses a moving target ball and intercepts it with its hands (d) (body model with skin) and feet (e), and draws a butterfly shape with its index finger (f).

research community as well.

1.3 Overview

The remainder of the dissertation is organized as follows: Chapter 2 reviews relevant prior work. Chapter 3 reviews our biomechanical human musculoskeletal model. Our novel sensorimotor control system for this model is developed in Chapter 4. Chapter 5 presents some of our experiments and results with our virtual human model. Chapter 6 concludes the thesis with a summary of the contributions of our work and our current plans for future work.

CHAPTER 2

Related Work

2.1 Anthropomorphic Modeling and Animation

Character modeling and animation has been a topic of interest in computer graphics since the early days of the field. Anthropomorphic character modeling is most relevant to this thesis. To this end, there exists a 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; Sifakis et al., 2005a), face (Sifakis et al., 2005b; Kähler et al., 2002; Lee et al., 1995a), and neck (Lee and Terzopoulos, 2006).

Furthermore, the modeling of muscle has also been an active research area, and researchers have proposed a variety of different modeling schemes (Ng-Thow-Hing, 2001; Irving et al., 2004). These prior efforts implemented different methods for each part of the human body, but they did not attempt full-body modeling and control.

The control of anthropomorphic models has been researched by Faloutsos et al. (2001a), Hodgins et al. (1995), and others. Most significantly to date, Lee et al. (2009) developed a comprehensive upper body, muscle-actuated biomechanical model, and a full-body version of this model was used by Si et al. (2014) (see also (Si, 2013)) to animate human swimming in simulated water with a CPG (Central Pattern Generator) control system.

2.2 Biomechanical Human Models

Waters (1987) and Lee et al. (1995b) first employed muscle based control system for face. The work by Faloutsos et al. (2001b)) articulated skeletal animation driven by joint torques, which

was the state-of-the-art in physics-based character animation two decades ago. Because of the complexity of the biological structures, it was not easy incorporate muscle into the control mechanism.

More recently, researchers have worked on constructing biomechanical human bodies. Sueda et al. (2008) worked on hand, Sifakis et al. (2005a) worked on torso, and Lee and Terzopoulos (2006) for neck. Upper body model was investigated by Lee et al. (2009). These prior efforts proposed different control schemes for human control system. The neck model consist of 9 bones and 72 muscle actuators while the full body consists of 103 bones with 163 DoF and 823 muscles. Hence; the challenge in those system comes from the complexity of the modeling of anatomically accurate human skeleton, joints, muscles, skin, and flesh models. Furthermore, we must also grapple with the difficulty in motor control problems due to the complexity of the human anatomy. We need muscle innervations with accurate muscle activations at every time step in order to control such an anatomical system. Recently biomechanical eyes was presented by Kepler and Linz (2004). They engineered a biomehcanical eye ball with mechanical muscle, and emulate the system of eye behavior.

2.3 Neuromuscular Control of Biomechanical Models

As pioneered by Lee and Terzopoulos (2006) in the context of neck-head control, neuromuscular learning approaches are the most biomimetic and promising in tackling high-dimensional, muscle-actuated biomechanical motor control problems. To control the neck-head system in gravity, rather than resorting to traditional inverse dynamics control, which is both unnatural and computationally expensive, these authors employed a conventional, artificial neural network with two hidden layers of sigmoidal units, which they trained offline through back-propagation learning. The training data comprised tens of thousands of random target head orientations as network inputs and the muscle activation levels required to achieve these target orientations of the skull as associated network outputs. Their trained neuromuscular controller was capable of efficiently controlling the biomechanical model online, thereby achieving a real-time neuromuscular neck-head control system. The neck-head biomechanical

model is actuated by 72 muscles, so their trained neural network controller maps 3 inputs (a target skull orientation) to 72 outputs (the muscle activations required to achieve this orientation) with adequate generalization. The back-propagation learning process was completed using their offline training procedure, which took approximately 10 hours to complete.

Conventional neural networks with only a few hidden layers have not proven effective at coping with the higher dimensionality and much greater quantities of training data required to learn to control biomechanical models of an order of magnitude greater complexity, such as the one presented by Lee et al. (2009). For this reason, Lee and Terzopoulos (2006) made no attempt to generalize their neck-head controller to deal with non-horizontally-oriented shoulders and, unsurprisingly, their trained neuromuscular controller fails to maintain satisfactory control unless the shoulders remain nearly horizontal. Lee et al. (2009) achieved an upper body control by solving inverse kinematics, inverse dynamics and muscle optimization problems; however, they did not achieve the control with neural network due to the complexity of the system with higher DoF.

We began to address this challenging problem through the use of deep learning Nakada and Terzopoulos (2015); the content of this paper is reproduced as Appendix A. Our system made it possible to have deeper architecture by having the unsupervised pre-training phase to initialize the parameters of a stacked denoising autoencoder optimally prior to the regular back-propagation based fine-tuning process. Our work proved that deeper network architectures could potentially overcome the obstacles that previous shallow neural networks faced in neuromuscular control systems.

2.4 Perception for Virtual Humans

In computer vision, Terzopoulos and Rabie (1995) proposed a biomimetic active vision system with foveated perception and sensorimotor control for biomechanically-simulated virtual animals. They also applied their "animat vision" sensorimotor system to virtual humans, demonstrating vision-guided bipedal locomotion and navigation (Rabie and Terzopoulos, 2000).

In computer graphics, Lee et al. (2002) proposed statistical eye movement model based on the empirical studies of eye saccade and achieved spatio-temporal saccade motions. They collected various eye movements using eye tracker, and use statistical approaches to generate eye movement for different situations (ex. conversations). Itti et al. (2003) proposed a system to track salient targets and have desired saccade eye motion in various video scenes, using a neurobiological model of eye movement and visual attention. They used recorded neck data of a monkey's movement and demonstrated successful eye tracking of salient targets in dynamic scenes with a virtual human face.

More recently, Yeo et al. (2012) presented a sensorimotor system for an anthropomorphic virtual character capable of visual target estimation tasks and demonstrating realistic ball catching actions, although their character is purely kinematic rather than biomechanically simulated, and it predicts the trajectories of thrown balls from their known positions and velocities in 3D space, without any biologically-inspired visual processing. They used Kalman filtering for visual target estimation tasks and achieved realistic ball interception movement. Although they developed the internal system to estimate the trajectory of the thrown ball, the 3D positions of the target were provided to the humanoid character without any biomimetic visual processing.

2.5 The Retina and Early Visual Processing

The retina is a light-sensitive layered tissue that converts incident light energy into nerve impulses, which are sent to the brain through the optic nerve. There are three layers of neural cells within the retina: photoreceptor cells (rods and cones), bipolar cells, and ganglion cells. The photoreceptors are spatially distributed in the retinal mosaic due to the fact that the photoreceptor cells tile the surface of the retina (Eglen, 2012; Galli-Resta, 1998). The mosaic serves as a sampling matrix that converts continuous light stimuli into discrete signals presented to the later stages of visual information processing in the brain (Curcio et al., 1990).

Unlike the regular, Cartesian grid arrangement of most artificial image sensors, primate

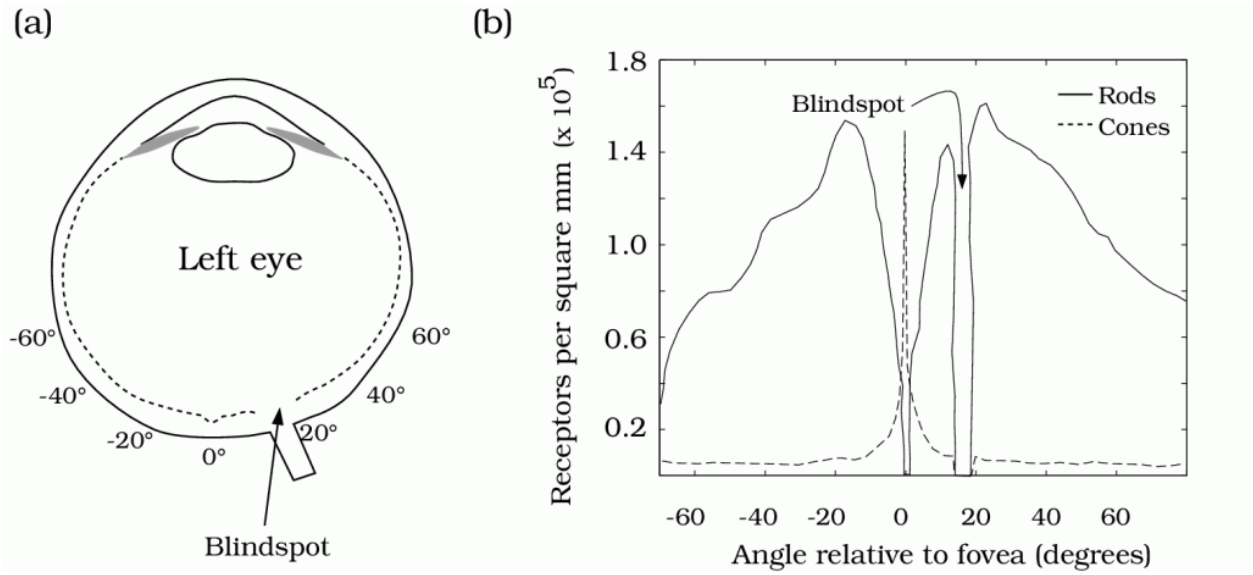


Figure 2.1: The distribution of rod and cone photoreceptors across the retina along a line passing through the fovea and blindspot of the human eye. (a) The density of the receptors is shown in degrees of visual angle relative to the position of the fovea for the left eye. (b) The cone receptors are concentrated in the fovea and their density decreases radially towards the periphery. The rod photoreceptors are absent from the fovea and reach their highest density 10 to 20 degrees peripheral to the fovea. No photoreceptors are present in the blindspot, the location in the retina from which the optic nerve exits the eye. From (Wandell, 1995).

visual sampling is fundamentally space variant (Schwartz, 1977), and this is ubiquitous in higher vertebrate visual systems (Hughes, 1977; Bonmassar and Schwartz, 1994). In the human retina, the density of cones, which mediate color, photopic vision, peaks in the foveal region and decreases radially toward the periphery (Figure 2.1). Sensed visual stimuli are encoded in the activity of ganglion cells that provide, through the optic nerve, all the visual information received by the brain. The ganglion cells encode information in a largely independent manner (Nirenberg et al., 2001) to remove correlation from the image and increase efficiency in encoding spike trains (Tokutake and Freed, 2008).

A review article by Bolduc and Levine (1998) surveys foveated computer vision systems at the time that active vision was a "hot" topic in the field. More recently, Grady (2004) explored biomimetic, space-variant image processing on a 2D graph-based network of photoreceptors. Rather than using any 2D image representation, the artificial vision system that we develop in this thesis eschews the spatial arrangement of photoreceptors, encoding the

visual signal as a one-dimensional Optic Nerve Vector (ONV). Subsequently, our artificial vision system applies a collection of perception DNNs that operate on the ONV to control eye movements and provide actionable information to the DNNs responsible for the neuromuscular control of the motor subsystem in our human sensorimotor model. Our perception DNNs are fully connected artificial neural networks, not the currently popular Convolutional Neural Networks (CNNs).[1]

[1]CNNs, such as LeNet (LeCun et al., 1998), were inspired by the Neocognitron architecture first proposed in 1979 by Fukushima (Fukushima, 1988), which was in turn inspired by the model of primary visual cortex proposed by Hubel and Wiesel in 1959 that emphasized its apparent cascade of image feature extractions. The advent of the GPUs (Graphical Processing Units) has enabled researchers in the field to train deeper CNNs and fully-connected NNs with more elaborate architectures, which has dramatically improved their performance in computer vision tasks.

CHAPTER 3

Biomechanical Human Musculoskeletal Model

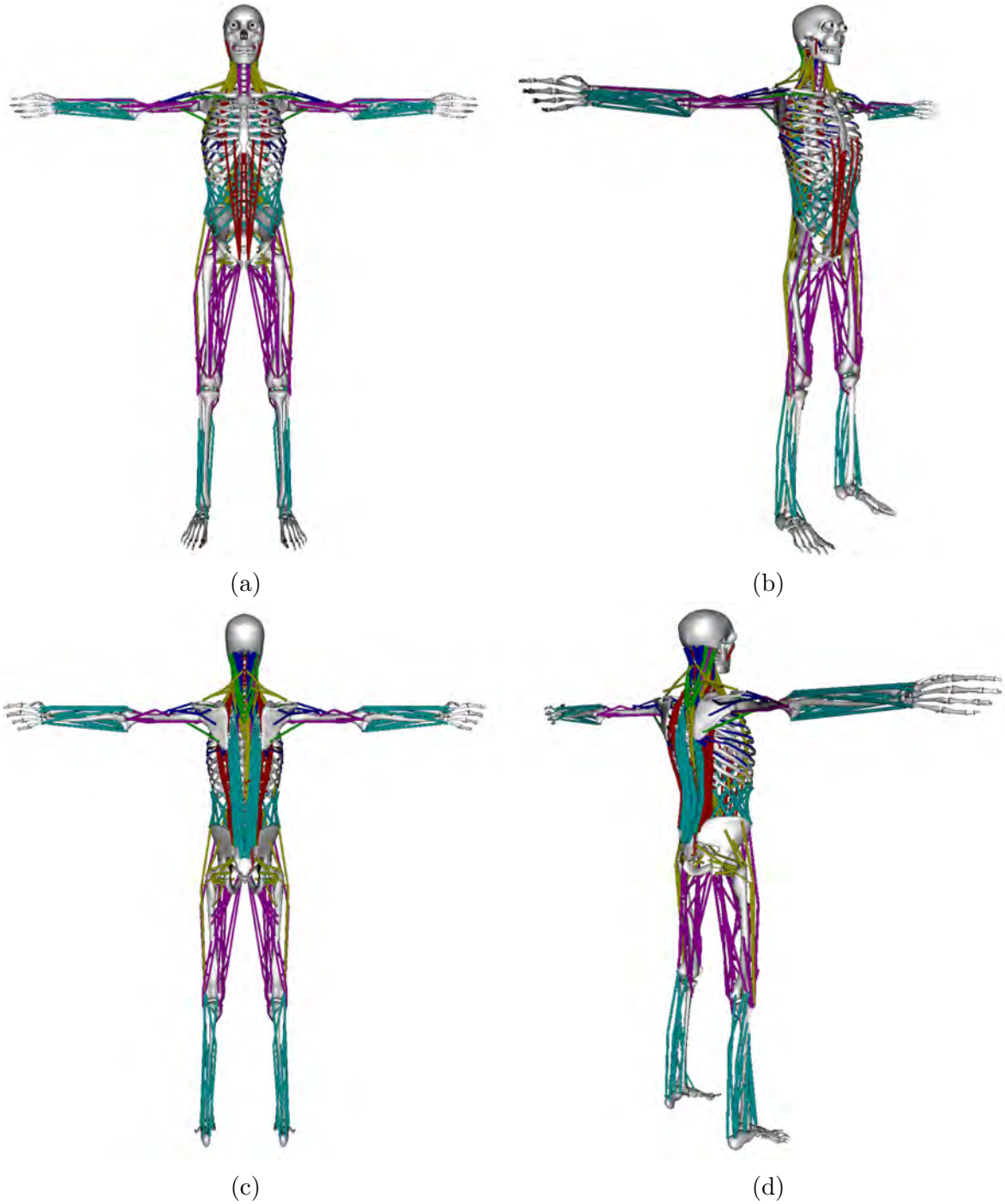
Figure 3.1 shows the musculoskeletal system of our anatomically accurate human model. It includes all of the relevant articular bones and muscles—103 bones connected by joints comprising 163 articular degrees of freedom, plus a total of 823 muscle actuators embedded in a finite element model of the musculotendinous soft tissues of the body.[1] Each skeletal muscle is modeled as a Hill-type uniaxial contractile actuator that applies forces to the bones at its points of insertion and attachment. The human model is numerically simulated as a force-driven articulated multi-body system (refer to (Lee et al., 2009) for the details).

Each muscle actuator is activated by an independent, time-varying, efferent activation signal $a(t)$. Given our human model, the overall challenge in neuromuscular motor control is to determine the activation signals for each of its 823 muscles necessary to carry out various motor tasks. For the purposes of our research to date, we mitigate complexity by placing our virtual human in a seated position, immobilizing the pelvis as well as the lumbar and thoracic spinal column vertebra and other bones of the torso, leaving the cervical column, arms, and legs free to articulate.

3.1 Neck-Head and Limbs Models

Neck-head: Figure 3.2 shows in greater detail the neck-head biomechanical complex, which is rooted at the thoracic vertebra (T1), with its seven vertabrae, C7 through C1 (atlas), progressing up the cervical spine to the skull, which is an end-effector of substantial mass.

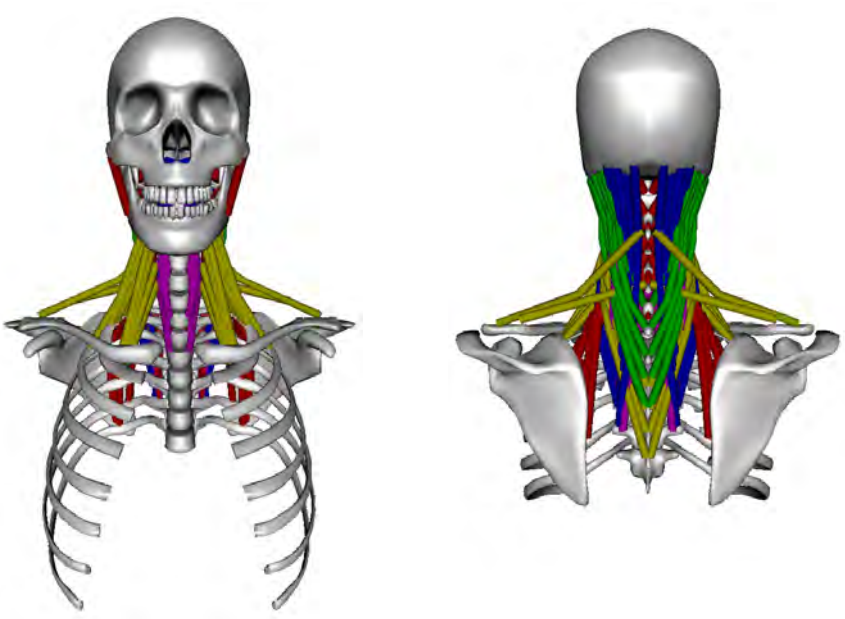
[1]For the purposes of the research reported in this thesis, the finite element soft-tissue simulation, which produces realistic flesh deformations, is unnecessary and it is excluded so as to reduce computational cost.



(a) (b)

(c) (d)

Figure 3.1: The biomechanical human musculoskeletal model, showing the skeletal system with its 103 bones and the 823 Hill-type muscle actuators.



(a) Anterior (b) Posterior

Figure 3.2: The biomechanical neck-head musculoskeletal model showing the 216 neck muscles, for some of which the displayed vertebra below C1 plus the ribs, clavicles, and scapulae serve as root attachments.

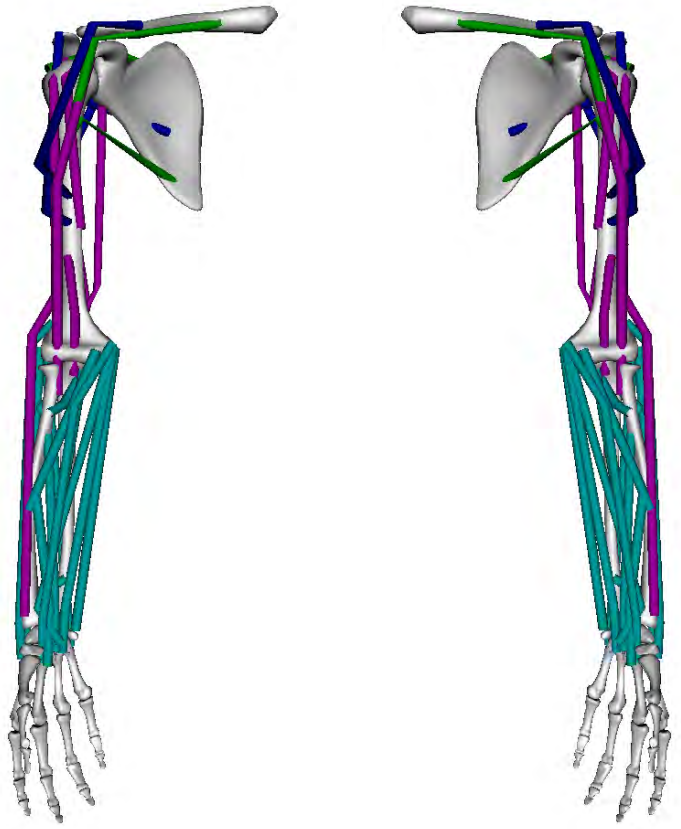
A total of 216 short, intermediate, and long hill-type uniaxial muscle actuators arranged in deep, intermediate, and superficial layers, respectively, actuate the seven 3-degree-of-freedom joints of the neck-head biomechanical complex.

Each joint is incorporated with damped rotational springs to approximate the stiffness of the intervertebral discs. The spring torque is computed as

$$\tau_s = -k_s(q - q_0) - k_d\dot{q}, \tag{3.1}$$

where q is the joint angle, q_0 is the joint angle in the natural pose, $\dot{q}$ is the joint angular velocity, k_s is the spring stiffness, and k_d is the damping coefficient. The passive behavior of the cervical column can be modified by adjusting the stiffness and damping coefficients.

Arms: Figure 3.3 shows the musculoskeletal model of the bilaterally symmetric arms, which are rooted at the clavicle and scapula, and include the humerus, ulnar, and radius bones. A total of 29 Hill-type uniaxial muscles actuate the shoulder, elbow, and wrist joints

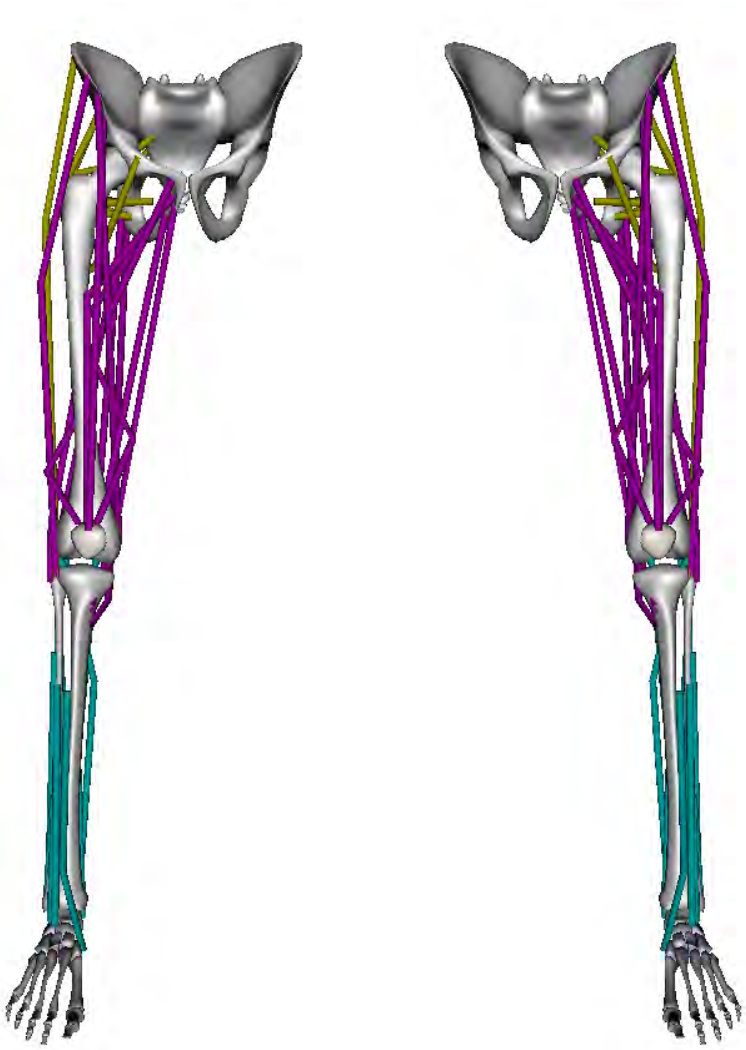


(a) Right arm (b) Left arm

Figure 3.3: The musculoskeletal anatomy of the arms, showing the 29 arm muscles, for some of which the clavicle and scapula serve as root attachments.

in each arm. The bones of the hand, which for the purposes of this thesis are actuated only kinematically, serve as the end effector.

Legs: Figure 3.4 shows the musculoskeletal model of the bilaterally symmetric legs, which are rooted at the pelvis and include the femur, patella, tibia, and fibula bones. A total of 39 Hill-type uniaxial muscles actuate the hip, knee, and ankle joints in each leg. The bones of the foot, which for the purposes of this thesis are actuated only kinematically, serve as the end effector.



(a) Right leg (b) Left leg

Figure 3.4: The musculoskeletal anatomy of the legs, showing the 39 leg muscles, for some of which the pelvis serves as a root attachment.

3.2 Simulation of the Musculoskeletal System

Next, we will briefly describe the simulation of the skeletal and muscle systems of our model. Additional details are provided in (Lee et al., 2009).

3.2.1 Skeletal System

The equations of motion of the skeletal system are written as

$$\boldsymbol{M}(\boldsymbol{q}) \begin{bmatrix} \ddot{\boldsymbol{q}}_m \\ \ddot{\boldsymbol{q}}_p \end{bmatrix} + \boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}}) = \begin{bmatrix} \boldsymbol{P}(\boldsymbol{q})\boldsymbol{f}_c \\ \boldsymbol{0} \end{bmatrix} + \boldsymbol{J}^T \boldsymbol{f}_e, \tag{3.2}$$

where $\dot{\boldsymbol{q}}$ and $\ddot{\boldsymbol{q}}$ are the joint velocities and accelerations, $\boldsymbol{M}$ is 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, which transforms applied external force to joint torque. The moment arm matrix $\boldsymbol{P}$ maps the contractile muscle force $\boldsymbol{f}_c$ to the space of joint torques. The computation technique for the moment arm matrix is introduced in (Gonzalez et al., 1997). Finally, (3.2) can be compactly written as

$$\ddot{\boldsymbol{q}} = \phi(\boldsymbol{q}, \dot{\boldsymbol{q}}, \boldsymbol{\tau}). \tag{3.3}$$

We use forward dynamics to compute ϕ by computing the $\ddot{\boldsymbol{q}}$ from the generated torque produced by the muscle force. 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

$$\dot{\boldsymbol{q}}(t + \Delta t) - \dot{\boldsymbol{q}}(t) = \Delta t \phi(\boldsymbol{q}(t + \Delta t), \dot{\boldsymbol{q}}(t + \Delta t), \tau). \tag{3.4}$$

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

$$\begin{aligned} \delta\dot{\boldsymbol{q}} &= \Delta t \left[\phi(\boldsymbol{q}(t), \dot{\boldsymbol{q}}(t), \tau) + \frac{\partial \phi}{\partial \boldsymbol{q}} \delta \boldsymbol{q} + \frac{\partial \phi}{\partial \dot{\boldsymbol{q}}} \delta \dot{\boldsymbol{q}} \right] \\ &= \Delta t \left[\phi(\boldsymbol{q}(t), \dot{\boldsymbol{q}}(t), \tau) + \frac{\partial \phi}{\partial \boldsymbol{q}} \Delta t (\dot{\boldsymbol{q}}(t) + \delta \dot{\boldsymbol{q}}) + \frac{\partial \phi}{\partial \dot{\boldsymbol{q}}} \delta \dot{\boldsymbol{q}} \right] \end{aligned} \tag{3.5}$$

Thus, we can compute the joint velocities at the next time step. Then the joint angles at

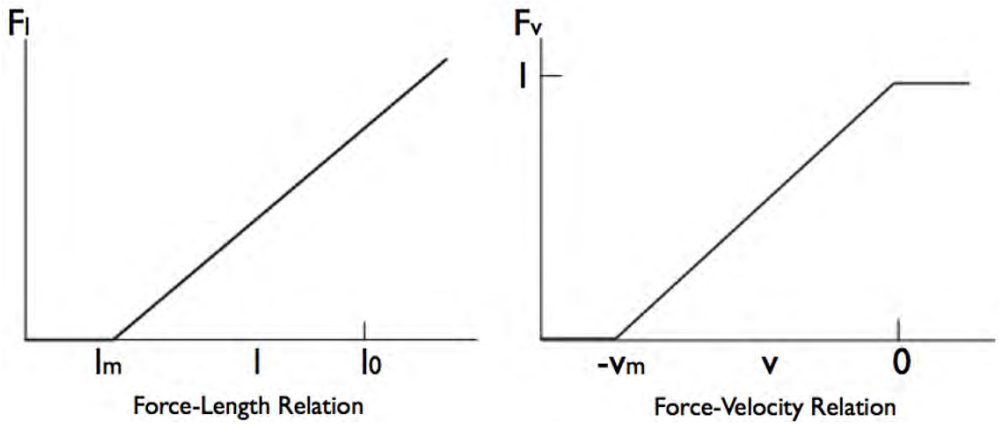


Figure 3.5: The force-length and force-velocity relations of the Hill-type muscle model.

the next time step can be computed with another time integration.

3.2.2 Muscle System

We use the Hill-type muscle model (Lee et al., 2009). The muscle force $f_m = f_P + f_C$ is the combination of two components. 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:

$$f_P = \max(0, k_s(\exp(k_c e) - 1) + k_d \dot{e}), \tag{3.6}$$

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

$$f_C = aF_l(l)F_v(\dot{l}), \tag{3.7}$$

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 3.5 plots the force-length and force-velocity relations. The former, F_l is represented as $F_l(l) = \max(0, k_{max}(l - l_m))$, where l_m is the minimum length for muscle to generate the force, k_{max} is the maximum stiffness of activated muscle. The latter, $F_v(\dot{l})$ is represented as

$F_v(\dot{l}) = \max(0, 1 + min(\dot{l}, 0)/v_m)$, 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 Muscle Control System

The biomechanical human musculoskeletal model includes a hierarchical control system. As we will explain in Chapter 4, high-level voluntary controllers generate efferent activation control signals for muscles using trained DNNs. Then, a low-level reflex control system adjusts the muscle activations by comparing current and desired muscle lengths and length rates.

Muscle reflex control, as employed in (Lee and Terzopoulos, 2006; Lee et al., 2009), improves the stability of the biomechanical system. Each muscle reflex controller inputs the signal from the associated voluntary controller as well as a feedback signal from the muscle, and it outputs a modified activation level for the muscle that reflects the current muscle state and yields higher stability, as follows:

$$a = \min(1, \max(0, a_f + a_b)), \tag{3.8}$$

where a_f is the incoming activation signal from the voluntary controller. The reflex controller computes the reflex activation signal

$$a_b = s(k_p(e - e_d) + k_d \operatorname{sat}_m(\dot{e} - \dot{e_d})), \tag{3.9}$$

where k_p and k_d are proportional and derivative gains, respectively, s is the feedback gain scaling factor, and e and $\dot{e}$ are the muscle strain and strain rate, and e_d and $\dot{e}_d$ are the desired strain and strain rate, respectively. The desired stain and strain rate are computed by the setpoint method in the voluntary controller (Lee et al., 2009). We set $k_d = 0.05, k_p = 8$. Large derivative feedback forces made the simulation unstable; therefore, we use the following function to maintain it within acceptable bounds:

$$\text{sat}_m(x) = \begin{cases} x & |x| < m \\ m\,\text{sgn}(x) & \text{otherwise,} \end{cases} \tag{3.10}$$

where m is set to 2.0.

The simulator activates the muscles with the activation input, and applies the muscle forces to the joint space of the skeletal system using the moment arm matrix. Using the system of equation in the previous section, we compute the dynamics with environmental force such as gravity and external forces and provide the output to the next time step. This output is passed to the setpoint generator, which returns the feedback strain and strain rate for the reflex controller to be employed in the next time step.

3.3 Synthesizing Training Data

The DNNs that implement the voluntary neuromuscular controllers in our sensorimotor control system, are trained offline in a supervised manner, as will be explained in Chapter 4. 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.,

$$\text{Training input-output pair} \begin{cases} \text{input:} & [\boldsymbol{e}, \boldsymbol{a}_c]; \\ \text{output:} & \Delta\boldsymbol{a}. \end{cases} \tag{3.11}$$

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:

$$\ddot{\boldsymbol{q}}^* = k_p(\boldsymbol{q}^* - \boldsymbol{q}) + k_d(\dot{\boldsymbol{q}}^* - \dot{\boldsymbol{q}}), \tag{3.12}$$

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).

CHAPTER 4

Sensorimotor System

Figure 4.1 presents an overview of the sensorimotor system, showing its perception and motor subsystems, and the figure caption describes the information flow and the functions of its 15 deep neural network (DNN) controllers. The following sections develop, first, the 5 recurrent motor DNNs (Figure 4.1(g),(h)). This is followed by the details of the eyes (Figure 4.1(e)) and their retinas (Figure 4.1(b)). Finally, we develop the 10 feedforward perception DNNs (Figure 4.1(c)L,(c)R,(d)L,(d)R)

4.1 Recurrent Motor Control DNNs (11–15)

The neck-head complex and limbs of our biomechanical human musculoskeletal model are actuated by groups of skeletal muscles stimulated by deep neural network (DNN) neuromuscular controllers that generate efferent signals, $a_i(t)$, $1 \leq i \leq n$, that activate the muscles. The function of the neck motor DNN (Figure 4.2) is to generate activation signals feeding the neck muscles to balance the mass of the head in gravity atop the flexible cervical column while actuating controlled head movements to achieve target head poses. The function of the four limb motor DNNs (Figure 4.3), is to generate activation signals feeding the muscles of each of the limbs in order to execute controlled arm and leg movements; in particular, extending any limb in gravity to reach out to a physical target.

Except for the sizes of the input and output layers, the architecture of the five motor DNNs is identical. Their input layers include units that represent errors relative to the target. They also include units that represent the current neuromuscular activation state, accounting for the fact that the control outputs should differ, even for the same movement,

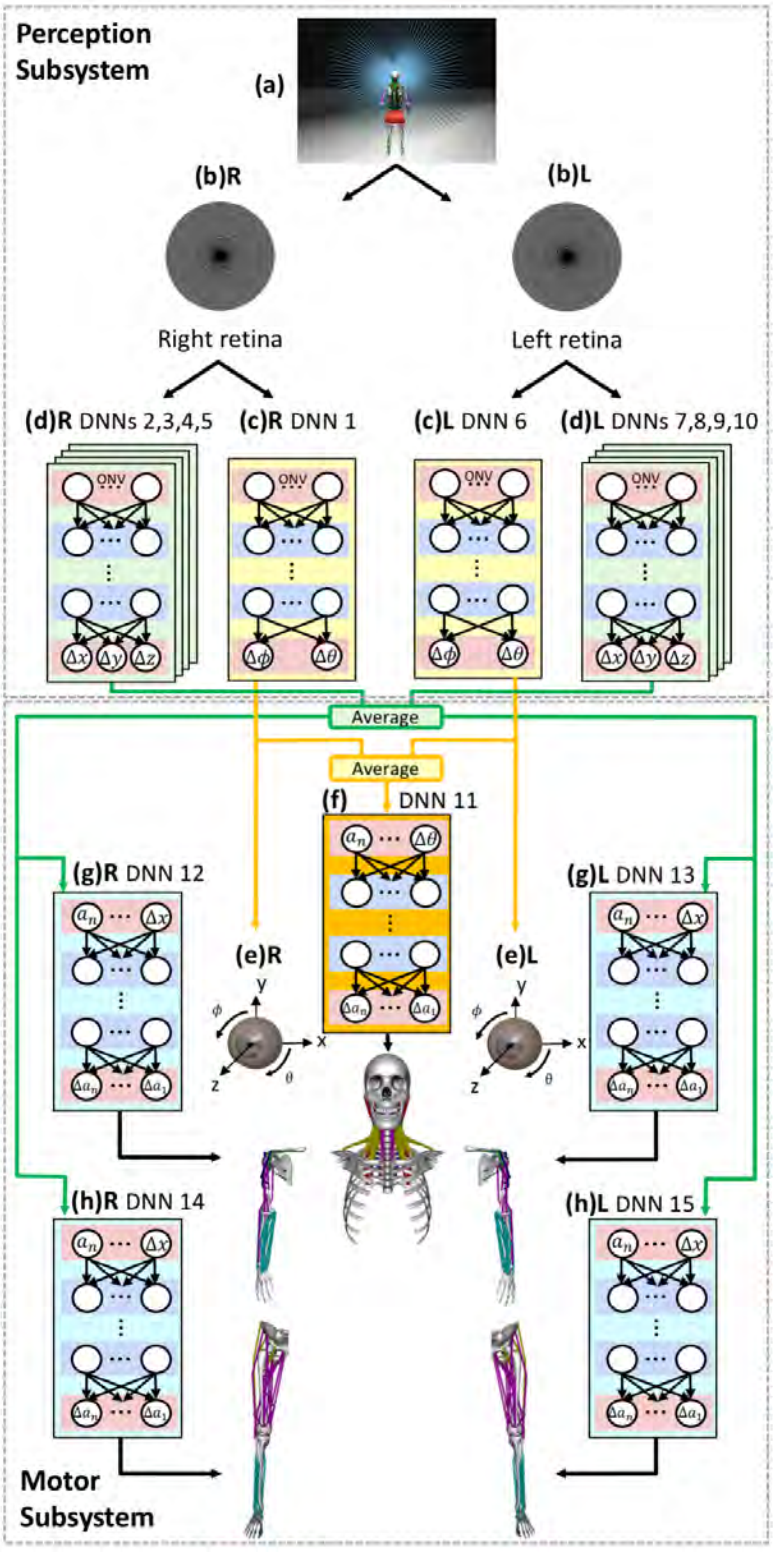


Figure 4.1: Sensorimotor system architecture, showing the modular neural network controllers in the perception subsystem (top) and motor subsystem (bottom). The controllers include a total of 15 DNNs, labeled 1–15, of four types, colored green, yellow, orange, and blue. In the perception subsystem, each photoreceptor on the retinas casts a ray into the virtual world (a), which computes and returns the irradiance at that photoreceptor. (b) The arrangement of the 3,600 photoreceptors (black dots) on the left (b)L and right (b)R foveated retinas. Each retina outputs the 10,800-dimensional Optic Nerve Vector (ONV). The two (yellow) feedforward perception DNNs (c) (1,6) input the ONV and output left-eye (c)L and right-eye (c)R error angles used to control the movements of the eyes (e) for the purposes of visual target foveation. The eight (green) feedforward perception DNNs (d), (d)L (7,8,9,10) for the left eye and (d)R (2,3,4,5) for the right eye also input the ONV and output the observed limb-to-target discrepancy estimations. In the motor subsystem, the (orange) neck muscle motor control DNN (f) inputs the average response of the left (c)L and right (c)R foveation DNNs along with the current activations of the 216 neck muscles and outputs the desired changes to the activations of those muscles. Similarly, the (blue) muscle motor control DNNs (g),(h) (12,13,14,15) for the four limbs input the average response of the left (d)L and right (d)R perception DNNs along with the current activation of the 29 arm or 39 leg muscles and outputs the desired changes to the activations of those muscles.

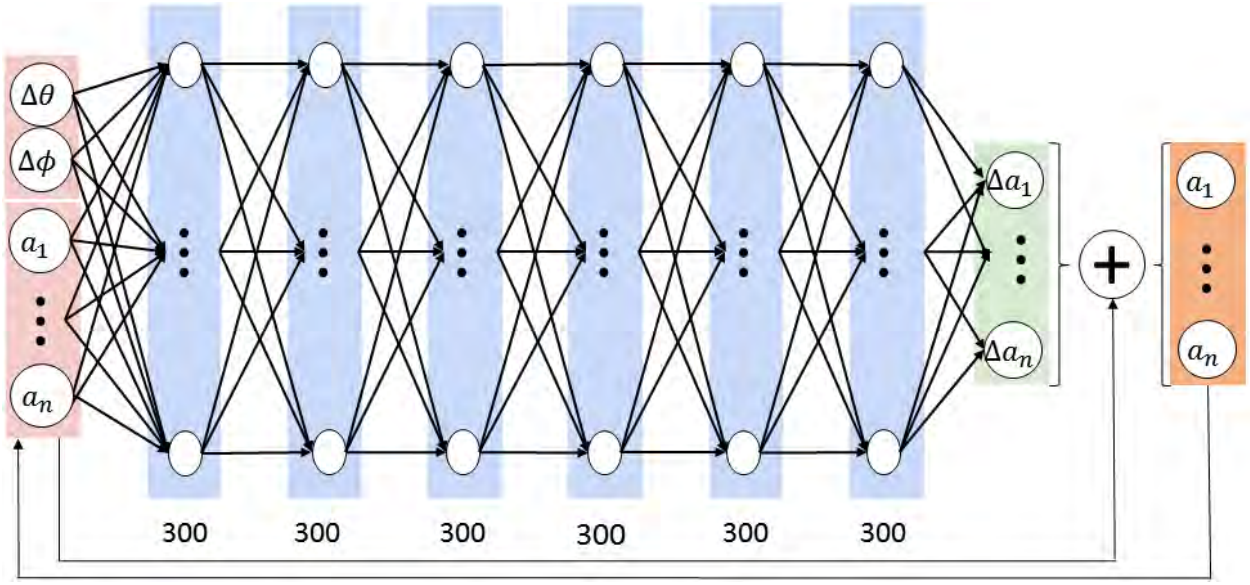


Figure 4.2: The fully-connected recurrent neuromuscular controller architecture. The network shown is for the motor control of the neck-head complex.

given different muscle activation states. The networks have 6 hidden layers, each of which include 300 units. The output layer consists of units that encode the changes $\Delta a_i(t)$ to the muscle activations, which then additively update the current muscle activations:

$$a_i(t + \Delta t) = a_i(t) + \Delta a_i(t). \tag{4.1}$$

The updated muscle activations provide proprioceptive feedback to the input layer, making these motor DNNs recurrent networks.

The motor DNNs are networks of rectified linear units (ReLU) whose initial weights are sampled from the zero-mean normal distribution with standard deviation $\sqrt{\frac{2}{fan\_in}}$, where *fan_in* is the number of input units in the weight tensor (He et al., 2015). For training, we apply Adaptive Moment Estimation (Adam) (Kingma and Ba, 2014) as the stochastic optimization method, using the following parameters: $lr = 0.000001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 1e-08$, $\alpha = 0.001$, where β_1 and β_2 represent the exponential decay rate for momentum estimates taking an average and an average of squared gradients, respectively, ϵ prevents a divide-by-zero error, lr is the learning rate and α is the step size. An early stopping condition is set to avoid overfitting and the loss function is mean squared error.

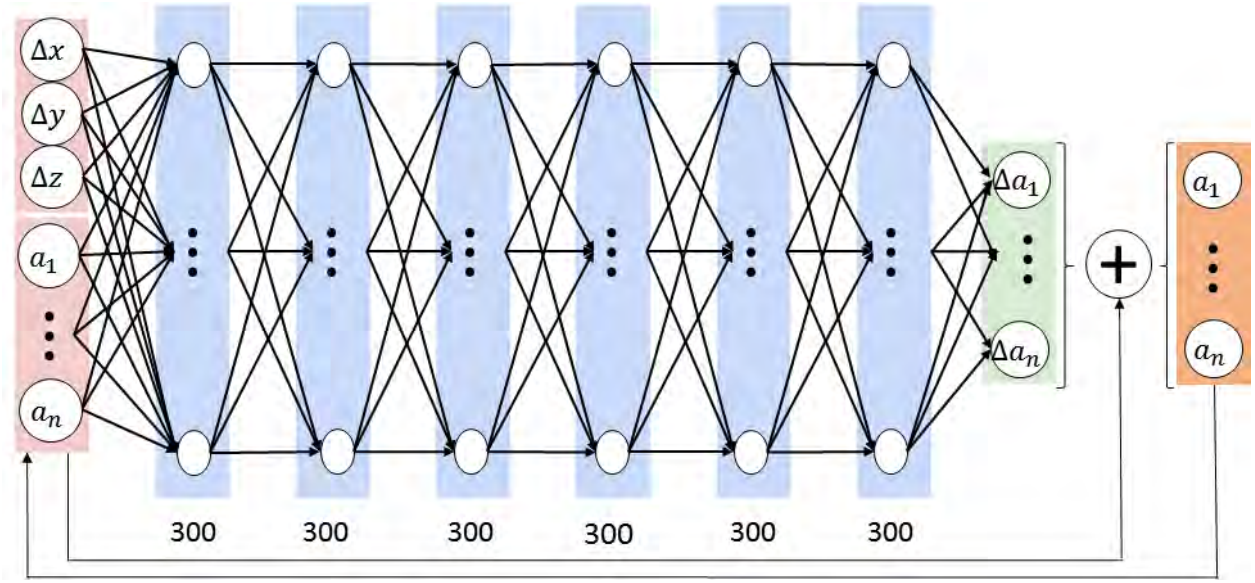


Figure 4.3: The fully-connected, recurrent neuromuscular controller architecture. The network shown is for the motor control of a limb (arm or leg).

In the following sections, we provide additional details about each of the DNNs and how they are trained.

4.1.1 Neck-Head Motor DNN (11)

DNN architecture: As shown in Figure 4.2, the input layer of the neck-head motor DNN consists of 218 units that comprise 2 units for the head pose target error angles, $\Delta\theta$ and $\Delta\phi$, and 216 units for the activations a_i, $1 \leq i \leq 216$, of the neck muscles. The output layer consists of 216 units providing the muscle activation changes Δa_i, $1 \leq i \leq 216$, which then update the muscle activations according to (4.1).

Offline training data synthesis and network training: To train the DNN, we use our biomechanical human musculoskeletal model to synthesize training data as follows: We specify target orientations for the neck and compute the angular differences $\Delta\theta$ and $\Delta\phi$ from the current neck orientation to the target. Given the desired angular differences as the target control input, we compute the desired muscle activations using inverse kinematics and inverse dynamics with muscle optimization applied to the biomechanical neck model, as

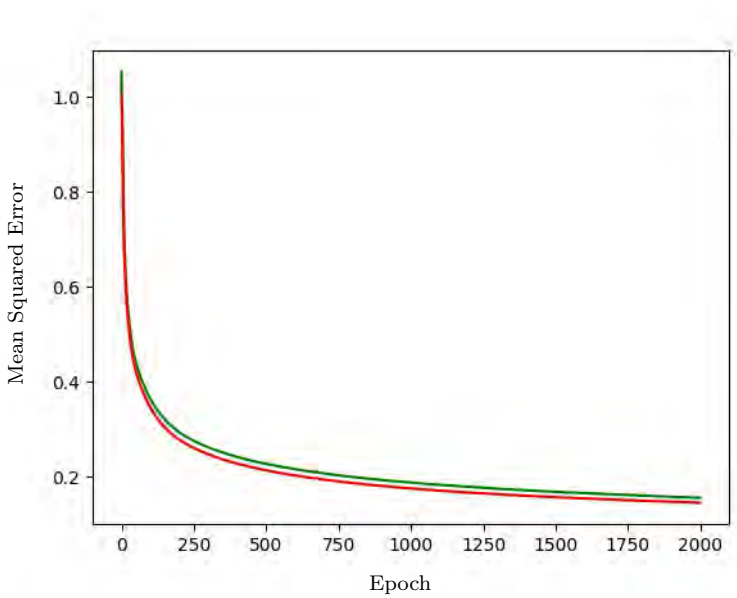


Figure 4.4: Progress of the backpropagation training process for the neck motor DNN on the training and validation datasets. The ratio of the sizes of the training and validation datasets is 9:1. The green plot shows the progress of the training process with the training dataset while the red plot shows its progress with the validation dataset.

described in Section 3.3, to determine the desired delta muscle activations, which constitute the desired output of the neck motor DNN.

The *offline* training data synthesis process takes 0.8 seconds to solve for 0.02 seconds simulation. The resulting input/output training pair then consists of an input comprising the concatenation of the desired angle differences $\Delta\theta$ and $\Delta\phi$ along with the current muscle activations a_i, $1 \leq i \leq 216$, and an associated output that consists of the desired delta muscle activations Δa_i, $1 \leq i \leq 216$. Repeatedly placing the target at random positions in the visual field, we generated a large training dataset of 1M input-output pairs.

The backpropagation DNN training process converged to a small error in 2,000 epochs. Figure 4.4 shows the progress of the training process. After the DNN is trained, it serves as the *online* motor controller.

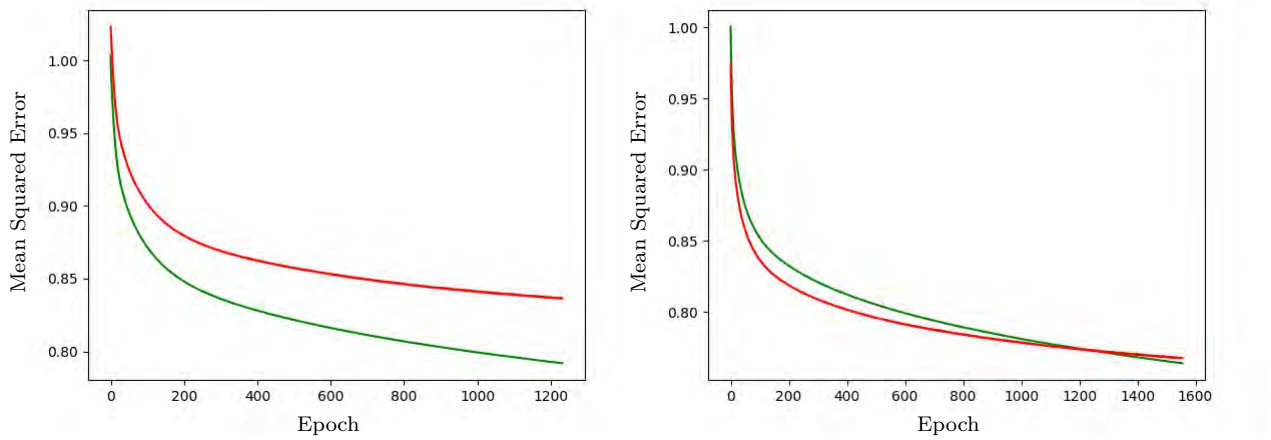
4.1.2 Limb Motor DNNs

DNN architecture: As shown in Figure 4.3, the input layer of the limb motor DNN consists of 3 units that specify, Δx, Δy, and Δz, the estimated discrepancy between the 3D positions of the end effector and the target, and the current activations of the 26 arm muscles, a_i, $1 \leq i \leq 26$, or those of the 36 leg muscles, a_i, $1 \leq i \leq 36$. The output layer consists of units providing the muscle activation changes Δa_i, which then update the muscle activations according to (4.1).

Offline training data synthesis and network training: To train the DNN, we use our biomechanical human musculoskeletal model to synthesize training data as follows: We present a target ball and extend a limb (arm or leg) towards the ball. We compute the 3D discrepancy, Δx, Δy, and Δz, between the *known* 3D positions of the end effector and the target. Given this 3D discrepancy plus the current muscle activations as its input, the desired output of the network is the delta muscle activations, which are computed by solving inverse kinematics, inverse dynamics, and muscle optimization, as described in Section 3.3, to reach the target ball. Repeatedly placing the target sphere at random positions and randomly articulating the limb to reach for it in space, we generated a large training dataset of 1M input-output training pairs.

Arm motor DNNs: Figure 4.5 plots the progress of the backpropagation training process for the left arm motor DNN. Using the 0.6M and 1M dataset, it converged to a small error after 1,223 epochs and 1,533 epochs, respectively, which triggered the early stopping condition to avoid overfitting. On the 0.6M dataset, however, the training process exhibited a large gap between the training and validation errors, and the trained network turned out to be insufficiently robust and stable as an online controller.

Figure 4.6 plots the progress of the backpropagation training process for the right arm motor DNN. It converged to a small error after 822 epochs, which triggered the early stopping condition to avoid overfitting.



(a) (b)

Figure 4.5: Progress of the backpropagation training of the left arm motor DNN on datasets with (a) 0.6M and (b) 1M data samples, respectively. The size ratio of the training dataset to the validation datset is 9:1. The green plot shows the progress of the training process with the training dataset while the red plot shows its progress with the validation dataset.

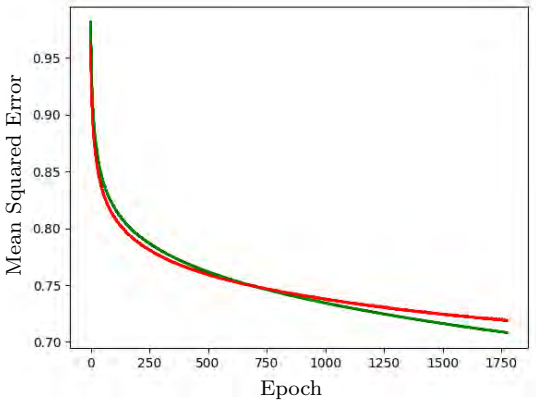
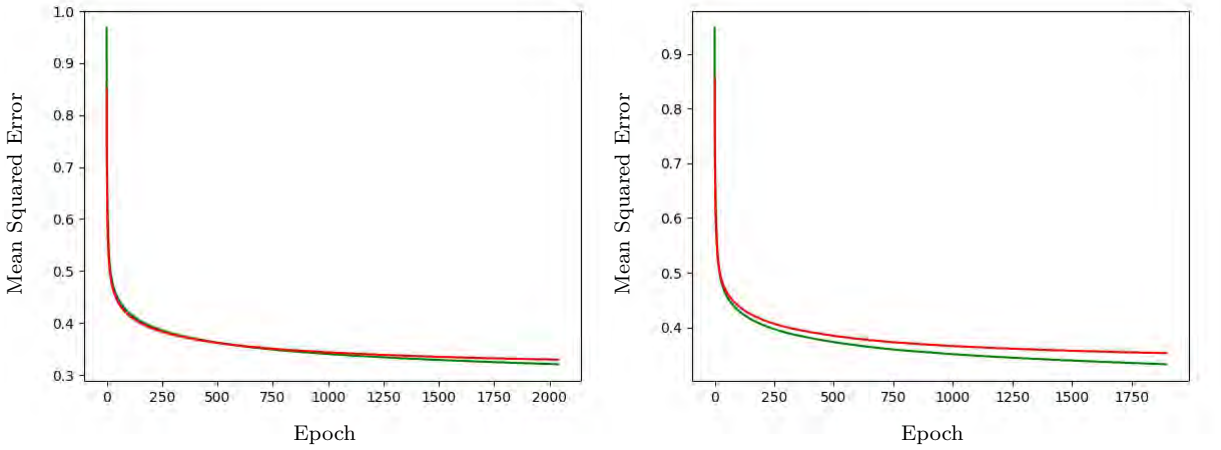


Figure 4.6: Progress of the backpropagation training of the right arm controller on a dataset of 1M input-output training pairs. The size ratio of the training dataset to the validation datset is 9:1. The green plot shows the progress of the training process with the training dataset while the red plot shows its progress with the validation dataset.



(a) (b)

Figure 4.7: Progress of the backpropagation training for (a) the left leg and (b) the right leg motor DNNs on a dataset of 1M input-output training pairs. The size ratio of the training dataset to the validation dataset is 9:1. The green plot shows the progress of the training process with the training dataset while the red plot shows its progress with the validation dataset.

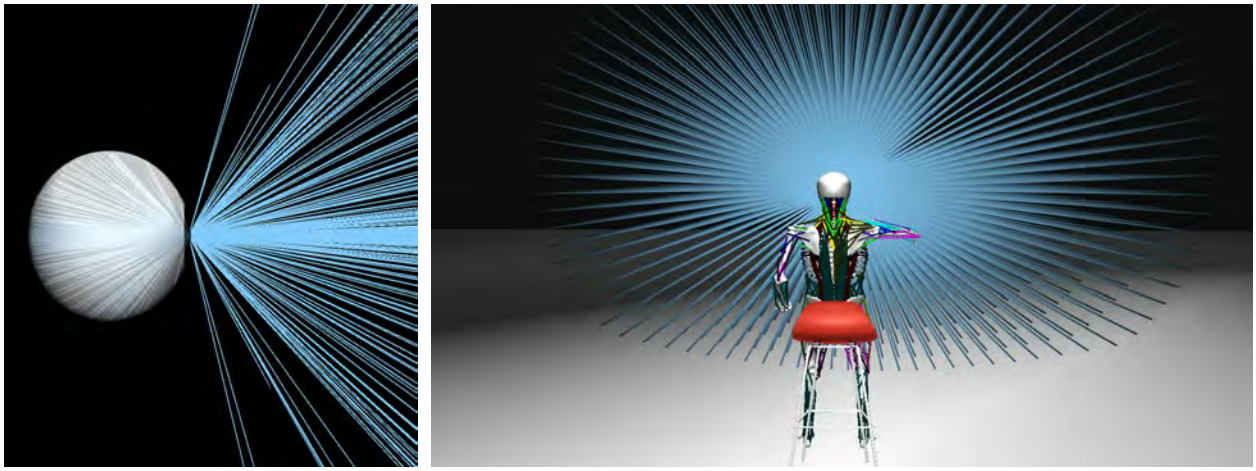
Leg motor DNNs: Figure 4.7 plots the progress of the backpropagation DNN training process for the leg motor controllers. For the left leg controller it converged to a small error after 2000 epochs and for the right leg controller it converged to a small error after 1850 epochs, which triggered the early stopping condition to avoid overfitting.

4.2 Eye and Retinal Models

4.2.1 Eye Model

We modeled the eyes by taking into consideration the physiological data from an average human.[1] As shown in Figure 4.1(e), we model the virtual eye as a sphere of radius of 12mm, 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

[1]The transverse size of an average eye is 24.2 mm and its sagittal size is 23.7 mm. at its center. The approximate field of view of an individual eye is 30 degrees to superior, 45 degrees to nasal, 70 degrees to inferior, and 100 degrees to temporal. When two eyes are combined, the field of view becomes about 135 degrees vertically and 200 degrees horizontally.



(a) (b)

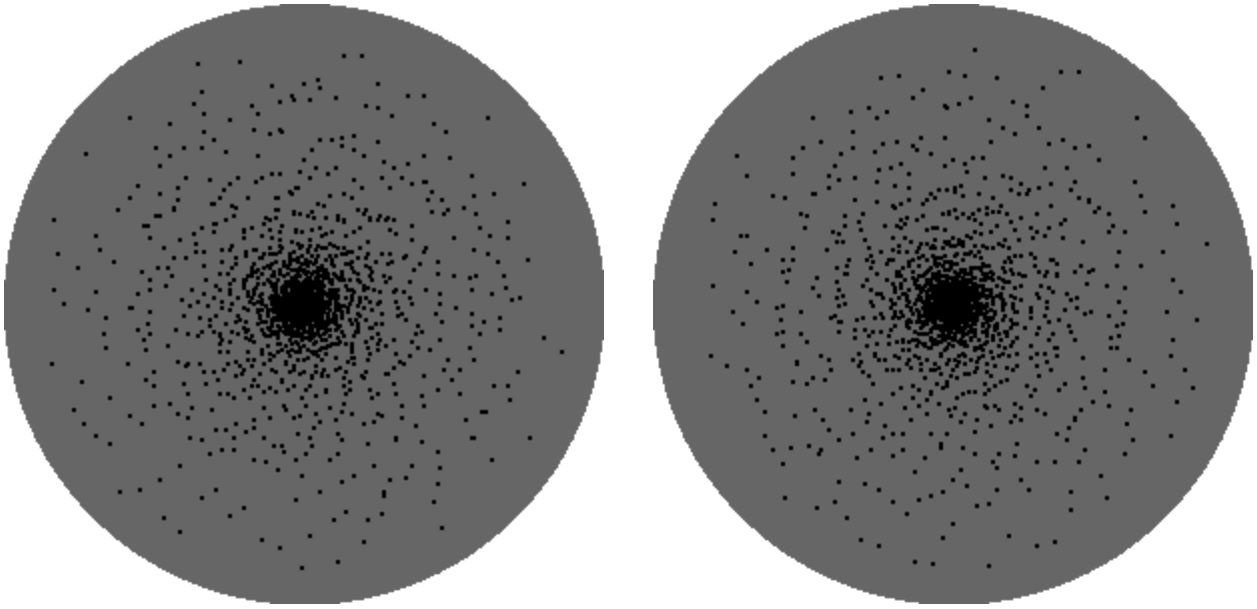
Figure 4.8: (a) Rays cast from the positions of photoreceptors on the retina through the ideal pinhole aperture and out into the scene by the raytracing procedure that computes the irradiance responses of the photoreceptors. (b) All the cast rays as the seated virtual human looks forward with both eyes.

positions looking straight ahead when $\theta = \phi = 0°$. At least for now, we model the eye as an idealized pinhole camera with aperture at the center of the pupil and with horizontal and vertical fields of view of 167.5°.

We can compute the irradiance at any point on the hemispherical retinal surface at the back of the eye using the conventional raytracing technique of computer graphics rendering (Shirley and Morley, 2003). Sample rays from the positions of photoreceptors on the hemispherical retinal surface are cast through the pinhole and out into the 3D virtual world where they recursively intersect with the visible surfaces of virtual objects and query the virtual light sources according to the Phong local illumination model. The irradiance values returned by these rays determine the light impinging upon the retina at the photoreceptor positions. Figure 4.8 illustrates this retinal imaging process.

4.2.2 Placement of the Photoreceptors

To simulate foveated perception, we use a Gaussian noise-perturbed log-polar distribution to model the nonuniform 2D positions of the photoreceptors on the retina. At present we



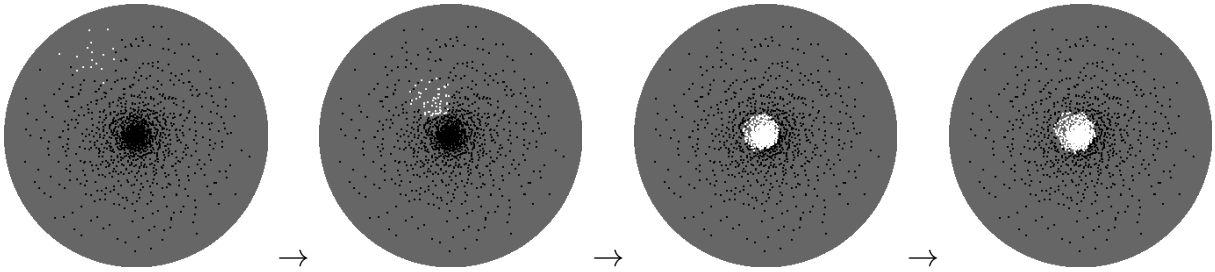
(a) Left retina (b) Right retina

Figure 4.9: Locations of the photoreceptors (black dots) on the left retina (a) and right retina (b) according to the noisy log-polar model.

include 3,600 photoreceptors situated at $\boldsymbol{d}_k$, where $1 \leq k \leq 3{,}600$, such that

$$\boldsymbol{d}_k = e^{\rho_i} \begin{bmatrix} \cos\theta_j \\ \sin\theta_j \end{bmatrix} + \begin{bmatrix} \mathcal{N}(\mu, \sigma^2) \\ \mathcal{N}(\mu, \sigma^2) \end{bmatrix}, \tag{4.2}$$

where $0 \leq \rho_i < 40$ in unit increments and $0 \leq \theta_j < 360°$ in $4°$ increments, and where the additive IID Gaussian noise $\mathcal{N}$ has mean μ and variance σ^2. Figure 4.9 illustrates the arrangement of the photoreceptors. By drawing different random numbers, the 3,600 photoreceptors can be placed in slightly different positions on each of the two hemispherical retinas. Of course, other placement patterns are possible, including more elaborate biomimetic procedural models or photoreceptor distributions empirically measured from biological eyes, all of which deviate dramatically from the uniformly-sampled Cartesian pixel images in common use in vision and graphics.



(a) t_0 (b) t_1 (c) t_2 (d) t_3

Figure 4.10: Time sequence (a)–(d) of photoreceptor responses in the left retina during a saccadic eye movement that foveates and tracks a moving white ball. At time t_0 the ball becomes visible in the periphery, at t_1 the eye movement is bringing the ball towards the fovea, and the moving ball is being fixated in the fovea at times t_2 and t_3.

4.2.3 Optic Nerve Vectors

The foveated retinal RGB "image" captured by each eye is output for further processing down the visual pathway, not as a 2D array of pixels, but as a 1D vector of length $3{,}600 \times 3 = 10{,}800$, which we call the Optic Nerve Vector (ONV). The raw sensory information encoded in this vector feeds the perceptual neural networks that directly control eye movements and whose outputs also feed the motor networks that control head motions and the reaching actions of the limbs.

4.3 Perception DNNs (1–10)

The perception subsystem includes two types of fully-connected feedforward DNNs that input the sensory information provided by the 10,800-dimensional ONV. The first type controls the eye movements, as well as the head movements via the aforementioned neck motor DNN. The second type produces arm-to-target 3D error vectors $[\Delta x, \Delta y, \Delta z]^T$ that drive the limbs via the aforementioned limb motor DNNs. Both types are described in the next two sections.

4.3.1 Foveation DNNs (1,6)

The first role of the left and right foveation DNNs is to generate changes in the gaze directions that drive saccadic eye movements to foveate visible objects of interest, thereby

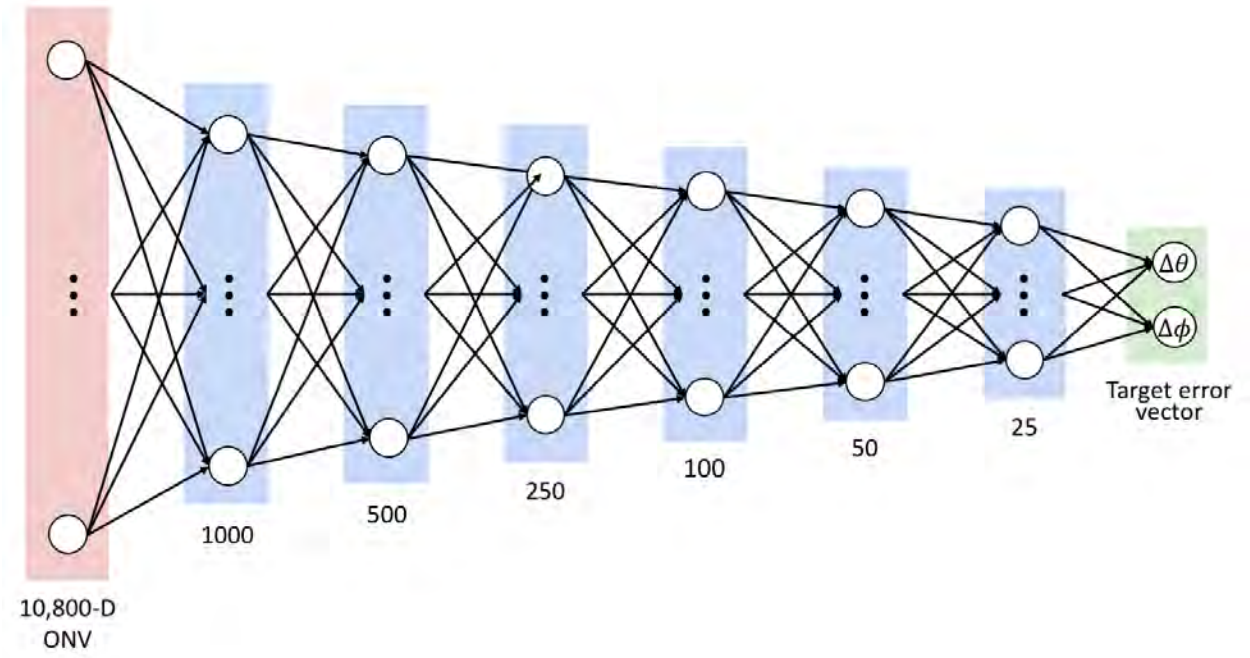


Figure 4.11: The fully-connected feedforward perception neural network architecture. The network shown is for foveation eye movements.

observing them with maximum visual acuity, as is illustrated in Figure 4.10 for a white ball in motion that enters the eye's field of view from the lower right, stimulating several peripheral photoreceptors at the upper left of the retina. The maximum speed of saccadic eye movements is 900 degrees/sec, and the eye almost instantly foveates the visual target. Fine adjustments comparable to microsaccades can be observed during fixation.

The eye movements are tightly coupled with head movements that facilitate foveation, fixation, and visual tracking. Hence, the second role of these two DNNs is to control head movement, which is accomplished by driving the aforementioned neck motor DNN (11) (Figure 4.1(f)) with the average of their outputs.

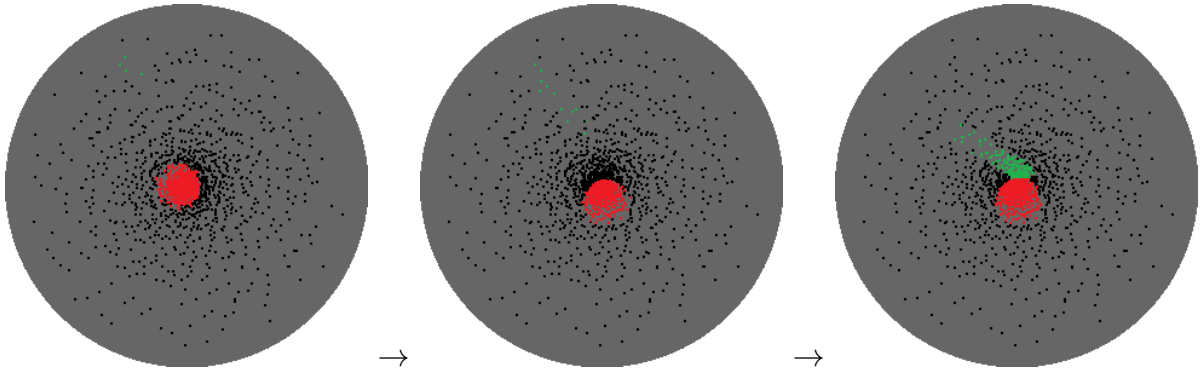
Network architecture: As shown in Figure 4.11, the input layer to this DNN comprises 10,800 units, due to the dimensionality of the ONV, the output layer has 2 units, $\Delta\theta$ and $\Delta\phi$, and there are 6 hidden layers. We conducted experiments with various DNN architectures, activation functions, and other parameters to determine a suitable architecture for our purposes. The detail is explained in Sec. 5.2. The DNN applies the rectified linear unit (ReLU)

activation function, and its initial weights are sampled from the zero-mean normal distribution with standard deviation $\sqrt{\frac{2}{fan\_in}}$, where fan_in is the number of input units in the weight tensor (He et al., 2015). We chose Adaptive Moment Estimation (Adam) (Kingma and Ba, 2014) as the stochastic optimization method, using the following parameters: $lr = 0.000001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 1e-08$, $\alpha = 0.001$, where β_1 and β_2 represent the exponential decay rate for momentum estimates taking an average and an average of squared gradients, respectively, ϵ prevents a divide-by-zero error, lr is the learning rate and α is the step size. An early stopping condition is set to avoid overfitting and mean squared error is applied as a loss function.

Training data synthesis and network training: We use our virtual human model to train the network, as follows: We presented a white sphere within the visual field. By raytracing the 3D scene, the photoreceptors in the retinas of each eye are stimulated, and the visual stimuli are presented as the RGB components of the respective ONV. Given this ONV as input, the desired output of the network is the angular differences $\Delta\theta$ and $\Delta\phi$ between the actual gaze directions of the eyes and the *known* gaze directions that would foveate the sphere. Repeatedly positioning the sphere at random locations in the visual field, we generated a large training dataset of 1M input-output pairs. The backpropagation DNN training process converged to a small error after 80 epochs, which triggered an early stopping condition (no improvement for 10 successive epochs) to avoid overfitting.

4.3.2 Limb Perception DNNs (2,3,4,5 & 7,8,9,10)

The role of the left and right limb (arm and leg) perception DNNs is to estimate the separation in 3D space between the position of the end effector (hand or foot) and the position of a visual target, thus driving the associated limb motor DNN to extend the limb to touch the target. This is illustrated in Figure 4.12 for a fixated red ball and a green arm that enters the eye's field of view from the lower right, stimulating several peripheral photoreceptors at the upper left of the retina.



(a) t_0 (b) t_1 (c) t_2

Figure 4.12: Photoreceptor responses during an arm reaching motion that deflects a moving ball. The photoreceptors are simultaneously stimulated by the fixated red ball and by the green arm entering the eye's field of view from the lower right (upper left on the retina).

Network architecture: The architecture of the limb perception DNN is identical to the foveation DNN in Figure 4.11, except for the size of the output layer, which has 3 units, Δx, Δy, and Δz, to encode the estimated discrepancy between the 3D positions of the end effector and the visual target.

Data synthesis and training: Again, we use our virtual human model to train the four limb networks, as follows: We present a red ball in the visual field and allow the trained foveation DNNs to foveate the ball. Then, we extend a limb (arm or leg) towards the ball. Again, by raytracing the 3D scene, the photoreceptors in the retinas of each eye are stimulated and the visual stimuli are presented as the RGB components of the respective ONV. Given this ONV as its input, the desired output of the network is the 3D discrepancy, Δx, Δy, and Δz, between the *known* 3D positions of the end effector and the visual target. Repeatedly placing the sphere at random positions in the visual field and randomly articulating the limb to reach for it in space, we again generated a large training dataset of 1M input-output pairs. The backpropagation DNN training process converged to a small error after 388 epochs, which triggered the early stopping condition to avoid overfitting. As expected, due to the greater complexity of this task, the training speed is significantly slower than that of the foveation DNN.

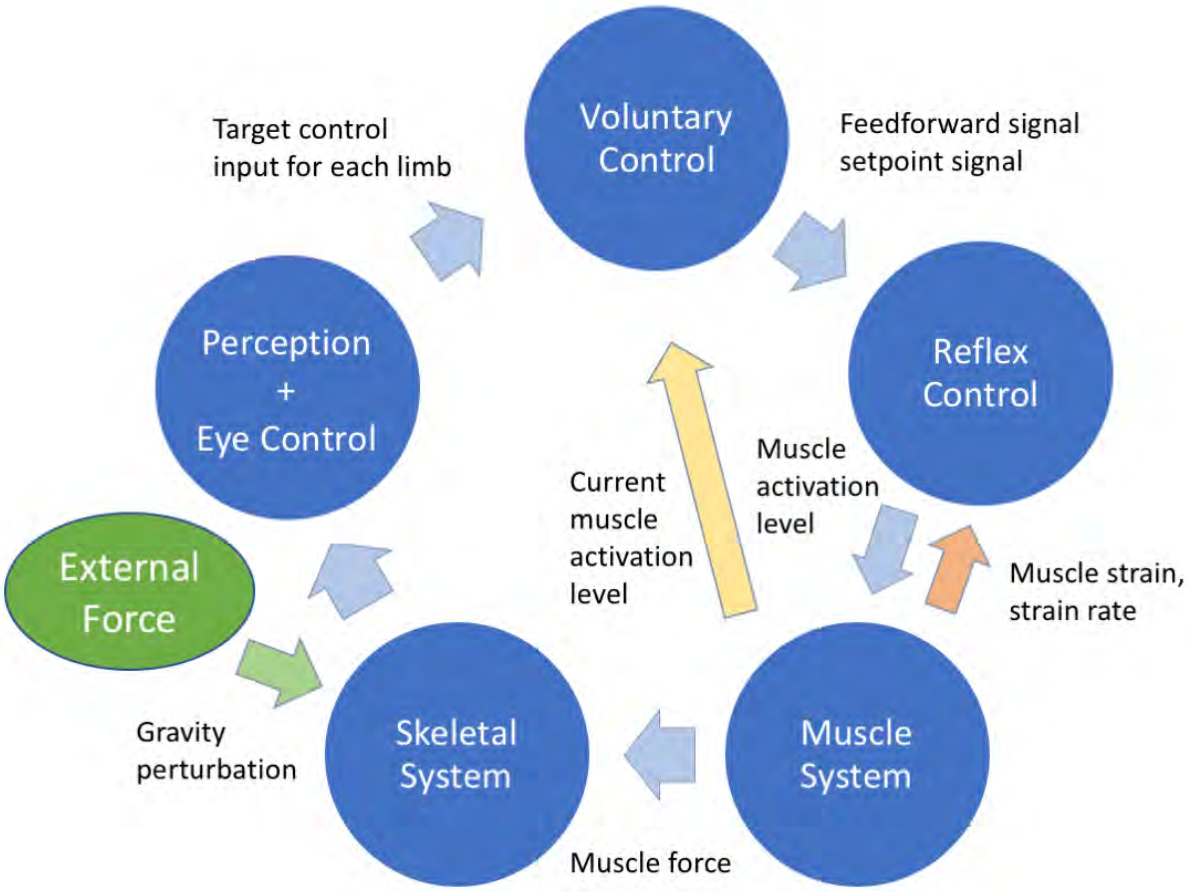


Figure 4.13: The simulation cycle. The scene is "imaged" through foveated perception and eye movements are generated to fixate a visual target. The neuromuscular motor controllers receive their inputs from the perception networks and they output muscle activation levels. The muscle reflex controllers adjust the activation signals to the muscles. Finally, the skeletal system is actuated by the generated muscle forces. This cycle repeats throughout the simulation.

4.4 The Sensorimotor Simulation Cycle

Figure 4.13 illustrates the flow of our embodied, sensorimotor simulation process. The ONVs from the eyes are processed by the perception DNNs to enable foveation and visual tracking of the existing target. Simultaneously fed by the perception DNNs, the motor DNNs effectively generate the voluntary control signals for neck and limbs. Those muscle signals are adjusted at the reflex controller. Finally, the muscles are activated and the force generated by those muscles articulate the skeletal movement. This process repeat till the simulation loop ends.

CHAPTER 5

Experiments and Results

5.1 Musculoskeletal Behavior Tests

Figure 5.1 shows a sequence of frames from a simulation of the skeletal motion of the left arm without muscles. As expected for a fully dynamic biomechanical model, the arm bones swing downward in gravity and oscillate until they come to rest.

5.1.1 Muscle Activation and Deactivation

Neck: Figure 5.2 shows a sequence of frames from the simulation of the neck with its trained neuromuscular controller in which the neck muscles are deactivated and reactivated, demonstrating fully dynamic control of the neck. When the muscles are deactivated, by setting all the muscle activations to 0.0, the head drops backward. As soon as we reactivate the neuromuscular controller, the neck lifts the head back up and regains its ability to balance the head and make controlled head movements.

Arm: Figure 5.3 shows a sequence of frames from a simulation of the left arm with its trained neuromuscular controller in which the arm muscles are deactivated and reactivated, demonstrating fully dynamic control of the arm.

Leg: Figure 5.4 shows a sequence of frames from a simulation of the left leg with its trained neuromuscular controller, in which the leg muscles are deactivated and reactivated, demonstrating fully dynamic control of the leg.

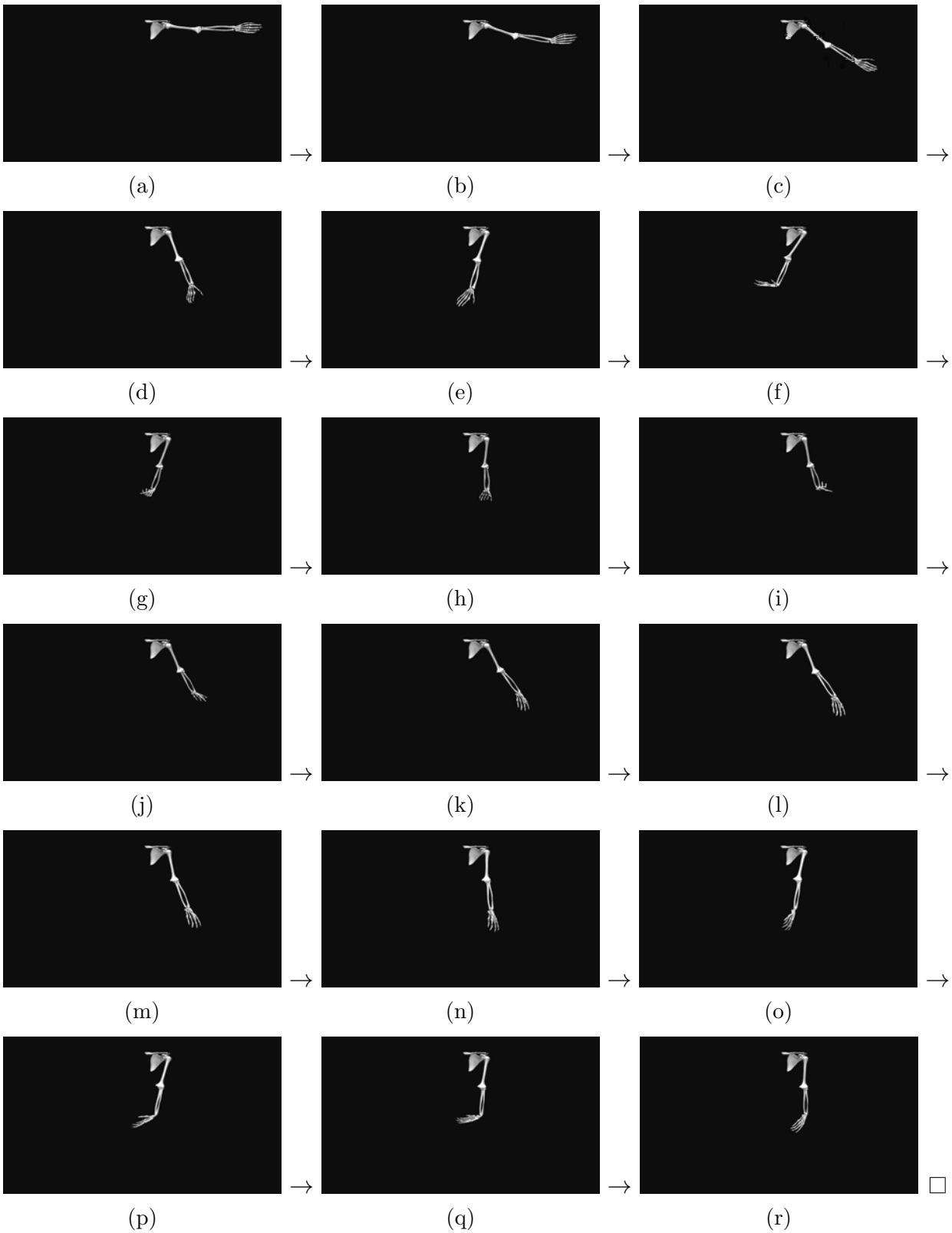


Figure 5.1: Sequence of frames from the simulation of the left arm without muscles in gravity.

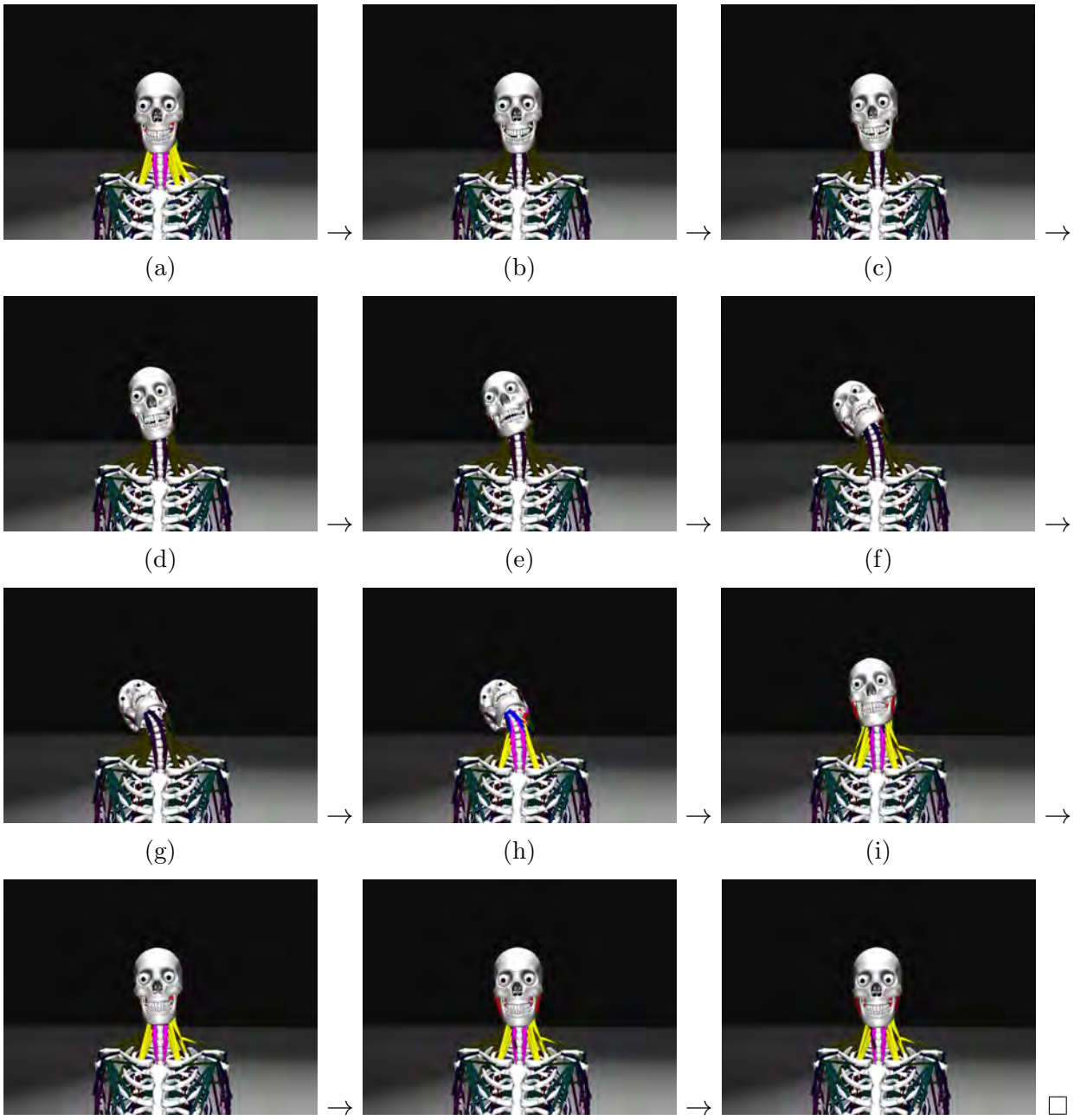


Figure 5.2: Sequence of frames from a simulation of the neck with its trained neuromuscular controller, in which the neck muscles are deactivated and reactivated.

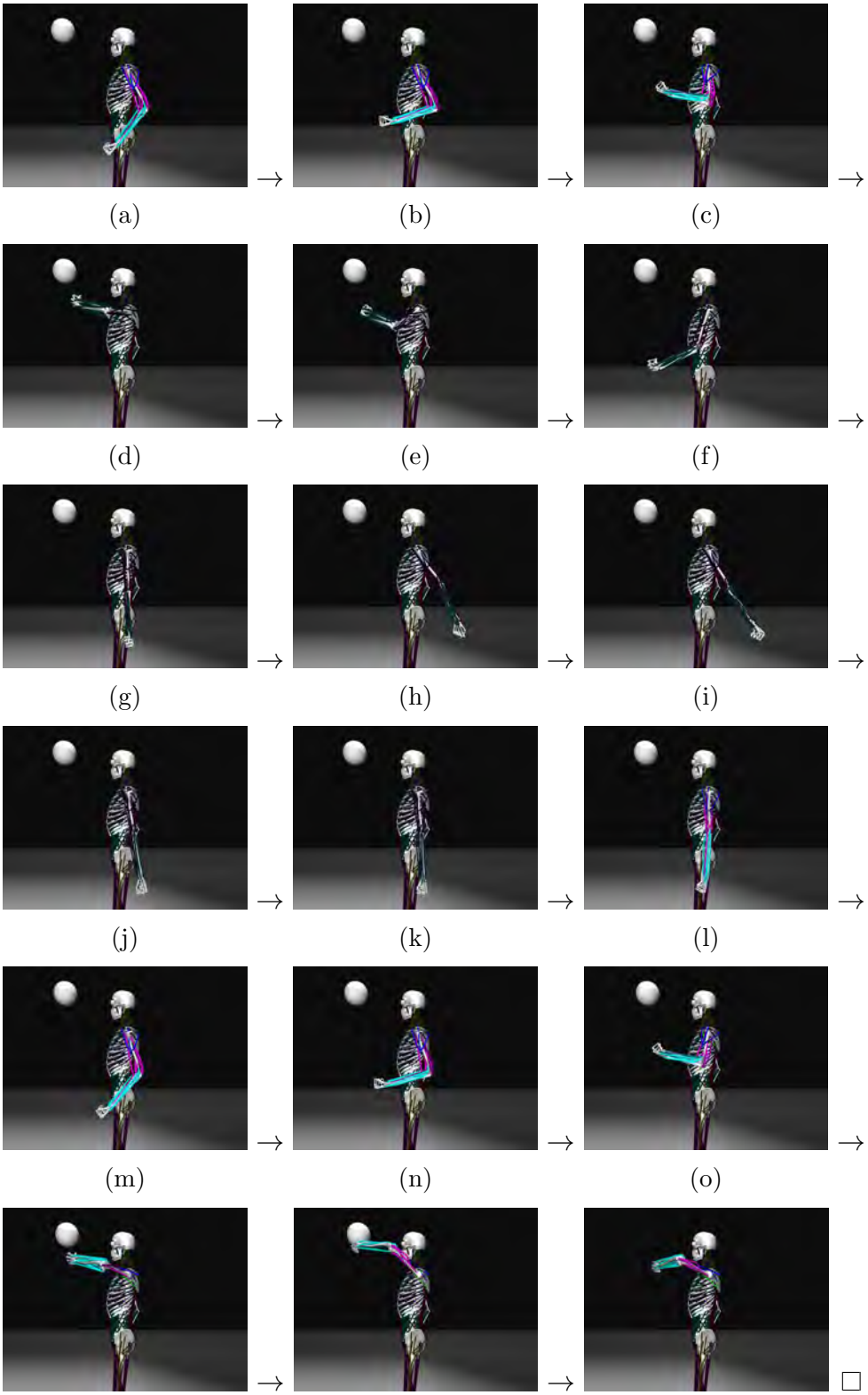


Figure 5.3: Sequence of frames from the simulation of the left arm, in which the arm muscles are deactivated and reactivated.

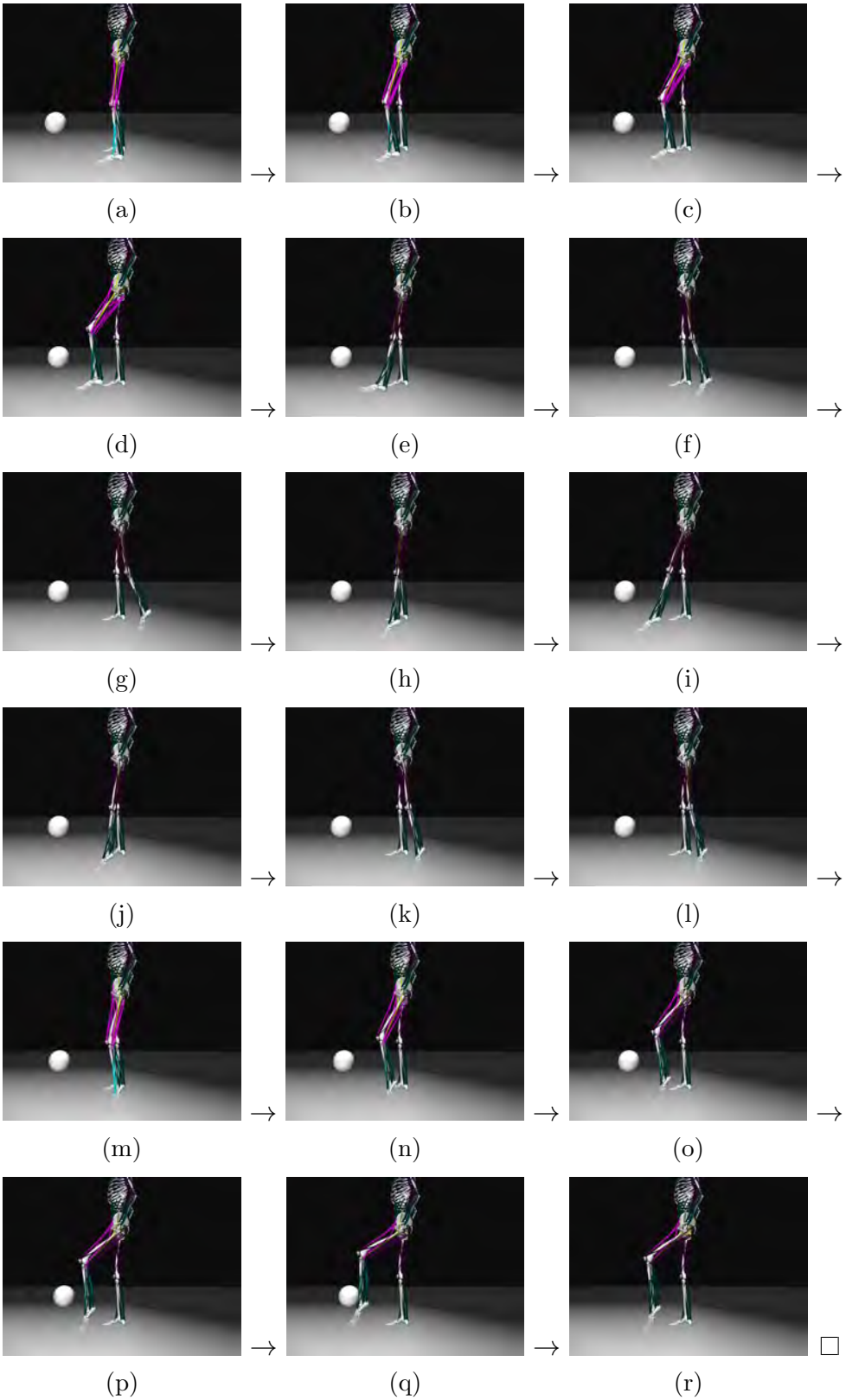


Figure 5.4: Sequence of frames from the simulation of the left leg, in which the leg muscles are deactivated and reactivated.

5.2 Experiments With Various DNN Architectures and Techniques

We conducted a number of experiments to compare the performance of various perception controller architectures and training techniques. The networks were implemented and trained using the Theano library (Bergstra et al., 2010) running on an NVIDIA Titan X GPU installed in a Ubuntu 16.04 system. The total number of weights in the DNNs ranges from 11.5–24.2 million. Unless stated otherwise, all networks employ the same weight initialization (He Normal), the same rectified linear units (ReLU), and the same stochastic optimization method (Adam).

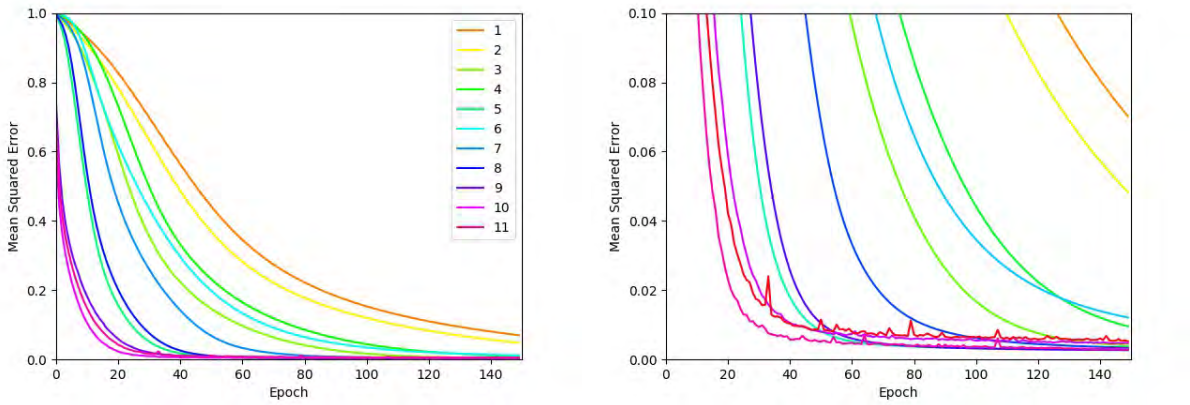
For each experiment, we randomly selected a 0.1M training dataset from an original 1M input-output-pair synthesized dataset. The smaller datasets that we employed reduced the computational load, resulting in more rapid experimentation.

5.2.1 Foveation DNN

Network shape: We conducted training experiments with the Foveation controller using 11 different network architectures with different shapes. The network shapes are as follows, where the leftmost number, rightmost number, and intermediate numbers, indicate the number of units in the input layer, output layer, and hidden layers, respectively:

1. 10,800 | 100 | 200 | 2
2. 10,800 | 200 | 100 | 2
3. 10,800 | 300 | 200 | 100 | 2
4. 10,800 | 100 | 200 | 300 | 2
5. 10,800 | 500 | 250 | 100 | 50 | 25 | 2
6. 10,800 | 25 | 50 | 100 | 250 | 500 | 2
7. 10,800 | 100 | 250 | 500 | 250 | 100 | 2
8. 10,800 | 500 | 250 | 100 | 250 | 500 | 2
9. 10,800 | 10,000 | 10,000 | 2
10. 10,800 | 20,000 | 10,000 | 2
11. 10,800 | 10,000 | 20,000 | 2

This list encompasses deep and shallow as well as narrow and wide network architectures with straight, regular, and inverse triangle and diamond shapes. Figure 5.5 plots the mean



(a) (b) Closer view of plot in (a)

Figure 5.5: Training progress of the foveation DNN experiment with various architecture shapes.

squared error as a function of the number of epochs during the training process on the validation datasets for each of the above listed network architectures, by number. All the training processes converged; however, the convergence speed and stability varied for each architecture.

The shallow-narrow neural networks (1,2,3,4) required the most epochs to converge. The progress of learning was stable but slow. The shallow-wide neural networks (9,10,11) where more efficient but less stable, in the sense that they required fewer epochs, but the error did not decrease monotonically, as we see in Figure 5.5(b). The deep architectures (5,6,7,8) converged quickly and stably. The deep tapered network (5) converged with the best speed and stability.

The average time spent for each epoch is shown in Figure 5.6. As expected, it increases with the number of weights, wide networks requiring 10 to 20 times more computation time per epoch than narrow ones; however, the use of a trained wide network for online control allows parallelization since the outputs of units in the same layer can be computed independently, although the layers must be computed sequentially, rendering deeper networks less parallelizable. Overall, the training time for the more elaborate architectures was similar to that for the simpler ones since the former required fewer epochs, albeit more time per epoch, whereas the latter required more epochs, but less time per epoch.

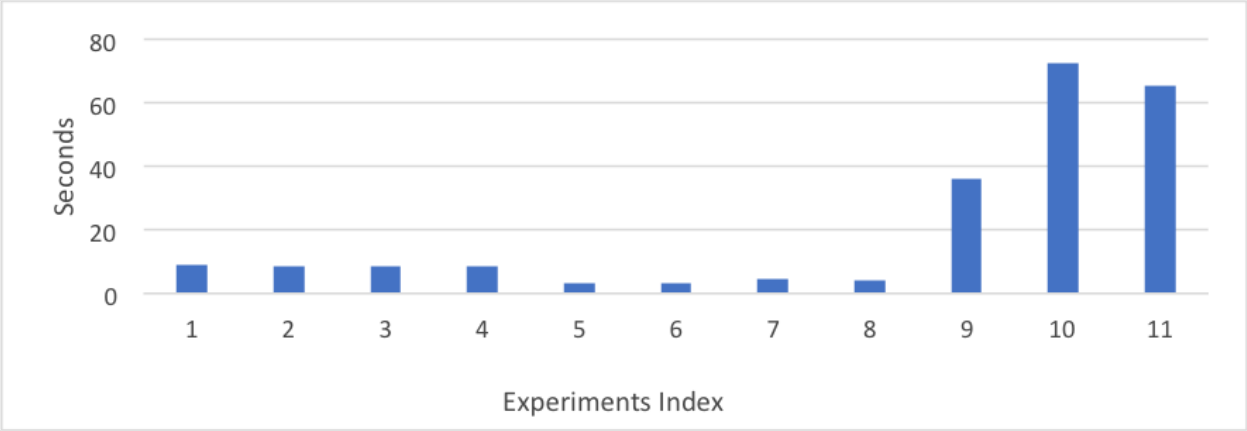


Figure 5.6: Average time per epoch for foveation DNN experiment with various neural network architecture shape.

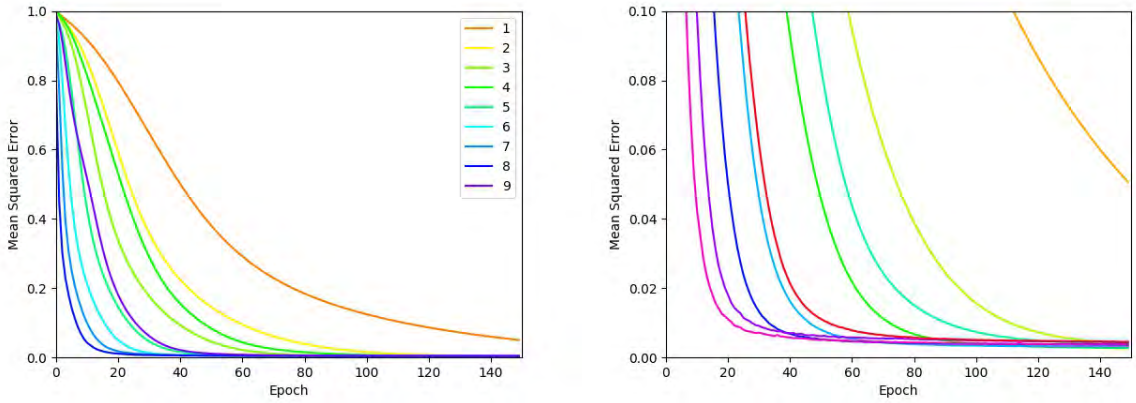
Network depth: The next set of experiments evaluated networks all with the same architecture but with different depths, as follows:

1. 10,800 | 200 | 100 | 2
2. 10,800 | 300 | 200 | 100 | 2
3. 10,800 | 400 | 300 | 200 | 100 | 2
4. 10,800 | 500 | 250 | 100 | 50 | 25 | 2
5. 10,800 | 1,000 | 500 | 250 | 100 | 50 | 25 | 2
6. 10,800 | 1,800 | 1,000 | 500 | 250 | 100 | 50 | 25 | 2
7. 10,800 | 3,000 | 1,800 | 1,000 | 500 | 250 | 100 | 50 | 25 | 2
8. 10,800 | 4,500 | 3,000 | 1,800 | 1,000 | 500 | 250 | 100 | 50 | 25 | 2
9. 10,800 | 680 | 550 | 430 | 320 | 230 | 150 | 80 | 40 | 20 | 10 | 2

Figure 5.7 plots the mean squared error as a function of the number of epochs during the training process. Interestingly, fewer epochs were necessary as the network depth increased. For each experiment, we included an extra larger hidden layer immediately after the input layer in order to maintain the triangular network shape as we increase the number of hidden layers.

In Experiment 9, we decreased the number of units in each layer by an order of magnitude, and we observed that the number of epochs required increased significantly.

These results show that both the number of layers and the number of units in each layers—that is, the total number of weights—contribute to the regression abilities of the



(a) (b) Closer view of plot in (a)

Figure 5.7: Training progress of the foveation DNN with different network depths.

perception network.

The average computational time spent on each epoch is shown in Figure 5.8. This was less than 3.7 seconds for each epoch until the number of hidden layers increases to 7. Networks with more than 7 hidden layers required significantly more time for each epoch. This is due to the complexity of the architecture and the greater number of weights to train.

For Experiment 9, with the tenfold decrease in the number of units, the training time for each epoch decreases to around 4 seconds again.

Hence, we decided to use a 6-hidden-layer network architecture for a good compromise between accuracy, efficiency, and stability.

Network width: The next set of experiments evaluated networks with the same depth but with different widths, as follows:

1. 10,800 | 100 | 100 | 2
2. 10,800 | 1,000 | 1,000 | 2
3. 10,800 | 10,000 | 10,000 | 2
4. 10,800 | 200 | 100 | 2
5. 10,800 | 2,000 | 1,000 | 2
6. 10,800 | 20,000 | 10,000 | 2
7. 10,800 | 100 | 200 | 2

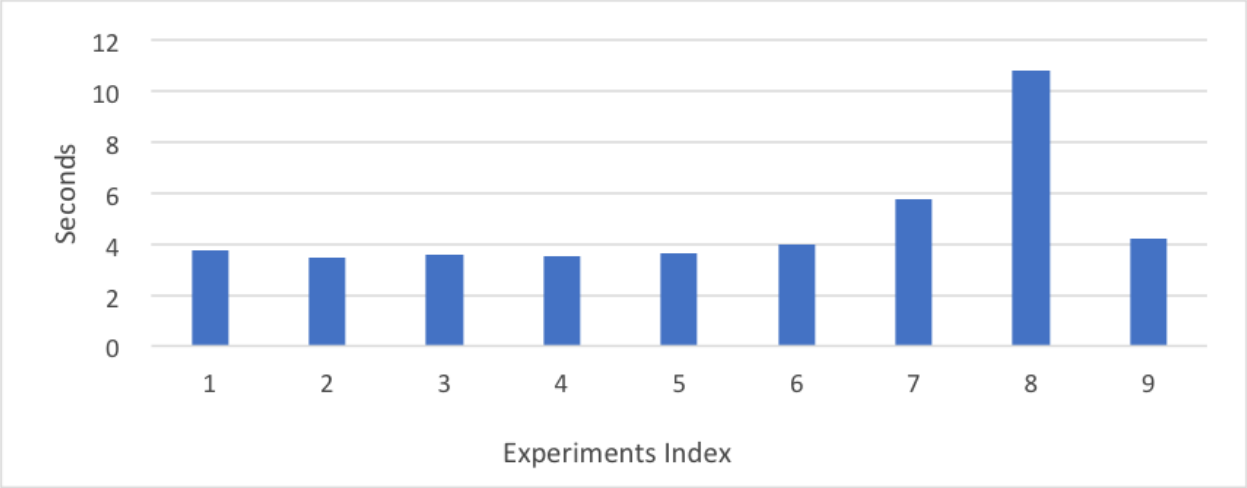


Figure 5.8: Average time per epoch for the foveation DNN with different network depths.

8. 10,800 | 1,000 | 2,000 | 2
9. 10,800 | 10,000 | 20,000 | 2

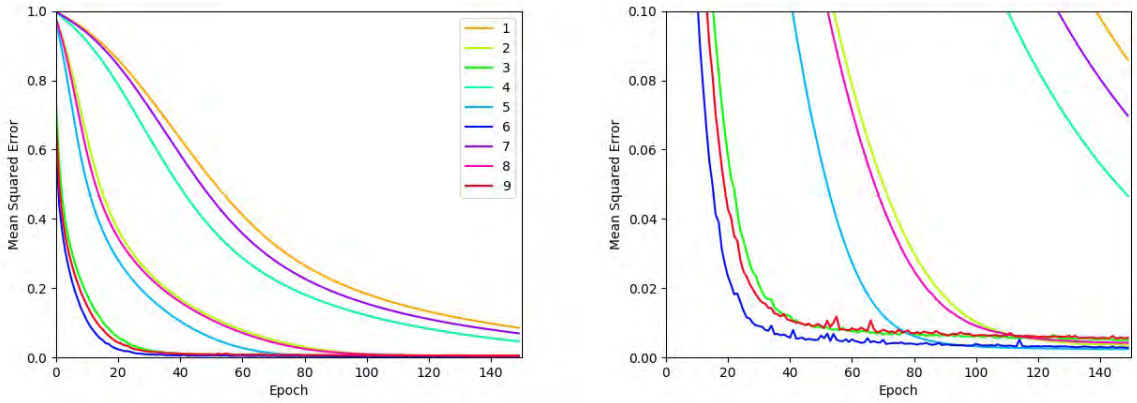
Figure 5.9 plots the network training convergences. It can be seen that the wider the architectures are, the fewer epochs are required. This is expected because the network's representation capability should increase as the number of nodes in each layer increases.

The results show that the shape of wide neural networks (straight, tapered, and inverse tapered) has only a minor influence. Although most of these networks converged well, the training progress was not as efficient and stable as it was for the deeper architectures.

Learning rate: We conducted experimental trainings with the foveation DNN using 6 different architectures to determine the best learning rate with Adam optimization.

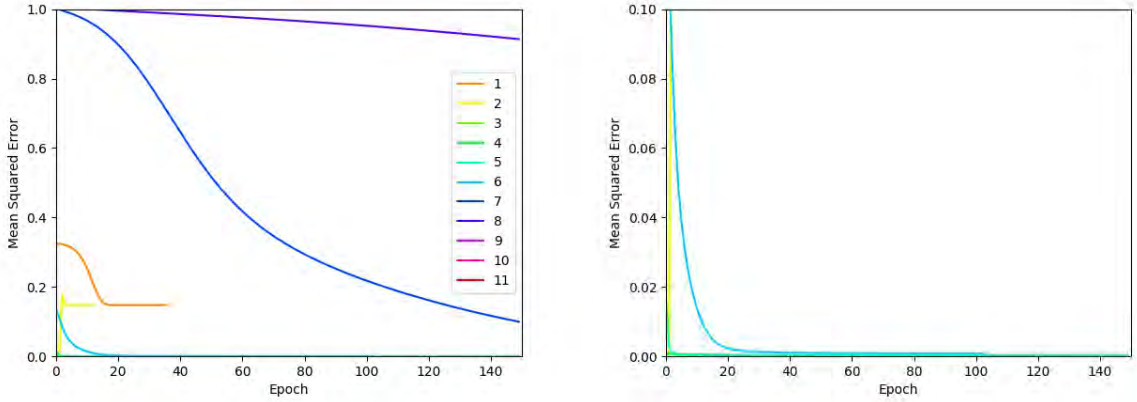
The number of hidden layers was fixed to 6. Only the learning rates for a stochastic optimization differ. We choose 11 different learning rates decreasing by factors of 10, as follows:

1. 1.0×10^{-1}
2. 1.0×10^{-2}
3. 1.0×10^{-3}
4. 1.0×10^{-4}
5. 1.0×10^{-5}



(a) (b) Closer view of Plot (a)

Figure 5.9: Training progress of the foveation DNN with different widths.

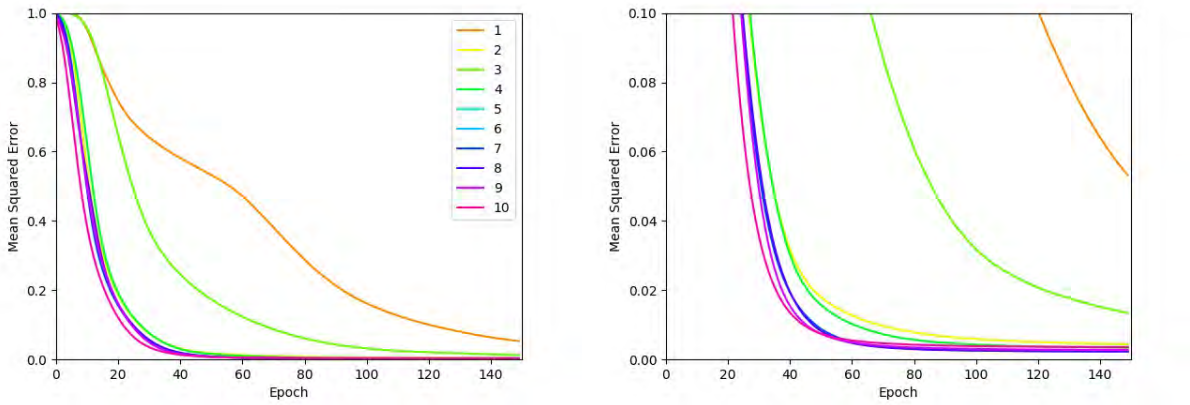


(a) (b) Closer view of plot in (a)

Figure 5.10: Training progress of the foveation DNN with different learning rates.

6. 1.0×10^{-6}
7. 1.0×10^{-7}
8. 1.0×10^{-8}
9. 1.0×10^{-9}
10. 1.0×10^{-10}
11. 1.0×10^{-11}

Figure 5.10 plots the network training convergences. It can be seen that the training is done most efficiently and stably when the learning rate is 1.0×10^{-6}. When the learning rate is less than 1.0×10^{-8}, the trainings do not converge. When the learning rate is greater than



(a) (b) Closer view of plot in (a)

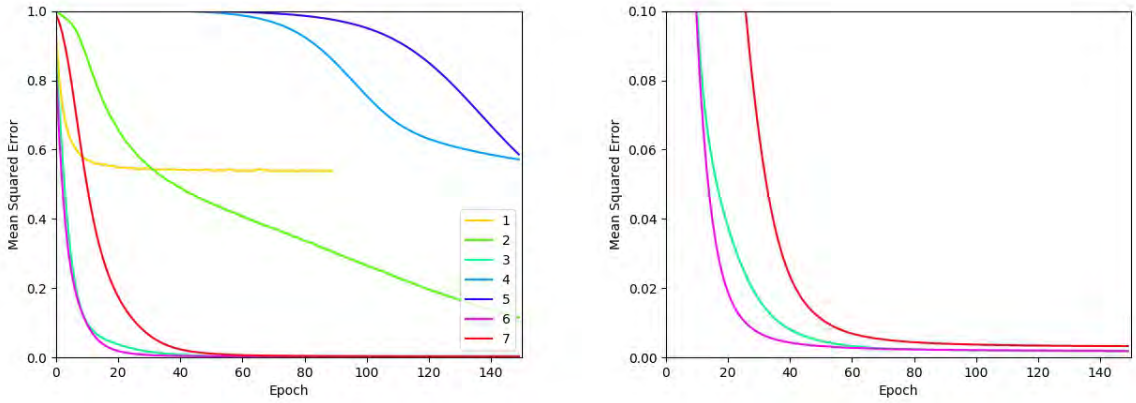
Figure 5.11: Training progress of the foveation DNN with different weight initialization methods.

1.0×10^{-4}, the trainings converged very quickly; however, they appear unstable and may not work robustly for other datasets. Hence, 1.0×10^{-6} was chosen as the Adam learning rate in our offline training process as it attains a good balance between stability and efficiency.

Weight initialization: The next set of experiments evaluated the following different weight initialization methods using networks with 6 hidden layers:

1. Uniform
2. LeCun Uniform
3. Normal
4. Orthogonal
5. Zeros
6. Ones
7. He Uniform
8. He Normal
9. Glorot Uniform
10. Glorot Normal

Figure 5.11 plots the network training convergences. Although most of the training processes worked well, those with Uniform and Normal weight initialization were slower to converge. The Zeros initialization method does not converge until near the end.



(a) Vertical axis ranges from 0.0 to 1.0 (b) Closer view of the plot in (a)

Figure 5.12: Training progress of the foveation DNN with different activation functions.

We decided to employ He Normal initialization because it yields fast and stable convergence and, indeed, it is one of the most popular weight initialization methods.

Activation function: The next set of experiments evaluated the following different activation functions using networks with 6 hidden layers, each containing the same number of units:

1. Linear
2. Soft Plus
3. Soft Sign
4. Hard Sigmoid
5. Sigmoid
6. Tanh
7. ReLU

Figure 5.12 plots the training convergences. Networks using ReLU, Soft Sign, and Tanh activation functions show fast convergence. Those using Linear, Sigmoid, and Hard Sigmoid do not converge. The network using Soft Plus converged, but slowly. Though Tanh and Soft Sign required fewer epochs in this experiment, we decided to use ReLU activation for its known advantages of sparsity of the representation, absence of vanishing gradients, greater biological plausibility, and computational simplicity, which leads to superior stability and robustness.

5.2.2 Left Arm Motor DNN

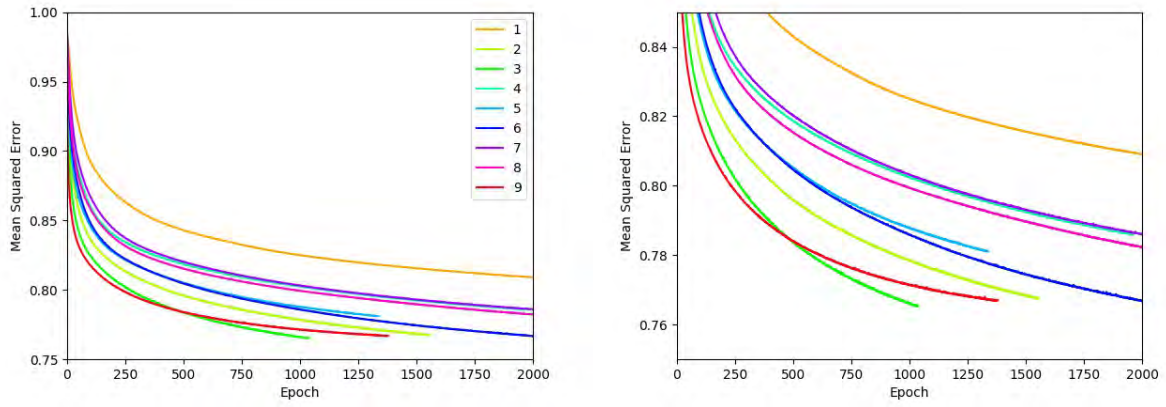
Network shape: We conducted training experiments with the left arm motor controller using 9 different network architectures with various shapes. The network shapes are as follows, where the leftmost number, rightmost number, and intermediate numbers, indicate the number of units in the input layer, output layer, and hidden layers, respectively:

1. 32 | 100 | 100 | 100 | 100 | 100 | 29
2. 32 | 300 | 300 | 300 | 300 | 300 | 29
3. 32 | 500 | 500 | 500 | 500 | 500 | 29
4. 32 | 300 | 250 | 200 | 150 | 100 | 29
5. 32 | 500 | 400 | 300 | 200 | 100 | 29
6. 32 | 100 | 200 | 300 | 400 | 500 | 29
7. 32 | 100 | 200 | 300 | 200 | 100 | 29
8. 32 | 300 | 200 | 100 | 200 | 300 | 29
9. 32 | 3000 | 3000 | 29

This list encompasses wide-shallow and deep network architectures with rectangular, triangular, and diamond shapes. Figure 5.13 plots the mean squared error as a function of the number of epochs during the training process on the validation datasets for each of the above listed network architectures, by number. All the training processes converged; however, the convergence speed and stability varied for each architecture.

The rectangular shaped neural networks (1,2,3) required fewer epochs to converge as the number of units in each hidden layer increased. The network with 100 units in each hidden layer did not converge well such that it was useful as a motor controller. The ones with 300 and 500 units in each hidden layer converged very well, especially the one with 500 units in each layer, which showed one of the best performances. The triangular and the diamond shaped architectures (4,5,6,7,8) converged in average speed. The learning progress was stable but slow. The wide-shallow neural network (9) was efficient in the sense that it required fewer epochs.

The average time spent for each epoch is shown in Figure 5.14. As expected, the required time increases as the number of weights increased. The wide-shallow network requiring 6 to 8 times more computation time per epoch than deep-narrow ones.



(a) (b) Closer view of plot in (a)

Figure 5.13: Training progress of left arm motor DNN experiment with various neural network architecture shapes.

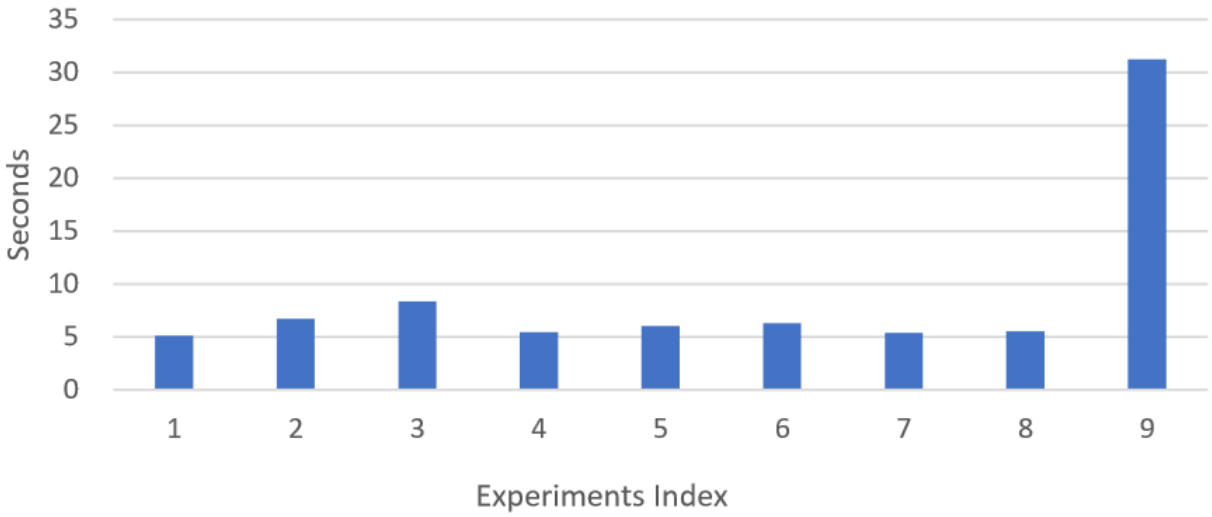


Figure 5.14: Average time per epoch for left arm DNN experiment with various neural network architecture shapes.

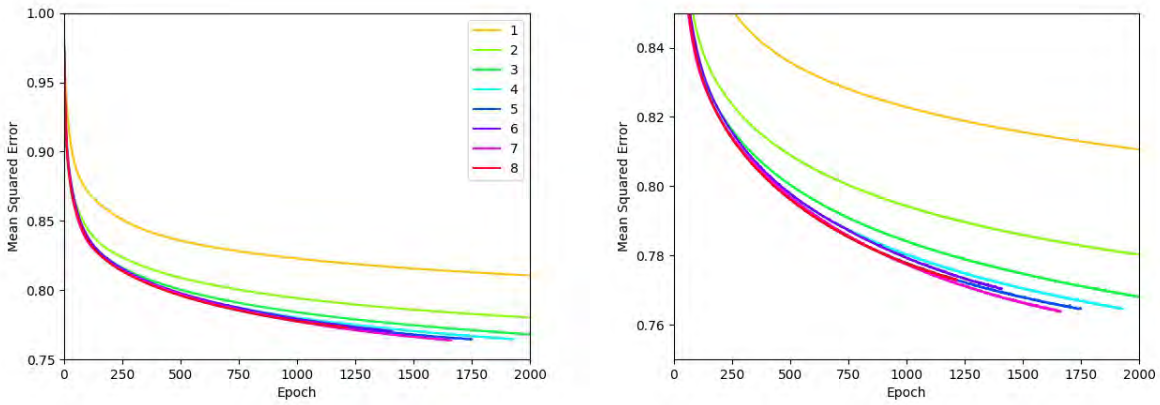
Network depth: The next experiment evaluated networks with the same architecture, but with different depths, as follows:

1. 32 | 1800 | 29
2. 32 | 900 | 900 | 29
3. 32 | 600 | 600 | 600 | 29
4. 32 | 450 | 450 | 450 | 450 | 29
5. 32 | 360 | 360 | 360 | 360 | 360 | 29
6. 32 | 300 | 300 | 300 | 300 | 300 | 300 | 29
7. 32 | 257 | 257 | 257 | 257 | 257 | 257 | 257 | 29
8. 32 | 225 | 225 | 225 | 225 | 225 | 225 | 225 | 225 | 29

To evaluate the effect of network depth independently of the total number of hidden units in the network, each of the above networks includes approximately 1,800 hidden units. Figure 5.15 plots the mean squared error as a function of the number of epochs of the training process run on the validation dataset. Fewer epochs were required as the network depth increased until the number of hidden layers reaches 6. However, the required number of epochs increased with 7 hidden layers. It then becomes less again with 8 hidden layers.

The average computational time spent on each epoch is shown in Figure 5.16. Interestingly, the required time for each epoch increases until the number of the hidden layers reaches 6, but the required times with more than 5 hidden layers are almost the same. The network with 7 hidden layers actually spent slightly less time than the one with 6 hidden layers, and the time increased for the network with 8 hidden layers.

Thus, this experiment shows that the number of hidden layers affects the regression abilities of the motor DNN as well the computational time for each training epoch, albeit much less substantially so beyond 6 hidden layers.



(a) (b) Closer view of plot in (a)

Figure 5.15: Experiments with different neural network depths.

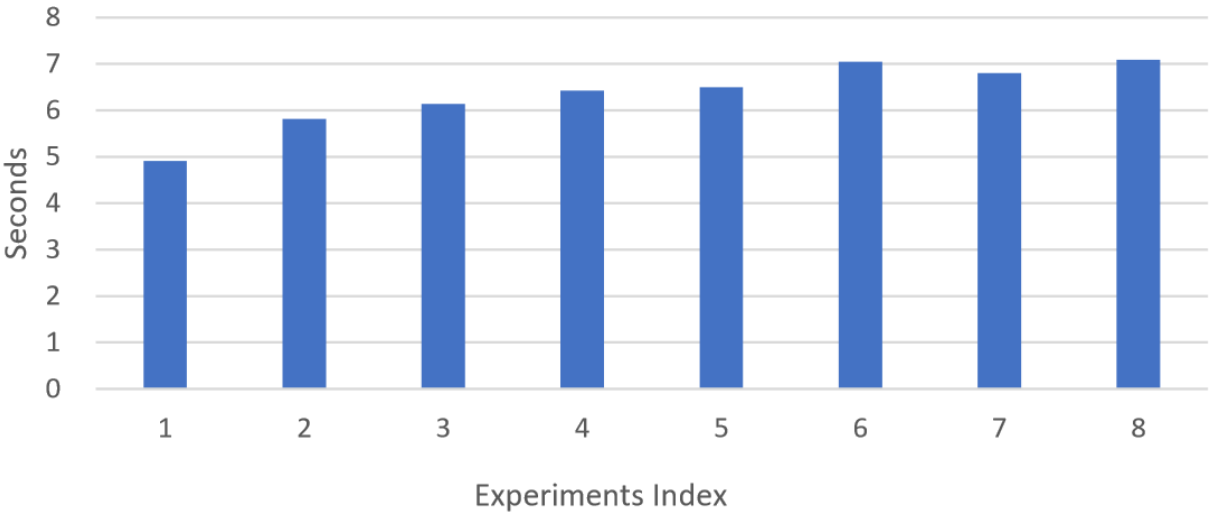


Figure 5.16: Average time per epoch for left arm motor DNNs with various depths.

5.3 Animation Results

5.3.1 Raytracing Animation

Figure 5.17 shows a sequence of frames from an animation of our biomechanically-simulated virtual human sitting on a stool and actively executing arms reaching motion. The blue lines show the rays from each photoreceptor which sample the scene and returns stimulus back to the photoreceptors. First, the ball is perceived by the eyes, then, processed by the perception DNNs, foveated and tracked through eye movements in conjunction with muscle-actuated

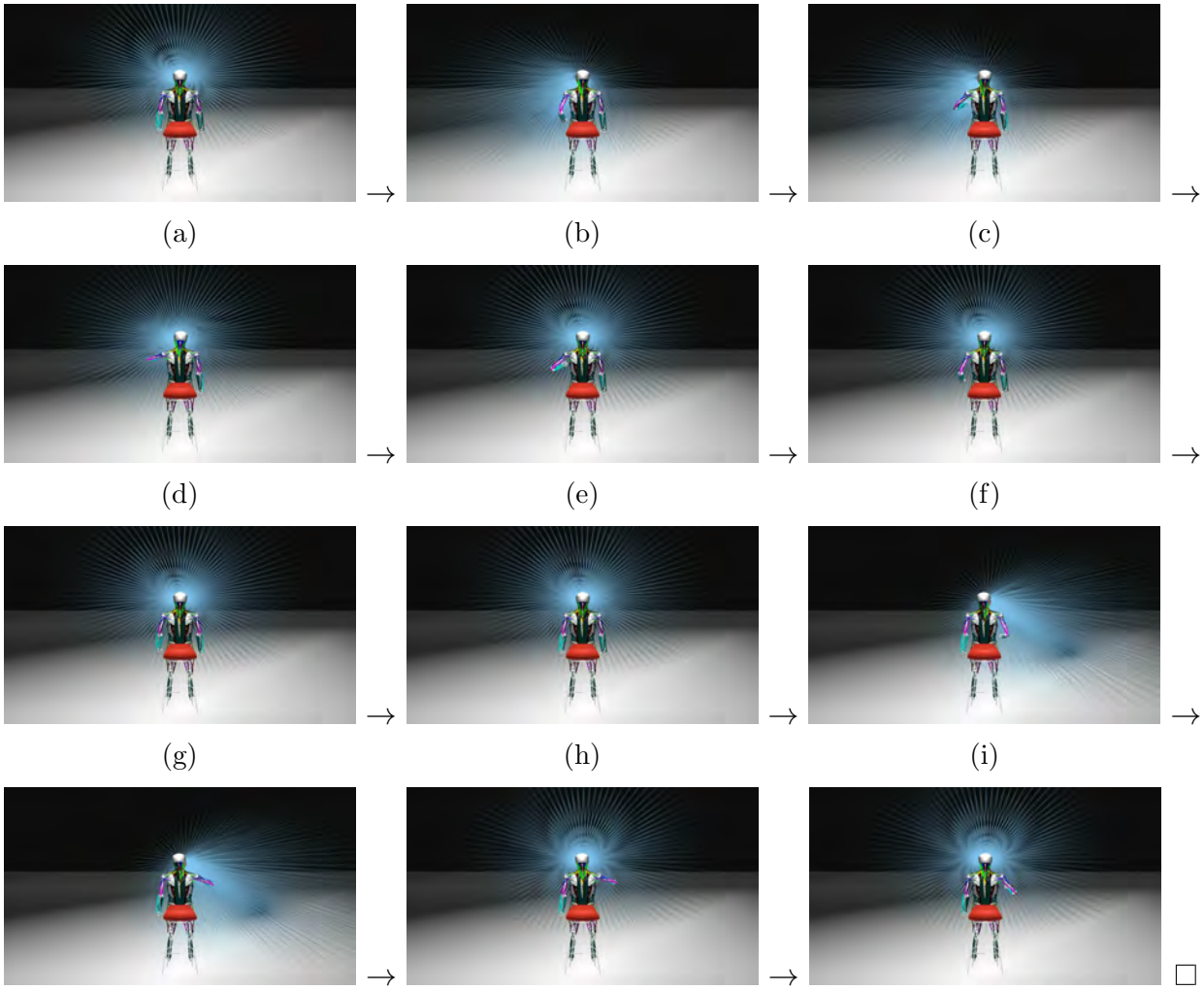


Figure 5.17: Frames from an animation of our biomechanically-simulated virtual human sitting on a stool and actively executing arms reaching motion. The blue lines show the rays from the photoreceptors which sample the scene. The ball is perceived by the eyes, processed by the perception DNNs, foveated and tracked through eye movements in conjunction with muscle-actuated head movements controlled by the neck-head motor DNN, and the reactive reaching motion is muscle-actuated and controlled by the arm motor DNN.

head movements controlled by the neck-head motor DNN, and the reactive reaching motion is muscle-actuated and controlled by the arm motor DNN.

5.3.2 Punching and Kicking an Incoming Ball

Figure 5.18 shows a sequence of frames from an animation demonstrating the active sensorimotor system. A cannon shoots a ball towards the virtual human, which actively perceives the ball on its foveated retinas. The ONVs from the eyes are processed by the perception DNNs to enable foveation and visual pursuit of the incoming ball. Simultaneously fed by the perception DNNs, the motor DNNs effectively control arms and legs to punch and kick the incoming ball, thus deflecting it out of the way. The virtual human successfully controls itself to carry out this complex online and dynamic task. Figure 4.13 illustrates the flow of this embodied, active vision simulation process. Aside from the high level objective of deflecting the incoming ball, no scripts were specified to generate the animation.

Figure 5.19 shows a similar frame sequence from a high-quality-rendered animation of a skinned version of our virtual human model deflecting balls shot by the cannon.

5.3.3 Punching a Ball Bouncing On Multiple Sloped Planes

Figure 5.20 shows a sequence of frames from an animation of our virtual human actively executing reaching motions to intercept a ball bouncing on multiple sloped planes. The eyes, in conjunction with the head, visually pursue the ball target and the right arm also naturally tracks the ball until it comes within a reachable distance. The neck, eye, and arm movements are automatically prompted by the motion of the ball. The eyes make fast saccadic movements to quickly fixate on the visual target, while the head moves cooperatively with the eyes, but more sluggishly because of its greater mass. In the early frames, the ball is too far away to reach with the arm; however, our virtual human automatically generates preparatory arm movements towards the approaching ball.

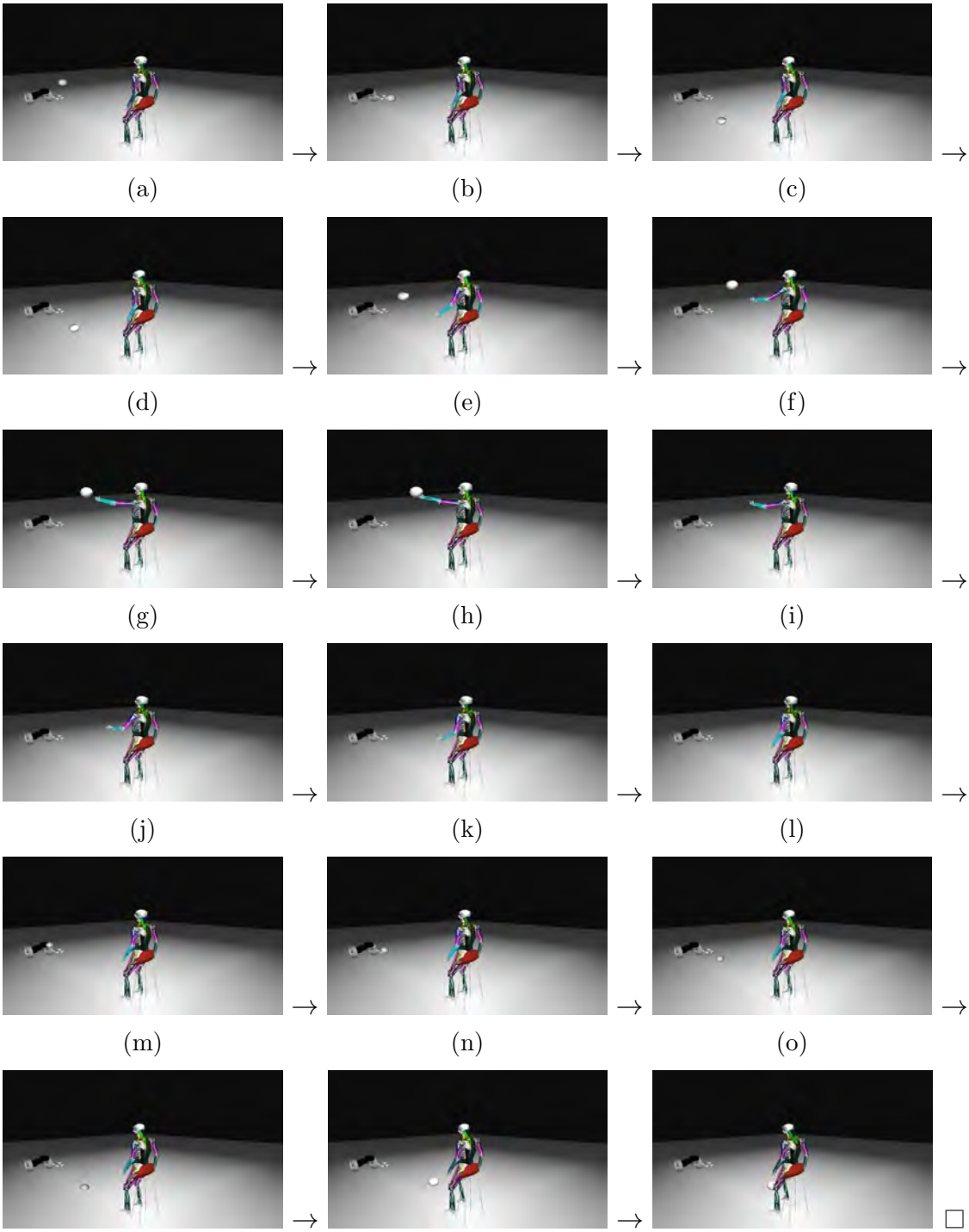


Figure 5.18: Frames from an animation of our biomechanically-simulated virtual human sitting on a stool and actively executing arm and leg reaching motions to deflect a ball shot by a cannon. The ball is perceived by the eyes, processed by the perception DNNs, foveated and smoothly pursued through eye movements in conjunction with muscle-actuated head movements controlled by the neck-head motor DNN, and the reactive reaching motion is muscle-actuated and controlled by the arm or leg motor DNNs. The red lines show the eye gaze directions.

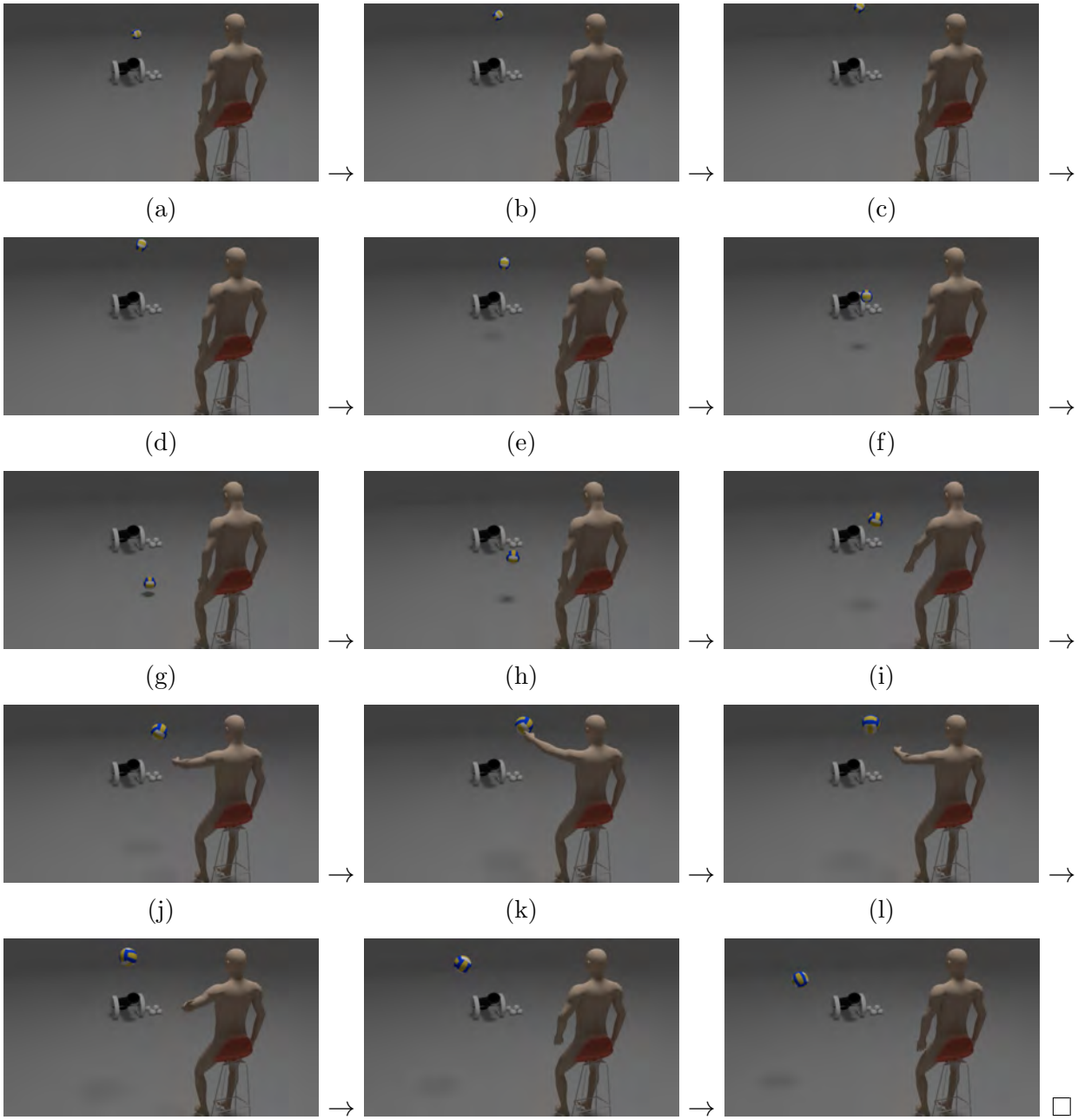


Figure 5.19: Frames from an animation of our biomechanically-simulated virtual human, rendered with a skin surface, sitting on a stool and actively executing a left-arm reaching motion that contacts a bouncing ball shot by a cannon. The ball is perceived by the eyes, processed by the perception DNNs, foveated and smoothly pursued through eye movements in conjunction with muscle-actuated head movements controlled by the neck-head motor DNN, and the reactive reaching motion is muscle-actuated and controlled by the left arm motor DNN.

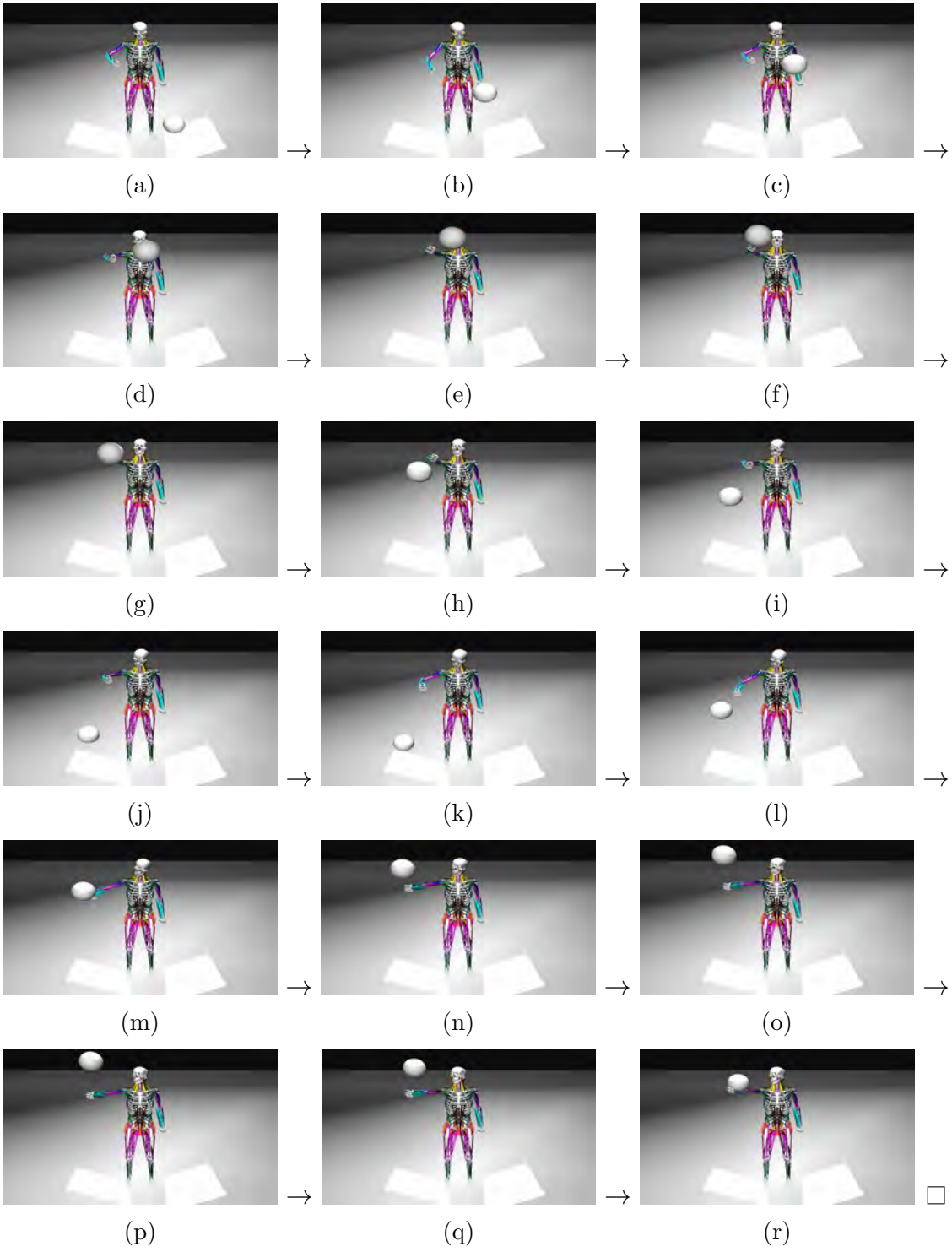


Figure 5.20: Frames from an animation of our biomechanically-simulated virtual human sitting on a stool and actively executing right-arm reaching motions to make contact with a ball bouncing on multiple sloped planes. The ball is perceived by the eyes, processed by the perception DNNs, foveated and smoothly pursued through eye movements in conjunction with muscle-actuated head movements controlled by the neck-head motor DNN, and the active reaching motion is muscle-actuated and controlled by the right arm motor DNN.

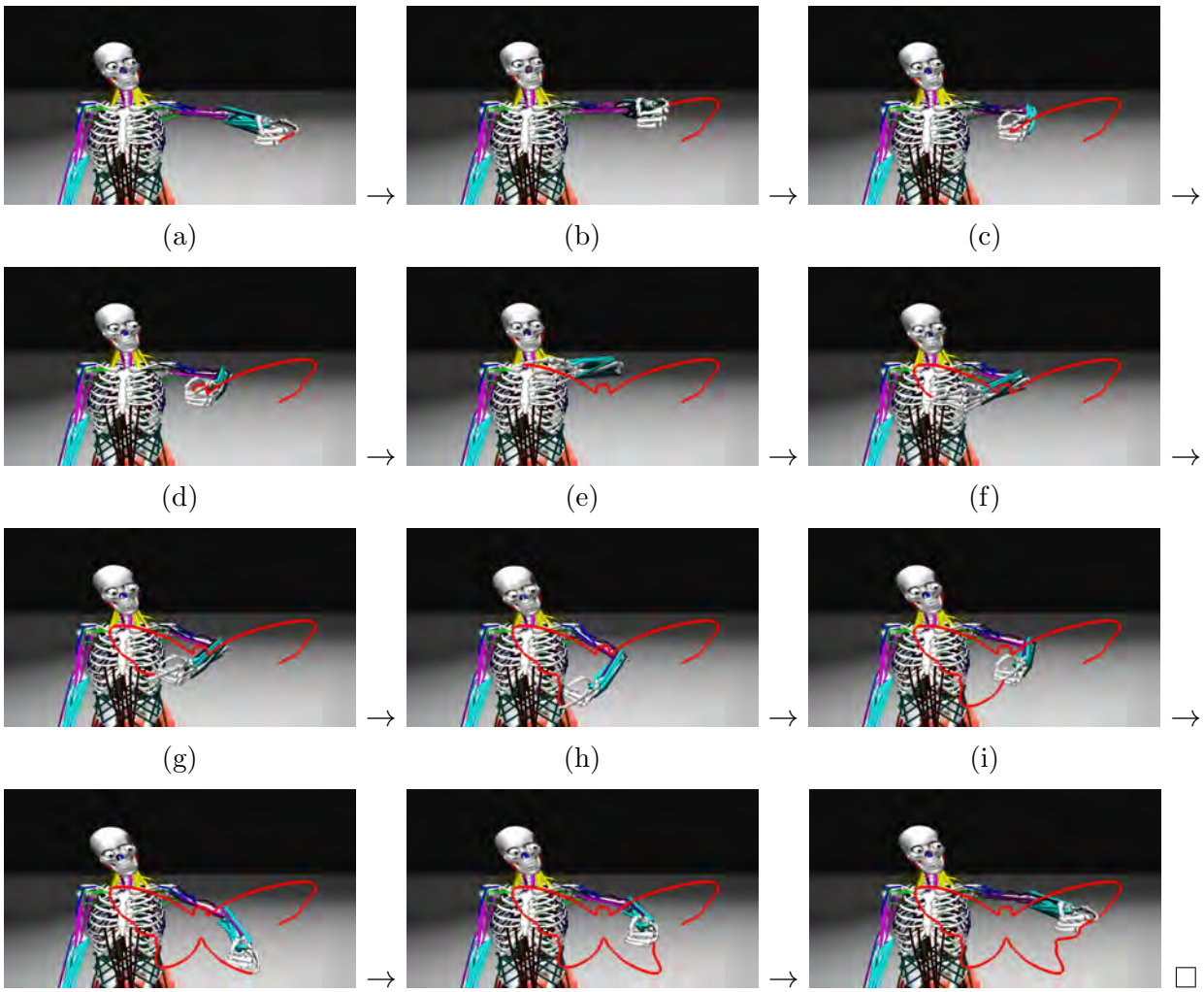


Figure 5.21: Frames from an animation of our biomechanically-simulated virtual human sitting on a stool and actively drawing a butterfly with its left arm and finger. Muscle-actuated head movements are controlled by the neck-head motor DNN, and the muscle-actuated arm reaching and drawing movements are controlled by the left arm motor DNN.

5.3.4 Drawing a Shape

Figure 5.21 shows a sequence of frames from an animation of our biomechanical human musculoskeletal model drawing the shape of a butterfly with its finger. The butterfly shape was provided to the human model as a predefined trajectory in space. Entirely through the control of its arm muscles, our virtual human is able to follow the specified trajectory with its finger.

5.4 Recognition Experiments With the Biomimetic Vision System

Certain computer vision algorithms are immediately compatible with the ONV output of our biomimetic, foveated perception model, in the context of which each retinal "image" may be regarded a point in a 10,800-dimensional photoreceptor response space. Among these are certain appearance-based recognition algorithms based on linear and multilinear PCA/ICA and more general manifold learning.

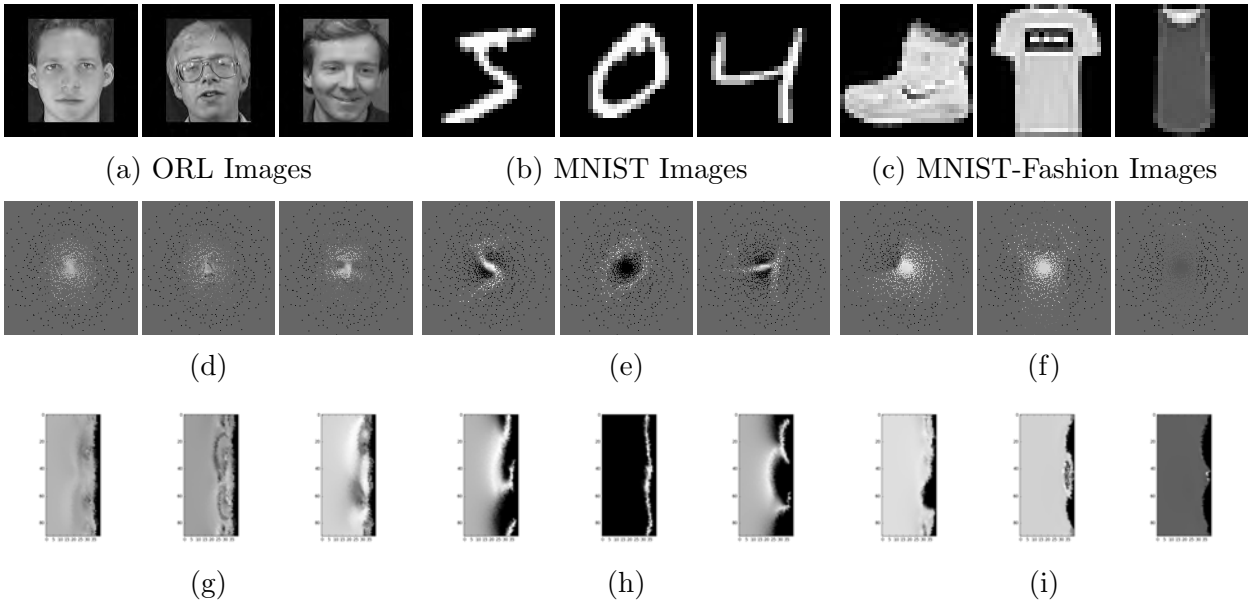
As a feasibility study, we applied our foveated perception model to image recognition. Figure 5.22(a)–(c) shows example images from the ORL[1] facial image database (Samaria and Harter, 1994), the MNIST[2] handwritten digit image database, and the Fashion-MNIST[3] clothing image database. As shown in Figure 5.22(d)–(f), the images from these databases are observed through our foveated retina model, centered on each image, yielding their associated ONVs. Figure 5.22(g)–(i) shows the idealized retino-cortical mapping (Schwartz, 1977) of each ONV. These maps are displayed as 40×90 images with horizontal equiangular lines and vertical equiradial lines, where the intensity of pixel (i, j) indicates the response of the photoreceptor situated on the retina at position $\boldsymbol{d}_k$ corresponding to ρ_i and θ_j in (4.2).

The ORL database consists of 10 different frontal face images of 40 subjects taken at different times, under different lightning conditions, and in different facial expressions. To

[1] http://www.cl.cam.ac.uk/research/dtg/attarchive/facedatabase.html

[2] http://yann.lecun.com/exdb/mnist/

[3] https://github.com/zalandoresearch/fashion-mnist



(a) ORL Images (b) MNIST Images (c) MNIST-Fashion Images

(d) (e) (f)

(g) (h) (i)

Figure 5.22: Example images from the ORL (a), the MNIST (b), and the Fashion-MNIST (c) databases. (d)–(f) The corresponding retinal "images" showing the intensity responses of the photoreceptors. (g)–(i) The corresponding 40×90 idealized retino-cortical map for each ONV.

test performance of our biomimetic perception model in the task of face recognition, we extended the ORL face database by including 3 rotated versions (15°, 30°, and 45°) of each image in the database, and all the images were resized to 145×145 pixels by padding with black pixels as necessary.

A fully-connected feedforward DNN of the type shown in Figure 4.11 but with an output layer comprised of 40 units, each unit associated with each of the 40 subjects, was trained with 30 ONVs of randomly-selected images from each subject. The remaining 10 ONVs of the subject were used for testing. We observed a recognition rate of 98.5%.

Additionally, the standard PCA image recognition algorithm was trained with 30 randomly-selected ONVs of images from each subject, while the remaining 10 ONVs of the subject were used for testing. We created a 40-dimensional linear model by retaining 40 eigenvectors, and employed the Nearest Neighbor Classifier (NNC) and Support Vector Machine (SVM) classifier. We observed recognition rates of nearly 93% and 95%, respectively.

Figure 5.23 summarizes the facial recognition performance results obtained in the aforementioned experiments.

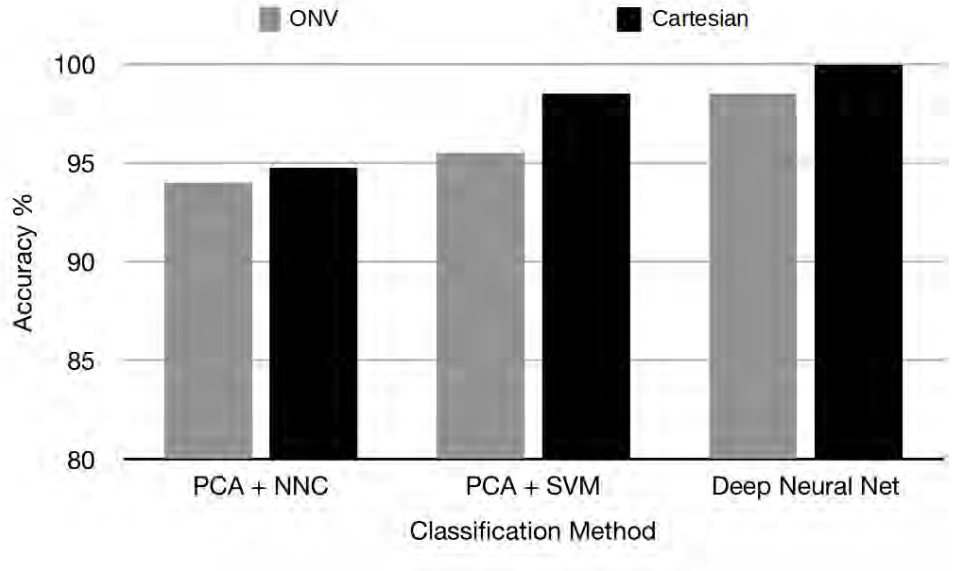


Figure 5.23: Comparison of the accuracy of face recognition using images and their ONVs, relative to other popular face recognition algorithms such as PCA+NNC, PCA+SVM, Deep multilayer perceptron, and VGG Face.

We also evaluated the recognition performance on the MNIST and Fashion-MNIST databases. Using a total of 50,000 ONVs of images from the training set of each database, we trained separate deep neural networks and tested them on 10,000 ONVs of images from the test set, obtaining accuracies of 98.50% and 89.16% on the MNIST and Fashion-MNIST images, respectively.

CHAPTER 6

Conclusion

We have introduced a biomimetic simulation framework for investigating human perception and sensorimotor control. Our framework is unique in that it features an anatomically accurate, biomechanically simulated human musculoskeletal model that is actuated by numerous contractile muscles. Our contributions in this thesis included the following primary ones to the fields of computer graphics and computer vision:

- The development of a biomimetic, foveated retina model, which is employed in a pair of human-like eyes capable of realistic eye movements, that uses raytracing to compute the irradiance captured by a multitude of nonuniformly arranged photoreceptors.
- The development of a prototype sensorimotor system, which (i) demonstrates deep-learning-based neuromuscular control of the neck-head, arms, and legs (under the influence of gravity) using 5 deep recurrent neural networks with proprioceptive feedback loops, and (ii) affords our virtual human with active visual capabilities using 10 deep feedforward neural networks driven by the optic nerve outputs of the eyes.
- Demonstration of the performance of our innovative sensorimotor system in nontrivial tasks that simultaneously involve eye movement control for saccadic foveation and smooth pursuit of a visual target in conjunction with appropriate dynamic head motion control, plus visually-guided dynamic limb control to generate natural limb reaching actions so as to make contact with a moving visual target.

Our approach was to train the set of 15 deep neural networks with very large quantities of training data that were synthesized by the biomechanical human model itself. We regard this achievement a contribution to the machine learning research community as well.

6.1 Future Work

Our work to date inevitably has some limitations that we plan to address in future work.

Our current eye models are idealized pinhole cameras. We plan to create a more realistic eye model that includes cornea and lens components to refract rays and, via active lens deformation, be capable of focusing onto the retina the rays passing through a finite-aperture pupil that can dilate and constrict to adjust the amount of incoming light, thus synthesizing depth of field phenomena.

Our current eye models are also purely kinematically rotating spheres. We plan to implement a fully dynamic eye model in which the sphere has the typical 7.5 gram mass of a human eyeball and is actuated by 6 synthetic extraocular muscles, including 4 rectus muscles to produce the θ, ϕ movement produced by our kinematic eyeball, but also the two oblique muscles that induce torsion in the gaze direction around the eye's z axis.

The jobs of the DNNs that must estimate from their ONV inputs the discrepancy between the 3D positions of the end effector and visual target are made difficult by the fact that 3D depth information is lost with projection onto the 2D retina and, in fact, the estimation of depth discrepancy is currently quite poor. This limitation provides an opportunity to explore binocular stereopsis with an enhanced version of our foveated perception model. For this, as well as for other types of subsequent visual processing, we will likely want to increase the number of photoreceptors, experiment with different nonuniform photoreceptor distributions, and automatically construct retino-cortical maps from the 1D ONV inputs.

Our system generates saccadic eye movements to foveate interesting objects in a variety of different scenarios. Hence, our model can be valuable in human visual attention research, a topic that we wish to explore in future work.

In future work we would also like to investigate the automatic recognition of objects, including faces, particularly active recognition perhaps along the lines of the active face recognition approach that we proposed in (Nakada et al., 2017) (see Appendix C).

We humans continue to learn from our ongoing experience; but once the DNNs in our

prototype sensorimotor system are trained offline, in advance, they remain unchanged as they perform their online sensorimotor control tasks. We would like to incorporate a continual, online deep reinforcement learning scheme into our model.

Medicine is a domain that our research can potentially benefit. For example, our biomechanical musculoskeletal model with its neuromuscular controllers can be useful in studies of muscle disorders. Such applications will require collaborations with researcher practitioners in the biomechanics and medical communities.

Currently, our musculoskeletal model has average human body form and physical parameters. One of the challenges is how to personalize the model to specific individuals, such as medical patients. The skeleton and muscle actuators can be easily morphed, but it is not clear if the trained neuromuscular controllers would continue to work robustly for a variety of body shapes. This is another reason to incorporate online learning, in order to adjust the neuromuscular controllers.

Finally, our ultimate goal is to free up our entire biomechanical musculoskeletal model, include additional motor DNNs to control its entire spine and enable it to stand upright, balance in gravity, locomote, and perform a variety of active vision tasks, including the aforementioned active recognition of objects and faces. We are excited by the seemingly endless possibilities along these lines of research.

APPENDIX A

Deep Learning of Neuromuscular Controllers Using Autoencoders

This appendix reproduces publication (Nakada and Terzopoulos, 2015).

Increasingly complex physics-based models enhance the realism of character animation in computer graphics, but they pose difficult motor control challenges. This is especially the case when controlling a biomechanically simulated virtual human with an anatomically realistic structure that is actuated in a natural manner by a multitude of contractile muscles. Graphics researchers have pursued machine learning approaches to neuromuscular control, but traditional neural network learning methods suffer limitations when applied to complex biomechanical models and their associated high-dimensional training datasets. We demonstrate that "deep learning" is a useful approach to training neuromuscular controllers for biomechanical character animation. In particular, we propose a deep neural network architecture that can effectively and efficiently control (online) a dynamic musculoskeletal model of the human neck-head-face complex after having learned (offline) a high-dimensional map relating head orientation changes to neck muscle activations. To our knowledge, this is the first application of deep learning to biomechanical human animation with a muscle-driven model.

A.1 Introduction

The modeling of graphical characters based on human anatomy is becoming increasingly important in the field of computer animation. Progressive fidelity in biomechanical modeling should, in principle, result in more realistic human animation. Given realistic biomechanical

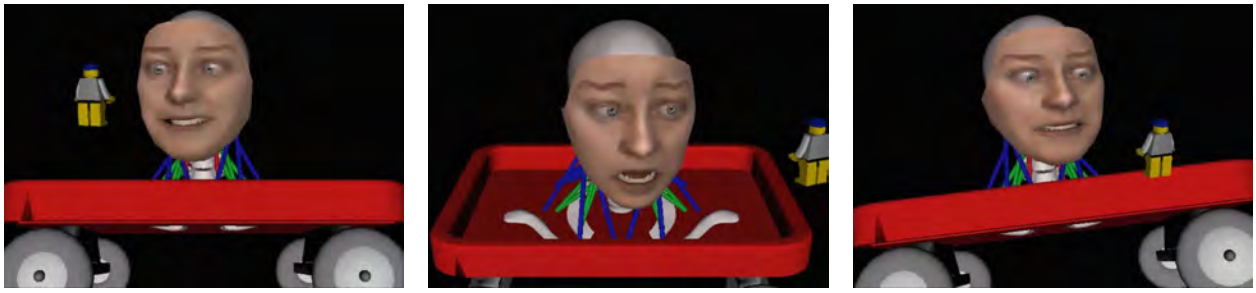


Figure A.1: Snapshots from an animation demonstrating robust online neuromuscular control of head orientation in a muscle-actuated biomechanical neck-head-face system. Despite the continuously varying orientation of its shoulders that are fixed to the unsteady wagon base, the character can visually track a moving target (doll) while naturally supporting the mass of its head in gravity atop its anatomically accurate flexible neck. Facial expressions are automatically produced in response to the observed movement of the doll—the character expresses awe when the doll is raised above the head, anger when the doll is shaken, and pleasure when the doll is held calmly at eye level.

models, however, we must confront a variety of difficult motor control problems due to the complexity of human anatomy.

In conjunction with the modeling of skeletal muscle (Ng-Thow-Hing, 2001; Irving et al., 2004), existing work in biomechanical human modeling has addressed the hand (van Nierop et al., 2007; Tsang et al., 2005; Sueda et al., 2008), torso (Sifakis et al., 2005a; DiLorenzo et al., 2008; Lee et al., 2009), face (Lee et al., 1995a; Kähler et al., 2002; Sifakis et al., 2005b), neck (Lee and Terzopoulos, 2006), etc. These prior efforts demonstrated different control schemes for individual parts of the human body. Further research is needed to achieve a fully robust control system for muscle-actuated biomechanical human animation.

The challenge in controlling biomechanical human models stems from anatomical intricacy, which complicates the kinematics, dynamics, and actuation of the character. For example, the neck-head biomechanical model developed in (Lee and Terzopoulos, 2006) has 10 bones and 72 muscle actuators (Figure A.2), while the full-body biomechanical model presented in (Si et al., 2014) includes 103 bones and 823 muscle actuators, an order of magnitude greater complexity. Controlling such models requires the specification of muscle innervations (i.e, activation signals) at every time step of the biomechanical simulation.

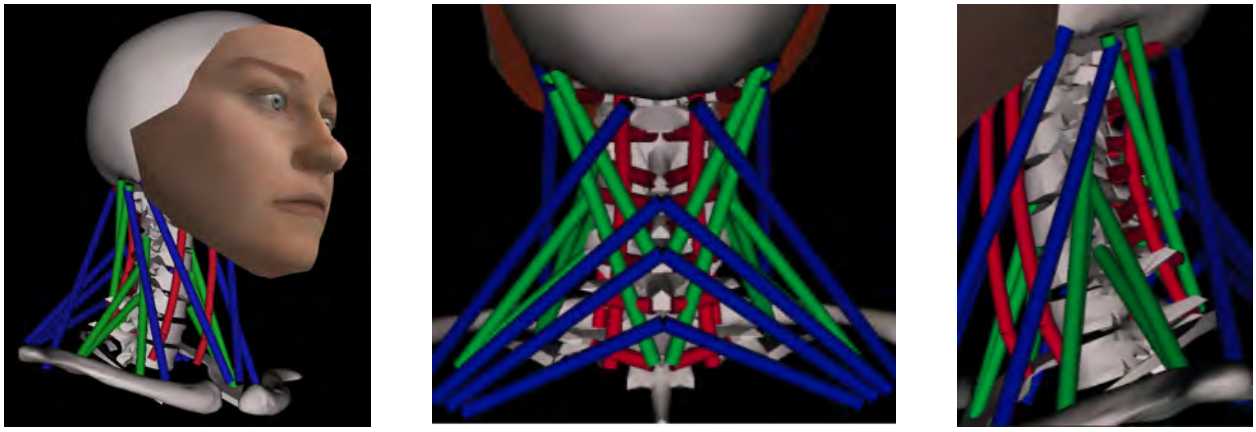


Figure A.2: Close-up views of the musculoskeletal structure of the biomechanical neck-head model, comprising 7 cervical vertebra and 72 actuators: 48 deep muscles (red), 12 intermediate muscles (green), and 12 superficial muscles (blue).

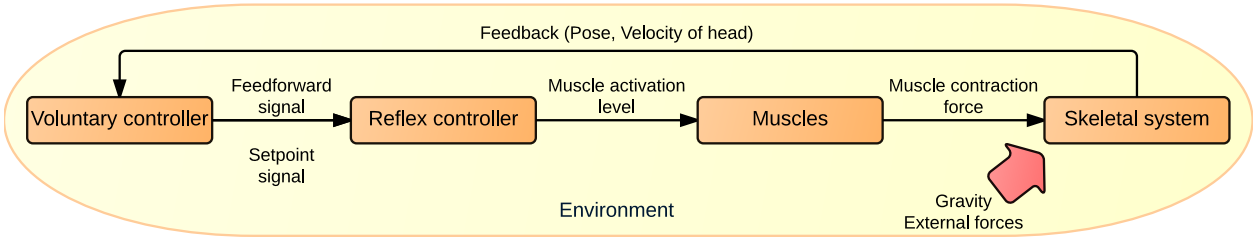


Figure A.3: Control flow for the neck-head model.

A.1.1 Neuromuscular Control

As pioneered by Lee and Terzopoulos (2006) in the context of neck-head control, neuromuscular learning approaches are the most biomimetic and promising in tackling high-dimensional, muscle-actuated biomechanical motor control problems. To control the neck-head system in gravity, rather than resorting to traditional inverse dynamics control, which is both unnatural and computationally expensive, these authors employed a conventional, artificial neural network with two hidden layers of sigmoidal units, which they trained offline through back-propagation learning. The training data comprised tens of thousands of random target head orientations as network inputs and the muscle activation levels required to achieve these target orientations as associated network outputs. Their trained neuromuscular controller was capable of efficiently controlling the biomechanical model online, thereby achieving a real-time neuromuscular neck-head control system. The neck-head biomechanical model is

actuated by 72 muscles, so their trained neural network controller maps 3 inputs (a target head orientation) to 72 outputs (the muscle activations required to achieve this orientation) with adequate generalization. Their *offline* training procedure took approximately 10 hours to complete the backpropagation learning process.

Conventional neural networks with only a few hidden layers have not proven effective at coping with the higher dimensionalities and necessarily much greater quantities of training data required to learn to control biomechanical models of an order of magnitude greater complexity, such as the one presented in (Si et al., 2014). For this reason, Lee and Terzopoulos (Lee and Terzopoulos, 2006) made no attempt to generalize their neck-head controller to deal with other than horizontally-oriented shoulders and, unsurprisingly, their trained neuromuscular controller fails to maintain satisfactory control unless the shoulders remain nearly horizontal.

A.1.2 Deep Learning of Neuromuscular Control

Deep neural networks with many hidden layers could not at first be trained due to technical difficulties with the backpropagation learning algorithm such as local minima and the vanishing gradient problem. Motivated by recent breakthroughs in "deep learning" that have mitigated these problems, we have applied deep neural network architectures to neuromuscular control. The long-range goal of our research is to overcome the aforementioned challenges in the neuromuscular control of complex biomechanical models.

In this paper, we adapt deep learning techniques to the neuromuscular control of the human neck-head biomechanical model described in (Lee and Terzopoulos, 2006). Our contributions include the introduction of a new neuromuscular controller, whose deep architecture differs markedly from the one in (Lee and Terzopoulos, 2006), with dramatically enhanced learning ability and efficiency even when employing much higher dimensional training data. This enables us to push significantly beyond the prior benchmark, training our neck-head neuromuscular controller to work well not merely for horizontal shoulders, but for a considerable range of shoulder orientations.

A.1.3 Overview

The remainder of this paper is organized as follows: In Section A.2, we briefly explain the neck-head model and its neuromuscular control system. We then develop our novel neuromuscular learning approach in Section A.3, and demonstrate the effectiveness of our approach through experiments reported in Section A.4. Section A.5 presents our conclusions and discusses future work.

A.2 Biomechanical Model and Control System

In our work, we use the anatomically accurate musculoskeletal model of the human neck-head complex implemented by Lee and Terzopoulos (2006), which is illustrated in Figure A.2. The model comprises a skull, whose mass is commensurate with that of an average adult male head, atop a cervical skeleton comprising 7 vertabrae connected by 3 degree-of-freedom rotational joints, and a clavicle base. This skeletal system is actuated by 72 contractile cervical muscles, modeled as Hill-type actuators, arranged in three layers (Figure A.2). Appendix B presents additional details about the biomechanical model. The associated neuromuscular control architecture (Figure A.3) involves a high-level voluntary controller that determines the head pose and a lower-level reflex controller that produces the necessary muscle activation inputs. Appendix B presents additional details about the hierarchical neuromuscular control system, as well as the gaze control mechanism.

The specific technical challenge that we address in our work is how best to implement and train the voluntary controller. The considerable kinematic and muscular redundancy of the biomechanical model makes it infeasible to control the cervical muscles online in real time using an inverse dynamics approach. Instead, as in (Lee and Terzopoulos, 2006), we pursue a biomimetic neuromuscular control approach. A neural network controller is trained offline on a large dataset comprising several tens of thousands of random head orientations (inputs) and the associated muscle activations (outputs). To generate training data, each 72-dimensional muscle activation output, which must achieve the associated target head pose

and sustain it against the pull of gravity, is synthesized off-line through (computationally expensive) inverse kinematics/dynamics computations as in (Lee and Terzopoulos, 2006).

Lee and Terzopoulos (2006) used a conventional, shallow neural network for their voluntary controller. It typically took on the order of 10 hours to train this network, which imposed practical limits on the dimensionality and size of the usable training dataset. To surmount this obstacle in neuromuscular control, we devise a deep neuromuscular control architecture that proves to be more efficient to train and is capable of learning more proficiently from larger and much higher-dimensional datasets. In principle, this makes it amenable to biomechanical models of much greater complexity—potentially even to a full-body biomechanical model with an order of magnitude more articular degrees of freedom and muscle actuators (Lee et al., 2009; Si et al., 2014).

At each time-step of the biomechanical simulation, muscle activations are computed by our trained deep neural network in the voluntary controller, and they are input to the reflex controller. The latter adjusts the muscle activations in accordance with the difference between the current state and the desired state of each muscle. The biomechanical simulator computes the muscle forces induced by the muscle activations, applies the muscle forces to the dynamic skeletal system, and numerically updates the accelerations, velocities, and positions of the articulated bones, thereby advancing the simulation in time.

A.2.1 Egocentric Head-Eye Control

Humans obtain visual information about objects from their foveated retinal images. Sensing a target of interest in the periphery of the visual field triggers a rapid eye rotation to foveate the target in conjunction with a comparatively sluggish head rotation (due to the greater mass of the head) toward the target. Lee and Terzopoulos (2006) synthesized desired head rotation trajectories through the spherical linear interpolation (slerp) of head Euler angles, which is unnatural.

By contrast, in our neuromuscular control system, we define the angular error vector as the difference of the current head orientation and the target head orientation. In each

time step, our neuromuscular controller acts to reduce this 2-D angular error vector (with horizontal angle θ and vertical angle ϕ). Our neck-head control system does not require knowledge of the position of the visual target nor of the orientation of the head relative to the world frame. Rather, the relevant variables are represented relative to the egocentric frame whose origin is at the head and which rotates with the head.

A.3 The Deep Neuromuscular Controller

Our neuromuscular controller differs substantially from the shallow (3-input, 72-output, 2-hidden-layer) network described in (Lee and Terzopoulos, 2006). We map to the 72 muscle activation outputs not only the target head orientation inputs, but also the 72 current muscle activations, so that the current muscle activation state affects the computation of the output muscle activations. Incorporating the current muscle activations as inputs is necessary in order to reduce the angular error vector in egocentric coordinates while achieving robust control regardless of shoulder orientation. The higher input dimensionality of our controller along with the much greater quantity of data necessary to train it at a variety of shoulder orientations exceeds the capabilities of the shallow network used in (Lee and Terzopoulos, 2006), and this problem is unlikely to be overcome without the use of a deeper network.

Our proposed deep network architecture includes 74 inputs, comprising the 2 angles, θ and ϕ, of the head orientation error vector plus the 72 current muscle activations, along with 72 muscle activation outputs. The intermediate, hidden layers of the network implement an n-stage stacked denoising autoencoder (SdA) (Bengio et al., 2007), which will be described in the next section. The bottom layer outputs the modified 72 cervical muscle activations (illustrated in Appendix B).

A more powerful neuromuscular controller learning method is needed to train our deep network. Our learning method has two phases. The first is a pre-training phase, where we apply unsupervised learning to the SdA. The second is a fine-tuning phase where we apply supervised learning to the multilayer perceptron. The first phase contributes to finding better initial parameters and the second phase tunes those parameters by comparing the

output from the network and the training example values.

The overall learning process proceeds as follows: We generate a training dataset comprising many input/output examples by numerically simulating the biomechanical model. Then, we run the SdA pre-training phase on this dataset. After pre-training, we initiate fine-tuning with the multilayer perceptron. Subsequently, we can employ the trained deep network online to control the biomechanical model.

Additional technical details about our deep neuromuscular controller learning process are presented in the following sections.

A.3.1 Stacked Denoising Autoencoder (SdA)

An autoencoder is a multilayer neural network that minimizes the difference between its input and output after encoding/decoding through its hidden layers. The encoding is as follows:

$$\boldsymbol{y}_e = \boldsymbol{\sigma}(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}), \tag{A.1}$$

where $\boldsymbol{x}$ is the input, $\boldsymbol{y}_e$ is the encoded output, $\boldsymbol{\sigma}$ is a nonlinearity with sigmoidal $\sigma(z) = 1/(1+e^{-z})$ components, $\boldsymbol{W}$ is the weight matrix, and $\boldsymbol{b}$ is the bias vector. Likewise for the decoder,

$$\boldsymbol{y}_d = \boldsymbol{\sigma}(\boldsymbol{W}'\boldsymbol{y}_e + \boldsymbol{b}'), \tag{A.2}$$

where $\boldsymbol{y}_d$ is the decoded value, and $\boldsymbol{W}'$ and $\boldsymbol{b}'$ are the weight matrix and bias vector. After encoding/decoding, the inputs are compared to the outputs, yielding the error function

$$E = \frac{1}{N}(\sum_{i=1}^{N} ||\boldsymbol{x}_i - \boldsymbol{y}_{di}||^2), \tag{A.3}$$

where i indexes over the N training examples. The error gradient ∇E is used to update the weights and biases through gradient descent (in our work, we set the learning rate to 0.1).

In mini-batch stochastic gradient descent, ∇E is estimated using a limited number of training examples. We include 32 training examples in each mini-batch. Processing all

the mini-batches in this way constitutes a training epoch. We use a tied-weight scheme, $\boldsymbol{W}' = \boldsymbol{W}^T$, which has a beneficial regularization effect and requires less memory. Hence, the parameters to be updated are $\boldsymbol{W}$, $\boldsymbol{b}$, and $\boldsymbol{b}'$.

A Denoising Autoencoder (dA) forces its hidden layers to discover more robust features and prevent learning the identity mapping, which the simple autoencoder may uselessly determine as being the zero-error optimal solution. To this end, one adds noise to the training data, and the dA tries to reduce the noise and recover the original data from the corrupted data. Associating a probability (0 implies certainty) with each unit in each layer, we set the probabilities to 0.1 for the hidden layers and randomly mask some inputs by setting them to zero.

We stack the dAs to form a Stacked Denoising Autoencoder (SdA), where the outputs of each dA are propagated as inputs to the next dA. The advantage of the SdA is that each dA is trained independently and sequentially, and the dAs may be stacked as deeply as necessary to produce a good output.

A.3.2 Pre-Training Phase

During the pre-training phase, the training starts at the input layer of the SdA and advances to the output layer, completing the training one layer at a time. We run 300 epochs for each layer sequentially such that each dA can learn a good representation and the input data are mapped to the output data with minimal information loss.

The initial weights are uniformly sampled from the interval $[-v, v]$, where $v = \sqrt{6/(n_i + n_o)}$, with n_i being the number of inputs and n_o the number of outputs in a layer. It is known that uniform sampling from this interval works well with sigmoidal units (Glorot and Bengio, 2010). This prevents the learning process from becoming trapped in a local minimum early in the pre-training phase and it yields better optimization and faster convergence.

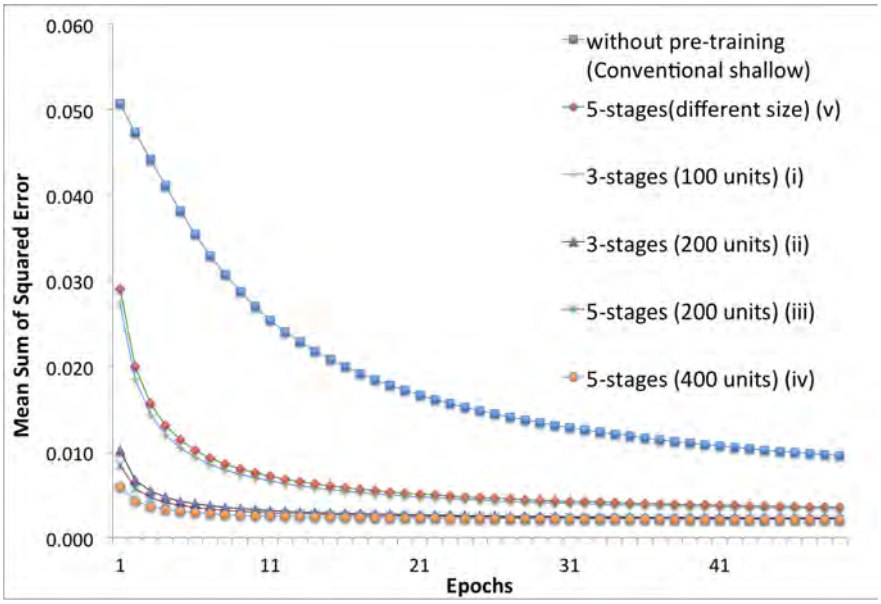


Figure A.4: Training error with different network structures when fine-tuning on the validation dataset.

A.3.3 Fine-Tuning Phase

During the fine-tuning phase, we start with the weights and biases obtained in the pre-training phase, and then we update them using the multilayer backpropagation algorithm. This phase is supervised, as we compare the outputs to the labeled training data and reduce the error by updating the weights and biases using mini-batch stochastic gradient descent with (mean sum-of-squared) error backpropagation.

The training process is stopped when it ceases to improve sufficiently on a validation dataset (we set the threshold in each training epoch to 0.005%). This "early stopping" prevents the training process from overfitting.

A.4 Experiments and Results

A.4.1 Fixed, Horizontal Shoulder Orientation

Our first set of experiments evaluates the performance of our deep neuromuscular controller relative to the shallow one reported in (Lee and Terzopoulos, 2006) with fixed horizon-

tal shoulder orientation. Figure A.4 plots the learning performance of different network structures. The plots indicate the superiority of our deep neuromuscular controller learning method with pre-training. Appendix B presents the details of this experiment.

A.4.2 Robust Control With Varying Shoulder Orientation

We verified that the shallow neuromuscular controller developed in (Lee and Terzopoulos, 2006) fails to maintain control of the head when the shoulders are not in a horizontal orientation. To facilitate comparison, the motion of the doll visual target is the same as the one in the head-eye gaze behavior demonstration illustrated in Figure 10 of reference (Lee and Terzopoulos, 2006). The more the orientation of the shoulders deviates from the horizontal, the more difficulty the shallow neuromuscular controller experiences in tracking the visual target. More strikingly, as the shoulders tilt back moderately, the neck-head skeleton collapses backward in gravity, resulting in catastrophic failure. We will next explain how we train our deep neuromuscular controller to accomplish this more challenging control task.

Synthesis of Training Data: First, with horizontal initial shoulder orientation, we synthesize training data by incrementing the head by 1° angles around the θ and ϕ axes in the range $-50^\circ < \theta < 50^\circ$ and $-50^\circ < \phi < 50^\circ$. Then, we repeat the same process with different shoulder orientations, varying the orientation of the shoulders by 5° from -25° to 25° around both the frontal x axis along the shoulders and the sagittal z axis. This yields 1M example target angular error vectors with associated muscle activations computed through inverse kinematics/dynamics. The training data synthesis took 6.5 days to complete. Finally, we partition the data into training, validation, and testing datasets in the ratio 70%:15%:15%, respectively.

We trained our deep neuromuscular controller with the aforementioned dataset and verified that, unlike the shallow neuromuscular controller, our controller enables the neck-head system to smoothly track the visual target while the orientation of the shoulders varies continuously in the range of -25° to 25° around both the x and y axes. We used the 5-stage SdA network with 400 units in each stage—network (iv) in Figure A.4—since it proves to

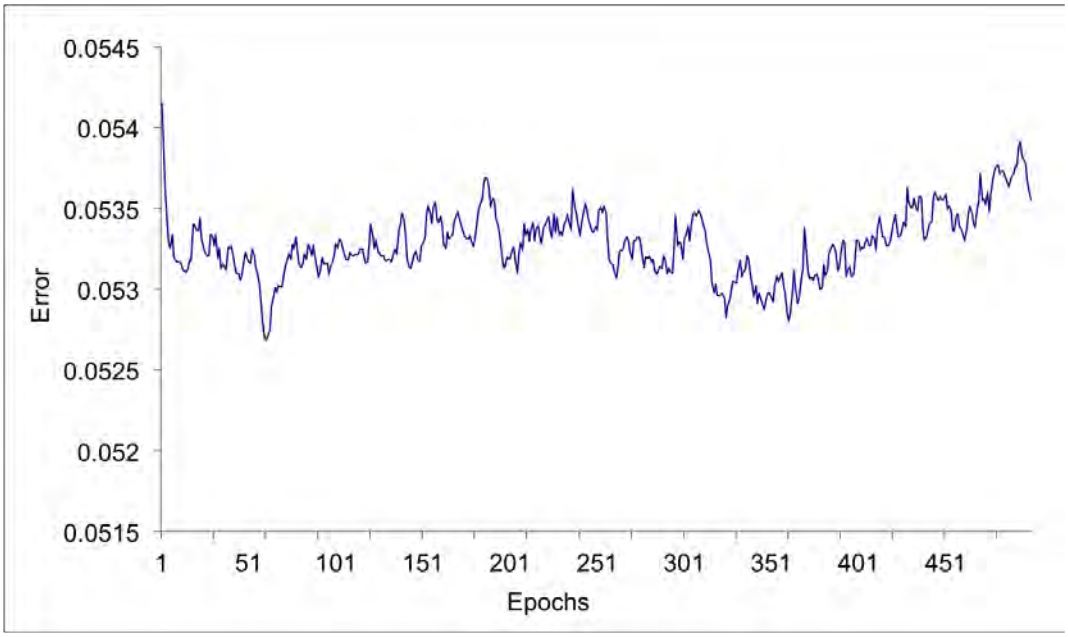


Figure A.5: Training error progress with large dataset using the previous neural network method.

learn best as is shown in the figure.

Figure A.1 shows snapshots from a demonstration video of our neck control system, demonstrating that it copes very well with continuously varying shoulder orientations. Although our training dataset was 33 times larger (and higher-dimensional) than the one used in (Lee and Terzopoulos, 2006), our off-line training process completed in only 23.0 hours, with the pre-training phase taking 19.3 hours and the fine-tuning phase taking 2.7 hours.

The pre-training phase worked well with our training data, and our deep neural network learned the appropriate initial weights and biases after 300 epochs of training for each autoencoder layer. Subsequently, the fine-tuning phase started with a relatively small 0.021 error (the error would have been much higher if we simply set random initial weights and biases). This reduced the time for the fine-tuning phase, which completed with 0.003 error after satisfying the early stopping condition in 82 epochs.

For a fair comparison, we replicated the shallow neural network presented in (Lee and Terzopoulos, 2006) and tested it with our larger dataset. Figure A.5 includes a plot of the training progress. It took 2.5 days to complete 500 training epochs, and the error fluctuated

between 0.0525 and 0.0545 without decreasing. None of the different parameter settings and shallow network structures that we tried resulted in successful convergence.

A.5 Conclusion and Future Work

We have introduced a new approach to the design and training of neuromuscular controllers of complex biomechanical human models for use in computer graphics animation. To our knowledge, this is the first application of deep neural networks to the problem. Our approach proves to be superior to previous methods both in terms of the efficiency of the learning process and, potentially more importantly, in the ability to learn effectively from much larger quantities of higher-dimensional training data. These benefits enabled us to train a deep neuromuscular neck-head controller to work well over a continuous range of shoulder orientations.

Our experiments demonstrated that our deep network, which incorporates significantly more units in its hidden layers, can learn effectively from our largest training datasets (in our case, one million input/output examples), reinforcing the notion that deep networks are generally more suitable than shallow networks for the regression, with good generalization, of massive quantities of training data.

We expect our deep learning approach to architecting and training neuromuscular controllers to work for other, similarly complex biomechanical models, such as those for arms and hands. Furthermore, our approach seems promising for the neuromuscular control of full-body human models with a dramatically higher dimensional muscle activation space, on the order of 1,000 muscle actuators.

APPENDIX B

Additional Details Relating to Appendix A

B.1 Neck-Head-Face Biomechanical Model

The neck-head-face model is comprised of a base link, C1-C7 links, a skull, and 72 muscles. The cervical spine and skull are modeled as a dynamic articulated body with appropriate mass distribution and three degree-of-freedom joints. The neck is actuated by muscles, which are modeled as Hill-type uniaxial actuators. The details of this biomechanical neck-head model are explained in (Lee and Terzopoulos, 2006), and the physics-based face model is described in (Lee et al., 1995a). This biomechanical system emulates the anatomy of the neck and yields realistic neck motions.

B.1.1 Cervical Skeleton

The skeletal structure is represented as an articulated multi-body system. It contains a base link, skull, and C1-C7 cervical links, as we show in Figure B.1.

Each joint is modeled as a 3-DOF rotational joint with damped springs to approximate the stiffness on the ligaments and disks. The spring torque is computed as

$$\tau_s = -k_s(q - q_0) - k_d\dot{q}, \tag{B.1}$$

where k_s is the spring stiffness and k_d is a damping coefficient, q is the joint angle, q_0 is the original joint angle in natural pose, $\dot{q}$ is the joint angular velocity. The system becomes more stiff as we increase the damping coefficients. It is necessary to keep the simulation stable. The physical parameters of the skeleton are tabulated in Figure B.2.

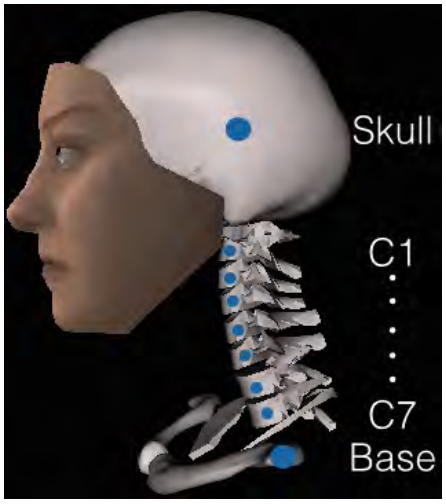


Figure B.1: Structure of the neck skeleton.

Bone	Mass	ks: x, z axes	ks:y axis
Skull	3.5	50	25
C1 - C7	0.21	60	30

Figure B.2: Anatomical parameters of the neck skeleton.

The equations of the motion of the skeletal system are

$$\boldsymbol{M}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{c}(\boldsymbol{q}, \dot{\boldsymbol{q}}) + \boldsymbol{K}_s + \boldsymbol{K}_d\dot{\boldsymbol{q}} - \boldsymbol{P}(\boldsymbol{q})\boldsymbol{f}_P = \boldsymbol{P}(\boldsymbol{q})\boldsymbol{f}_C + \boldsymbol{J}(\boldsymbol{q})^T\boldsymbol{f}_e, \tag{B.2}$$

where $\boldsymbol{M}(\boldsymbol{q})$ is an inertia matrix of the skeleton and function $\boldsymbol{c}$ represents the combination of Coriolis forces, centrifugal forces, and gravity. The 24-dimensional vectors $\boldsymbol{q}$, $\dot{\boldsymbol{q}}$, and $\ddot{\boldsymbol{q}}$ are angle, angular velocity, angular acceleration of the joints, respectively. $\boldsymbol{K}_s$ and $\boldsymbol{K}_d$ are the stiffness and damping coefficient matrices of the joints. $\boldsymbol{J}(\boldsymbol{q})$ is the Jacobian matrix, which transforms the external force to joint torques. The muscle force is divided into two components, which are $\boldsymbol{f}_P$ and $\boldsymbol{f}_C$. The former is the passive force produced by the material properties when it is deformed, and the later is the contractile force generated by muscle activated with the activation input. $\boldsymbol{P}(\boldsymbol{q})$ is the moment arm matrix which maps muscle force to joint torques, which is computed by following the principle of virtual work. All forces need to be represented in the joint space in our system.

We can compute $\ddot{\boldsymbol{q}}$ with Featherstone's algorithm. Then, we obtain $\dot{\boldsymbol{q}}$ and $\boldsymbol{q}$ at next time

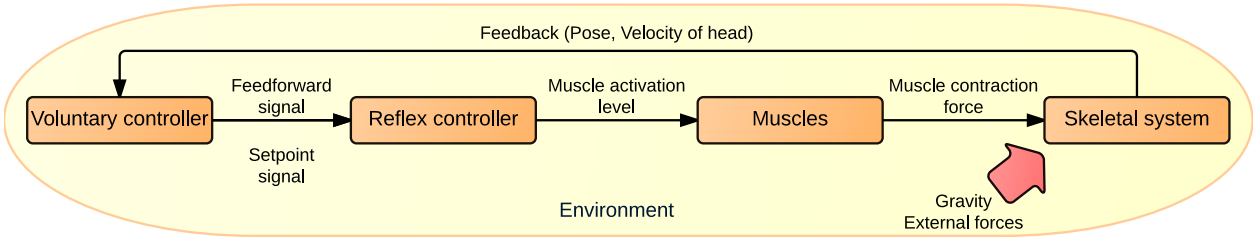


Figure B.3: Control flow chart of neck-head-face model.

step by numerically integrating in time with the explicit Euler method.

B.1.2 Neck Muscle System

The neck model uses the same muscle model described in Section 3.2.2 for the full-body model.

B.2 Neck Control System

The neck control system is described in (Lee and Terzopoulos, 2006). The system comprises the skeleton, muscles, and hierarchical controller. This hierarchical controller has two layers. The first is a voluntary controller which generates the feedforward signal and setpoint control signal. This feedforward signal is determined by the desired pose. The setpoint signal specifies the desired strain and strain rate of the muscle and it also feeds back the gain to compensate for the error in the next step. The second layer of the controller is the reflex layer, which plays a role in adjusting the adequate control taking the information of the current state of the joints. It compares the strain and strain rate against the desired value, and generates the feedback signal so that it can minimize the difference between the current pose and the desired pose. This feedback signal is added to the feedforward signal and used to compute the desired muscle activation. Given the input muscle activation, each muscle generates the contractile force, which depends on the length and velocity of the muscle. The control flow is shown in Figure B.3.

B.2.1 Hierarchical Control System

The control system has two controllers, a reflex controller and a voluntary controller.

1. Reflex Control: The reflex controller is the same model as the one used in full-body model.

2. Voluntary Control: the neck system, the muscle controller generates signals—the pose signal defines the required activation to achieve specified pose, and the tone signal defines the stiffness of the muscle, which is necessary to keep the neck upright against gravity and other external forces. Each signal is computed using different sub-controllers. The sum of the two components provides the activation value of the system at each time step.

B.3 Learning Network Structure

The architecture of our deep neural network is illustrated in Figure B.4. It constitutes an input layer, hidden layers, and an output layer. Each pair of adjacent layers is trained as a denoising autoencoder in the pre-training phase, and then we improve the weights and biases in the fine-tuning phase. In this second phase, the network is trained as a multilayer perceptron. The structure of the network remains unchanged in the two phases; however, we use different learning techniques, The controller takes head orientation errors (2 inputs) and current cervical muscle activations (72 inputs) as the inputs, and outputs the 72 required muscle activations

B.4 Experiments and Results

We covered the case of variable shoulder orientation in Appendix A. In this section, we explain our method for fixed horizontal shoulder orientation.

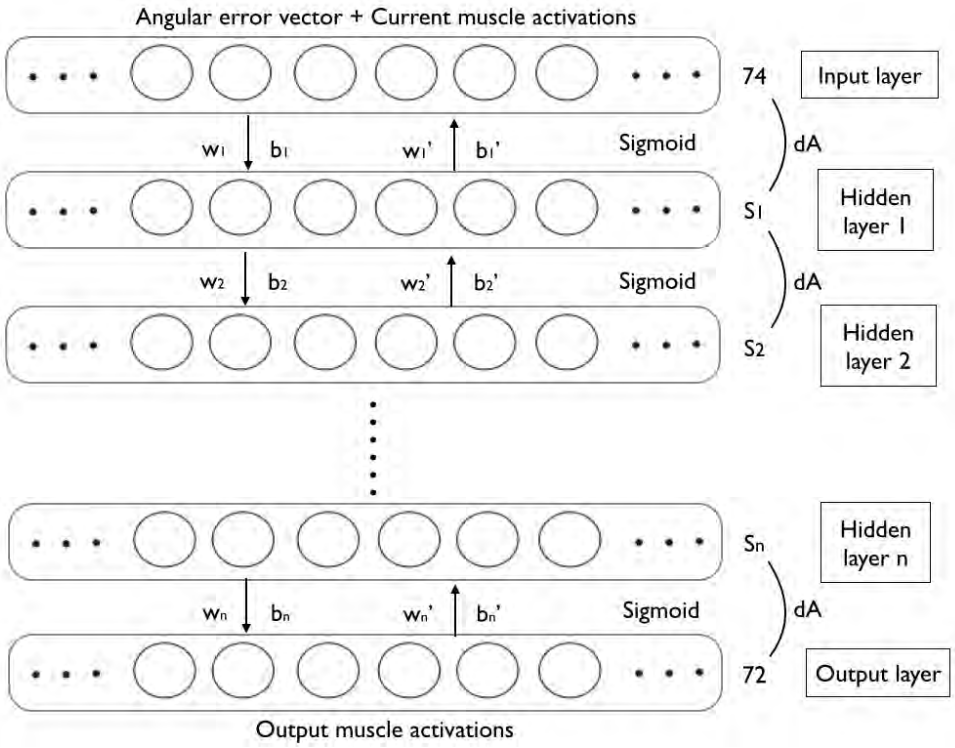


Figure B.4: Structure of the deep neuromuscular controller. The head orientation error (2 inputs) and current cervical muscle activations (72 inputs) comprise the 74 inputs to the controller (top). The controller outputs 72 muscle activations (bottom). The input layer, n hidden layers, and output layer constitute a stack of denoising autoencoders (dAs).

B.4.1 Fixed, Horizontal Shoulder Orientation

Synthesis of Training Data: We synthesized the controller training data offline using inverse kinematics and dynamics, a necessarily expensive method for computing the required muscle activations to achieve a target head orientation and sustain it against the downward pull of gravity. To create the training dataset, we sample the head orientation space by incrementing the target orientations by 0.33° around each axis in the range of $-50° < \theta, \phi < 50°$, where θ and ϕ are the horizontal and vertical angles in the eye-centered coordinate system, respectively, with $\theta = \phi = 0$ being the neutral (forward-looking) head pose. This yields 30,000 samples of target head pose error vectors and associated muscle activations. Then, we partition the synthesized data into three subsets—a training dataset, a validation dataset, and a testing dataset—in the ratio 70%:15%:15%, respectively.

The results plotted in Figure A.4 confirm the superiority of our deep neuromuscular

Figure B.5: Snapshots from an animation that demonstrates the neck-head-face system fixating and tracking the moving doll visual target using online neuromuscular control of head orientation.

controller learning method. For this set of experiments, we employed a training dataset that is approximately the same size as the one used in (Lee and Terzopoulos, 2006). The conventional multilayer perceptron with random initial weights and biases (no pre-training) started with a error 0.051. By contrast, for our 3-stage SdA network with 200 units in each stage (purple triangle plot in Figure A.4), we were able to start the fine-tuning phase with a error 0.010 by virtue of the weights and biases initialized by pre-training with the stacked denoising autoencoder. Our learning process then converged to an error 0.002 with the validation dataset, satisfying the early stopping condition in 68 epochs. On a PC with a 2.4GHz Intel Core i5 CPU, the total training time was 26.33 minutes, consisting of 21.17 minutes for the pre-training and 5.16 minutes for the fine-tuning. This learning process is dramatically faster than that for the network proposed in (Lee and Terzopoulos, 2006), which reportedly took approximately 10 hours to train on a PC with a 3.2GHz CPU. For a fair comparison, we ran their training method on our machine, and it took 11.8 hours to complete.

We tested our method with four other network structures, resulting in the following five cases:

Case (i) — a 3-stage SdA with 100 units per stage,

Case (ii) — a 3-stage SdA with 200 units per stage,

Case (iii) — a 5-stage SdA with 200 units per stage,

Case (iv) — a 5-stage SdA with 400 units per stage, and

Case (v) — a 5-stage SdA with different stage widths:

$S_1 = 200$, $S_2 = 600$, $S_3 = 1000$, $S_4 = 600$, and $S_5 = 200$ units. The results are shown in

Figure A.4. The deep network in Case (iv), 5 hidden layers with 400 units in each layer, worked the most effectively. After the pre-training phase, the fine-tuning started with error 0.006 and satisfied the early stopping condition in 51 epochs, taking a total of 67.44 minutes to finish the training process. This experiment indicates that using the same number of units in each hidden layer works better than a pyramid-shaped network structure. Furthermore, we showed that the deeper network was more effective by comparing Case (ii) and Case (iii).

Figure B.5 shows snapshots from the demonstration video of our neck control system. Our trained neuromuscular controller is able to activate the 72 neck muscles to balance the head in gravity atop the cervical column and smoothly track a visual target (a doll) that moves in 3D space in front of the neck-head system. The animations resulting from the use of our controller are as smooth and natural looking as the ones presented in (Lee and Terzopoulos, 2006), which suggests that our method works well with the neck/head biomechanical model, even though our controller uses egocentric coordinates and requires dramatically shorter learning times.

APPENDIX C

AcFR: Active Face Recognition Using Convolutional Neural Networks

This appendix reproduces publication (Nakada et al., 2017).

We propose AcFR, an active face recognition system that employs a convolutional neural network and acts consistently with human behaviors in common face recognition scenarios. AcFR comprises two main components—a recognition module and a controller module. The recognition module uses a pre-trained VGG-Face CNN to extract facial image features, along with a nearest-neighbor identity recognition criterion. The controller module can make three different decisions based on the results—greet a recognized individual, disregard an unknown individual, or acquire a different viewpoint from which to reassess the subject, which are natural reactions when people observe passers-by. Evaluated on the CMU PIE face database, our recognition module yields higher accuracy on images acquired at angles more similar to those saved in memory. The view-dependent accuracy provides evidence for the proper design of the controller module.

C.1 Introduction

Face recognition is a classic computer vision problem. A big challenge is improving recognition accuracy given limited information. In this context, the active perception approach in computer vision (Bajcsy, 1988; Aloimonos et al., 1988; Ballard, 1991; Terzopoulos and Rabie, 1997) seems promising. For example, when observing the profile of someone who looks like a friend, before presuming to greet the individual, one might move to a better viewpoint from which to see more of the face of interest in order to ascertain the person's

identity. The idea of viewpoint-dependent recognition has been studied in neurophysiology (Andrews and Ewbank, 2004), and Wang et al. (1998) found face-selective neurons tuned to specific viewpoints. These considerations have motivated us to develop an active face recognition (AcFR) system that models human behaviors in common face recognition scenarios. Unlike most existing face recognition systems, it does not rely solely on a single input image; instead, like people in real-world scenarios, it acquires additional images from different viewpoints to improve recognition accuracy.

AcFR comprises two main components, a face recognition module and a behavior controller module. The face recognition module, which employs VGG-Face, a popular convolutional neural network (CNN), in conjunction with a nearest-neighbor identity recognition criterion, evaluates the input image and provides the data needed to make decisions. The controller module uses these data to drive its follow-up behavior, including whether to greet the (known) observed individual, disregard the (unknown) individual, or obtain a different viewpoint from which to reassess the subject.

The main contributions of this study are twofold: First, we propose the AcFR approach, which actively recognizes faces by mimicking human behaviors. Second, the use of a CNN makes the visual processing more biologically plausible, which is relevant in future applications of AcFR to biomechanical virtual human models.

C.2 Related Work

Our work is relevant to human modeling and simulation. Early efforts on this topic focused on human face and body modeling. Over time, technologies from computer graphics, computer vision, machine learning, psychology, etc., have been converging on this area. Some recent work, including ours, considers brain functionalities such as visual recognition, learning, and communication abilities.

With regard to recognition, much work has been done on object recognition, especially face recognition. Learning-based face recognition has recently made major strides. Traditional approaches represent faces using hand-crafted features extracted from the image, such

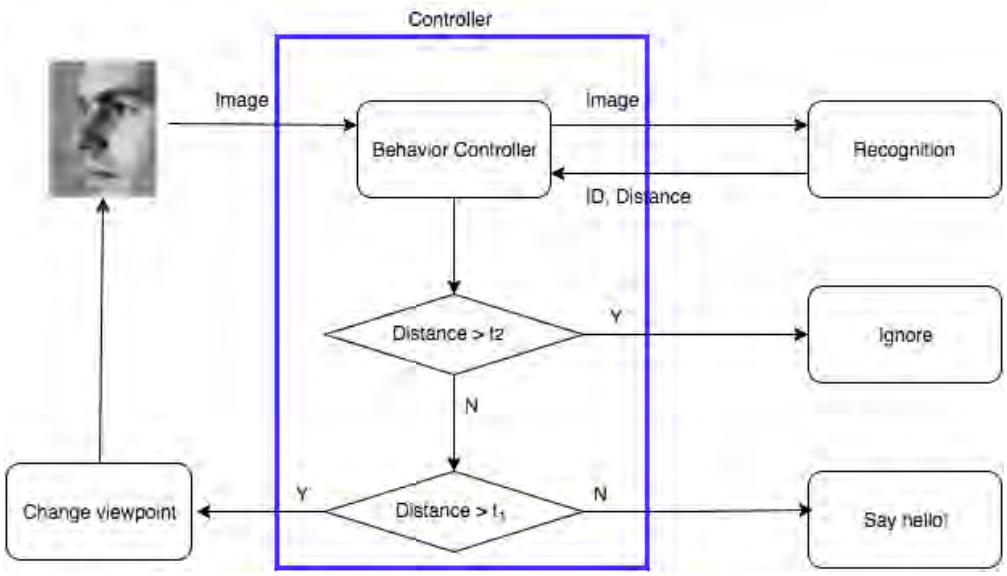


Figure C.1: System architecture

as SIFT, LBP, and HOG (Sivic et al., 2005; Wolf et al., 2011; Lu and Tang, 2014), which are then employed in the classification step. So-called deep learning approaches employ artificial neural networks that learn appropriate features automatically via training on massive quantities of image data. Among neural networks, CNNs are preferred by the computer vision community in part because the mechanisms underlying their architectural design are suggestive of cortical mechanisms in biological vision systems.

Although CNNs have been applied to face recognition as far back as 1997 (Lawrence et al., 1997), the recent availability of massive image datasets have revealed their power. A representative work of this class of approaches is Deep-Face (Taigman et al., 2014), which uses a 9-layer CNN trained on 4.4 million labeled facial images including over 4,000 identities. It has achieved outstanding performance in both the Labeled Faces in the Wild (LFW) (Lu and Tang, 2014) and YouTube Faces (YTF) (Wolf et al., 2011) benchmarks. Subsequently, Parkhi et al. (2015) proposed the VGG-Face network, which we have adopted in our work. It uses a 16-layer CNN trained on 2.6 million images and achieves even better accuracy in these benchmarks.

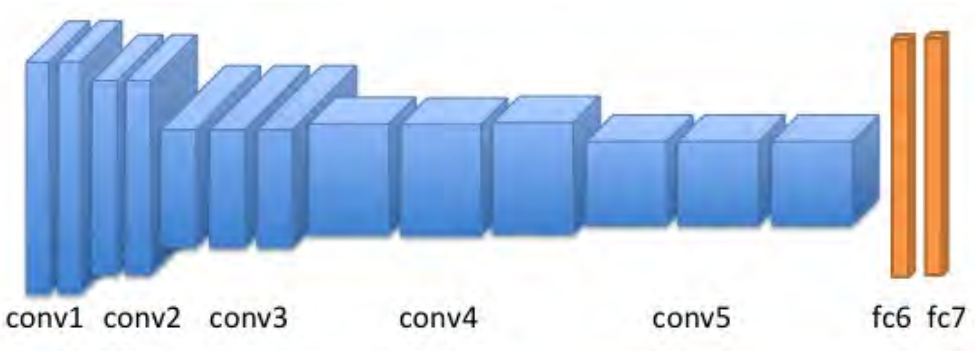


Figure C.2: VGG-Face network architecture

C.3 Methodology

As illustrated in Figure C.1, our AcFR system consists of two main modules—a behavior controller module, which is served by a face recognition module. Given a facial image, the recognition module will attempt to determine the subject's identity and will provide its results to the controller. The controller has two functionalities: to re-evaluate the face whenever the viewpoint changes and to model human behaviors based on the results of recognition.

C.3.1 Face Recognition

The modern face recognition pipeline usually consists of four stages: detection, alignment, representation, and classification. Detection and alignment are often included as preprocessing steps. Given a good facial representation, the system can predict identity through classification algorithms.

C.3.1.1 Preprocessing

Face detection and alignment algorithms are often employed because many face recognition algorithms require the input images to be carefully positioned into a canonical pose. Sometimes the assumption is made that the detection step has provided rough alignment. In this project, faces are detected using the algorithm by Mathias et al. (2014).

C.3.1.2 Face Representation

Face representation heavily influences the performance of the face recognition system and is also the focus of current recognition-related research. In this project, we employ the VGG-Face network (Figure C.2), a 16-layer CNN that is trained on over 2 million celebrity images. In addition to its outstanding performance in benchmarks, we chose it over other CNNs because (i) the image dataset that VGG-Face used is similar to ours, which makes its performance more reliable in our application, and (ii) the pre-trained VGG-Face model is available in the Caffe (Jia et al., 2014) Zoo Library, which makes it easy to use. Using VGG-Face, we are able to extract suitable image features from the output of the fc-6 layer and use them in our subsequent classification stage. In this way, each 224×224 image is represented by a 4,096-dimensional feature vector.

C.3.1.3 Classification

Given the extracted features, we experimented with various classification algorithms. With half the dataset used for training, Support Vector Machines and Linear Regression yielded poor accuracy (below 20%), whereas K-Nearest-Neighbor (KNN) classification achieved 90% accuracy with $K = 30$. We determined that the performance gap is attributable to the "curse of dimensionality". Furthermore, maintaining only a single frontal image per person in the gallery G results in an improvement over KNN. Accordingly, we decided to use the Nearest-Neighbor (NN) classifier. Given the feature vector θ associated with an unknown image, NN will compute its Euclidean distance from each of the feature vectors θ_i stored in the gallery G and output the identification of the image as

$$\text{ID} = \arg\min_{\theta_i \in G} \parallel \theta - \theta_i \parallel . \tag{C.1}$$

We also used Euclidean distance in the subsequent behavioral model.

Figure C.3: A sequence of 9 images of the same person

C.3.2 Behavior Modeling

In active face recognition scenarios, human behaviors can be roughly categorized into three types: change viewpoint, greet, and ignore. If a person of interest is likely to be someone we know, but we are uncertain, we will seek a better position from which to observe the subject's face until we can confidently recognize the subject or relegate the subject a stranger, at which point we would choose to greet or ignore the subject, respectively. Hence, our controller module is designed to model such behaviors based on the results of facial image recognition.

The controller module is initialized with two distance thresholds, t_1 and t_2. Given the output of the recognition module, the controller can model the aforementioned behaviors in the following simple way:

$$\text{Behavior} = \begin{cases} \textit{Greet} & \text{if } d \leq t_1, \\ \textit{Ignore} & \text{if } d \geq t_2, \\ \textit{ChangeView} & \text{if } t_1 < d < t_2, \end{cases} \tag{C.2}$$

where $d = \min_{\theta_i \in G} \parallel \theta - \theta_i \parallel$. In the first and second cases, our system is confident that the subject is a friend or a stranger, respectively, while in the third case it must acquire more information via a change in viewpoint.

C.4 Experiments

C.4.1 Experimental Setup

We used Caffe (Jia et al., 2014) to implement our recognition module. It is one of most popular neural network frameworks and is widely used by many large-scale computer vision

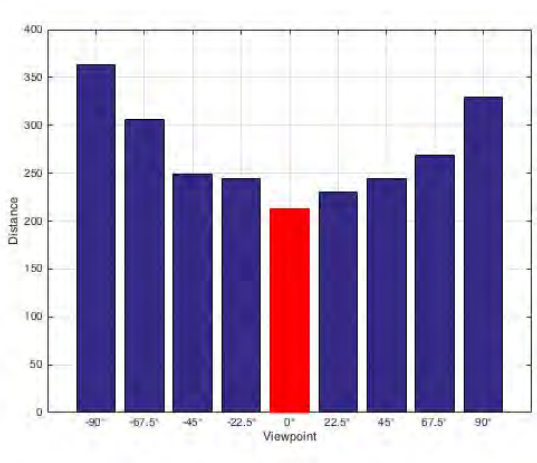


Figure C.4: Distance for different viewpoints

applications. Additionally, Caffe provides various popular pre-trained neural networks in its Model Zoo. Considering the high computational costs of CNNs and the complexity of installing Caffe, we decided to develop this project on Elastic Compute Cloud (EC2), which provides a scalable computing capability on Amazon Web Service (AWS), and we used the g2.2xlarge GPU instance, which has Caffe already installed, that takes advantage of the high-performance parallel processing capabilities of NVIDIA GPUs.

C.4.2 Dataset

The PIE facial image database used in this project was collected by Carnegie Mellon University's Robotics Institute (Sim et al., 2003). It contains 41,368 images of 68 people taken under 43 different illumination conditions, 4 different facial expressions, and 13 imaging viewpoints (from 13 cameras) ranging from -90° to 90°. Among the 13 cameras, 9 are positioned at head height in an arc from approximately a full left profile to a full right profile. As shown in Figure C.3, images acquired from these 9 viewpoints may be used to model a sequence of faces seen by an active observer.

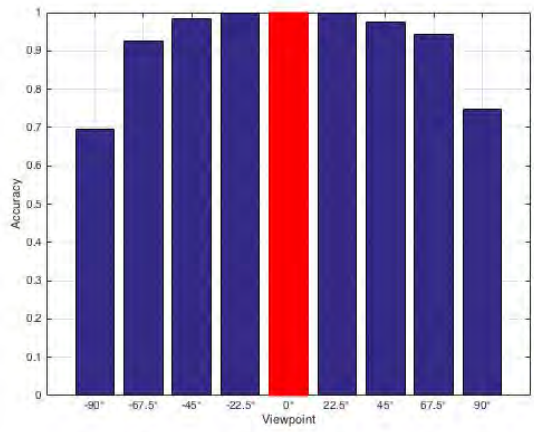


Figure C.5: Accuracy for different viewpoints

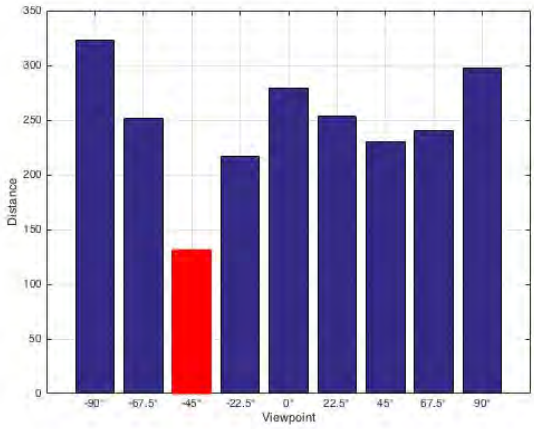


Figure C.6: Distance for different viewpoints for side-view gallery

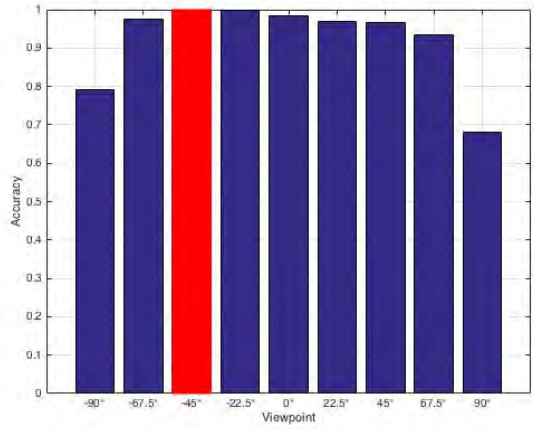


Figure C.7: Accuracy for different viewpoints for side-view gallery

C.4.3 Face Recognition Module Performance

To evaluate the performance of our recognition module, we reserved one frontal image of each person, saved its VGG-Face feature vector in the gallery and used the remaining images for testing. Figure C.4 and Figure C.5 show that the accuracy and distance are view-dependent, as expected. For images with angles closer to the frontal view, the accuracy can reach 100% and the distance is also minimized. This shows that views can be represented by the features extracted by VGG-Face and also provides evidence for the view-dependent design of our behavior model.

To investigate further the influence of the gallery on the results, we changed the stored feature vectors in the gallery from frontal views to -45° side views. Figure C.6 and Figure C.7 show that the optimal view (the view that results in the highest accuracy and minimal distance) changes accordingly. Also, due to the similarity between the left and right side views, a distance local minimum occurs at 45° and the accuracy remains roughly the same for angles near -45° and 45°. This is reasonable because the images most familiar to the active observer are the images (feature vectors) in memory—the gallery.

To test how our AcFR system works when observing strangers, we removed 10 people randomly from the gallery to relegate them strangers. For these strangers, the average distances under different views range from 286 to 350, which are close to the distance of the worst views at -90° and 90°. This shows that our system is able to distinguish strangers from friends. Thus, for properly set thresholds in the behavior controller module, it exhibits appropriate behavioral modeling.

C.4.4 Behavior Controller Module Performance

For each of the 68 subjects in the PIE database, we choose a sequence of images, one per viewpoint angle, under one specific illumination condition (Figure C.3), albeit sometimes with different expressions. This is because viewpoint changes usually happen quickly, which means that illumination is likely to remain unchanged, although expression may change. Then our AcFR system plays the role of an active observer with the feature vectors associated

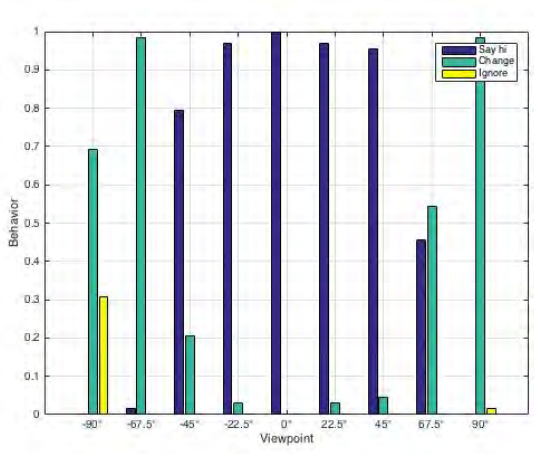


Figure C.8: Behaviors for different initial viewpoints

with the frontal images of all 68 subjects in memory. We tested the active observer's first reaction when acquiring images from different viewpoint angles.

Modifying the thresholds in our behavior model changes the "personality" of our AcFR system. As expected, the system greets the subject more often for larger t_1, and ignores the subject more often for smaller t_2. By default, we set the thresholds to $t_1 = 250$ and $t_2 = 325$.

Figure C.8 shows the behaviors when the initial observation is at different angles. Obviously, the frontal view is the best from which the observer can immediately recognize the subject. By contrast, the full left profile and full right profile are the most difficult views to recognize and they sometimes even lead to incorrect results. This mimics human active face recognition in the real world.

C.4.5 Time Complexity

Computational efficiency is an important concern for us because large latencies undermine real-time behavioral modeling. The initialization of our AcFR system takes approximately 2.2 seconds and the recognition of each image takes only about 0.067 seconds. This implies that our system is capable of performing real-time recognition, which makes it suitable for virtual human modeling applications.

C.5 Summary, Limitations, and Future Work

We have proposed an approach to simulating human behaviors in facial recognition. Motivated by real-life face recognition scenarios and related psychological findings, we assumed that the recognition strategy should be active; therefore, the controller module in our prototype Active Face Recognition (AcFR) system was designed to perform facial recognition in a view-driven sequential manner. Our use of a convolutional neural network makes the recognition module of the AcFR system more biomimetic and more powerful compared to alternative approaches. The experimental results support our design decisions.

The direction of movement when our AcFR system decides to change its viewpoint was not carefully investigated. This can be a problem because the system may decide to move from a side view to look at the subject from behind, whereas people normally move towards the front of a subject in order to see the face more clearly. Therefore, the direction of the face must be estimated for more realistic active face recognition.

In future work, we will incorporate our AcFR system into a biomechanical human model of the face-head-neck complex (Nakada and Terzopoulos, 2015). To this end, the behavioral repertoire of the system's controller module will need to be expanded.

Aloimonos, J., Weiss, I., and Bandopadhay, A. (1988). Active vision. *International Journal of Computer Vision*, 1(4):333–356. 87

Andrews, T. J. and Ewbank, M. P. (2004). Distinct representations for facial identity and changeable aspects of faces in the human temporal lobe. *Neuroimage*, 23(3):905–913. 88

Bajcsy, R. (1988). Active perception. *Proceedings of the IEEE*, 76(8):966–1005. 87

Ballard, D. H. (1991). Animate vision. *Artificial Intelligence*, 48(1):57–86. 87

Bengio, Y., Lamblin, P., Popovici, D., and Larochelle, H. (2007). Greedy layer-wise training of deep networks. In *Advances in Neural Information Processing Systems 19 (NIPS'07)*, pages 153–160. 73

Bergstra, J., Breuleux, O., Bastien, F., Lamblin, P., Pascanu, R., Desjardins, G., Turian, J., Warde-Farley, D., and Bengio, Y. (2010). Theano: A CPU and GPU math compiler in Python. In *Proc. 9th Python in Science Conference*, pages 1–7. 43

Bolduc, M. and Levine, M. D. (1998). A review of biologically motivated space-variant data reduction models for robotic vision. *Computer Vision and Image Understanding*, 69(2):170–184. 10

Bonmassar, G. and Schwartz, E. (1994). Geometric invariance in space-variant vision systems: the exponential chirp transform. In *Pattern Recognition, 1994. Vol. 3-Conference C: Proc. 12th IAPR International Conference on Signal Processing*, pages 204–207. IEEE. 10

Curcio, C. A., Sloan, K. R., Kalina, R. E., and Hendrickson, A. E. (1990). Human photoreceptor topography. *Journal of Comparative Neurology*, 292(4):497–523. 9

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, 68

Eglen, S. J. (2012). Cellular spacing: Analysis and modelling of retinal mosaics. In *Computational Systems Neurobiology*, pages 365–385. Springer. 9

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, New York, New York, USA. ACM Press. 6

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

Featherstone, R. (2014). *Rigid Body Dynamics Algorithms*. Springer. 21

Fukushima, K. (1988). Neocognitron: A hierarchical neural network capable of visual pattern recognition. *Neural networks*, 1(2):119–130. 11

Galli-Resta, L. (1998). Patterning the vertebrate retina: The early appearance of retinal mosaics. *Seminars in Cell and Developmental Biology*, 9(3):279–284. 9

Glorot, X. and Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. In *Proc. International Conference on Artificial Intelligence and Statistics*, volume 9, pages 249–256. 75

Gonzalez, R. V., Buchanan, T. S., and Delp, S. L. (1997). How muscle architecture and moment arms affect wrist flexion-extension moments. *Journal of Biomechanics*, 30(7):705–712. 17

Grady, L. J. (2004). *Space-Variant Computer Vision: A Graph-Theoretic Approach*. PhD thesis, Boston University. 10

He, K., Zhang, X., Ren, S., and Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proc. IEEE International Conference on Computer Vision*, pages 1026–1034. 25, 35

Hodgins, J. K., Wooten, W. L., Brogan, D. C., and O'Brien, J. F. (1995). Animating human athletics. In *Proc. 22nd Annual Conference on Computer Graphics and Interactive Techniques - SIGGRAPH '95*, pages 71–78, New York, New York, USA. ACM Press. 6

Hughes, A. (1977). The topography of vision in mammals of contrasting life style: Comparative optics and retinal organisation. In *The Visual System in Vertebrates*, pages 613–756. Springer. 10

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. ACM Press. 6, 68

Itti, L., Dhavale, N., and Pighin, F. (2003). Realistic avatar eye and head animation using a neurobiological model of visual attention. Technical report, Department of Computer Science, University of Southern California, Los Angeles. 9

Jia, Y., Shelhamer, E., Donahue, J., Karayev, S., Long, J., Girshick, R., Guadarrama, S., and Darrell, T. (2014). Caffe: Convolutional architecture for fast feature embedding. In *Proc. ACM International Conference on Multimedia*, pages 675–678. 91, 92

Kähler, K., Haber, J., Yamauchi, H., and Seidel, H. (2002). Head shop: Generating animated head models with anatomical structure. In *Proc. 2002 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 55–63. 6, 68

Kepler, J. and Linz, U. (2004). Biomechanical modelling of the human eye. *Netzwerkfür Forschung, Lehreund Praxs, Linz*, 231. 7

Kingma, D. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*. 25, 35

Lawrence, S., Giles, C. L., Tsoi, A. C., and Back, A. D. (1997). Face recognition: A convolutional neural-network approach. *IEEE Transactions on Neural Networks*, 8(1):98–113. 89

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324. 11

Lee, S.-H., Sifakis, E., and Terzopoulos, D. (2009). Comprehensive biomechanical modeling and simulation of the upper body. *ACM Transactions on Graphics*, 28(4):99:1–17. 1, 6, 7, 8, 12, 16, 18, 19, 20, 22, 68, 72

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. 1, 6, 7, 8, 19, 68, 69, 70, 71, 72, 73, 76, 77, 78, 80, 82, 85, 86

Lee, S. P., Badler, J. B., and Badler, N. I. (2002). Eyes alive. *ACM Transactions on Graphics (TOG)*, 21(3):637–644. 9

Lee, Y., Terzopoulos, D., and Waters, K. (1995a). Realistic modeling for facial animation. In *Proc. 22nd Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '95)*, volume 95, pages 55–62. 6, 68, 80

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

Lu, C. and Tang, X. (2014). Surpassing human-level face verification performance on LFW with GaussianFace. *arXiv Preprint arXiv:1404.3840*. 89

Mathias, M., Benenson, R., Pedersoli, M., and Van Gool, L. (2014). Face detection without bells and whistles. In *Proc. European Conference on Computer Vision (ECCV)*, pages 720–735. 90

Nakada, M. and Terzopoulos, D. (2015). Deep learning of neuromuscular control for biomechanical human animation. In *Proc. International Symposium on Visual Computing*, Lecture Notes in Computer Science, Vol. 9474, pages 339–348. Springer. 8, 67, 97

Nakada, M., Wang, H., and Terzopoulos, D. (2017). AcFR: Active face recognition using convolutional neural networks. In *Proc. Third IEEE Workshop on Vision Meets Cognition: Functionality, Physics, Intentionality, and Causality (FPIC)*, pages 35–40, Honolulu, HI. 65, 87

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, 68

Nirenberg, S., Carcieri, S. M., Jacobs, A. L., and Latham, P. E. (2001). Retinal ganglion cells act largely as independent encoders. *Nature*, 411(6838):698–701. 10

Parkhi, O. M., Vedaldi, A., and Zisserman, A. (2015). Deep face recognition. In *Proc. British Machine Vision Conference (BMVC)*, pages 41:1–12. 89

Rabie, T. F. and Terzopoulos, D. (2000). Active perception in virtual humans. In *Proc. Vision Interface 2000*, pages 16–22, Montreal, Canada. 8

Samaria, F. S. and Harter, A. C. (1994). Parameterisation of a stochastic model for human face identification. In *Proc. 2nd IEEE Workshop on Applications of Computer Vision*, pages 138–142. IEEE. 61

Schwartz, E. L. (1977). Spatial mapping in the primate sensory projection: Analytic structure and relevance to perception. *Biological Cybernetics*, 25(4):181–194. 10, 61

Shirley, P. and Morley, R. K. (2003). *Realistic Ray Tracing*. A. K. Peters, Ltd., Natick, MA, USA, 2 edition. 31

Si, W. (2013). *Realistic Simulation and Control of Human Swimming and Underwater Movement*. PhD thesis, University of California, Los Angeles, Computer Science Department. 6

Si, W., Lee, S.-H., Sifakis, E., and Terzopoulos, D. (2014). Realistic biomechanical simulation and control of human swimming. *ACM Transactions on Graphics*, 34(1):10:1–15. 6, 68, 70, 72

Sifakis, E., Neverov, I., and Fedkiw, R. (2005a). Automatic determination of facial muscle activations from sparse motion capture marker data. *ACM Transactions on Graphics (TOG)*, 24(3):417–425. 6, 7, 68

Sifakis, E., Neverov, I., and Fedkiw, R. (2005b). Automatic determination of facial muscle activations from sparse motion capture marker data. *ACM Transactions on Graphics (TOG)*, 1(212):417–425. 6, 68

Sim, T., Baker, S., and Bsat, M. (2003). The CMU pose, illumination, and expression database. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25(12):1615–1618. 93

Sivic, J., Everingham, M., and Zisserman, A. (2005). Person spotting: Video shot retrieval for face sets. In *Proc. International Conference on Image and Video Retrieval*, pages 226–236. 89

Sueda, S., Kaufman, A., and Pai, D. K. (2008). Musculotendon simulation for hand animation. *ACM Transactions on Graphics (TOG)*, 27(3):83. 6, 7, 68

Taigman, Y., Yang, M., Ranzato, M., and Wolf, L. (2014). Deepface: Closing the gap to human-level performance in face verification. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1701–1708. 89

Terzopoulos, D. and Rabie, T. F. (1995). Animat vision: Active vision with artificial animals. In *Proc. Fifth International Conference on Computer Vision (ICCV'95)*, pages 840–845, Cambridge, MA. 8

Terzopoulos, D. and Rabie, T. F. (1997). Animat vision: Active vision in artificial animals. *Videre: Journal of Computer Vision Research*, 1(1):2–19. 87

Tokutake, Y. and Freed, M. A. (2008). Retinal ganglion cells–spatial organization of the receptive field reduces temporal redundancy. *European Journal of Neuroscience*, 28(5):914–923. 10

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, 68

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, 68

Wandell, B. A. (1995). *Foundations of Vision*. Sinauer Associates, Sunderland, MA. 10

Wang, G., Tanifuji, M., and Tanaka, K. (1998). Functional architecture in monkey inferotemporal cortex revealed by in vivo optical imaging. *Neuroscience Research*, 32(1):33–46. 88

Waters, K. (1987). A muscle model for animation three-dimensional facial expression. *Computer Graphics (Proc. ACM SIGGRAPH 87)*, 21(4):17–24. 6

Wolf, L., Hassner, T., and Maoz, I. (2011). Face recognition in unconstrained videos with matched background similarity. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 529–534. 89

Yeo, S., Lesmana, M., Neog, D., and Pai, D. (2012). Eyecatch: Simulating visuomotor coordination for object interception. *ACM Transactions on Graphics (TOG) - SIGGRAPH 2012 Conference Proceedings*, 31(4):1–10. 9